Package 'autovarCore'

August 29, 2016

Type Package

Title Automated Vector Autoregression Models and Networks

Version 1.0-0

Date 2015-06-29

BugReports https://github.com/roqua/autovarcore/issues

Maintainer Ando Emerencia <ando.emerencia@gmail.com>

Description Automatically find the best vector autoregression models and networks for a given time series data set. 'AutovarCore' evaluates eight kinds of models: models with and without log transforming the data, lag 1 and lag 2 models, and models with and without day dummy variables. For each of these 8 model configurations, 'AutovarCore' evaluates all possible combinations for including outlier dummies (at 2.5x the standard deviation of the residuals) and retains the best model. Model evaluation includes the Eigenvalue stability test and a configurable set of residual tests. These eight models are further reduced to four models because 'AutovarCore' determines whether adding day dummies improves the model fit.

License MIT + file LICENSE

Suggests testthat, roxygen2

Imports Rcpp (>= 0.11.4), Amelia, jsonlite, parallel, stats, urca, vars

LinkingTo Rcpp

NeedsCompilation yes

Author Ando Emerencia [aut, cre]

Repository CRAN

Date/Publication 2015-07-01 14:41:42

R topics documented:

`autovarCore-package` *Automated Vector Autoregression Networks*

Description

Find contemporaneous and dynamic relations in time series data and return them as a graph.

Details

Package:	autovarCore
Type:	Package
Version:	1.0-0
Date:	2015-03-03
License:	Unlimited

Please see help of the `autovar` function for information on how to use this package.

Author(s)

Ando Emerencia

Maintainer: Ando Emerencia <ando.emerencia@gmail.com>

`apply_ln_transformation` *Applies the natural logarithm to the data set*

Description

This applies the ln function columnwise to the given input matrix and returns the modified matrix. If necessary, columns undergo a linear translation to ensure that all resulting values are >= 0.

Usage

```
apply_ln_transformation(data_matrix)
```

Arguments

`data_matrix`	The original data matrix.

Value

The log-transformed data matrix.

Examples

```
data_matrix <- matrix(1:10, dimnames = list(NULL, 'some_val'))
data_matrix
autovarCore:::apply_ln_transformation(data_matrix)
```

`assess_joint_sktest` *Tests the skewness and kurtosis of a VAR model*

Description

This function tests the joint skewness and kurtosis for the residuals of the endogenous variables in the specified VAR model. This function uses an implementation equivalent to STATA's `sktest`. Of the p-levels resulting from assessing the significance of the joint sktest for the residuals of that variable, the minimum is returned.

Usage

```
assess_joint_sktest(varest)
```

Arguments

varest	A varest model.

Value

This function returns a p-level.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::assess_joint_sktest(varest)
```

---

assess_kurtosis *Tests the kurtosis of a VAR model*

Description

This function tests the kurtosis for the residuals of the endogenous variables in the specified VAR model. This function uses an implementation equivalent to STATA's sktest. Of the p-levels resulting from assessing the significance of the kurtosis for the residuals of that variable, the minimum is returned.

Usage

```
assess_kurtosis(varest)
```

Arguments

varest	A varest model.

Value

This function returns a p-level.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::assess_kurtosis(varest)
```

assess_portmanteau *Tests the white noise assumption for a VAR model using a portmanteau test on the residuals*

Description

This function tests the white noise assumption for the residuals of the endogenous variables in the specified VAR model. This function implements the portmanteau test known as the Ljung-Box test, and results are comparable with STATA's `wntestq`. Of the p-levels resulting from assessing the white noise assumption for the residuals of that variable, the minimum is returned.

Usage

```
assess_portmanteau(varest)
```

Arguments

`varest`	A `varest` model.

Value

This function returns a p-level.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::assess_portmanteau(varest)
```

assess_portmanteau_squared *Tests the homeskedasticity assumption for a VAR model using a portmanteau test on the squared residuals*

Description

This function tests the homeskedasticity assumption for the residuals of the endogenous variables in the specified VAR model. This function implements the portmanteau squared test known as the Ljung-Box test, and results are comparable with STATA's `wntestq`. Of the p-levels resulting from assessing the homeskedasticity assumption for the squared residuals of that variable, the minimum is returned.

Usage

```
assess_portmanteau_squared(varest)
```

Arguments

varest	A varest model.

Value

This function returns a p-level.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::assess_portmanteau_squared(varest)
```

---

assess_skewness *Tests the skewness of a VAR model*

Description

This function tests the skewness for the residuals of the endogenous variables in the specified VAR model. This function uses an implementation equivalent to STATA's sktest. Of the p-levels resulting from assessing the significance of the skewness for the residuals of that variable, the minimum is returned.

Usage

```
assess_skewness(varest)
```

Arguments

varest	A varest model.

Value

This function returns a p-level.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::assess_skewness(varest)
```

autovar *Return the best VAR models found for a time series data set*

Description

This function evaluates possible VAR models for the given time series data set and returns a sorted list of the best models found. The first item in this list is the "best model" found.

Usage

```
autovar(raw_dataframe, selected_column_names, significance_levels = c(0.05,
  0.01, 0.005), test_names = c("portmanteau", "portmanteau_squared",
  "skewness"), criterion = "AIC", imputation_iterations = 30,
  measurements_per_day = 1)
```

Arguments

raw_dataframe	The raw, unimputed data frame. This can include columns other than the selected_column_names, as those may be helpful for the imputation.
selected_column_names	The endogenous variables in the models, specified as a vector of character strings. This argument is required. The selected column names should be a subset of the column names of raw_dataframe.
significance_levels	A vector with descending p values that indicate cut-offs placing models in different buckets. If it is not specified, this parameter defaults to c(0.05, 0.01, 0.005). For example, with the default configuration, a model whose worst (lowest) p-level for any test is 0.03 is always seen as a better model than one whose worst p-level for any test is 0.009, no matter the AIC/BIC score of that model. Also, the lowest significance level indicates the minimum p-level for any test of a valid model. Thus, if a test for a model has a lower p-level than the minimum specified significance level, it is considered invalid.
test_names	The residual tests that should be performed, specified as a vector of character strings. If not specified, this parameter defaults to c('portmanteau', 'portmanteau_squared', 'skew The possible tests are c('portmanteau', 'portmanteau_squared', 'skewness', 'kurtosis', 'jo In addition to the residual tests, please note that the Eigenvalue stability test is always performed.
criterion	The information criterion used to sort the models. Valid options are 'AIC' (the default) or 'BIC'.
imputation_iterations	The number of times we average over one Amelia call for imputing the data set. Since one Amelia call averages over five imputations on its own, the actual number of imputations is five times the number specified here. The default value for this parameter is 30.

`measurements_per_day`
: The number of measurements per day in the time series data. The default value for this parameter is `1`. If this value is `0`, then daypart- and day-dummies variables are not included for any models.

Details

AutovarCore evaluates eight kinds of models: models with and without log transforming the data, lag 1 and lag 2 models, and with and without day dummy variables. For each of these 8 model configurations, we evaluate all possible combinations for including outlier dummies (at 2.5x the standard deviation of the residuals) and retain the best model (the procedure for selecting the best model is described in more detail below).

These eight models are further reduced to four models because AutovarCore determines whether adding daydummies improves the model fit (considering only significance bucket and the AIC/BIC score, NOT the number of outlier dummy columns), and only when the best model found with day dummies is a "better model" than the best model found without day dummies (other parameters kept the same), do we include the model with daydummies and discard the one without day dummies. Otherwise, we include only the model without daydummies and discard the one with daydummies.

Thus, AutovarCore always returns four models (assuming that we find enough models that pass the Eigenvalue stability test: models that do not pass this test are immediately discarded). There are three points in the code where we determine the best model, which is done according to what we refer to as algorithm A or algorithm B, which we explain below.

When evaluating all possible combinations of outlier dummies for otherwise identical model configurations, we use algorithm A to determine the best model. When comparing whether the best model found without day dummy columns is better than the best model found with day dummy columns, we use algorithm B. For sorting the two models with differing lag but both being either logtransformed or not, we again use algorithm A. Then at the end we merge the two lists of two models (with and without logtransform) to obtain the final list of four models. The sorting comparison here uses algorithm B.

The reason for the different sorting algorithms is that in some cases we want to select the model with the fewest outlier dummy columns (i.e., the model that retains most of the original data), while in other cases we know that a certain operation (such as adding day dummies or logtransforming the data set) will affect the amount of day dummies in the model and so a fair comparison would exclude this property.

Algorithm A applies the following rules for comparing two models in order:

1. We first consider the significance bucket. If the two models being compared are in different significance buckets, choose the one with the highest significance bucket, otherwise proceed to step 2.

   The significance buckets are formed between each of the (decreasingly sorted) specified `significance_levels` in the parameters to the autovar function call. For example, if the `signifance_levels` are `c(0.05, 0.01, 0.005)`, then the significance buckets are `(0.05 <= x)`, `(0.01 <= x < 0.05)`, `(0.005 <= x < 0.01` and `(x < 0.005)`. The metric used to place a model into a bucket is the maximum p-level that can be chosen as cut-off for determining whether an outcome is statistically significant such that all residual tests will still pass ("pass" meaning not invalidating the assumption that the residuals are normally distributed). In other words: it is the minimum p-value of all three residual tests of all endogenous variables in the model.

2. If the two models being compared are in the same significance bucket, the number of outlier columns is most important. If the two models being compared have a different amount of outlier columns, choose the one with the least amount of outlier columns, otherwise proceed to step 4.

   For this count of outlier columns, the following rules apply:

   - Day-part dummies do not add to the count. This is because when they are included, they are included for each model and thus never have any discriminatory power.
   - Day dummies count as one outlier column in total (so including day dummies will add one outlier column). This is because we do not necessarily want to punish models if the data happens to exhibit weekly cyclicity, but models that do not need the day dummies and are equally likely should be preferred.
   - Outlier dummy variables are split up such that each time point that is considered an outlier has its own dummy outlier variable and adds one to the count of outlier columns. The outliers are, for each variable, the measurements at >2.5 times the standard deviation away from the mean of the residuals or of the squared residuals. Checks are in place to ensure that a time point identified as an outlier by multiple variables only adds a single dummy outlier column to the equation.

3. When the bucket and number of outlier columns for the two models being compared are the same, select the one with the lowest AIC/BIC score. Whether the AIC or BIC is used here depends on the `criterion` option specified in the parameters to the autovar function call.

In the end, we should have one best logtransformed model and one best nonlogtransformed model. We then compare these two models in the same way as we have compared all other models up to this point with one exception: we do not compare the number of outlier columns. Comparing the number of outliers would have likely favored logtransformed models over models without logtransform, as logtransformations typically have the effect of reducing the outliers of a sample.

Algorithm B is identical to algorithm A, except that we skip the step comparing the number of outlier dummy variables. Thus, we instead compare by bucket first and AIC/BIC score second. Notice that, if we may assume that the presence or absence of day dummies does not vary between the four models for any particular invocation of the autovar method (which is not an unreasonable assumption to make), that then the arbitrary choice of letting all daydummy columns together add one to the outlier count does not matter at all, since the only times where we are comparing the outlier dummy counts is when both models either both have or both do not have day dummy columns.

We are able to compare the AIC/BIC scores of logtransformed and nonlogtransformed models fairly because we compensate the AIC/BIC scores to account for the effect of the logtransformation. We compensate for the logtransformation by adjusting the loglikelihood score of the logtransformed models in the calculation of their AIC/BIC scores (by subtracting the sum of the logtransformed data).

Value

A sorted list of "valid" models. A "model" is a list with the properties `logtransformed`, `lag`, `varest`, `model_score`, `bucket`, and `nr_dummy_variables`.

Examples

```
## Not run:
data_matrix <- matrix(nrow = 40, ncol = 3)
```

```
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
while (sum(is.na(data_matrix)) == 0)
  data_matrix[as.logical(round(runif(ncol(data_matrix) * nrow(data_matrix), -0.3, 0.7)))] <- NA
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
dataframe <- as.data.frame(data_matrix)
autovar(dataframe, selected_column_names = c('rumination', 'happiness'),
                   significance_levels = c(0.05, 0.01, 0.005),
                   test_names = c('portmanteau',
                                  'portmanteau_squared',
                                  'skewness'),
                   criterion = 'AIC',
                   imputation_iterations = 30,
                   measurements_per_day = 1)

## End(Not run)
```

`compete` *Returns the winning model*

Description

This function returns the best model as explained in the documentation for the `autovar` function.

Usage

```
compete(best, challenger, compare_outliers)
```

Arguments

`best`	A model given as a list with at least the properties `model_score`, `nr_dummy_variables`, and `bucket`.
`challenger`	Another model, also given as a list with properties `model_score`, `nr_dummy_variables`, and `bucket`.
`compare_outliers`	A boolean. When `FALSE`, the model comparison does not take the number of dummy variables into account.

Value

This function returns the best model of the two given models.

Examples

```
model1 <- list(logtransformed = FALSE, lag = 1, nr_dummy_variables = 1,
               model_score = 100, bucket = 0.05)
model2 <- list(logtransformed = FALSE, lag = 2, nr_dummy_variables = 2,
               model_score = 200, bucket = 0.01)
autovarCore:::compete(model1, model2, TRUE)
```

daypart_dummies *Calculate day-part dummy variables*

Description

This function returns either NULL (if measurements_per_day is 0 or 1) or a matrix of dummy variables for the specified input configuration.

Usage

```
daypart_dummies(number_of_rows, measurements_per_day)
```

Arguments

number_of_rows	the number of rows in the input data set.
measurements_per_day	the number of measurements per day in the input data set.

Value

Either NULL or a matrix with number_of_rows rows and measurements_per_day - 1 columns.

Examples

```
autovarCore:::daypart_dummies(10, 3)
```

day_dummies *Calculate day dummy variables*

Description

This function returns either NULL (if measurements_per_day is 0) or a matrix of daydummy variables specified number of rows and measurements per day. In the latter case, we return a matrix of six columns.

Usage

```
day_dummies(number_of_rows, measurements_per_day)
```

Arguments

number_of_rows	the number of rows in the input data set.
measurements_per_day	the number of measurements per day in the input data set.

Value

Either NULL or a matrix with number_of_rows rows and 6 columns.

Examples

```
autovarCore:::day_dummies(16, 2)
```

---

explode_dummies *Explode dummies columns into separate dummy variables*

Description

This function takes a matrix with dummy outlier columns, where there are possibly multiple ones. We first merge these columns to one and then explode them to obtain one dummy variable per column.

Usage

```
explode_dummies(outlier_dummies)
```

Arguments

outlier_dummies	A matrix of outlier dummy variables in columns.

Value

A matrix with dummy variables in columns, each having one nonzero index. The columns are named outlier_x, with x being the 1-based row index of the position that this dummy variable is masking.

Examples

```
outlier_dummies <- matrix(NA,
                          nrow = 5,
                          ncol = 3,
                        dimnames = list(NULL, c('rumination', 'happiness', 'activity')))
outlier_dummies[, 1] <- c(0, 0, 1, 0, 1)
outlier_dummies[, 2] <- c(0, 1, 1, 0, 0)
outlier_dummies[, 3] <- c(1, 0, 0, 0, 1)
outlier_dummies
autovarCore:::explode_dummies(outlier_dummies)
```

`impute_datamatrix` *Imputes the missing values in the input data*

Description

This function uses `Amelia::amelia` to try and impute missing (`NA`) values in the input data set. Amelia averages over five iterations, which is multiplied by the given `imputation_iterations` parameter.

Usage

```
impute_datamatrix(data_matrix, measurements_per_day, imputation_iterations)
```

Arguments

`data_matrix`	The raw, unimputed data matrix.
`measurements_per_day`	The number of measurements per day. This variable is used for adding day part dummy variables to aid the imputation.
`imputation_iterations`	The amount of times the Amelia call should be averaged over. The actual number of imputations is five times the value for `imputation_iterations`, since Amelia's values are already averaged over five runs.

Value

This function returns the modified matrix.

Examples

```
# create a matrix with some missing values
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
while (sum(is.na(data_matrix)) == 0)
  data_matrix[as.logical(round(runif(ncol(data_matrix) * nrow(data_matrix), -0.3, 0.7)))] <- NA
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
data_matrix
autovarCore:::impute_datamatrix(data_matrix, 1, 30)
```

invalid_mask *Calculate a bit mask to identify invalid outlier dummies*

Description

Invalid outlier dummy variables are dummy variables that are all zeros (where the original variable had no outliers at the 2.5x standard deviation for either the residuals or the squared residuals. Interpreting the leftmost column as bit 0 and continuing with higher bits going from left to right in the matrix, this function returns a bit mask that has a 1 on all positions in the matrix where the dummy column is invalid. We use this in later functions to easily filter out theinvalid outlier masks from the valid ones.

Usage

```
invalid_mask(outlier_dummies)
```

Arguments

outlier_dummies	A matrix of outlier dummy variables in columns.

Value

An integer mask indicating the invalid columns according to the procedure describe above.

Examples

```
resid_matrix <- matrix(rnorm(39 * 3),
                       nrow = 39,
                       ncol = 3,
                       dimnames = list(NULL, c('rumination', 'happiness', 'activity')))
outlier_dummies <- autovarCore:::residual_outliers(resid_matrix, 40)
autovarCore:::invalid_mask(outlier_dummies)
```

model_is_stable *Eigenvalue stability condition checking*

Description

This function returns whether the given model satisfies the Eigenvalue stability condition. The Eigenvalue stability condition is satisfied when all eigen values lie in the unit circle.

Usage

```
model_is_stable(varest)
```

Arguments

varest	A varest model.

Value

This function returns TRUE if the model satisfies the Eigenvalue stability condition and FALSE otherwise.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::model_is_stable(varest)
```

---

model_score *Return the model fit for the given varest model*

Description

This function returns the model fit for the given model as either an AIC or BIC score. We compensating for logtransformation so that the model scores of logtransformed and non-logtransformed models can be compared with each other directly. This compensation is implemented by subtracting the logtransformed data from the log-likelihood score and using the result as log-likelihood score for the AIC/BIC calculations.

Usage

```
model_score(varest, criterion, logtransformed)
```

Arguments

varest	A varest model.
criterion	A character string being either 'AIC' or 'BIC'.
logtransformed	A boolean, either TRUE or FALSE, indicating whether the input data for the model has been logtransformed.

Value

This returns a floating point that is either the AIC or BIC criterion for the model. A lower number corresponds to a better model fit.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::model_score(varest, 'AIC', FALSE)
```

---

needs_trend *Determines if a trend is required for the specified VAR model*

Description

This function uses the Phillips-Perron Unit Root Test to determine whether a trend is required for a VAR model based on the given matrix of endogenous variables and the given lag. All variables are assessed individually. This function returns TRUE if any of the endogenous variables requires a trend.

Usage

```
needs_trend(endo_matrix, lag)
```

Arguments

endo_matrix	The matrix of endogenous variables in the model.
lag	An integer specifying the lag length of the model.

Value

A boolean indicating whether a trend is required for the specified VAR model.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, 10)
data_matrix[, 3] <- (1:40) + rnorm(40)
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
data_matrix
autovarCore:::needs_trend(data_matrix, 1)
```

---

rcpp_hello_world *Simple function using Rcpp*

Description

Simple function using Rcpp

Usage

```
rcpp_hello_world()
```

`residual_outliers` *Calculate dummy variables to mask residual outliers*

Description

This function returns a matrix with columns that have a `1` at indices where the residuals have an outlier, and a `0` everywhere else. Outliers are calculated per variable (column) separately. We consider residual outliers the rows in the column of residuals or in the column of squared residuals that are more than 2.5 times the standard deviation away from the mean (standard deviation and mean are calculated separately per column and for residuals/squared residuals). The dummy columns are prepended with zeros to match the size of the other input variables to the model.

Usage

```
residual_outliers(resid_matrix, number_of_rows)
```

Arguments

`resid_matrix`	A matrix of residuals. Column names are copied over to the returned result.
`number_of_rows`	The number of measurements that were input to the model. Since the length of the residual matrix is shorter depending on the amount of lags in the model, we use `number_of_rows` to specify the number of rows in the returned matrix.

Value

A matrix with dummy variables in columns following the procedure described above.

Examples

```
resid_matrix <- matrix(rnorm(39 * 3),
                       nrow = 39,
                       ncol = 3,
                       dimnames = list(NULL, c('rumination', 'happiness', 'activity')))
resid_matrix[13, 2] <- 48
resid_matrix[23, 2] <- -62
resid_matrix[36, 2] <- 33
resid_matrix[27, 3] <- 75
resid_matrix
autovarCore:::residual_outliers(resid_matrix, 40)
```

run_tests *Execute a series of model validity assumptions*

Description

This function returns the given suite of tests on for the given VAR model. For each test, the result is the minimum p-level of all the assumptions and p-levels checked within the test. In other words, the result of a test is the p-level that should be used as a threshold below which outcomes are considered statistically significant (e.g., a result of 0.06 is better than a result of 0.03). The `run_tests` function returns a vector of results, one for each test, in the order corresponding to the `test_names` argument.

Usage

```
run_tests(varest, test_names)
```

Arguments

`varest`	A `varest` model.
`test_names`	A vector of names of tests given as character strings. Supported tests are specified in the `autovarCore:::supported_test_names()` function.

Value

This function returns a vector of p-levels.

Examples

```
data_matrix <- matrix(nrow = 40, ncol = 3)
data_matrix[, ] <- runif(ncol(data_matrix) * nrow(data_matrix), 1, nrow(data_matrix))
colnames(data_matrix) <- c('rumination', 'happiness', 'activity')
varest <- autovarCore:::run_var(data_matrix, NULL, 1)
autovarCore:::run_tests(varest, 'portmanteau')
```

run_var *Calculate the VAR model and apply restrictions*

Description

This function calls the `vars::var` function to calculate the VAR model and applies restrictions if needed. We set the intercept to 1 for restricted equations because calculations go wrong otherwise (this is a bug in the vars library).

Usage

```
run_var(endo_matrix, exo_matrix, lag)
```

Arguments

endo_matrix	A numeric matrix of endogenous data.
exo_matrix	Either NULL or a numeric matrix of exogenous data.
lag	A nonnegative integer specifying the lag length of the model. Specifying 0 for the lag results in calculating a lag 1 model with all lag-1 terms restricted.

Value

A varest object with the VAR estimation result.

Examples

```
endo_matrix <- matrix(rnorm(120), ncol = 2, nrow = 60,
                      dimnames = list(NULL, c("rumination", "activity")))
autovarCore:::run_var(endo_matrix, NULL, 1)
```

---

selected_columns *Convert an outlier_mask to a vector of column indices*

Description

This function returns an ordered vector of all the 1-toggled bits in the outlier_mask offset by 1.

Usage

```
selected_columns(outlier_mask)
```

Arguments

outlier_mask	An integer representing the outlier mask.

Value

A vector of column indices corresponding to the outlier mask.

Examples

```
outlier_mask <- 7
autovarCore:::selected_columns(outlier_mask)
```

select_valid_masks *Select and return valid dummy outlier masks*

Description

Valid dummy outlier masks are integers whose bitwise AND with the given `invalid_mask` is zero. This function returns the subset of integers of the given vector that does not share any bits with the given `invalid_mask`.

Usage

```
select_valid_masks(all_outlier_masks, invalid_mask)
```

Arguments

`all_outlier_masks`	A vector of possible outlier masks (integers).
`invalid_mask`	An integer encoding the invalid columns.

Value

The valid outlier_masks as a vector of integers.

Examples

```
all_outlier_masks <- c(0, 1, 2, 3, 4, 5, 6, 7)
invalid_mask <- 1
autovarCore:::select_valid_masks(all_outlier_masks, invalid_mask)
```

trend_columns *Construct linear and quadratic trend columns*

Description

This function returns a matrix of linear and quadratic trends.

Usage

```
trend_columns(number_of_rows)
```

Arguments

`number_of_rows`	the number of rows in the input data set.

Value

A matrix with number_of_rows rows and 2 columns, one for linear trends and one for quadratic trends.

Examples

```
autovarCore:::trend_columns(10)
```

---

validate_params *Validates the params given to the autovar function*

Description

This function uses a list of default params that may be overwritten the params argument. stop() errors are thrown when invalid params are supplied.

Usage

```
validate_params(data_matrix, params)
```

Arguments

data_matrix	The raw, unimputed data matrix. This parameter is supplied so that we can verify the selected column names.
params	A list with the following named entries:

- selected_column_names - The endogenous variables in the models, specified as a vector of character strings. This argument is required. The selected column names should be a subset of the column names of data_matrix.
- significance_levels - A vector with descending p values that indicate cut-offs placing models in different buckets. If it is not specified, this parameter defaults to c(0.05, 0.01, 0.005). For example, with the default configuration, a model whose worst (lowest) p-level for any test is 0.03 is always seen as a better model than one whose worst p-level for any test is 0.009, no matter the AIC/BIC score of that model. Also, the lowest significance level indicates the minimum p-level for any test of a valid model. Thus, if a test for a model has a lower p-level than the minimum specified significance level, it is considered invalid.
- test_names - The residual tests that should be performed, specified as a vector of character strings. If not specified, this parameter defaults to c('portmanteau', 'portmanteau_squared', 'skewness'). The possible tests are returned by the function autovarCore:::supported_test_names(). In addition to the residual tests, please note that the Eigenvalue stability test is always performed.
- criterion - The information criterion used to sort the models. Valid options are 'AIC' (the default) or 'BIC'.

- `imputation_iterations` - The number of times we average over one Amelia call for imputing the data set. Since one Amelia call averages over five imputations on its own, the actual number of imputations is five times the number specified here. The default value for this parameter is `30`.
- `measurements_per_day` - The number of measurements per day in the time series data. The default value for this parameter is `1`.

Value

A list containing augmented params.

Examples

```
data_matrix <- matrix(ncol = 3, nrow = 5)
data_matrix[, 1] <- 1
data_matrix[, 2] <- c(1, 3, 5, 6, 7)
data_matrix[, 3] <- c(1, 0, 1, NA, 1)
colnames(data_matrix) <- c('id', 'tijdstip', 'home')
autovarCore:::validate_params(data_matrix,
                              list(selected_column_names = c('tijdstip', 'home'),
                                   imputation_iterations = 20))
```

---

validate_raw_dataframe

Validates the dataframe given to the autovar function

Description

This function returns the given data frame as a numeric matrix, using `as.numeric` to convert any columns in the data frame that are not numeric. A `stop()` error is thrown if there is not enough data in the data frame.

Usage

```
validate_raw_dataframe(raw_dataframe)
```

Arguments

`raw_dataframe`	The raw, unimputed data frame.

Value

A numeric matrix with converted values and names taken from the data frame.

Examples

```
raw_dataframe <- data.frame(id = rep(1, times = 5),
  tijdstip = c(1, 3, 5, 6, 7),
  home = c(1, 0, 0, NA, 1))
autovarCore:::validate_raw_dataframe(raw_dataframe)
```

Index