Package: ldt (via r-universe)

October 5, 2024

Title Automated Uncertainty Analysis

Version 0.5.3

Description Methods and tools for model selection and multi-model inference (Burnham and Anderson (2002) <doi:10.1007/b97636>, among others). 'SUR' (for parameter estimation), 'logit'/'probit' (for binary classification), and 'VARMA' (for time-series forecasting) are implemented. Evaluations are both in-sample and out-of-sample. It is designed to be efficient in terms of CPU usage and memory consumption.

License GPL (>= 3)

URL https://github.com/rmojab63/LDT

VignetteBuilder knitr

Encoding UTF-8

SystemRequirements C++17

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Imports Rcpp, tdata, Rdpack (>= 0.7), MASS

Suggests knitr, testthat, rmarkdown, kableExtra, moments, systemfit

LinkingTo BH, Rcpp

Config/testthat/edition 3

LazyData true

RdMacros Rdpack

NeedsCompilation yes

Author Ramin Mojab [aut, cre], Stephen Becker [cph] (BSD 3-clause license. Original code for L-BFGS-B algorithm. The L-BFGS-B algorithm was written in the 1990s (mainly 1994, some revisions 1996) by Ciyou Zhu (in collaboration with R.H. Byrd, P. Lu-Chen and J. Nocedal). L-BFGS-B Version 3.0 is an algorithmic update from 2011, with coding changes by J. L. Morales)

Maintainer Ramin Mojab <rmojab63@gmail.com>

Repository CRAN

Date/Publication 2024-07-06 15:10:02 UTC

Contents

`adjust_indices_after_remove` *Adjust Indices in a List*

Description

This function adjusts a list of indices after certain indices have been removed. The new indices will point to the same elements as the original indices. If an index is removed, it will also be removed from the indices list.

Usage

```
adjust_indices_after_remove(indicesList, removedIndices)
```

Arguments

`indicesList`	A list of integer vectors, each representing a set of indices.
`removedIndices`	A vector of integers representing the indices to be removed.

Value

A list of adjusted indices. Each set of indices is adjusted separately.

`AIC.ldt.estim` *Akaike Information Criterion*

Description

This function extracts Akaike information criterion from an `ldt.estim` object.

Usage

```
## S3 method for class 'ldt.estim'
AIC(object, ..., k = NA)
```

Arguments

`object`	An object of class `ldt.estim`
`...`	Additional arguments.
`k`	Unused parameter.

Value

The value of AIC for the whole system. #' @importFrom stats AIC

`BIC.ldt.estim` *Bayesian Information Criterion*

Description

This function extracts Baysian information criterion from an `ldt.estim` object.

Usage

```
## S3 method for class 'ldt.estim'
BIC(object, ...)
```

Arguments

`object`	An object of class `ldt.estim`
`...`	Additional arguments.

Value

The value of BIC for the whole system.

---

boxCoxTransform *Box-Cox Transformation of Numeric Matrix*

Description

This function applies the Box-Cox transformation to the columns of a numeric matrix.

Usage

```
boxCoxTransform(data, lambdas, ...)
```

Arguments

`data`	A numeric matrix to be transformed.
`lambdas`	A numeric vector, a single number, NA, or NULL indicating the lambda parameter(s) for the Box-Cox transformation. Use `NULL` for no transformation, `NA` for estimating the lambda parameter for each variable, a single number for equal lambda parameter for all variables, and a numeric vector for distinct lambda parameters for the corresponding variables.
`...`	additional parameters for `MASS::boxcox` function.

Value

`data`	transformed data
`lambdas`	final lambda vector used in the calculations.

Examples

```
data <- matrix(rnorm(40), ncol = 2)
result <- ldt:::boxCoxTransform(data, c(0.5, 0.5))
```

coef.ldt.estim *Extract Coefficients Matrix*

Description

This function extracts coefficient matrix from an `ldt.estim` object.

Usage

```
## S3 method for class 'ldt.estim'
coef(object, equations = NULL, removeZeroRest = FALSE, ...)
```

Arguments

`object`	An object of class `ldt.estim`
`equations`	A number, a numeric array or a string array specifying the indices or names of the endogenous variables in the equations. `NULL` means all equations.
`removeZeroRest`	If `TRUE` and model is restricted, zero restrictions are removed.
`...`	Additional arguments.

Value

If zero restrictions are not removed, it is a matrix containing the coefficients of the system. Each column of the matrix belongs to an equation. Explanatory variables are in the rows. Otherwise, coefficients of different equations are reported in a list.

coefs.table *Create Table of Coefficients*

Description

This function summarizes a list of estimated models (output of `estim.?` functions) and creates a table of coefficients.

Usage

```
coefs.table(
  estimList,
  depList = NULL,
  tableFun = "coef_star",
  formatNumFun = NULL,
  regInfo = NULL,
  textFun = NULL,
  textFun_sub = NULL,
  textFun_max = 20,
```

```
  expList = NA,
  latex = TRUE,
  numFormat = "%.2f"
)
```

Arguments

estimList	A named list where each element is output from a `estim.?` function, all belonging to a common analysis.
depList	List of endogenous variable name to be included in the columns of the table. If `NULL`, everything is added.
tableFun	Function with arguments `coef`, `std`, `pvalue`, `minInColm`, `maxInCol` for formatting estimated coefficients. Can also use one of the following character strings for predefined formatting: `sign`, `sign_star`, `coef`, `coef_star`, `coef_star_std`.
formatNumFun	Function to format numbers if `tableFun` uses predefined formatting. It takes two arguments: `colIndex` (determines the column index) and `x` (determines the value).
regInfo	List of keys (such as `num_eq`, `num_dep`, ...) to determine information at bottom of table. Use "" (empty) for empty rows. A list of available options is given in the details section.
textFun	Function to change any text in columns or rows of the table to a more informative text. It has two arguments: `text` and `type`.
textFun_sub	List for replacing special characters. If `NULL`, ’list(c("%", "\\%"), c("", "\\"))’ is used.
textFun_max	Maximum length for texts in the table.
expList	Determines the name of the explanatory variables to insert in table. If `NA`, it inserts all coefficients. If it is an integer, it insert that number of explanatory variables in the table. It can be a list of available explanatory variables. Use `...` for empty rows.
latex	If `TRUE`, default options are for ’latex’, otherwise, ’html’.
numFormat	default formatting for the numbers.

Details

The first part of the table is the header, followed by the coefficients. At the bottom, you can insert the following items by specifying `regInfo`:

- An empty character string (i.e., "") for inserting empty line.
- `"sigma2"` for the covariance of regression, if it is available.
- An available metric name in the row names of `estimList[[...]]$metrics`.

Furthermore, second argument in `textFun` can be:

- `hname`: shows that the text is a header name from the `estimList` elements.
- `dname`: shows that the text is an endogenous variable name from the columns of `coefs` matrix.
- `rname`: shows that the text is a key given in `regInfo`.
- `ename`: shows that the text is an explanatory variable name from the rows of `coefs` matrix.
- `NULL`: shows that the text is a specific code or something else.

Value

A data frame with (formatted) requested information.

Examples

```
# See 'search.?' or 'estim.?' functions for some examples.
```

---

combine.search — *Combine a List of* `ldt.search` *Objects*

Description

Combine a List of `ldt.search` Objects

Usage

```
combine.search(list, method)
```

Arguments

`list`	A list of `ldt.search` objects
`method`	Method in objects

Value

the combined `ldtsearch` object

---

data.berka — *Berka and Sochorova (1993) Dataset for Loan Default*

Description

This dataset is a part of the Berka and Sochorova (1993) study, which contains information on loan defaults. The data was generated using the code in the '/data-raw/data-berka.R' file.

Usage

```
data.berka
```

Format

A list with the following items:

y A vector representing the labels. It contains 1 for default and 0 for non-default. Any observation with default (i.e., 'default' and 'finished with default') is considered to be a positive.

x A matrix with explanatory variables in the columns.

w A vector with the weight of each observation. This is mathematically generated to balance the observations.

descriptions A list that describes the column names.

Source

Berka and Sochorova (1993)

`data.pcp` *IMF's Primary Commodity Prices*

Description

This is a subset of the IMF's Primary Commodity Prices dataset (non-index data is omitted). The data was generated using the code in the '/data-raw/data-pcp.R' file.

Usage

```
data.pcp
```

Format

A list with the following items:

data A data frame with monthly variables in the columns.

descriptions A list that describes the columns of `data`.

datatypes A character array that describes the type of data in the columns of `data`.

start A number that indicates the frequency of the first observation in `data`.

Source

International Commodity Prices (2023)

`data.wdi` *Long-run Growth from World Development Indicator Dataset*

Description

This dataset is derived from the World Development Indicator (WDI) dataset. It contains information on long-run output growth after 2006 and its potential explanatory variables before that year. The data was generated using the code in the '/data-raw/data-wdi.R' file.

Usage

```
data.wdi
```

Format

A list with the following items:

y A vector representing the long-run output growth after 2006. Each element represents a country.

x A matrix with explanatory variables in the columns. Each row represents a country.

splitYear A number that indicates how `y` and `x` are calculated.

names A list of pairs that describe the code and the name of variables in `y` and `x`.

Source

World Bank (2022)

`endogenous` *Extract Endogenous Variable(s) Data*

Description

This function extracts data of a endogenous variable(s) from an estimated model.

Usage

```
endogenous(object, equations = NULL, ...)
```

Arguments

`object`	An object of class `ldt.estim`.
`equations`	A number, a numeric array or a string array specifying the indices or names of the endogenous variables in the equations. `NULL` means all equations.
`...`	Additional arguments.

Value

A data frame containing the endogenous data.

eqList2Matrix *Convert a List of Equations to a Matrix*

Description

This function takes a list of equations and a data frame, and returns a matrix where the response variables are in the first columns, and the predictor variables are in the subsequent columns.

Usage

```
eqList2Matrix(equations, data, addIntercept = FALSE)
```

Arguments

`equations`	A formula or a list of formula objects representing the equations.
`data`	A matrix or a data frame containing the variables in the equations.
`addIntercept`	Logical. If `TRUE`, an intercept column is added after the response variables. Default is `FALSE`.

Details

The function checks for duplicate response variables across equations and throws an error if any are found. It also ensures that predictor variables are not duplicated in the final matrix.

Value

`result`	A matrix with response variables in the first columns, predictor variables in subsequent columns, and optionally an intercept column. The matrix does not contain duplicate columns.
`numResponse`	Number of response variables in the first columns.

Examples

```
data <- data.frame(income = c(50000, 60000, 80000, 85000, 65000),
                   age = c(25, 32, 47, 51, 36),
                   education = c(16, 18, 20, 20, 16),
                   savings = c(20000, 25000, 30000, 35000, 40000))
equations <- list(as.formula("income ~ age + education"),
                  as.formula("savings ~ age + education"))
matrix_data <- ldt:::eqList2Matrix(equations, data, addIntercept = TRUE)
print(matrix_data)
```

estim.bin *Estimate a Binary Choice Model*

Description

Use this function to estimate a binary choice model.

Usage

```
estim.bin(
  data,
  linkFunc = c("logit", "probit"),
  pcaOptionsX = NULL,
  costMatrices = NULL,
  optimOptions = get.options.newton(),
  aucOptions = get.options.roc(),
  simFixSize = 0,
  simTrainFixSize = 0,
  simTrainRatio = 0.75,
  simSeed = 0,
  weightedEval = FALSE,
  simMaxConditionNumber = Inf
)
```

Arguments

`data`	A list that determines data and other required information for the model search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
`linkFunc`	A character string that shows the probability assumption. It can be `logit` or `probit`.
`pcaOptionsX`	A list of options to use principal components of the `x`, instead of the actual values. Set `NULL` to disable. Use `get.options.pca()` for initialization.
`costMatrices`	A list of numeric matrices where each one determines how to score the calculated probabilities. See and use search.bin for more information and initialization.
`optimOptions`	A list for Newton optimization options. Use get.options.newton function to get the options.
`aucOptions`	A list of options for AUC calculation. See and use `[get.options.roc()]` for more information and initialization.
`simFixSize`	An integer that determines the number of out-of-sample simulations. Use zero to disable the simulation.
`simTrainFixSize`	An integer representing the number of data points in the training sample in the out-of-sample simulation. If zero, `trainRatio` will be used.

`simTrainRatio` A number representing the size of the training sample relative to the available size, in the out-of-sample simulation. It is effective if `trainFixSize` is zero.

`simSeed` A seed for the random number generator. Use zero for a random value.

`weightedEval` If `TRUE`, weights will be used in evaluations.

`simMaxConditionNumber`
A number for the maximum value for the condition number in the simulation.

Details

As documented in chapter 12 in Greene and Hensher (2010), binary regression is a statistical technique used to estimate the probability of one of two possible outcomes for a variable such as y, i.e., $p = P(y = 1)$ and $q = P(y = 0)$. The most commonly used binary regression models are the logit and probit models. In general, a binary regression model can be written as $f(p) = z'\gamma + v$, where the first element in γ is the intercept and $f(p)$ is a link function. For logit and probit models we have $f(p) = \ln \frac{p}{1-p}$ and $f(p) = \Phi^{-1}(p)$ respectively, where Φ^{-1} is the inverse cumulative distribution function of the standard normal distribution.

Given an independent sample of length N, the parameters of the binary regression model are estimated using maximum likelihood estimation. Assuming that some observations are more reliable or informative than others and w_i for $i = 1, \ldots, N$ reflects this fact, the likelihood function is given by:

$$L(\gamma) = \prod_{i=1}^{N} (p_i)^{w_i y_i} (1 - p_i)^{w_i(1-y_i)},$$

where $p_i = \frac{\exp \gamma z_i}{1+\exp \gamma z_i}$ for logit model and $p_i = \Phi(\gamma z_i)$ for probit model. `ldt` uses feasible GLS to calculate the initial value of the coefficients and a weighted least squares estimator to calculate the initial variance matrix of the error terms (see page 781 in Greene (2020)). The condition number of the estimation is calculated by multiplying 1-norm of the observed information matrix at the maximum likelihood estimator and its inverse (e.g., see page 94 in Trefethen and Bau (1997)). Furthermore, if x is a new observations for the explanatory variables, the predicted probability of the positive class is estimated by $p_i = \frac{\exp \gamma x}{1+\exp \gamma x}$ for logit model and $p_i = \Phi(\gamma x)$ for probit model.

Note that the focus in `ldt` is model uncertainty and the main purpose of exporting this method is to show the inner calculations of the search process in search.bin function.

References

Greene WH (2020). *Econometric analysis*, 8th edition. Pearson Education Limited, New York. ISBN 9781292231136.

Greene WH, Hensher DA (2010). *Modeling ordered choices: A primer*. Cambridge University Press. ISBN 9780511845062, doi:10.1017/cbo9780511845062.

Trefethen LN, Bau D (1997). *Numerical linear algebra*. Society for Industrial and Applied Mathematics. ISBN 9780898714876.

See Also

search.bin

Examples

```
# Example 1 (simulation, small model):
set.seed(123)
sample <- sim.bin(3L, 100)
print(sample$coef)

data <- data.frame(sample$y, sample$x)

#   Estimate using glm
fit <- glm(Y ~ X1 + X2, data = data, family = binomial())
print(fit)

#   Estimate using 'ldt::estim.bin'
fit <- estim.bin(data = get.data(data = data,
                                 equations = list(Y ~ X1 + X2)),
                  linkFunc = "logit")
print(fit)
plot_data <- plot(fit, type = 1)
#   See 'plot.ldt.estim()' function documentation


# Example 2 (simulation, large model with PCA analysis):
sample <- sim.bin(30L, 100, probit = TRUE)
data <- data.frame(sample$y, sample$x)
colnames(data) <- c(colnames(sample$y),colnames(sample$x))
pca_options <- get.options.pca(ignoreFirst = 1, exactCount = 3)
fit <- estim.bin(data = get.data(cbind(sample$y, sample$x),
                                  endogenous = ncol(sample$y),
                                  addIntercept = FALSE),
                  linkFunc = "probit",
                  pcaOptionsX = pca_options)
print(fit)
plot_data <- plot(fit, type = 2)
```

`estim.binary.model.string` *Get Model Name*

Description

Get Model Name

Usage

```
estim.binary.model.string(object)
```

Arguments

`object`	A `estim.bin` object

Value

model string

---

estim.sur *Estimate a SUR Model*

Description

Use this function to estimate a Seemingly Unrelated Regression model.

Usage

```
estim.sur(
  data,
  searchSigMaxIter = 0,
  searchSigMaxProb = 0.1,
  restriction = NULL,
  pcaOptionsY = NULL,
  pcaOptionsX = NULL,
  simFixSize = 0,
  simTrainFixSize = 0,
  simTrainRatio = 0.75,
  simSeed = 0,
  simMaxConditionNumber = Inf
)
```

Arguments

`data` A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.

`searchSigMaxIter` An integer for the maximum number of iterations in searching for significant coefficients. Use 0 to disable the search.

`searchSigMaxProb` A number for the maximum value of type I error to be used in searching for significant coefficients. If p-value is less than this, it is interpreted as significant.

`restriction` A `km x q` matrix of restrictions where `m` is the number of endogenous data, `k` is the number of exogenous data, and `q` is the number of unrestricted coefficients.

`pcaOptionsY` A list of options to use principal components of the endogenous data, instead of the actual values. Set `NULL` to disable. Use `get.options.pca()` for initialization.

`pcaOptionsX` A list of options to use principal components of the exogenous data, instead of the actual values. Set `NULL` to disable. Use `get.options.pca()` for initialization.

`simFixSize` An integer that determines the number of out-of-sample simulations. Use zero to disable the simulation.

`simTrainFixSize`
: An integer representing the number of data points in the training sample in the out-of-sample simulation. If zero, `trainRatio` will be used.

`simTrainRatio`
: A number representing the size of the training sample relative to the available size, in the out-of-sample simulation. It is effective if `trainFixSize` is zero.

`simSeed`
: A seed for the random number generator. Use zero for a random value.

`simMaxConditionNumber`
: A number for the maximum value for the condition number in the simulation.

Details

As described in section 10.2 in Greene (2020), this type of statistical model consists of multiple regression equations, where each equation may have a different set of exogenous variables and the disturbances between the equations are assumed to be correlated. The general form with m equations can be written as $y_i = z_i'\gamma_i + v_i$ and $E(v_i v_j) = \sigma_{ij}^2$ for $i = 1, \dots m$. Assuming that a sample of N independent observations is available, we can stack the observations and use the following system for estimation:

$$Y = XB + V, \quad \text{vec}B = R\gamma,$$

where the columns of $Y : N \times m$ contain the endogenous variables for each equation and the columns of $X : N \times k$ contain the explanatory variables, with k being the number of unique explanatory variables in all equations. Note that X combines the z_i variables, and the restrictions imposed by $R : mk \times q$ and $\gamma : q \times 1$ determine a set of zero constraints on $B : k \times m$, resulting in a system of equations with different sets of exogenous variables.

Imposing restrictions on the model using the R matrix is not user-friendly, but it is suitable for use in this package, as users are not expected to specify such restrictions, but only to provide a list of potential regressors. Note that in this package, most procedures, including significance search, are supposed to be automated.

The unrestricted estimators (i.e., $\hat{B} = (X'X)^{-1}X'Y$, and $\hat{\Sigma} = (\hat{V}'\hat{V})/N$ where $\hat{V} = Y - X\hat{B}$) are used to initialize the feasible GLS estimators:

$$\tilde{B} = RW^{-1}R'[\hat{V} - 1 \otimes x']\text{vec}Y, \quad \tilde{\Sigma} = (\tilde{V}'\tilde{V})/N,$$

where $W = R'[\hat{V}^{-1} \otimes X'X]R$ and $\tilde{V} = Y - X\tilde{B}$. The properties of these estimators are discussed in proposition 5.3 in Lütkepohl (2005). See also section 10.2 in Greene (2020). The maximum likelihood value is calculated by $-\frac{N}{2}(m(\ln 2\pi + 1) + \ln|\tilde{\Sigma}|)$. The condition number is calculated by multiplying 1-norm of W and its inverse (e.g., see page 94 in Trefethen and Bau (1997)). Furthermore, given an out-of-sample observation such as $x : k \times 1$, the prediction is $y^f = \tilde{B}'x$, and its variance is estimated by the following formula:

$$\text{var}y^f = \tilde{V} + (x' \otimes I_m)RW^{-1}R'(x \otimes I_m).$$

Note that the focus in `ldt` is model uncertainty and for more sophisticated implementations of the FGLS estimator, you may consider using other packages such as `systemfit`.

Finally, note that the main purpose of exporting this method is to show the inner calculations of the search process in search.sur function.

References

Greene WH (2020). *Econometric analysis*, 8th edition. Pearson Education Limited, New York. ISBN 9781292231136.

Lütkepohl H (2005). *New introduction to multiple time series analysis*. Springer, Berlin. ISBN 3540401725, doi:10.1007/9783540277521.

Trefethen LN, Bau D (1997). *Numerical linear algebra*. Society for Industrial and Applied Mathematics. ISBN 9780898714876.

See Also

search.sur

Examples

```
# Example 1 (simulation, small model):
set.seed(123)
sample <- sim.sur(sigma = 2L, coef = 3L, nObs = 100)
print(sample$coef)
print(sample$sigma)

data <- data.frame(sample$y, sample$x)

#    Use systemfit to estimate:
exp_names <- paste0(colnames(sample$x), collapse = " + ")
fmla <- lapply(1:ncol(sample$y), function(i) as.formula(paste0("Y", i, " ~ -1 +", exp_names)))
fit <- systemfit::systemfit(fmla, data = data, method = "SUR")
print(fit)

#    Use 'ldt::estim.sur' function
fit <- estim.sur(data = get.data(cbind(sample$y, sample$x),
                                  endogenous = ncol(sample$y),
                                  addIntercept = FALSE))

# or, by using formula list:
fit <- estim.sur(data = get.data(data = data,
                                 equations = fmla,
                                 addIntercept = FALSE))

print(fit)
print(fit$estimations$sigma)
plot_data <- plot(fit, equation = 1)


# Example 2 (simulation, large model with significancy search):
num_obs <- 100
sample <- sim.sur(sigma = 2L, coef = 3L, nObs = num_obs)
print(sample$coef)

#   create irrelevant data:
```

```
num_x_ir <- 20
x_ir <- matrix(rnorm(num_obs * num_x_ir), ncol = num_x_ir)
data_x <- data.frame(sample$x, x_ir)
colnames(data_x) <- c(colnames(sample$x), paste0("z", 1:num_x_ir))

fit <- estim.sur(data = get.data(cbind(sample$y, data_x),
                                 endogenous = ncol(sample$y),
                                 addIntercept = FALSE),
                 searchSigMaxIter = 100,
                 searchSigMaxProb = 0.05)

print(fit$estimations$coefs)
# coefficient matrix, with lots of zero restrictions

# Example 3 (simulation, large model with PCA):
#   by using data of the previous example
fit <- estim.sur(data = get.data(cbind(sample$y, data_x),
                                 endogenous = ncol(sample$y),
                                 addIntercept = FALSE),
                 pcaOptionsX = get.options.pca(2,4))
print(fit$estimations$coefs)
#  coefficients are: intercept and the first exogenous variable and 4 PCs
```

---

estim.varma *Estimate a VARMA Model*

Description

Use this function to estimate a Vector Autoregressive Moving Average model.

Usage

```
estim.varma(
  data,
  params = NULL,
  seasonsCount = 0,
  lbfgsOptions = get.options.lbfgs(),
  olsStdMultiplier = 2,
  pcaOptionsY = NULL,
  pcaOptionsX = NULL,
  maxHorizon = 0,
  simFixSize = 0,
  simHorizons = NULL,
  simUsePreviousEstim = FALSE,
  simMaxConditionNumber = Inf
)
```

Arguments

`data`	A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
`params`	An integer vector that determines the values for the parameters of the VARMA model: `(p,d,q,P,D,Q)`. If `NULL`, `c(1,0,0,0,0,0)` is used.
`seasonsCount`	An integer value representing the number of observations per unit of time.
`lbfgsOptions`	A list containing L-BFGS optimization options. Use get.options.lbfgs function for initialization.
`olsStdMultiplier`	A number used as a multiplier for the standard deviation of OLS, used for restricting maximum likelihood estimation.
`pcaOptionsY`	A list of options to use principal components of `y`, instead of the actual values. Set to `NULL` to disable. Use `get.options.pca()` for initialization.
`pcaOptionsX`	A list of options to use principal components of `x`, instead of the actual values. Set to `NULL` to disable. Use `get.options.pca()` for initialization.
`maxHorizon`	An integer representing the maximum prediction horizon. Set to zero to disable prediction.
`simFixSize`	An integer that determines the number of out-of-sample simulations. Use zero to disable simulation.
`simHorizons`	An integer vector representing the prediction horizons to be used in out-of-sample simulations. See also `get.search.metrics()`.
`simUsePreviousEstim`	If `TRUE`, parameters are initialized only in the first step of the simulation. The initial values of the n-th simulation (with one more observation) are the estimations from the previous step.
`simMaxConditionNumber`	A number representing the maximum value for the condition number in simulation.

Details

The VARMA model can be used to analyze multivariate time series data with seasonal or non-seasonal patterns. According to Lütkepohl (2005), it considers interdependencies between the series, making it a powerful tool for prediction. The specification of this model is given by:

$$\Delta^d \Delta_s^D y_t = c + \sum_{i=1}^{p} A_i y_{t-i} + \sum_{i=1}^{q} B_i \epsilon_{t-i} + C x_t + \sum_{i=1}^{P} A_{is} y_{t-is} + \sum_{i=1}^{Q} B_{is} v_{t-is} + v_t,$$

where $y_t : m \times 1$ is the vector of endogenous variables, $x_t : k \times 1$ is the vector exogenous variables, s is the number of seasons and (p, d, q, P, D, Q) determines the lag structure of the model. Furthermore, c, C, A_i and B_i for all available i determines the model's parameters. v_t is the disturbance vector and is contemporaneously correlated between different equations, i.e., $E(v_t v_t') = \Sigma$. Given a sample of size T, the model can be estimated using maximum likelihood estimation. However, to ensure identifiability, it is necessary to impose additional constraints on the parameters (see chapter 12 in Lütkepohl (2005)). In this function, diagonal MA equation form is used (see Dufour and

Pelletier (2022)). In this function, the feasible GLS estimator is used to initialize the maximum likelihood, and the OLS estimator is used to calculate the initial value of the variance matrix of the error term. The condition number is calculated similar to the other models (see estim.sur or e.g., page 94 in Trefethen and Bau (1997)). Furthermore, given a prediction horizon and required exogenous data, prediction is performed in a recursive schema, in which the actual estimated errors are used if available and zero otherwise. The variance of the predictions is also calculated recursively. Note that this function does not incorporate the coefficients uncertainty in calculation of the variance (see section 3.5 in Lütkepohl (2005)).

Finally, note that the main purpose of exporting this method is to show the inner calculations of the search process in search.varma function.

Value

A nested list with the following items:

`counts`	Information about different aspects of the estimation such as the number of observation, number of exogenous variables, etc.
`estimations`	Estimated coefficients, standard errors, z-statistics, p-values, etc.
`metrics`	Value of different goodness of fit and out-of-sample performance metrics.
`prediction`	Information on the predicted values.
`simulation`	Information on the simulations.
`info`	Some other general information.

References

Dufour J, Pelletier D (2022). "Practical methods for modeling weak VARMA processes: Identification, estimation and specification with a macroeconomic application." *Journal of Business & Economic Statistics*, **40**(3), 1140–1152. doi:10.1080/07350015.2021.1904960.

Lütkepohl H (2005). *New introduction to multiple time series analysis*. Springer, Berlin. ISBN 3540401725, doi:10.1007/9783540277521.

Trefethen LN, Bau D (1997). *Numerical linear algebra*. Society for Industrial and Applied Mathematics. ISBN 9780898714876.

See Also

search.varma

Examples

```
# Example 1 (simulation, ARMA):
num_eq <- 1L
num_ar <- 2L
num_ma <- 1L
num_exo <- 1L
sample <- sim.varma(num_eq, arList = num_ar, maList = num_ma, exoCoef = num_exo, nObs = 110)
# estimate:
```

```
fit <- estim.varma(data = get.data(cbind(sample$y, sample$x)[1:100,],
                                   endogenous = num_eq,
                                   newData = sample$x[101:110,, drop=FALSE]),
                   params = c(num_ar, 0, num_ma, 0, 0, 0),
                   maxHorizon = 10,
                   simFixSize = 5,
                   simHorizons = c(1:10))
print(fit)

pred <- predict(fit, actualCount = 10)
plot(pred, simMetric = "mape")

# split coefficient matrix:
get.varma.params(fit$estimations$coefs, numAR = num_ar, numMA = num_ma, numExo = num_exo)

# Example 2 (simulation, VARMA):
num_eq <- 3L
num_ar <- 2L
num_ma <- 1L
num_ma <- 1L
num_exo <- 2L
sample <- sim.varma(num_eq, arList = num_ar, maList = num_ma, exoCoef = num_exo, nObs = 110)
# estimate:
fit <- estim.varma(data = get.data(cbind(sample$y, sample$x)[1:100,],
                                   endogenous = num_eq,
                                   newData = sample$x[101:110,]),
                   params = c(num_ar, 0, num_ma, 0, 0, 0),
                   maxHorizon = 10,
                   simFixSize = 5,
                   simHorizons = c(1:10))

pred <- predict(fit, actualCount = 10)
plot(pred, simMetric = "mape")

# split coefficient matrix:
get.varma.params(fit$estimations$coefs, numAR = num_ar, numMA = num_ma, numExo = num_exo)
```

---

estim.varma.model.string

Get the Specification of an `ldt.estim.varma` *Model*

Description

Use this function to get the name of a VARMA model, such that: If It is multivariate, it will be VAR, otherwise AR; If moving average terms are present, it will be ARMA or VARMA; If it is seasonal, it will be S-ARMA or S-VARMA; If it is integrated, it will be S-ARMA (D=?,d=?); ..., and any possible combination. Parameters will be reported in parenthesis after the name of the model.

Usage

```
estim.varma.model.string(obj)
```

Arguments

`obj` AN object of class `ldt.estim.varma`.

Value

A character string representing the specification of the model.

`exogenous` *Extract Exogenous Variable(s) Data*

Description

This function extracts data of an exogenous variable(s) in an equation from an estimated model. It takes zero restrictions imposed into account.

Usage

```
exogenous(object, equation = 1, ...)
```

Arguments

`object`	An object of class `ldt.estim`.
`equation`	A number or a string specifying the equation with exogenous data.
`...`	Additional arguments.

Value

A matrix containing the exogenous data.

`fan.plot` *Fan Plot Function*

Description

This function creates a fan plot.

Usage

```
fan.plot(
  data,
  dist = "normal",
  lambda = NA,
  quantiles = c(0.05, 0.1, 0.25, 0.75, 0.9, 0.95),
  gradient = FALSE,
  ylimSuggest = c(NA, NA),
  ylimExpand = 0.1,
  newPlot = TRUE,
  boundColor = "blue",
  plotArgs = list(),
  actualArgs = list(),
  medianArgs = list(),
  polygonArgs = list(border = NA)
)
```

Arguments

`data`	A matrix where columns represent the parameters of distributions. E.g., for normal distribution, the columns will be `mean` and `variance`. The first rows can be actual values given in the first column.
`dist`	A string indicating the type of distribution. Currently, it can be either "normal" or "log-normal". Default is "normal".
`lambda`	A numeric value for Box-Cox transformation. If `NA`, no transformation is applied. Default is `NULL`.
`quantiles`	A numeric vector of quantiles for shading. Default is c(0.05, 0.1, 0.25, 0.75, 0.9, 0.95).
`gradient`	A logical value indicating whether to create a gradient fan plot. If FALSE, a standard fan plot is created. Default is FALSE.
`ylimSuggest`	A numeric vector of length 2 indicating the suggested y-axis limits. Use `NA` for automatic calculation. Default is c(NA, NA).
`ylimExpand`	A numeric value indicating the proportion to expand the y-axis limits. Default is 0.1.
`newPlot`	A logical value indicating whether to create a new plot. If FALSE, the fan plot is added to the existing plot. Default is TRUE.
`boundColor`	A string indicating the color of the boundary of the fan plot. Default is "blue".
`plotArgs`	A list of additional arguments passed to the plot function when creating a new plot.
`actualArgs`	A list of additional arguments passed to the lines function when plotting actual values.
`medianArgs`	A list of additional arguments passed to the lines function when plotting median values.
`polygonArgs`	A list of additional arguments passed to the polygon function when creating the fan plot.

Value

This function does not return a value but creates a fan plot as a side effect.

---

fitted.ldt.estim — *Extract Fitted Data*

Description

This function calculates and returns fitted values for an `ldt.estim` object.

Usage

```
## S3 method for class 'ldt.estim'
fitted(object, equations = NULL, ...)
```

Arguments

`object`	An object of class `ldt.estim`.
`equations`	A number, a numeric array or a string array specifying the equations with residual data. If `NULL`, residuals in all equations are returned.
`...`	Additional arguments.

Value

A matrix containing the exogenous data.

---

get.combinations — *Define Combinations for Search Process*

Description

This function defines a structure for a two-level nested loop used in a model search (or screening) process. The outer loop is defined by a vector of sizes and all the combinations of the variables are generated automatically. The inner loop is defined by a list of predefined combinations of the variables. Each variable can belong to either endogenous or exogenous variables based on their usage.

Usage

```
get.combinations(
  sizes = c(1),
  partitions = NULL,
  numFixPartitions = 0,
  innerGroups = list(c(1)),
  numTargets = 1,
  stepsNumVariables = c(NA),
  stepsFixedNames = NULL,
  stepsSavePre = NULL
)
```

Arguments

- `sizes` A numeric vector or a list of numeric vectors that determines the sizes of outer loop combinations. For example, if the outer loop belongs to the endogenous variables, `c(1, 2)` means all models with 1 and 2 equations. If the outer loop belongs to exogenous variables, `c(1,2)` means all regressions with 1 and 2 exogenous variables. It can also be a list of numeric vectors for step-wise search. Each vector determines the size of the models in a step. In the next step, a subset of potential variables is selected by using `stepsNumVariables` argument.
- `partitions` A list of numeric vectors or character vectors that partitions the outer loop variables. No model is estimated with two variables from the same partition.
- `numFixPartitions` A single number that determines the number of partitions at the beginning of `partitions` to be included in all models.
- `innerGroups` A list of numeric vectors or character vectors that determines different combinations of the variables for the inner loop. For example, if the inner loop belongs to exogenous data, `list(c(1), c(1, 2))` means estimating all models with just the first exogenous variable and all models with both first and second exogenous variables.
- `numTargets` An integer for the number of target variables at the first columns of the data matrix. Results of a search process are specific to these variables. A model is not estimated if it does not contain a target variable.
- `stepsNumVariables` A numeric vector. If `sizes` is a list (i.e., a step-wise search), this vector must be of equal length and determines the number of variables (with best performance) in each step.
- `stepsFixedNames` A character vector. If `sizes` is a list (i.e., a step-wise search), this vector determines the name of variables to be included in all steps.
- `stepsSavePre` A name for saving and loading progress, if `sizes` is a list. Each step's result is saved in a file (name=`paste0(stepsSavePre,i)`) where `i` is the index of the step.

Details

The `get.combinations` function in the `ldt` package uses a two-level nested loop to iterate over different combinations of endogenous and exogenous variables. This is similar to running the following code:

```
for (endo in list(c(1), c(1, 2)))
  for (exo in list(c(1), c(1, 2)))
    Estimate a model using \code{endo} and \code{exo} indexation
```

However, predefining both loops is not memory efficient. Therefore, `ldt` uses a running algorithm to define the outer loop. It asks for the desired size of endogenous or exogenous variables in the model (i.e., `sizes`) and creates the outer groups using all possible combinations of the variables. The `partitions` and `numFixPartitions` parameters can be used to restrict this set.

For the inner loop, you must provide the desired combination of variables (endogenous or exogenous). Given `m` as the number of variables, you can generate all possible combinations using the following code:

```
m <- 4
combinations <- unlist(lapply(1:m, function(i) {
 t(combn(1:m, i, simplify = FALSE))
}), recursive = FALSE)
```

You can use this as the `innerGroups` argument. However, this might result in a large model set.

Note that in `ldt`, if the data matrix does not have column names, default names for the endogenous variables are `Y1, Y2, ...`, and default names for the exogenous variables are `X1, X2, ...`. See `get.data()` function for more details.

Also note that `ldt` ensure that all possible models can be estimated with the given number of partitions and sizes. If it's not possible, it will stop with an error message.

Value

A list suitable for use in `ldt::search.?` functions. The list contains:

- `sizes` The sizes of outer loop combinations.
- `partitions` The partitions of outer loop variables.
- `numFixPartitions` The number of fixed partitions at the beginning.
- `innerGroups` Different combinations of variables for inner loop.
- `numTargets` The number of target variables at first columns.
- `stepsNumVariables` The number of variables in each step for step-wise search.
- `stepsFixedNames` The names of fixed variables in each step for step-wise search.
- `stepsSavePre` The name for saving and loading progress for step-wise search.

Examples

```
# Some basic examples are given in this section. However, more practical examples are available
# for the \code{search.?} functions.

# Example 1:
combinations1 <- get.combinations(sizes = c(1, 2))
# The function will generate all possible combinations of sizes 1 and 2.

# Example 2: Using partitions
combinations2 <- get.combinations(sizes = c(1, 2), partitions = list(c(1, 2), c(3, 4)))

# Here, we're specifying partitions for the variables.
# The function will generate combinations such that no model is estimated with two variables
# from the same partition.

# Example 3: Specifying inner groups
combinations3 <- get.combinations(sizes = c(1, 2), innerGroups = list(c(1), c(1, 2)))

# In this example, we're specifying different combinations of variables for the inner loop.
# For instance, \code{list(c(1), c(1, 2))} means estimating all models with just the first
# variable and all models with both first and second variables.

# Example 4: Step-wise search
combinations4 <- get.combinations(sizes = list(c(1), c(1, 2)), stepsNumVariables = c(NA, 1))

# This example demonstrates a step-wise search. In the first step (\code{sizes = c(1)}), all
# models with one variable are estimated.
# In the next step (\code{sizes = c(1, 2)}), a subset of potential variables is selected based
# on their performance in the previous step and all models with both first and second variables
# are estimated.
```

---

get.data *Transform and Prepare Data for Analysis*

Description

This function prepares a data matrix for analysis. It applies a Box-Cox transformation to the endogenous variables, adds an intercept column, and optionally includes new rows with exogenous data.

Usage

```
get.data(
  data,
  endogenous = 1,
  equations = NULL,
  weights = NULL,
  lambdas = NULL,
```

```
  newData = NULL,
  addIntercept = TRUE,
  ...
)
```

Arguments

`data`	A data.frame or a numeric matrix that serves as the primary data source.
`endogenous`	A single number indicating the number of endogenous variables in the first columns, or a list of names specifying the endogenous variables. The remaining variables will be treated as exogenous.
`equations`	A formula or a list of formula objects that represent the equations to be used instead of `endogenous`. If provided, the final data will be a matrix where the response variables are in the first columns and the predictor variables are in the subsequent columns.
`weights`	A numeric vector or a column matrix representing weights of observations. Not all applications implement this parameter.
`lambdas`	A numeric vector, a single number, NA, or NULL indicating the lambda parameter(s) for the Box-Cox transformation. Use `NULL` for no transformation, `NA` for estimating the lambda parameter for each variable, a single number for an equal lambda parameter for all variables, and a numeric vector for distinct lambda parameters for corresponding variables.
`newData`	A data.frame or a numeric matrix representing new data for exogenous variables. It should have a structure similar to `data`, excluding endogenous or response variables.
`addIntercept`	A logical value indicating whether to add an intercept column to the final matrix.
`...`	Additional parameters for the `MASS::boxcox` function.

Details

This function is designed to prepare a data matrix for model search (or screening) analysis. It performs several operations to transform and structure the data appropriately.

The function first checks if the input data is a matrix or a data frame. If new data is provided, it also checks its type. It then extracts the frequency of the first observation from the `ldtf` attribute of the data, if available.

If no equations are provided, the function assumes that the endogenous variables are in the first columns of the data. It checks if an intercept is already present and throws an error if one is found and `addIntercept` is set to TRUE. It then validates the number of endogenous variables and converts the data to a numeric matrix.

If column names are missing, they are added based on the number of endogenous and exogenous variables. If new data is provided, it checks its structure and matches it with the exogenous part of the original data.

If equations are provided, they are used to transform the original data into a matrix where response variables are in the first columns and predictor variables in subsequent columns. The new data is also transformed accordingly.

The function then applies a Box-Cox transformation to the endogenous variables if lambda parameters are provided. Weights are added if provided, and an intercept column is added if `addIntercept` is set to TRUE.

Finally, the function returns a list containing all relevant information for further analysis. This includes the final data matrix, number of endogenous and exogenous variables, number of observations in original and new data, lambda parameters used in Box-Cox transformation, and flags indicating whether an intercept or weights were added.

Value

A list suitable for use in `ldt::search.?` functions. The list contains:

- `data` The final data matrix. Endogenous variables are in the first columns, followed by weights (if provided), then the intercept (if added), and finally the exogenous variables.
- `numEndo` The number of endogenous variables in the data.
- `numExo` The number of exogenous variables in the data (including 'intercept' if it is added).
- `newX` The matrix of new observations for exogenous variables.
- `lambdas` The lambda parameters used in the Box-Cox transformation.
- `hasIntercept` Indicates whether an intercept column is added to the final matrix.
- `hasWeight` Indicates whether there is a weight column in the final matrix.
- `startFrequency` Frequency of the first observation, extracted from `ldtf` attribute of `data`, if available. This will be used in time-series analysis such as VARMA estimation.

Examples

```
# Example 1:
data <- matrix(1:24, ncol = 6)
result <- get.data(data, endogenous = 1)
print(result$data)

# Example 2:
data <- matrix(1:24, ncol = 6,
               dimnames = list(NULL,c("V1", "V2", "V3", "V4", "V5", "V6")))
result <- get.data(data, endogenous = c("V6", "V1"))
print(result$data)

# Example 3:
data <- data.frame(matrix(1:24, ncol = 6))
colnames(data) <- c("X1", "X2", "Y2", "X3", "Y1", "X4")
equations <- list(
   Y1 ~ X2 + X1,
   Y2 ~ X4 + X3)
result <- get.data(data, equations = equations)
print(result$data)
```

`get.data.append.newX` *Append* `newX` *to* `data$data` *matrix.*

Description

Use it for VARMA estimation

Usage

```
get.data.append.newX(data, maxHorizon = NA)
```

Arguments

`data`	The output of get.data function.
`maxHorizon`	Number of expected new data

Value

The input `data` with updated data matrix

`get.data.check.discrete` *Check if a column is discrete*

Description

For example, it checks if the endogenous variable in binary model is 0 and 1 (number of choices is 2)

Usage

```
get.data.check.discrete(data, colIndex = 1)
```

Arguments

`data`	Output of get.data function
`colIndex`	The index of column to be checked.

Value

Number of choices in the model, if no error occured. This means, the maximum value for the discrete data will be the output minus one.

`get.data.check.intercept`
Check for an intercept in a matrix

Description

This function checks if any column in the matrix is intercept.

Usage

```
get.data.check.intercept(matrix)
```

Arguments

`matrix`	data matrix

Value

The index of the intercept. '-1' in intercept is not found.

`get.data.keep.complete`
Remove Rows with Missing Observations from Data

Description

Remove Rows with Missing Observations from Data

Usage

```
get.data.keep.complete(data, warn = TRUE)
```

Arguments

`data`	Output of get.data function
`warn`	If true, warning message about the indices of the removed rows is shown

Value

The input `data` but with updated `data$data` and `data$obsCount`

get.indexation *Get Numeric Indices in a Combination*

Description

This function takes the output of the get.combinations function and a numeric matrix with given column names. It converts all character vectors in `innerGroups` or `partitions` to numeric vectors based on the index of the columns.

Usage

```
get.indexation(combinations, data, isInnerExogenous)
```

Arguments

`combinations`	A list returned by the get.combinations function.
`data`	A list returned by get.data function.
`isInnerExogenous`	Use `TRUE` if outer loop is defined over the endogenous variables and `FALSE` if it is for exogenous.

Value

A list similar to the input `combinations`, but with all character vectors in `innerGroups` or `partitions` converted to numeric vectors based on the index of the columns in the `data` matrix. It sums the exogenous indexes with the number of endogenous variables and returns zero-based indexation for C code.

get.options.lbfgs *Get Options for L-BFGS Optimization*

Description

Use this function to get optimization options in estim.varma or search.varma functions.

Usage

```
get.options.lbfgs(
  maxIterations = 100,
  factor = 1e+07,
  projectedGradientTol = 0,
  maxCorrections = 5
)
```

Arguments

`maxIterations` A positive integer representing the maximum number of iterations.

`factor` A number that determines the condition for stopping the iterations. Use, for example, 1e12 for low accuracy, 1e7 (default) for moderate accuracy, and 1e1 for extremely high accuracy. The default is 1e7.

`projectedGradientTol` A number used to stop the iteration using the projected gradient. The default is zero.

`maxCorrections` The maximum number of variable metric corrections allowed in the limited memory matrix. The default is 5.

Value

A list with the given options.

---

get.options.neldermead *Options for Nelder-Mead Optimization*

Description

Use this function to get the required options when Nelder-Mead optimization is needed such as s.gld.from.moments function.

Usage

```
get.options.neldermead(
  maxIterations = 100,
  tolerance = 1e-06,
  reflection = 1,
  expansion = 2,
  contraction = 0.5,
  shrink = 1
)
```

Arguments

`maxIterations` (int) Maximum number of iterations.

`tolerance` A small number to determine the convergence. The algorithm terminates when the difference between the best and worst points in the simplex is less than this value.

`reflection` A number for reflection coefficient. It controls how far the worst point is reflected through the centroid of the remaining points.

`expansion` A number that determines the expansion coefficient. It controls how far the reflected point is expanded along the line connecting it to the centroid.

`contraction`	A number that determines the contraction coefficient. It controls how far the worst point is contracted towards the centroid.
`shrink`	A number that determines the shrink coefficient. It controls how much the simplex is shrunk towards the best point when all other moves are rejected.

Value

A list with the given options.

`get.options.newton` *Get Options for Newton Optimization*

Description

Use this function to get optimization options in estim.bin or search.bin functions.

Usage

```
get.options.newton(
  maxIterations = 100,
  functionTol = 1e-04,
  gradientTol = 0,
  useLineSearch = TRUE
)
```

Arguments

`maxIterations`	An integer representing the maximum number of iterations.
`functionTol`	A small value used to test the convergence of the objective function.
`gradientTol`	A small value used to test the convergence of the gradient.
`useLineSearch`	If `TRUE`, line search is used.

Value

A list with the given options.

`get.options.pca` *Get Options for PCA*

Description

Use this function to get PCA options in estim.bin, estim.sur, estim.varma, or s.pca functions.

Usage

```
get.options.pca(ignoreFirst = 1, exactCount = 0, cutoffRate = 0.8, max = 1000)
```

Arguments

`ignoreFirst`	A number representing the number of variables to exclude at the beginning of data matrices (such as intercept) from PCA.
`exactCount`	A number that determines the number of components to be used. If zero, the number of components is determined by the `cutoffRate`.
`cutoffRate`	A number between 0 and 1 that determines the cutoff rate for the cumulative variance ratio in order to determine the number of PCA components. It is not used if `exactCount` is positive.
`max`	A number representing the maximum number of components when `cutoffRate` is used.

Details

See details of s.pca function.

Value

A list with the given options.

See Also

estim.bin, estim.sur, estim.varma, s.pca

Examples

```
# See 's.pca' function.
```

`get.options.roc` *Get Options for ROC and AUC Calculations*

Description

Use this function to get the required options for search.bin, estim.bin, or s.roc functions.

Usage

```
get.options.roc(
  lowerThreshold = 0,
  upperThreshold = 1,
  epsilon = 1e-12,
  pessimistic = FALSE,
  costs = NULL,
  costMatrix = NULL
)
```

Arguments

`lowerThreshold`	A number representing the lower bound for calculating partial AUC.
`upperThreshold`	A number representing the upper bound for calculating partial AUC.
`epsilon`	A small number used to ignore small floating point differences when comparing scores.
`pessimistic`	If `TRUE`, sequences of equally scored instances are treated differently and a pessimistic metric is calculated (see Fawcett (2006) An introduction to ROC analysis, fig. 6).
`costs`	The cost of each observation. If `NULL`, the cost of all observations will be 1.
`costMatrix`	A `2x2` cost matrix in which: (1,1) is the cost of TN, (2,2) is the cost of TP, (1,2) is the cost of FP and (2,1) is the cost of FN. The first column is multiplied by the corresponding value in the costs vector (see Fawcett (2006), ROC graphs with instance-varying costs).

Details

See details of s.roc function.

Value

A list with the given options.

See Also

search.bin, estim.bin, s.roc

Examples

```
# See 's.roc' function.
```

`get.search.items` *Specify the Purpose of the Model Search Process*

Description

Use this function to list the required items and information that should be saved and retrieved from the model set search process in `search.?` functions.

Usage

```
get.search.items(
  model = TRUE,
  type1 = FALSE,
  type2 = FALSE,
  bestK = 1,
  all = FALSE,
  inclusion = FALSE,
  cdfs = numeric(0),
  extremeMultiplier = 0,
  mixture4 = FALSE
)
```

Arguments

`model`	If `TRUE`, some information about the models is saved.
`type1`	If `TRUE` and implemented, extra information is saved. This can be the coefficients in the SUR search or predictions in the VARMA search.
`type2`	If `TRUE` and implemented, extra information is saved. This is similar to `type1`. **It is reserved for future updates.**
`bestK`	The number of best items to be saved in `model`, `type1`, or `type2` information.
`all`	If `TRUE`, all models' information is saved.
`inclusion`	If `TRUE`, inclusion weights are saved.
`cdfs`	Weighted average of the CDFs at each given point is calculated (for `type1` and `type2` cases).
`extremeMultiplier`	A number that determines the multiplier in the extreme bound analysis (for `type1` and `type2` cases). Use zero to disable it.
`mixture4`	If `TRUE`, the first four moments of the average distributions are calculated in `type1` and `type2` cases.

Value

A list with the given options.

---

get.search.metrics *Get Options for Measuring Performance*

Description

Use this function to get measuring options in `search.?` functions.

Usage

```
get.search.metrics(
  typesIn = c("aic"),
  typesOut = NULL,
  simFixSize = 2,
  trainRatio = 0.75,
  trainFixSize = 0,
  seed = 0,
  horizons = c(1L),
  weightedEval = FALSE,
  minMetrics = list(aic = 0)
)
```

Arguments

`typesIn`	A list of evaluation metrics when the model is estimated using all available data. It can be `aic`, `sic`, `frequencyCostIn`, `brierIn`, or `aucIn`. `NULL` means no metric.
`typesOut`	A list of evaluation metrics in a out-of-sample simulation. It can be `sign`, `direction`, `rmse`, `rmspe`, `mae`, `mape`, `crps`, `frequencyCostOut`, `brierOut`, or `aucOut`. Null means no metric.
`simFixSize`	An integer that determines the number of out-of-sample simulations. Use zero to disable the simulation.
`trainRatio`	A number representing the size of the training sample relative to the available size, in the out-of-sample simulation. It is effective if `trainFixSize` is zero.
`trainFixSize`	An integer representing the number of data points in the training sample in the out-of-sample simulation. If zero, `trainRatio` will be used.
`seed`	A seed for the random number generator. Use zero for a random value. It can be negative to get reproducible results between the `search.?` function and the `estim.?` function.
`horizons`	An array of integers representing the prediction horizons to be used in out-of-sample simulations, if the model supports time-series prediction. If `NULL`, `c(1)` is used.

`weightedEval` If `TRUE`, weights are used in evaluating discrete-choice models.

`minMetrics` a list of minimum values for adjusting the weights when applying the AIC weight formula. It can contain the following members: `aic`, `sic`, `brierIn`, `rmse`, `rmspe`, `mae`, `mape`, `crps`, `brierOut`. Members can be numeric vectors for specifying a value for each target variable. See details.

Details

An important aspect of `ldt` is model evaluation during the screening process. This involves considering both in-sample and out-of-sample evaluation metrics. In-sample metrics are computed using data that was used in the estimation process, while out-of-sample metrics are computed using new data. These metrics are well documented in the literature, and I will provide an overview of the main computational aspects and relevant references.

Value

A list with the given options.

AIC and SIC

According to Burnham and Anderson (2002) or Greene (2020), AIC and SIC are two commonly used metrics for comparing and choosing among different models with the same endogenous variable(s). Given L^* as the maximum value of the likelihood function in a regression analysis with k estimated parameters and N observations, AIC is calculated by $2k - 2\ln L^*$ and SIC is calculated by $k \ln N - 2\ln L^*$. SIC includes a stronger penalty for increasing the number of estimated parameters in the model.

These metrics can be converted into weights using the formula $w = \exp(-0.5x)$, where x is the value of the metric. When divided by the sum of all weights, w can be interpreted as the probability that a given model is the best model among all members of the model set (see section 2.9 in Burnham and Anderson (2002)). Compared to the Burnham and Anderson (2002) discussion and since $f(x) = exp(-0.5x)$ transformation is invariant to translation, the minimum AIC part is removed in the screening process. This is an important property because it enables the use of running statistics and parallel computation.

MSE, RMSE, MSPE, and RMSPE

According to Hyndman and Athanasopoulos (2018), MSE and RMSE are two commonly used scale-dependent metrics, while MAPE is a commonly used unit-free metric. `ldt` also calculates the less common RMSPE metric. If there are n predictions and $e_i = y_i - \hat{y}_i$ for $i = 1 \dots n$ is the prediction error, i.e., the distance between actual values (y_i) and predictions ($\hat{y}_i$), these metrics can be expressed analytically by the following formulas:

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |e_i|$$

$$\mathrm{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{e_i}{y_i} \right| \times 100$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i)^2}$$

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{e_i}{y_i}\right)^2} \times 100$$

Note that, first MAPE and RMSPE are not defined if y_i is zero and may not be meaningful or useful if it is near zero or negative. Second, although these metrics cannot be directly interpreted as weights, they are treated in a manner similar to AIC in the `ldt` package.. Third, caution is required when target variables are transformed, for example to a logarithmic scale. `ldt` provides an option to transform the data back when calculating these metrics.

Brier

The Brier score measures the accuracy of probabilistic predictions for binary outcomes. It is calculated as the mean squared difference between the actual values (y_i) and the predicted probabilities (p_i). Assuming that there are n predictions, its formula is given by:

$\text{Brier} = \frac{\sum(y_i - \hat{p}_i)^2}{n}$,

where p_i is the predicted probability that the i-th observation is positive. The value of this metric ranges from 0 to 1, with lower values indicating better predictions. In the screening process in `ldt`, both in-sample and out-of-sample observations can be used to calculate this metric. Although this metric cannot be directly interpreted as a weight, it is treated in a manner similar to AIC.

AUC

As described by Fawcett (2006), the receiver operating characteristic curve (ROC) plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at different classification thresholds. The area under this curve is known as the AUC. Its value ranges from 0 to 1, with higher values indicating that the model is better at distinguishing between the two classes Fawcett (2006, 2006). In the screening process in `ldt`, both in-sample and out-of-sample observations can be used to calculate this metric. There is also an option to calculate the pessimistic or an instance-varying costs version of this metric. Although this metric does not have a direct interpretation as weights, in `ldt` its value is considered as weight.

CRPS

According to Gneiting et al. (2005), the continuous ranked probability score (CRPS) is a metric used to measure the accuracy of probabilistic predictions. Unlike MAE, RMSE, etc., CRPS takes into account the entire distribution of the prediction, rather than focusing on a specific point of the probability distribution. For n normally distributed predictions with mean $\hat{y}_i$ and variance $\text{var}(\hat{y}_i)$, this metric can be expressed analytically as:

$\text{CRPS} = \sum_{i=1}^{n} \sigma \left(\frac{1}{\sqrt{\pi}} - 2\Phi(z_i) + z_i(2\phi(z_i) - 1)\right)$,

where $z_i = (y_i - \hat{y}_i)/\sqrt{\text{var}(\hat{y}_i)}$, and Φ and ϕ are CDF and density functions of standard normal distribution. Although this metric cannot be directly interpreted as a weight, it is treated in a manner similar to AIC in the `ldt` package.

Other metrics

There are some other metrics in `ldt`. One is "directional prediction accuracy", which is calculated as the proportion of predictions that correctly predict the direction of change relative to the previous observation. Its value ranges from 0 to 1, with higher values indicating better performance of the model. Its value is used as the weight of a model. Note that this is applicable only to time-series data.

Another similar metric is "sign prediction accuracy", which reports the proportion of predictions that have the same sign as the actual values. It is calculated as the number of correct sign predictions divided by the total number of predictions. Its value ranges from 0 to 1, with higher values indicating better performance of the model. Its value is used as the weight of a model.

References

Burnham KP, Anderson DR (2002). *Model selection and multimodel inference*. Springer, New York. ISBN 0387953647, doi:10.1007/b97636.

Fawcett T (2006). "An introduction to ROC analysis." *Pattern Recognition Letters*, **27**(8), 861–874. doi:10.1016/j.patrec.2005.10.010.

Fawcett T (2006). "ROC graphs with instance-varying costs." *Pattern Recognition Letters*, **27**(8), 882–891. doi:10.1016/j.patrec.2005.10.012.

Gneiting T, Raftery AE, Westveld AH, Goldman T (2005). "Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation." *Monthly Weather Review*, **133**(5), 1098–1118. doi:10.1175/mwr2904.1.

Greene WH (2020). *Econometric analysis*, 8th edition. Pearson Education Limited, New York. ISBN 9781292231136.

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: Principles and practice*. OTexts. `https://otexts.com/fpp2/`.

`get.search.modelchecks` *Set Options to Exclude a Model Subset*

Description

Use this function to determine which models should be skipped in the search process.

Usage

```
get.search.modelchecks(
  estimation = TRUE,
  maxConditionNumber = Inf,
  minObsCount = 0,
```

```
  minDof = 0,
  minOutSim = 0,
  minR2 = -Inf,
  maxAic = Inf,
  maxSic = Inf,
  prediction = FALSE,
  predictionBound = 10
)
```

Arguments

`estimation` If `TRUE`, the model is estimated with all data and is ignored if this estimation fails. If `FALSE`, you might get a 'best model' that cannot be estimated.

`maxConditionNumber` A number used to ignore an estimation that has a high condition number (if implemented in the search).

`minObsCount` An integer used to ignore an estimation where the number of observations (after dealing with `NA`) is low. Use 0 to disable this check.

`minDof` An integer used to ignore an estimation with low degrees of freedom (equation-wise). Use 0 to disable this check.

`minOutSim` An integer used to ignore estimations with a low number of out-of-sample simulations (if implemented in the search).

`minR2` A number used to ignore estimations with a low value for 'R2' (if implemented in the search).

`maxAic` A number used to ignore estimations with a high 'AIC' (if implemented in the search).

`maxSic` A number used to ignore estimations with a high 'SIC' (if implemented in the search).

`prediction` If `TRUE`, model data is predicted given all data and is ignored if this process fails. If `FALSE`, you might get a 'best model' that cannot be used for prediction.

`predictionBound` A list containing two matrices: `lower` and `upper`, which represent the bounds for checking predictions. Each column corresponds to a target variable, and each row corresponds to a horizon. If the data has been transformed using a Box-Cox transformation, these bounds will be compared with the transformed data. Alternatively, `predictionBound` can be a numeric value. In this case, the bounds are created by creating a confidence interval, assuming normality and using mean and standard errors of the growth rates. Any model that produces a prediction outside of these bounds will be ignored. To disable this check, set `predictionBound` to `NULL`.

Value

A list with the given options.

`get.search.options` *Get Extra Options for Model Search Process*

Description

Use this function to determine how the model search is performed.

Usage

```
get.search.options(parallel = FALSE, reportInterval = 0)
```

Arguments

`parallel`	If `TRUE`, a parallel search algorithm is used. This generally changes the speed and memory usage.
`reportInterval`	An integer representing the time interval (in seconds) for reporting progress (if any significant change has occurred). Set to zero to disable reporting.

Value

A list with the given options.

`get.varma.params` *Split VARMA parameter into its Components*

Description

Use this function to extract AR, MA, intercept, and exogenous coefficients from the VARMA estimation.

Usage

```
get.varma.params(
  coef,
  numAR = 1,
  numMA = 0,
  numExo = 0,
  intercept = TRUE,
  numAR_s = 0,
  numMA_s = 0,
  numSeasons = 1
)
```

Arguments

coef	A matrix of coefficients with dimensions numEq x numAR * numEq + numMA * numEq + numExo + ifelse(intercept, 1, 0).
numAR	A non-negative integer scalar specifying the number of AR lags.
numMA	A non-negative integer scalar specifying the number of MA lags.
numExo	A non-negative integer scalar specifying the number of exogenous variables.
intercept	A logical scalar indicating whether an intercept is included in the model.
numAR_s	A non-negative integer scalar specifying the number of seasonal AR lags.
numMA_s	A non-negative integer scalar specifying the number of seasonal MA lags.
numSeasons	A non-negative integer scalar specifying the number of seasons.

Value

A list with the following items:

- arList: A list containing the AR coefficients for each lag.
- intercept: A numeric vector of length numEq containing the intercept, or NULL if intercept = FALSE.
- exoCoef: A matrix of dimensions numEq x numExo containing the exogenous coefficients, or NULL if numExo = 0.
- maList: A list containing the MA coefficients for each lag.

See Also

estim.varma

Examples

```
# see 'search.varma' or 'estim.varma' functions.
```

---

logLik.ldt.estim *Extract Maximum Log-Likelihood*

Description

This function extracts maximum log-likelihood from an ldt.estim object.

Usage

```
## S3 method for class 'ldt.estim'
logLik(object, ...)
```

Arguments

`object`	An object of class `ldt.estim`
`...`	Additional arguments.

Value

The value of the maximum log-likelihood for the whole system.

`plot.ldt.estim` *Plot Diagnostics for* `ldt.estim` *Object*

Description

This function creates diagnostic plots for estimated regression models of `ldt.estim` class.

Usage

```
## S3 method for class 'ldt.estim'
plot(
  x,
  equation = 1,
  type = c(1, 2, 3, 4, 5, 6),
  ablineArgs = list(col = "lightblue"),
  textArgs = list(pos = 3, cex = 0.7, col = "red"),
  ...
)
```

Arguments

`x`	An object of type `ldt.estim`.
`equation`	A number or a name of endogenous variable specifying an equation in the estimated system.
`type`	One of these numbers: 1, 2, 3, or 5. See `which` argument in plot.lm documentation.
`ablineArgs`	A list of additional arguments to customize the "text" function used for labeling influential observations.
`textArgs`	A list of additional arguments to customize the "abline" function.
`...`	additional arguments to be passed to "plot" (or "qqnorm" function for `type=2`, or "barplot" for `type=4`).

Details

This function is designed to be similar to plot.lm function. However, note that an `ldt.estim` object might be a system estimation.

Some plots use standardized residuals. Note that they are not calculated in a system estimation context. See residuals.ldt.estim documentation for a description. Cook's distance is also calculated equation-wise. Its formula is:

$$d = \frac{r_i^2}{k * var(r)} \frac{h_{ii}}{(1 - h_{ii})^2}$$

where r_i and h_{ii} are residual and leverage in i-th observation, respectively. $var(r)$ is variance of residuals and k is the number of estimated coefficients in the equation. Note that Cook's distance is not implemented for weighted observations.

Value

This function creates diagnostic plots for regression models. It also returns a list with `x` and `y` data used in plot functions.

---

plot.ldt.varma.prediction

Plot Predictions from a VARMA model

Description

Plot Predictions from a VARMA model

Usage

```
## S3 method for class 'ldt.varma.prediction'
plot(
  x,
  variable = 1,
  xAxisArgs = list(),
  fanPlotArgs = list(),
  simMetric = NULL,
  simLineArgs = list(),
  simPointsArgs = list(),
  ...
)
```

Arguments

`x`	An object of class `ldt.varma.prediction`, which is the output of predict.ldt.estim.varma function.
`variable`	Index or name of the variable to be plotted.
`xAxisArgs`	Arguments to pass to `axis` function

fanPlotArgs	Additional arguments for the fan.plot function. lambda is added automatically.
simMetric	Name of metric to plot its details, provided that simulation details are available. If NULL, simulation details are not plotted.
simLineArgs	Arguments to pass to line function for simulation lines (if available).
simPointsArgs	Arguments to pass to points function for simulation points (if available).
...	Additional parameters (unused)

Value

This function does not return a value.

---

predict.ldt.estim *Extract Prediction Results*

Description

This function extracts predicted mean and its variance from an ldt.estim object. new data must be provided while estimating the model.

Usage

```
## S3 method for class 'ldt.estim'
predict(object, ...)
```

Arguments

object	An object of class ldt.estim
...	Additional arguments.

Value

A list containing the predicted (projected) means and variances.

`predict.ldt.estim.varma` *Extract Prediction Results from a* `ldt.estim.varma` *Object*

Description

This function extracts predicted mean and its variance from an `ldt.estim.varma` object. new data must be provided while estimating the model.

Usage

```
## S3 method for class 'ldt.estim.varma'
predict(object, actualCount = 0, startFrequency = NULL, ...)
```

Arguments

`object`	An object of class `ldt.estim.varma`
`actualCount`	Number of actual observations to be included in the result.
`startFrequency`	Frequency of the first observation used in the estimation. This is object of class `ldtf`.
`...`	Additional arguments.

Details

If estimation data undergoes a Box-Cox transformation, the resulting values will not be transformed accordingly.

Value

An object of class `ldt.varma.prediction`, which is a list with predicted `means` and (if available) `variances`.

`print.ldt.estim` *Prints an* `ldt.estim` *object*

Description

Prints the main results in an `ldt.estim` object.

Usage

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

Arguments

`x`	An object of class `ldt.estim`
`...`	Additional arguments

Details

An `ldt.search` object is an output from one of the `search.?` functions (see `search.sur`, `search.varma`, or `search.bin`).

Value

This function has no output.

`print.ldt.estim.projection`
Prints an `ldt.estim.projection` *object*

Description

An `ldt.estim.projection` object is the output of `predict.ldt.estim()` function.

Usage

```
## S3 method for class 'ldt.estim.projection'
print(x, ...)
```

Arguments

`x`	An object of class `ldt.estim.projection`
`...`	Additional arguments

Value

This function has no output.

`print.ldt.list` *Prints an* `ldt.list` *object*

Description

Prints an `ldt.list` object

Usage

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

Arguments

`x`	An object of class `ldt.list`
`...`	Additional arguments

Value

This function has no output.

`print.ldt.search` *Prints an* `ldt.search` *object*

Description

Prints the main results in an `ldt.search` object. This includes information about the first best models and significant coefficients.

Usage

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

Arguments

`x`	An object of class `ldt.search`
`...`	Additional arguments

Details

An `ldt.search` object is an output from one of the `search.?` functions (see `search.sur`, `search.varma`, or `search.bin`).

Value

This function has no output.

print.ldt.varma.prediction *Prints an* `ldt.varma.prediction` *object*

Description

An `ldt.varma.prediction` object is the output of `predict.ldt.estim.varma()` function.

Usage

```
## S3 method for class 'ldt.varma.prediction'
print(x, ...)
```

Arguments

`x`	An object of class `ldt.varma.prediction`
`...`	Additional arguments

Value

This function has no output.

rand.mnormal *Generate Random Samples from a Multivariate Normal Distribution*

Description

Use this function to get random samples from a multivariate normal distribution.

Usage

```
rand.mnormal(n, mu = NULL, sigma = NULL, p = NULL, byRow = TRUE)
```

Arguments

`n`	The number of samples to generate.
`mu`	The mean vector of the distribution. If `NULL`, it defaults to a zero vector of length `p`. If `NA`, it is set to a random vector.
`sigma`	The covariance matrix of the distribution. If `NULL`, it defaults to an identity matrix of size `p x p`. If `NA`, it is set to a random positive definite matrix.
`p`	The dimension of the distribution, if both `mu` and `sigma` are `NA` or `NULL`.
`byRow`	If `TRUE`, generated samples are stored in the rows. Otherwise, they are stored in the columns.

Value

A list containing the generated sample (p x n), mu, and sigma.

Examples

```
s1 <- rand.mnormal(10, mu = c(0, 0), sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2))
s2 <- rand.mnormal(10, mu = c(1,1), sigma = NA, p = 2)
s3 <- rand.mnormal(10, p = 2, byRow = FALSE) #standard normal
```

---

residuals.ldt.estim *Extract Residuals Data*

Description

This function returns residuals from or calculates the standardized residuals for an ldt.estim object.

Usage

```
## S3 method for class 'ldt.estim'
residuals(object, equations = NULL, standardized = FALSE, pearson = TRUE, ...)
```

Arguments

object	An object of class ldt.estim.
equations	A number, a numeric array or a string array specifying the equations with residual data. If NULL, residuals in all equations are returned.
standardized	If TRUE, standardized residuals are returned. See details.
pearson	If TRUE, it returns (or uses) Pearson residuals for binary choice regression.
...	Additional arguments.

Details

The standardized residuals have identical variance. In order to calculate the standardized residuals, each residual is divided by $s\sqrt{w_i(1-h_{ii})}$ where s is the standard error of residuals and h_{ii} is the leverage of i-th observation. w_i is the weight of the i-th observation if data is weighted, and 1 otherwise. Note that while the residuals are estimated in a system, the h_{ii} is calculated in a univariate context as the i-th diagonal of $X(X'X)^{-1}X'$ matrix, where X is the exogenous variables in the corresponding equation.

Value

A matrix containing the residuals data.

s.cluster.h   *Hierarchical Clustering*

Description

This function performs hierarchical clustering on a group of variables, given their distances from each other.

Usage

```
s.cluster.h(distances, linkage = "single")
```

Arguments

distances	Lower triangle of a symmetric distance matrix (without the diagonal). This can be the output of `s.distance` function.
linkage	Character string specifying the method for calculating the distance in a left-right node merge. It can be `single`, `complete`, `uAverage`, `wAverage`, or `ward`.

Details

The main purpose of exporting this statistics helper method is to show the inner calculations of the package.

Value

A list with the following items:

merge	An integer matrix representing the merge matrix.
height	A numeric vector representing the heights.
order	An integer vector representing the orders.

Examples

```
n <- 10
data <- data.frame(x = rnorm(n), y = rnorm(n), z = rnorm(n))
distances <- s.distance(data)
clusters <- s.cluster.h(distances)
```

`s.cluster.h.group` *Group Variables with Hierarchical Clustering*

Description

This function groups the columns of a numeric matrix based on the hierarchical clustering algorithm.

Usage

```
s.cluster.h.group(
  data,
  nGroups = 2,
  threshold = 0,
  distance = "correlation",
  linkage = "single",
  correlation = "pearson"
)
```

Arguments

`data`	A numeric matrix with variables in the columns.
`nGroups`	Integer value specifying the number of required groups.
`threshold`	Numeric value specifying a threshold for omitting variables. If the distance between two variables in a group is less than this value, the second one will be omitted. Note that a change in the order of the columns might change the results.
`distance`	Character string specifying how distances are calculated. It can be `correlation`, `absCorrelation`, `euclidean`, `manhattan`, or `maximum`. See `s.distance` function.
`linkage`	Character string specifying how distances are calculated in a left-right node merge. It can be `single`, `complete`, `uAverage`, `wAverage`, or `ward`. See `s.cluster.h` function.
`correlation`	Character string specifying the type of correlation if `distance` is correlation. It can be `pearson` or `spearman`. See `s.distance` function.

Details

The results might be different from R's 'cutree' function. (I don't know how 'cutree' works) Here this function iterates over the nodes and whenever a split occurs, it adds a group until the required number of groups is reached.

Value

A list with the following items:

`groups`	A list of integer vectors representing the indexes of variables in each group.
`removed`	An integer vector representing the indexes of removed variables.

`s.combine.stats4` *Combine Mean, Variance, Skewness, and Kurtosis This function combines two sets of mean, variance, skewness, and kurtosis and generates the combined statistics.*

Description

Combine Mean, Variance, Skewness, and Kurtosis This function combines two sets of mean, variance, skewness, and kurtosis and generates the combined statistics.

Usage

```
s.combine.stats4(list1, list2)
```

Arguments

list1	A list representing the first `mean`, `variance`, `skewness`, `kurtosis`, `weight`, and `count`.
list2	A list representing the second distribution (similar to `list1`).

Details

Assume there are two samples with $mean_i$, $variance_i$, $skewness_i$, and $kurtosis_i$ for $i = 1, 2$, this function calculates the mean, variance, skewness, and kurtosis of the combined sample. It does not need the data itself. It is based on population variance, skewness, and kurtosis and calculates the population statistics. Note that the kurtosis is not excess kurtosis.

Value

A list similar to `list1`.

Examples

```
n <- 1000 # sample size (increase it for more accurate result)
sample1 <- rchisq(n,3)
sample2 <- rchisq(n,5)

d1 <- list(mean = mean(sample1),
           variance = var(sample1),
           skewness = moments::skewness(sample1),
           kurtosis = moments::kurtosis(sample1),
           count=length(sample1),
           weight = length(sample1))
d2 <- list(mean = mean(sample2),
           variance = var(sample2),
           skewness = moments::skewness(sample2),
           kurtosis = moments::kurtosis(sample2),
           count=length(sample2),
           weight = length(sample2))
```

```
c <- s.combine.stats4(d1,d2)

# we can compare the results:
combined <- c(sample1,sample2)
mean_c = mean(combined)
variance_c = var(combined)
skewness_c = moments::skewness(combined)
kurtosis_c = moments::kurtosis(combined)
```

`s.distance` *Get the Distances Between Variables*

Description

This function calculates the distances between the columns of a numeric matrix.

Usage

```
s.distance(
  data,
  distance = "correlation",
  correlation = "pearson",
  checkNan = TRUE
)
```

Arguments

`data`	A numeric matrix with variables in the columns.
`distance`	Character string specifying the type of distance. It can be `correlation`, `absCorrelation`, `euclidean`, `manhattan`, or `maximum`.
`correlation`	Character string specifying the type of correlation if `distance` is correlation. It can be `pearson` or `spearman`.
`checkNan`	Logical value indicating whether to check for `NA`s (and omit them if any exist).

Details

The main purpose of exporting this statistics helper method is to show the inner calculations of the package.

Value

A symmetric matrix (lower triangle as a vector).

Examples

```
n <- 10
data <- data.frame(x = rnorm(n), y = rnorm(n), z = rnorm(n))
distances <- s.distance(data)
```

`s.gld.density.quantile`

GLD Density-Quantile Function

Description

This function calculates the densities of a Generalized Lambda Distribution (FKLM) given a vector of probabilities.

Usage

```
s.gld.density.quantile(probs, p1, p2, p3, p4)
```

Arguments

probs	A numeric vector representing the probabilities.
p1	Numeric value representing the first parameter (location) of the distribution.
p2	Numeric value representing the second parameter (scale) of the distribution.
p3	Numeric value representing the third parameter (skewness) of the distribution.
p4	Numeric value representing the fourth parameter (kurtosis) of the distribution.

Details

It is a helper statistics method in this package and is generally used to plot density function of a GLD distribution.

Value

A numeric vector representing the densities for each probability in `probs`.

See Also

s.gld.quantile

Examples

```
# In this example we use this function and plot the density function for
# standard normal distribution:
probs <- seq(0.1,0.9,0.1)
x <- s.gld.quantile(probs, 0,1,0,0)
y <- s.gld.density.quantile(probs, 0,1,0,0)
plot(x,y)
lines(x,y)
```

s.gld.from.moments *Get the GLD Parameters from the moments*

Description

Calculates the parameters of the generalized lambda distribution (FKML), given the first four moments of the distribution.

Usage

```
s.gld.from.moments(
  mean = 0,
  variance = 1,
  skewness = 0,
  excessKurtosis = 0,
  type = 0,
  start = NULL,
  nelderMeadOptions = get.options.neldermead()
)
```

Arguments

`mean`	A number for the mean of the distribution.
`variance`	A number for the variance of the distribution.
`skewness`	A number for the skewness of the distribution.
`excessKurtosis`	A number for the excess kurtosis of the distribution.
`type`	An integer to restrict the shape of the distribution. See details section.
`start`	A numeric vector of size 2 for the starting value.
`nelderMeadOptions`	A list of options for Nelder-Mead algorithm. Use get.options.neldermead for initialization.

Details

The type of the distribution is determined by one or two restrictions:

- **type 0:** general, no restriction
- **type 1:** symmetric 'type 0', p3 == p4
- **type 2:** uni-modal continuous tail, p3 < 1 & p4 < 1
- **type 3:** symmetric 'type 2', p3 == p4
- **type 4:** uni-modal continuous tail finite slope, p3 <= 0.5 & p4 <= 0.5
- **type 5:** symmetric 'type 4', p3 == p4
- **type 6:** uni-modal truncated density curves, p3 >= 2 & p4 >= 2 (includes uniform distribution)
- **type 7:** symmetric 'type 6', p3 == p4

- **type 8:** S shaped, (p3 > 2 & 1 < p4 < 2) or (1 < p3 < 2 & p4 > 2)
- **type 9:** U shaped, (1 < p3 <= 2) and (1 < p4 <= 2)
- **type 10:** symmetric 'type 9', p3 == p4
- **type 11:** monotone, p3 > 1 & p4 <= 1

Value

A vector of length 5. The first 4 elements are the parameters of the GLD distribution. The last one is the number of iterations.

Examples

```
res <- s.gld.from.moments(0,1,0,0, start = c(0,0), type = 4)
probs <- seq(0.1,0.9,0.1)
x <- s.gld.quantile(probs, res[1],res[2],res[3],res[4])
y <- s.gld.density.quantile(probs, res[1],res[2],res[3],res[4])
plot(x,y)
lines(x,y)
```

---

s.gld.quantile *GLD Quantile Function*

Description

This function calculates the quantiles of a Generalized Lambda Distribution (FKML).

Usage

```
s.gld.quantile(probs, p1, p2, p3, p4)
```

Arguments

probs	A numeric vector of probabilities.
p1	Numeric value representing the first parameter of the distribution (location of the distribution).
p2	Numeric value representing the second parameter of the distribution (scale of the distribution).
p3	Numeric value representing the third parameter of the distribution (skewness of the distribution).
p4	Numeric value representing the fourth parameter of the distribution (kurtosis of the distribution).

Details

It is a helper statistics method in this package and is generally used to plot density function of a GLD distribution. See the example of `s.gld.density.quantile` function for more details.

Value

A numeric vector representing the quantiles for each probability in `probs`.

See Also

s.gld.density.quantile

Examples

```
res = s.gld.quantile(c(0.1,0.5,0.95), 0,1,0,0) # standard normal distribution
```

`s.metric.from.weight` *Convert a Weight to Metric*

Description

This function converts a weight to its metric equivalent.

Usage

```
s.metric.from.weight(value, metricName, minValue = 0)
```

Arguments

`value`	Numeric value of the weight.
`metricName`	Character string specifying the name of the metric. See `get.search.metrics` function for the list of available options.
`minValue`	A minimum value used in exponential weight formula.

Details

See s.weight.from.metric and get.search.metrics for more details.

Note that the main purpose of exporting this statistics helper method is to show the inner calculations of the package.

Value

A numeric value representing the converted weight.

See Also

s.weight.from.metric

Examples

```
weight <- s.weight.from.metric(-3.4, "sic")
metric <- s.metric.from.weight(weight, "sic")
```

`s.pca` *Principal Component Analysis*

Description

This function performs PCA on the columns of a matrix.

Usage

```
s.pca(x, center = TRUE, scale = TRUE, newX = NULL)
```

Arguments

`x`	A numeric matrix with variables in the columns.
`center`	Logical value indicating whether to demean the columns of `x`.
`scale`	Logical value indicating whether to scale the columns of `x` to unit variance.
`newX`	A numeric matrix to be used in projection. Its structure must be similar to `x`.

Details

The main purpose of exporting this statistics helper method is to show the inner calculations of the package.

Value

A list with the following items:

`removed0Var`	An integer vector showing the zero-based indices of removed columns with zero variances.
`directions`	Directions matrix.
`stds`	An integer vector showing the standard deviation of the principal components.
`stds2Ratio`	Shows `stds^2/sum(stds^2)`.
`projections`	Projections matrix if `newX` is provided.

See Also

get.options.pca

Examples

```
set.seed(340)
data <- matrix(rnorm(500), nrow = 50, ncol = 10)

# using prcomp function
resR = prcomp(data, center = TRUE, scale. = TRUE)

# using s.pca in this package
res = s.pca(data,TRUE,TRUE,data)

# res$projections and resR$x must be equal
# res$directions and t(resR$rotation) must be equal

# ----- ANOTHER EXAMPLE: PCA where there is a constant variable:
data <- data.frame( x = rnorm(100), y = rnorm(100), z = rep(0, 100))

# using s.pca in this package
res <- s.pca(data)

# using prcomp function
res_invalid <- try(prcomp(data, center = TRUE,
                          scale. = TRUE))
# Fails, we should remove 'z' first
```

`s.roc` *Get ROC Curve Data for Binary Classification*

Description

This function calculates the required points for plotting the ROC curve and the AUC.

Usage

```
s.roc(y, scores, weights = NULL, options = get.options.roc())
```

Arguments

`y`	A numeric vector (`Nx1`) representing the actual values.
`scores`	A numeric vector (`Nx1`) representing the calculated probabilities for the **negative** observations.
`weights`	A numeric vector (`Nx1`) representing the weights of the observations. Use `NULL` for equal weights.
`options`	A list from `get.options.roc` function for more options.

Details

This is generally a statistics helper method in this package and it shows the inner calculations. See AUC section in get.search.metrics for a discussion.

Value

A list with the following items:

`n`	Number of observations.
`auc`	Value of AUC.
`points`	Points for plotting ROC.

Examples

```
y <- c(1, 0, 1, 0, 1, 1, 0, 0, 1, 0)
scores <- c(0.1, 0.2, 0.3, 0.5, 0.5, 0.5, 0.7, 0.8, 0.9, 1)
res1 <- s.roc(y,scores)
costs <- c(1,2,1,4,1,5,1,1,0.5,1)
costMatrix <- matrix(c(0.02,-1,-3,3),2,2)
opt <- get.options.roc(costs = costs, costMatrix = costMatrix)
res2 <- s.roc(y,scores,NULL,options = opt)
```

`s.weight.from.metric` *Convert a Metric to Weight*

Description

This function converts a metric to its weight equivalent.

Usage

```
s.weight.from.metric(value, metricName, minValue = 0)
```

Arguments

`value`	Numeric value of the metric.
`metricName`	Character string specifying the name of the metric. See get.search.metrics function for the list of available options.
`minValue`	A minimum value to be used for metrics with exponential weight formula.

Details

Given a collection of models for the data, a metric is not generally a metric of the relative quality of a model. This function converts the value of a metric to such a number. see get.search.metrics for more details.

The main purpose of exporting this statistics helper method is to show the inner calculations of the package.

Value

A numeric value representing the converted metric.

See Also

s.metric.from.weight

Examples

```
weight <- s.weight.from.metric(-3.4, "sic")
metric <- s.metric.from.weight(weight, "sic")
```

---

search.bin — *Create a Model Set for Binary Choice Models*

Description

Use this function to create a binary choice model set and search for the best models (and other information) based on in-sample and out-of-sample evaluation metrics.

Usage

```
search.bin(
  data,
  combinations,
  metrics = get.search.metrics(),
  modelChecks = get.search.modelchecks(),
  items = get.search.items(),
  options = get.search.options(),
  costMatrices = NULL,
  searchLogit = TRUE,
  searchProbit = FALSE,
  optimOptions = get.options.newton(),
  aucOptions = get.options.roc()
)
```

Arguments

`data`	A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
`combinations`	A list that determines the combinations of the exogenous variables in the search process. Use `get.combinations()` function to define it.
`metrics`	A list of options for measuring performance. Use get.search.metrics function to get them.
`modelChecks`	A list of options for excluding a subset of the model set. Use get.search.modelchecks function to get them.
`items`	A list of options for specifying the purpose of the search. Use get.search.items function to get them.

`options`	A list of extra options for performing the search. Use get.search.options function to get them.
`costMatrices`	A list of numeric matrices where each one determines how to score the calculated probabilities. Given the number of choices `n`, a frequency cost matrix is an `m x n+1` matrix. The first column determines the thresholds. Elements in the `j`-th column determine the costs corresponding to the `j-1`-th choice in `y`. It can be `NULL` if it is not selected in `metrics`.
`searchLogit`	If `TRUE`, logit regressions are added to the model set.
`searchProbit`	If `TRUE`, probit regressions are added to the model set.
`optimOptions`	A list for Newton optimization options. Use get.options.newton function to get the options.
`aucOptions`	A list for AUC calculation options. Use get.options.roc function to get the options.

Value

A nested list with the following members:

`counts`	Information about the expected number of models, number of estimated models, failed estimations, and some details about the failures.
`results`	A data frame with requested information in `items` list.
`info`	The arguments and some general information about the search process such as the elapsed time.

Note that the output does not contain any estimation results, but minimum required data to estimate the models (Use `summary()` function to get the estimation).

See Also

estim.bin

Examples

```
# We simulate some data for this example:
# sample data:
n = 50 # number of observations
num_x_r <- 3L # number of relevant explanatory variables
num_x_ir <- 20 # (relatively large) number of irrelevant explanatory variables
set.seed(340)
sample <- sim.bin(num_x_r, n)
x_ir <- lapply(1:num_x_ir, function(x) rnorm(n))

# prepare data:
data <- data.frame(sample$y, sample$x, x_ir)
colnames(data) <- c("Y", colnames(sample$x), paste0("z", 1:num_x_ir))

# Use glm function to estimate and analyse:
fit <- glm(Y ~ . - Y, data = data, family = binomial())
summary(fit)
```

```
# You can also use this package estimation function:
data0 <- get.data(data,
                equations = list(Y ~ . - Y),
                addIntercept = FALSE)
fit <- estim.bin(data = data0)
# format and print coefficients:
print(fit)

# Alternatively, You can define a binary choice model set:
x_sizes = c(1:3) # assuming we know the number of relevant explanatory variables is less than 3
metric_options <- get.search.metrics(typesIn = c("sic")) # We use SIC for searching
search_res <- search.bin(data = data0,
                         combinations = get.combinations(sizes = x_sizes),
                         metrics = metric_options)
print(search_res)

# Use summary function to estimate the best model:
search_sum <- summary(search_res, y = sample$y, x = data[,3:ncol(data)])

# format and print coefficients:
s_fit <- summary(search_res)
print(s_fit$results[[1]]$value)

# Try a step-wise search for creating a larger model set:
search_res <- search.bin(data = data0,
                         combinations = get.combinations(
                           sizes = list(c(1, 2, 3), c(4)),
                           stepsNumVariables = c(NA, 7)),
                         metrics = metric_options)
# combinations argument is different

print(search_res)
# Use summary like before.
```

---

search.rfunc *Create a Model Set for an R Function*

Description

Use this model to create a model set for an R function.

Usage

```
search.rfunc(
  data = get.data(),
  combinations = get.combinations(),
  metrics = get.search.metrics(),
  modelChecks = get.search.modelchecks(),
```

```
  items = get.search.items(),
  options = get.search.options(),
  rFuncName,
  length1,
  isInnerExogenous
)
```

Arguments

- `data` — A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
- `combinations` — A list that determines the combinations of endogenous and exogenous variables in the search process. Use `get.combinations()` function to define it.
- `metrics` — A list of options for measuring performance. Use get.search.metrics function to get them.
- `modelChecks` — A list of options for excluding a subset of the model set. See and use get.search.modelchecks function to get them.
- `items` — A list of options for specifying the purpose of the search. See and use get.search.items function to get them.
- `options` — A list of extra options for performing the search. See and use get.search.options function to get them.
- `rFuncName` — Name of a function that uses column indices and number of endogenous variables with respect to `data`. It should estimate a model and return a list with required performance statistics. See details.
- `length1` — An integer for the length of requested information. This can be the number of exogenous variables.
- `isInnerExogenous` — If `TRUE`, exogenous indices are defined by `innerGroups` in the `combinations` argument.

Details

The central part of calling this function is to write a function with `rFuncName` name. This function must have the following arguments:

- `columnIndices`: determines the variables to be used in the current iteration. These indices point to the column of `data$data` matrix. E.g., you can create a matrix of available data by using `data$data[,colIndices]`. It contains weight column index (at `numEndo+1`), if `data$hasWeight` is `TRUE`.
- `numEndo`: can be used to divide the `columnIndices` into endogenous and exogenous indices.
- `data, metrics, modelChecks, items`: The arguments of current function which are passed to this function.

The `rFuncName` function should use these arguments and estimate or predict by using any available R function.

This function must return a `List` with the following items:

- `error` (Character string or NULL): It not `NULL` or empty, it is considered as a failed estimation with the given message.
- `metrics` (Numeric Matrix): Model performance for each target variable. Available target variables must be in the columns and metrics in the rows.
- `extra` (Numeric Vector or NULL): Extra information in form of integers, which defines the current model.
- `type1means` (Numeric Matrix or NULL): Means of `type1` (coefficients or predictions) for each target variable. Target variables must be in the columns. Make sure to skip the rows which the model does not present any information.
- `type1vars` (Numeric Matrix or NULL): similar to `type1means` but for reporting the variances.

Value

A nested list with the following members:

`counts`	Information about the expected number of models, number of estimated models, failed estimations, and some details about the failures.
`results`	A data frame with requested information in `items` list.
`info`	The arguments and some general information about the search process such as the elapsed time.

Note that the output does not contain any estimation results, but minimum required data to estimate the models (Use `summary()` function to get the estimation).

`search.steps` *Step-wise estimation*

Description

This function uses the calculated inclusion weights and selects a subset of variables in each step. Note that it uses the values for the first target variable and first metric and might not be suitable for multi-target or multi-metric searches.

Usage

```
search.steps(method, isInnerExogenous, ...)
```

Arguments

`method`	sur, bin or varma
`isInnerExogenous`	Determines if the inner indices are for exogenous variables.
`...`	Additional arguments for the search function.

Value

the result

search.sur *Create a Model Set for SUR Models*

Description

Use this function to create a Seemingly Unrelated Regression model set and search for the best models (and other information) based on in-sample and out-of-sample evaluation metrics.

Usage

```
search.sur(
  data = get.data(),
  combinations = get.combinations(),
  metrics = get.search.metrics(),
  modelChecks = get.search.modelchecks(),
  items = get.search.items(),
  options = get.search.options(),
  searchSigMaxIter = 0,
  searchSigMaxProb = 0.1
)
```

Arguments

- `data` A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
- `combinations` A list that determines the combinations of endogenous and exogenous variables in the search process. Use `get.combinations()` function to define it.
- `metrics` A list of options for measuring performance. Use get.search.metrics function to get them.
- `modelChecks` A list of options for excluding a subset of the model set. Use get.search.modelchecks function to get them.
- `items` A list of options for specifying the purpose of the search. Use get.search.items function to get them.
- `options` A list of extra options for performing the search. Use get.search.options function to get them.
- `searchSigMaxIter` Maximum number of iterations in searching for significant coefficients. Use 0 to disable the search.
- `searchSigMaxProb` Maximum value of type I error to be used in searching for significant coefficients. If p-value is less than this, it is interpreted as significant.

Value

A nested list with the following members:

counts	Information about the expected number of models, number of estimated models, failed estimations, and some details about the failures.
results	A data frame with requested information in items list.
info	The arguments and some general information about the search process such as the elapsed time.

Note that the output does not contain any estimation results, but minimum required data to estimate the models (Use `summary()` function to get the estimation).

See Also

estim.sur

Examples

```
num_y <- 2L # number of equations
num_x_r <- 3L # number of relevant explanatory variables
num_x_ir <-
  10 # (relatively large) number of irrelevant explanatory variables
num_obs = 100  # number of observations

# create random data
sample <- sim.sur(sigma = num_y, coef = num_x_r, nObs = num_obs)
x_ir <- matrix(rnorm(num_obs * num_x_ir), ncol = num_x_ir) # irrelevant data

# prepare data for estimation
data <- data.frame(sample$y, sample$x, x_ir)
colnames(data) <- c(colnames(sample$y), colnames(sample$x), paste0("z", 1:num_x_ir))

# Use systemfit to estimate and analyse:
exp_names <- paste0(colnames(data)[(num_y + 1):(length(colnames((data))))], collapse = " + ")
fmla <- lapply(1:num_y, function(i) as.formula(paste0("Y", i, " ~ -1 + ", exp_names)))
fit <- systemfit::systemfit(fmla, data = data, method = "SUR")
summary(fit)

# You can also use this package estimation function:
fit <- estim.sur(data = get.data(data, endogenous = num_y, addIntercept = FALSE))
print(fit)

# Alternatively, You can define an SUR model set:
x_sizes = c(1:3) # assuming we know the number of relevant explanatory variables is less than 3
num_targets = 2
metric_options <- get.search.metrics(typesIn = c("sic")) # We use SIC for searching
search_res <- search.sur(data = get.data(data, endogenous = num_y, addIntercept = FALSE),
                         combinations = get.combinations(numTargets = num_targets,
                                                         sizes = x_sizes,
                                                        innerGroups = list(c(1), c(2))),
                         metrics = metric_options)
```

```
print(search_res)

# Use summary function to estimate the best models:
search_sum <- summary(search_res)

# Print the best model:
print(search_sum$results[[2]]$value)
#   see 'estim.sur' function

# Using a step-wise search to build a larger model set:
x_sizes_steps = list(c(1, 2, 3), c(4))
counts_steps = c(NA, 7)
search_step_res <- search.sur(data = get.data(data, endogenous = num_y, addIntercept = FALSE),
                              combinations = get.combinations(numTargets = num_targets,
                                                              sizes = x_sizes_steps,
                                                       stepsNumVariables = counts_steps,
                                                           innerGroups = list(c(1,2))),
                              metrics = metric_options)
# combinations argument is different

print(search_step_res)
```

---

search.varma *Create Model Set for VARMA Models*

Description

Use this function to create a Vector Autoregressive Moving Average model set and search for the best models (and other information) based on in-sample and out-of-sample evaluation metrics.

Usage

```
search.varma(
  data = get.data(),
  combinations = get.combinations(),
  metrics = get.search.metrics(),
  modelChecks = get.search.modelchecks(),
  items = get.search.items(),
  options = get.search.options(),
  maxParams = c(1, 0, 0, 0, 0, 0),
  seasonsCount = 0,
  maxHorizon = 1,
  simUsePreviousEstim = FALSE,
  olsStdMultiplier = 2,
  lbfgsOptions = get.options.lbfgs()
)
```

Arguments

- `data` — A list that determines data and other required information for the search process. Use `get.data()` function to generate it from a `matrix` or a `data.frame`.
- `combinations` — A list that determines the combinations of endogenous and exogenous variables in the search process. Use `get.combinations()` function to define it.
- `metrics` — A list of options for measuring performance. Use get.search.metrics function to get them.
- `modelChecks` — A list of options for excluding a subset of the model set. Use get.search.modelchecks function to get them.
- `items` — A list of options for specifying the purpose of the search. Use get.search.items function to get them.
- `options` — A list of extra options for performing the search. Use get.search.options function to get them.
- `maxParams` — An integer vector that determines the maximum values for the parameters of the VARMA model: `(p,d,q,P,D,Q)`. If `NULL`, `c(2,0,0,0,0,0)` is used.
- `seasonsCount` — An integer value representing the number of observations per unit of time.
- `maxHorizon` — An integer value representing the maximum value for the prediction horizon if `type1` is `TRUE` in the `modelChecks` argument. Also, it is used as the maximum prediction horizon in checking predictions.
- `simUsePreviousEstim` — If `TRUE`, parameters are initialized only in the first step of the simulation. The initial values of the n-th simulation (with one more observation) are the estimations from the previous step.
- `olsStdMultiplier` — A number used as a multiplier for the standard deviation of OLS, used for restricting maximum likelihood estimation.
- `lbfgsOptions` — A list containing L-BFGS optimization options. Use get.options.lbfgs function for initialization.

Value

A nested list with the following members:

- `counts` — Information about the expected number of models, number of estimated models, failed estimations, and some details about the failures.
- `results` — A data frame with requested information in `items` list.
- `info` — The arguments and some general information about the search process such as the elapsed time.

Note that the output does not contain any estimation results, but minimum required data to estimate the models (Use `summary()` function to get the estimation).

See Also

estim.varma

Examples

```
# We simulate some data for this example:
set.seed(340)
n = 100
num_eq <- 3L
num_ar <- 2L
num_ma <- 1L
num_ma <- 1L
num_exo <- 2L
sample <- sim.varma(num_eq, arList = num_ar, maList = num_ma, exoCoef = num_exo, nObs = n)

# (relatively large) number of irrelevant explanatory variables:
num_y_ir <- 10
y_ir <- lapply(1:num_y_ir, function(x) rnorm(n))

# prepare data:
data <- data.frame(sample$y, y_ir, sample$x)
colnames(data) <- c(colnames(sample$y), paste0("w", 1:num_y_ir), colnames(sample$x))


# Create a VARMA model set:
y_sizes = 3 # assuming we know the number of relevant endogenous variables
metric_options <- get.search.metrics(typesIn = c("aic")) # We use SIC for searching
search_res <- search.varma(data = get.data(data, endogenous = num_eq + num_y_ir),
                           combinations = get.combinations(sizes = y_sizes,
                                                           numTargets = 3),
                           metrics = metric_options,
                           maxHorizon = 0)
print(search_res)
```

---

sim.bin *Generate Random Sample from a DC Model*

Description

This function generates a random sample from an discrete choice regression model.

Usage

```
sim.bin(
  coef = 2L,
  nObs = 100,
  probit = FALSE,
  maxWeight = 1,
  pPos = 0.5,
  sampleFactor = 4,
  toNumeric = TRUE
)
```

Arguments

coef	Either a single integer specifying the number of variables in the model, or a numeric vector of coefficients for the regression.
nObs	The number of observations to generate.
probit	Logical value indicating whether to generate data from a probit model (if TRUE) or a logit model (if FALSE).
maxWeight	Integer value indicating the maximum weight of the observations. If 1, observations are not weighted. If larger than 1, a vector of weights is generated and included in the return list. The weights are drawn from a discrete uniform distribution with a maximum value determined by maxWeight. If weighted, a larger sample is created (nObs * sampleFactor * maxWeight) and a subset of them is randomly selected, where the probability of selection is determined by the weight.
pPos	The percentage of positive observations (y=1) in the endogenous variable y. Must be between 0 and 1. In the current implementation, this is independent of the weights, if maxWeight is larger than 1.
sampleFactor	The factor used to control the size of the initial sample. A larger value generates a larger initial sample, which can increase the accuracy of the generated sample but also takes more time and memory.
toNumeric	If TRUE, y and w are transformed to have numeric vector. Otherwise, they contain an integer vector.

Value

A list with the following items:

y	The endogenous variable.
x	The exogenous variables.
w	The weights of the observations. It is NULL if weighted is FALSE.
p1	Prob(Y=1)
coef	The coefficients of the regression.
probit	Logical value indicating whether data was generated from a probit model.
pPos	The percentage of negative observations in y.

See Also

estim.bin, search.bin

Examples

```
# Generate data from a logit model with 3 variables
sample <- sim.bin(3L, 100)

# see the examples in 'estim.bin' or 'search.bin' functions
```

sim.sur — *Generate Random Sample from an SUR Model*

Description

This function generates a random sample from an Seemingly Unrelated Regression model.

Usage

```
sim.sur(sigma = 1L, coef = 1L, nObs = 100, intercept = TRUE)
```

Arguments

sigma	covariance matrix of the errors. If it is an integer value, it specifies the number of equations in the SUR model and covariance matrix is generated randomly.
coef	Coefficients of the model. If it is an integer value, it specifies the number of exogenous variables in each equation of the SUR model and coefficient matrix is generated randomly.
nObs	Number of observations to generate.
intercept	If TRUE, an intercept is included in the model as the first exogenous variable.

Value

A list with the following items:

y	matrix, the generated endogenous variable(s).
x	matrix, the generated exogenous variable(s).
e	matrix, the generated errors.
sigma	matrix, the covariance matrix of the disturbances.
coef	matrix, the coefficients used in the model.
intercept	logical, whether an intercept was included in the model.

See Also

sim.varma,estim.sur,search.sur

Examples

```
num_y <- 2L
num_x <- 3L
n_obs = 100
data <- sim.sur(sigma = num_y, coef = num_x, nObs = n_obs)

# see the examples in 'estim.sur' or 'search.sur' functions
```

sim.varma *Generate Random Sample from a VARMA Model*

Description

This function generates a multivariate time series using a VARMA process.

Usage

```
sim.varma(
  sigma = 2L,
  arList = 1L,
  maList = 0L,
  exoCoef = 0L,
  nObs = 100,
  nBurn = 10,
  intercept = TRUE,
  d = 0,
  startFrequency = NULL,
  seasonalCoefs = NULL
)
```

Arguments

`sigma`	A positive definite matrix representing the covariance matrix of the white noise series or an integer representing the dimension of a random covariance matrix to generate.
`arList`	A list of matrices representing the AR coefficients of the VARMA model or an integer representing the number of random AR coefficients to generate.
`maList`	A list of matrices representing the MA coefficients of the VARMA model or an integer representing the number of random MA coefficients to generate. For identification purposes, it generates diagonal matrices.
`exoCoef`	A matrix representing the coefficients of the exogenous variables or an integer representing the number of random exogenous coefficients to generate.
`nObs`	An integer representing the number of observations to generate.
`nBurn`	An integer representing the number of burn-in observations to remove from the generated time series.
`intercept`	A numeric vector representing the intercept of the VARMA model or a logical value indicating whether to generate a random intercept.
`d`	An integer representing the order of integration.
`startFrequency`	The frequency of the first observation in the data.
`seasonalCoefs`	An integer vector of size 4: `(P,D,Q,s)` where `P` is the number of random seasonal AR coefficients to generate, `Q` is the number of random seasonal MA coefficients to generate, `D` is the order of seasonal integration, and `s` is the number of seasons. These are effective if `arList` and `maList` are randomly generated within the function.

Value

A list with the following items:

y	The simulated endogenous data.
x	The simulated exogenous data.
e	The simulated white noise series.
sigma	The covariance matrix of the white noise series.
arList	The list of autoregressive coefficients.
maList	The list of moving average coefficients.
exoCoef	The matrix of exogenous coefficients.
intercept	The intercept vector.
d	The order of the integration.
seasonalCoefs	The argument seasonalCoefs
nObs	The number of observations generated.
nBurn	The number of burn-in observations removed.

Examples

```
sample1 <- sim.varma(2L, 3L, 2L)

ar1 <- matrix(c(0.7,0.2,-0.4,0.3),2,2)
ar2 <- matrix(c(-0.4,0.1,0.2,-0.3),2,2)
ma1 <- matrix(c(0.5,-0.1,0.3,0.4),2,2)
Sigma <- matrix(c(1,0.3,0.3,1),2,2)
B <- matrix(c(0.5,-0.3),2)

sample2 <- sim.varma(Sigma, list(ar1, ar2), list(ma1), exoCoef = B ,
                    nObs =100, nBurn =10 , intercept = c(1,-1))

# Plot the y series
matplot(sample2$y,type = "l")

# see the examples in 'estim.varma' or 'search.varma' functions
```

---

summary.ldt.search *Summary for an* ldt.search *object*

Description

Use this function to get the full estimation of the models reported in the output of a search process.

Usage

```
## S3 method for class 'ldt.search'
summary(object, test = FALSE, ...)
```

Arguments

`object`	An `ldt.search` object.
`test`	If `TRUE` and applicable (e.g., in model estimation), it checks the metrics and throws error for any inconsistencies between the current estimation and the one calculated in the search process (Provided that negative seed is used).
`...`	Additional arguments.

Details

An `ldt.search` object is an output from one of the `search.?` functions (see `search.sur`, `search.varma`, or `search.bin`).

Value

The output replaces the value of `object$results` with the summary from summary.ldt.search.item.

`summary.ldt.search.item` *Summary for an* `ldt.search.item` *object*

Description

While you can get a summary of an item in a search result, this function is mainly designed to be called from print.ldt.search function. Its main job is to estimate the full model using the reported indices from the search process.

Usage

```
## S3 method for class 'ldt.search.item'
summary(object, searchResult = NULL, test = FALSE, ...)
```

Arguments

`object`	An `ldt.search.item` object.
`searchResult`	Parent list of `object`, which is an `ldt.search` object.
`test`	If `TRUE` and applicable (e.g., in model estimation), it checks the metrics and throws error for any inconsistencies between the current estimation and the one calculated in the search process.
`...`	Additional arguments.

Details

An `ldt.search.item` object is a member of `ldt.search` object. An `ldt.search` object is an output from one of the `search.?` functions (see `search.sur`, `search.varma`, or `search.bin`).

Value

If the object contains the indices of endogenous variables of an estimated model, it returns the estimation output. Otherwise, it returns object.

Index