Package ‘lmvar’

October 13, 2022

Type Package

Title Linear Regression with Non-Constant Variances

Version 1.5.2

Author Posthuma Partners <`info@posthuma-partners.nl`>

Maintainer Marco Nijmeijer <`nijmeijer@posthuma-partners.nl`>

Description Runs a linear-like regression with in which both the expected value and the variance can vary per observation. The expected values mu follows the standard linear model mu = X_mu * beta_mu. The standard deviation sigma follows the model log(sigma) = X_sigma * beta_sigma. The package comes with two vignettes: 'Intro' gives an introduction, 'Math' gives mathematical details.

License GPL-3

LazyData TRUE

Imports Matrix (>= 1.2-4), matrixcalc (>= 1.0-3), maxLik (>= 1.3-4), stats (>= 3.2.5), parallel (>= 3.3.0), graphics (>= 3.3.0), grDevices (>= 3.3.0)

RoxygenNote 6.1.1

Suggests testthat, knitr, rmarkdown, R.rsp, MASS, plotly (>= 4.7.1)

VignetteBuilder knitr, R.rsp

ByteCompile true

NeedsCompilation no

Repository CRAN

Date/Publication 2019-05-16 10:10:10 UTC

R topics documented:

`AIC.lmvar` *AIC for an object of class 'lmvar'*

Description

AIC (Aikaike's 'An Information Criterion') for an object of class 'lmvar'

Usage

```
## S3 method for class 'lmvar'
AIC(object, ..., k = 2)
```

Arguments

`object`	Object of class 'lmvar'
`...`	For compatibility with `AIC` generic
`k`	Numeric, the penalty per parameter to be used. The default k = 2 is the classical AIC.

Value

the AIC of the object

Examples

```
## Not run:

# Let 'fit' be an object of class 'lmvar'. The classical AIC is
AIC(fit)

# To calculate the AIC with penalty-parameter k = 3 run
AIC(fit, k = 3)

## End(Not run)
```

`alias.lmvar_no_fit` *Aliased coefficients in an 'lmvar' object*

Description

Returns the columns present in the user-specified model-matrices X_μ and X_σ that were removed by `lmvar` to make the matrices full-rank.

Usage

```
## S3 method for class 'lmvar_no_fit'
alias(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class 'lmvar_no_fit' (hence it can also be of class 'lmvar')
`mu`	Boolean, specifies whether the aliased columns from the model matrix X_μ must be returned
`sigma`	Boolean, specifies whether the aliased columns from the model matrix X_σ must be returned
`...`	Additional arguments, not used in the current implementation

Details

If `mu = TRUE` and `sigma = TRUE`, the function returns the aliased columns of both X_μ and X_σ. The string "_s" is appended to the aliased column names from X_σ if at least one of those names also appears in X_μ

If `mu = TRUE` and `sigma = FALSE`, the function returns the aliased columns of X_μ.

If `mu = FALSE` and `sigma = TRUE`, the function returns the aliased columns of X_σ.

Value

A character vector containing the names of the aliased columns

Examples

```
# Create matrix columns
my_intercept = rep(1, 20)
v1 = c(rep(1, 10), rep(0, 10))
v2 = c(rep(0, 10), rep(1, 10))

# Create model matrices
X = cbind(my_intercept, v1, v2)
X_s = X

# Rename the last column of the model matrix 'X_s' to make this example more clear.
colnames(X_s)[3] = ”v3”

# Create response vector
y = rnorm(20)

# Perform fit
fit = lmvar(y, X, X_s)

# The column 'my_intercept' is identical to '(Intercept)' added by 'lmvar'
# to the model matrix 'X'. Column 'v2' is equal to '(Intercept)' minus 'v1'.
# The same holds for the model matrix 'X_s'.
alias(fit)

# The aliased columns are left out if you extract the coefficients from a summary
coef(summary(fit))

# Only return the aliased colums in the model matrix for the expectation values
alias(fit, sigma = FALSE)

# Only return the aliased colums in the model matrix for the standard deviations
alias(fit, mu = FALSE)

# It also works on an object of class 'lmvar_no_fit'
no_fit = lmvar_no_fit(y, X, X_s)
alias(no_fit, mu = FALSE)
```

`beta_sigma_names` *Unique names for beta_sigma*

Description

Returns adapted names for the coefficients β_σ to distinguish them from the names of the coefficients β_μ. This is a helper function which is used in situations where it is necessary or convenient for the coefficient names of β_σ to be different from β_μ.

Usage

```
beta_sigma_names(beta_mu_names, beta_sigma_names, ...)
```

Arguments

`beta_mu_names`	Character vector with the names of the coefficients β_μ
`beta_sigma_names`	Character vector with the names of the coefficients β_σ
`...`	Additional arguments, not used in the current implementation

Details

When the name of at least one coefficient in β_σ is equal to one of the names of the coefficients in β_μ, the string '_s' is appended to the names of all coefficients in β_σ. Otherwise, the names of the coefficients in β_σ are left unchanged.

Value

Named character vector with the names of the coefficients β_σ. The name of a vector element is the original name of the coefficient. The value is the adapted name. The name and the value are equal if no adaptation was needed.

Examples

```
# If the names in beta_sigma are all different from all of the names in
# beta_mu, the function returns the names of beta_sigma
mu_names = c("(Intercept)", "age", "gender")
sigma_names = c("(Intercept_s)", "smoker", "job_code")

beta_sigma_names(mu_names, sigma_names)

# If at least one of the names in beta_sigma is equal to a name in
# beta_mu, all the names in beta_sigma get the string '_s' appended,
# except for '(intercept_s)'
sigma_names = c("(Intercept_s)", "age", "job_code")

beta_sigma_names(mu_names, sigma_names)
```

`coef.lmvar` *Extracts coefficients from an 'lmvar' object.*

Description

Extracts maximum-likelihood estimators for β_μ and β_σ from an 'lmvar' object.

Usage

```
## S3 method for class 'lmvar'
coef(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class 'lmvar'
`mu`	Boolean, specifies whether or not to return the maximum-likelihood estimator for β_μ
`sigma`	Boolean, specifies whether or not to return the maximum-likelihood estimator for β_σ
`...`	For compatibility with `coef` generic

Details

When both `mu = TRUE` and `sigma = TRUE`, the names of the coefficients in β_σ are adapted to distinguish them from the names in β_μ, if needed.

Value

When `mu = TRUE` and `sigma = TRUE`, a named numeric vector with the elements of β_μ, followed by the elements of β_σ.

When `mu = TRUE` and `sigma = FALSE`, a named numeric vector with the elements of β_μ.

When `mu = FALSE` and `sigma = TRUE`, a named numeric vector with the elements of β_σ.

See Also

`beta_sigma_names` for the adaptation of the names of the coefficients in β_σ.

`confint` for the calculation of confidence intervals of β_μ and β_σ.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist, attenu$mag + attenu$dist)
colnames(X) = c("mag", "dist", "mag+dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Extract all coefficients
coef(fit)

# Extract only the coefficients corresponding to the (non-aliased)
# columns in the model matrix for the expected values
coef(fit, sigma = FALSE)
```

```
# Extract only the coefficients corresponding to the (non-aliased)
# columns in the model matrix for standard deviations
coef(fit, mu = FALSE)
```

`convergence_precheck` *Pre-check model matrices for convergence issues*

Description

The model matrices X_μ and X_σ are checked to see if problems with the convergence of the fit can be anticipated. If so, it is determined which columns must be removed from X_σ to attempt to avoid convergence issues.

Usage

```
convergence_precheck(y, X_mu, X_sigma)
```

Arguments

`y`	Numeric, response vector y
`X_mu`	Model matrix for the expected values
`X_sigma`	Model matrix for the standard deviations. This must be a full-rank matrix.

Details

A matrix can be of class 'matrix', 'Matrix' or 'numeric' (in case it is a matrix of one column only).

An intercept term must be included in the model matrices if the model is such.

Value

A list with the following members:

- `column_numbers` The numbers of the columns of `X_sigma` that can be kept
- `column_names` The names of the columns of `X_sigma` that can be kept

Numbers and names refer to the same columns. They are supplied both for convenience.

`cv.lm` *Cross-validation for an object of class 'lm'*

Description

k-fold cross-validation for an object of class 'lm'

Usage

```
cv.lm(object, k = 10, ks_test = FALSE, fun = NULL, log = FALSE,
  seed = NULL, max_cores = NULL, ...)
```

Arguments

`object`	Object of class 'lm'
`k`	Integer, number of folds
`ks_test`	Boolean, if `TRUE`, a Kolmogorov-Smirnov test is carried out. See details.
`fun`	User-specified function for which cross-validation results are to be obtained. See details.
`log`	Boolean, specifies whether `object` contains a fit to the response vector Y or its logarithm $\log Y$
`seed`	Integer, seed for the random number generator. The seed is not set when `seed` equals `NULL`.
`max_cores`	Integer, maximum number of CPU-cores that can be used. For the default value `NULL`, the number is set to the number of available cores minus one.
`...`	Other parameters, not used in the current implementation.

Details

Cross-validations: The function `cv.lm` carries out a k-fold cross-validation for a linear model (i.e. a 'lm' model). For each fold, an 'lm' model is fit to all observations that are not in the fold (the 'training set') and prediction errors are calculated for the observations in the fold (the 'test set'). The prediction errors are the absolute error $|y - \mu|$ and its square $(y - \mu)^2$. The average prediction errors over the observations in the fold are calculated, and the square root of the average of the squared errors is taken. Optionally, one can calculate a user-specified function `fun` for the test set and the 'lmvar' model resulting from the training set. Optionally, one can also calculate the Kolmogorov-Smirnov (KS) distance for the test set and its p-value.

The results for the k folds are averaged over the folds and standard deviations are calculated from the k results.

Requirements on the 'lm' object: `object` must contain the list-members `x` and `y`. I.e., it must be created by running `lm` with the options `x = TRUE` and `y = TRUE`.

User defined function: The argument `fun` allows a user to specify a function for which cross-validation results must be obtained. This function must meet the following requirements.

- Its arguments are:
  - `object_t` an object of class ’lm’,
  - `y` a numerical vector of response values and
  - `X` the model matrix for the response vector `y`.
- It returns a single numerical value.

Carrying out a k-fold cross-validation, the function is called k times with `object_t` equal to the fit to the training set, `y` equal to the response vector of the test set, and `X_mu` the design matrix of the test set.

If the evaluation of `fun` gives an error, `cv.lm` will give a warning and exclude that evaluation from the mean and the standard deviation of `fun` over the k folds. If the evaluation of `fun` gives a warning, it will be ignored.

In the cross-validations, `object_t` contains the design matrix used in the fit to the training set as `object_t$x`.

Kolmogorov-Smirnov test: When `ks_test = TRUE`, a Kolmogorov-Smirnov (KS) test is carried out for each fold. The test checks whether the standardized residuals $(y - \mu)/\sigma$ in a fold are distributed as a standard normal distribution. The KS-distance and the corresponding p-value are calculated for each fold. The test uses the function `ks.test`. The expectation values μ and standard deviation σ are calculated from the model matrices for the test set (the fold) and the ’lm’ fit to the training set.

Other: The number of available CPU cores is detected with `detectCores`.

Value

An object of class ’cvlmvar’, which is a list with the following items:

- `MAE` a list with two items
  - `mean` the sample mean of the absolute prediction error over the k folds
  - `sd` the sample standard deviation of the absolute prediction error over the k folds
- `MSE` a list with two items
  - `mean` the sample mean of the mean squared prediction error over the k folds
  - `sd` the sample standard deviation of the mean squared prediction error over the k folds
- `MSE_sqrt` a list with two items
  - `mean` the sample mean of the root mean squared prediction error over the k folds
  - `sd` the sample standard deviation of the root mean squared prediction error over the k folds
- `KS_distance` a list with two items
  - `mean` the sample mean of the Kolmogorov-Smirnov distance over the k folds
  - `sd` the sample standard deviation of the Kolmogorov-Smirnov distance over the k folds
- `KS_p.value` a list with two items
  - `mean` the sample mean of the p-value of Kolmogorov-Smirnov distance over the k folds
  - `sd` the sample standard deviation of the p-value of the Kolmogorov-Smirnov distance over the k folds

- `fun` a list with two items
  - `mean` the sample mean of the user-specified function `fun`
  - `sd` the sample standard deviation of the of the user-specified function over the k folds

The items `KS_distance` and `KS_p.value` are added only in case `ks_test = TRUE`. The item `fun` is added only in case a function `fun` has been specified.

See Also

`cv.lmvar` is the equivalent function for an object of class ’lmvar’. It is supplied in case one wants to compare an ’lmvar’ fit with an ’lm’ fit.

`print.cvlmvar` provides a print-method for an object of class ’cvlmvar’.

Examples

```
# Create an object of class 'lm'. We use a model matrix obtained from the 'cats' dataframe,
# an arbitrary parameter vector beta and a generated response vector y for the purpose of the
# example.
library(MASS)

X = model.matrix(~ Sex + Bwt, cats)
beta_mu = c(-0.1, 0.3, 4)

mu = X %*% beta_mu

y = rnorm( nrow(X), mean = mu, sd = 0.5)

fit = lm(y ~ ., as.data.frame(X[,-1]), x = TRUE, y = TRUE)

# Carry out a cross-validation
cv.lm(fit)

# Carry out a cross-validation using a single CPU-core
cv.lm(fit, max_cores = 1)

# Carry out a cross-validation including a Kolmogorov-Smirnov test, using at most two CPU-cores
cv.lm(fit, ks_test = TRUE, max_cores = 2)

# Carry out a cross-validation with 5 folds and control the random numbers used
cv.lm(fit, k = 5, seed = 5483, max_cores = 1)


# Calculate cross-validation results for the fourth moment of the residuals, using a
# user-specified function
fourth = function(object, y, X){
  mu = predict(object, as.data.frame(X))
  residuals = y - mu
  return(mean(residuals^4))
}
cv.lm(fit, fun = fourth)
rm(fourth)
```

```
# Use option 'log = TRUE' if you fit the log of the response vector and require error estimates for
# the response vector itself
fit = lm(log(y) ~ ., as.data.frame(X[,-1]), x = TRUE, y = TRUE)
cv = cv.lm(fit, log = TRUE)

# Print 'cv' using the print-method print.cvlmvar
cv

# Print 'cv' with a specified number of digits
print(cv, digits = 2)
```

`cv.lmvar` *Cross-validation for an object of class 'lmvar'*

Description

k-fold cross-validation for an object of class 'lmvar'

Usage

```
cv.lmvar(object, k = 10, ks_test = FALSE, fun = NULL, log = FALSE,
  seed = NULL, sigma_min = NULL, exclude = NULL,
  slvr_options = list(), max_cores = NULL, ...)
```

Arguments

`object`	Object of class 'lmvar'
`k`	Integer, number of folds
`ks_test`	Boolean, if `TRUE`, a Kolmogorov-Smirnov test is carried out. See details.
`fun`	User-specified function for which cross-validation results are to be obtained. See details.
`log`	Boolean, specifies whether `object` contains a fit to the response vector Y or its logarithm $\log Y$
`seed`	Integer, seed for the random number generator. The seed is not set when seed equals `NULL`.
`sigma_min`	Minimum value for the standard deviations. Can be a single number which applies to all observations, or a vector giving a minimum per observation. In case of the the default value `NULL`, the value is the same as the value in `object`.
`exclude`	Numeric vector with observations that must be excluded for error statistics. The default `NULL` means no observations are excluded. See 'Details' for more information.
`slvr_options`	List of options passed on to the function `maxLik` which carries out the fits for the k folds. See 'Details' for more information.
`max_cores`	Integer, maximum number of CPU-cores that can be used. For the default value `NULL`, the number is set to the number of available cores minus one.
`...`	Other parameters, not used in the current implementation.

Details

Cross-validations: The function `cv.lmvar` carries out a k-fold cross-validation for an 'lmvar' model. For each fold, an 'lmvar' model is fit to all observations that are not in the fold (the 'training set') and prediction errors are calculated for the observations in the fold (the 'test set'). The prediction errors are the absolute error $|y - \mu|$ and its square $(y - \mu)^2$. The average prediction errors over the observations in the fold are calculated, and the square root of the average of the squared errors is taken. Optionally, one can calculate a user-specified function `fun` for the test set and the 'lmvar' model resulting from the training set. Optionally, one can also calculate the Kolmogorov-Smirnov (KS) distance for the test set and its p-value.

The results for the k folds are averaged over the folds and standard deviations are calculated from the k results.

User defined function: The argument `fun` allows a user to specify a function for which cross-validation results must be obtained. This function must meet the following requirements.

- Its arguments are:
  - `object_t` an object of class 'lmvar',
  - `y` a numerical vector of response values and
  - `X_mu` the model matrix for the expected values of the response vector `y`.
  - `X_sigma` the model matrix for the standard deviations of the response vector `y`.
- It returns a single numerical value.

Carrying out a k-fold cross-validation, the function is called k times with `object_t` equal to the fit to the training set, `y` equal to the response vector of the test set, and `X_mu` and `X_sigma` the design matrices of the test set.

If the evaluation of `fun` gives an error, `cv.lmvar` will give a warning and exclude that evaluation from the mean and the standard deviation of `fun` over the k folds. If the evaluation of `fun` gives a warning, it will be ignored.

In the cross-validations, `object_t` contains the design matrices of the training set as `object_t$X_mu` and `object_t$X_sigma`. `object_t$X_mu` was formed by taking `object$X_mu` and removing the fold-rows. In addition, columns may have been removed to make the matrix full-rank. Therefore, `object_t$X_mu` may have fewer columns than `object$X_mu`. The same is true for `object_t$X_sigma` compared to `object$X_sigma`.

Kolmogorov-Smirnov test: When `ks_test = TRUE`, a Kolmogorov-Smirnov (KS) test is carried out for each fold. The test checks whether the standardized residuals $(y - \mu)/\sigma$ in a fold are distributed as a standard normal distribution. The KS-distance and the corresponding p-value are calculated for each fold. The test uses the function `ks.test`. The expectation values μ and standard deviations σ are calculated from the model matrices for the test set (the fold) and the 'lmvar' fit to the training set.

Excluding observations: The observations specified in the argument `exclude` are not used to calculate the error statistics MAE (mean absolute error), MSE (mean squared error) and the square root of MSE. They are also not used to calculate the statistics for the user-defined function `fun`. This is useful when there are a few observations with such large residuals that they dominate the error estimates. Note that the excluded observations are not excluded from the training sets. It is only in the calculation of the statistics of the test sets that the observations are excluded. They are not excluded from the KS-test: when observations have large residuals, they should have large standard deviations as well, to give the standardized residuals normal values.

Minimum sigma: The argument `sigma_min` gives the option to enforce a minimum standard deviation. This is useful when, in a cross-validation, a fit fails because the maximum likelihood occurs when the standard deviation of one or more observations becomes zero. When a minimum standard deviation is specified, all fits are carried out under the boundary condition that the standard deviation is larger than the minimum. If `sigma_min = NULL` the same value is used as was used to create `object`.

Other: The fits are carried out with the options `slvr_options` stored in the 'lmvar' object `object`. However, these options can be overwritten with an explicit argument `slvr_options` in the call of `cv.lmvar`. Some of the options are affected by a `sigma_min` larger than zero, see `lmvar` for details.

The argument `slvr_options` is a list, members of which can be a list themselves. If members of a sublist are overwritten, the other members of the sublist remain unchanged. E.g., the argument `slvr_options = list(control = list(iterlim = 600))` will set `control$iterlim` to 600 while leaving other members of the list `control` unchanged.

The number of available CPU cores is detected with `detectCores`.

Value

In case none of the fits in the cross-validations returns an error or a warning, a 'cvlmvar' object is returned. This is a list with the following items:

- `MAE` a list with two items
  - `mean` the sample mean of the absolute prediction error over the k folds
  - `sd` the sample standard deviation of the absolute prediction error over the k folds
- `MSE` a list with two items
  - `mean` the sample mean of the mean squared prediction error over the k folds
  - `sd` the sample standard deviation of the mean squared prediction error over the k folds
- `MSE_sqrt` a list with two items
  - `mean` the sample mean of the root mean squared prediction error over the k folds
  - `sd` the sample standard deviation of the root mean squared prediction error over the k folds
- `KS_distance` a list with two items
  - `mean` the sample mean of the Kolmogorov-Smirnov distance over the k folds
  - `sd` the sample standard deviation of the Kolmogorov-Smirnov distance over the k folds
- `KS_p.value` a list with two items
  - `mean` the sample mean of the p-value of Kolmogorov-Smirnov distance over the k folds
  - `sd` the sample standard deviation of the p-value of the Kolmogorov-Smirnov distance over the k folds
- `fun` a list with two items
  - `mean` the sample mean of the user-specified function `fun`
  - `sd` the sample standard deviation of the of the user-specified function over the k folds

The items `KS_distance` and `KS_p.value` are added only in case `ks_test = TRUE`.

In case a fit returns an error or a warning, the return value of `cv.lmvar` lists the arguments of the first call to `lmvar` which failed. In addition, it lists the row number of the observations in `object` that formed the training set for which the fit returned an error or warning. These items are returned as a list:

- `y` the argument `y` of the failing call
- `X_mu` the argument `X_mu` of the failing call
- `X_sigma` the argument `X_sigma` of the failing call
- `intercept_mu` the argument `intercept_mu` of the failing call
- `intercept_sigma` the argument `intercept_sigma` of the failing call
- `sigma_min` the argument `sigma_min` of the failing call
- `slvr_options` the argument `slvr_options` of the failing call
- `control` the argument `control` of the failing call
- `training_rows` numeric vector containing the rows of the observations in `object` that were used in the failing fit

See Also

See `lmvar` for the options `slvr_options` stored in an 'lmvar' object.

`cv.lm` is the equivalent function for an object of class 'lm'. It is supplied in case one wants to compare an 'lmvar' fit with an 'lm' fit.

`print.cvlmvar` provides a print-method for an object of class 'cvlmvar'.

Examples

```
# Create an object of class 'lmvar'. We use a model matrix obtained from the 'cats' dataframe,
# arbitrary parameter vectors beta and a generated response vector y for the purpose of the
# example.


library(MASS)

X = model.matrix(~ Sex + Bwt, cats)
beta_mu = c(-0.1, 0.3, 4)
beta_sigma = c(-0.5, -0.1, 0.3)

mu = X %*% beta_mu
log_sigma = X %*% beta_sigma

y = rnorm( nrow(X), mean = mu, sd = exp(log_sigma))

fit = lmvar(y, X_mu = X[,-1], X_sigma = X[,-1])

# Carry out a cross-validation
cv.lmvar(fit)

# Carry out a cross-validation using a single CPU-core
```

```
cv.lmvar(fit, max_cores = 1)

# Carry out a cross-validation including a Kolmogorov-Smirnov test, using at most two CPU-cores
cv.lmvar(fit, ks_test = TRUE, max_cores = 2)

# Carry out a cross-validation with 5 folds and control the random numbers used
cv.lmvar(fit, k = 5, seed = 5483, max_cores = 1)

# Carry out a cross-validation and exclude observations 5, 11 and 20 from the calculation of
# the error statistics
cv.lmvar(fit, exclude = c(5, 11, 20), max_cores = 1)

# Calculate cross-validation results for the fourth moment of the residuals, using a
# user-specified function
fourth = function(object, y, X_mu, X_sigma){
  mu = predict(object, X_mu[,-1], X_sigma[,-1], sigma = FALSE)
  residuals = y - mu
  return(mean(residuals^4))
}
cv.lmvar(fit, fun = fourth)
rm(fourth)

# Carry out a cross-validation and specify the maximization routine and maximum number of iterations
cv.lmvar(fit, slvr_options = list( method = "NR", control = list(iterlim = 500)))

# Use option 'log = TRUE' if you fit the log of the response vector and require error estimates for
# the response vector itself
fit = lmvar(log(y), X_mu = X[,-1], X_sigma = X[,-1])
cv = cv.lmvar(fit, log = TRUE)

# Print 'cv' using the print-method print.cvlmvar
cv

# Print 'cv' with a specified number of digits
print(cv, digits = 2)
```

`dfree` *Degrees of freedom for an object of class 'lmvar'*

Description

Degrees of freedom for the model in an object of class 'lmvar'. The degrees of freedom are defined as the rank of the model matrix X_μ for the expectation values, plus the rank of the model matrix X_σ for the standard deviations.

Usage

```
dfree(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class ’lmvar_no_fit’ (hence it can also be of class ’lmvar’)
`mu`	Boolean, specifies whether the degrees of freedom for the model for the expectation values must be included.
`sigma`	Boolean, specifies whether the degrees of freedom for the model for the standard deviations must be included.
`...`	Additional arguments, not used in the current implementation

Details

If `mu` = `TRUE` and `sigma` = `TRUE`, the function returns the rank of the model-matrix X_μ plus the rank of the model matrix X_σ.

If `mu` = `TRUE` and `sigma` = `FALSE`, the function returns the rank of the model-matrix X_μ.

If `mu` = `FALSE` and `sigma` = `TRUE`, the function returns the rank of the model-matrix X_σ.

Both model matrices contain a column corresponding to an intercept term. This column is added by `lmvar`. See also the vignette ’Intro’.

Value

An integer containing the degrees of freedom for the model in `object`.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c(”mag”, ”dist”)

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c(”mag”, ”dist_inv”)

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# The degrees of freedom are
dfree(fit)

# The degrees of freedom of the expected values are
dfree(fit, sigma = FALSE)

# The degrees of freedom of the standard deviations are
dfree(fit, mu = FALSE)

# Function also works on object of class 'lmvar_no_fit'
no_fit = lmvar_no_fit(attenu$accel, X, X_s)
```

```
dfree(no_fit)
```

`fisher` *Fisher information matrix for an object of class 'lmvar'*

Description

Fisher information matrix for an object of class 'lmvar'.

Usage

```
fisher(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class 'lmvar'
`mu`	Specifies whether or not the block-matrix for β_μ is included in the returned matrix
`sigma`	Specifies whether or not the block-matrix for β_σ is included in the returned matrix
`...`	Additional arguments, not used in the current implementation

Details

The Fisher information matrix is calculated as minus $-E[H]/n$ with $E[H]$ the expected value of the Hessian matrix H of the log-likelihood and n the number of observations.

The matrix is calculated using the maximum-likelihood estimators of μ and σ.

If `mu = TRUE` and `sigma = TRUE`, the full Fisher information matrix is returned.

If `mu = TRUE` and `sigma = FALSE`, only the left-upper block-matrix is returned, corresponding to the part of the Fisher information matrix pertaining to β_μ.

If `mu = FALSE` and `sigma = TRUE`, only the right-lower block-matrix is returned, corresponding to the part of the Fisher information matrix pertaining to β_σ.

Value

An object of class 'matrix' containing the Fisher information matrix of `object`.

See Also

`vcov.lmvar` calculates the covariance matrix for the maximum-likelihood estimators of β_μ and β_μ

`nobs.lmvar_no_fit` for the number of observations in an object of class 'lmvar'

`coef.lmvar` for the coefficients β_μ and β_σ

`fitted.lmvar` for the expectation values μ and standard deviations σ.

See the vignette "Math" (to be viewed with `vignette("Math", "lmvar")`) for details.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# The complete Fisher information matrix is
fisher(fit)

# The left-upper block matrix relating to the expected values is
fisher(fit, sigma = FALSE)

# The right-lower block matrix relating to the variances is
fisher(fit, mu = FALSE)
```

`fitted.lmvar` *Fitted values for an 'lmvar' object*

Description

Estimators and confidence intervals for the expected values and standard deviations of the response vector Y. Prediction intervals for Y. Alternatively, estimators and intervals can be for e^Y.

Usage

```
## S3 method for class 'lmvar'
fitted(object, mu = TRUE, sigma = TRUE, log = FALSE,
  interval = c("none", "confidence", "prediction"), level = 0.95, ...)
```

Arguments

`object`	An 'lmvar' object
`mu`	Boolean, specifies whether or not to return estimators and intervals for the expected values
`sigma`	Boolean, specifies whether or not to return estimators and intervals for the standard deviations
`log`	Boolean, specifies whether estimators and intervals should be for Y (`log = FALSE`) or for e^Y (`log = TRUE`).

`interval`	Character string, specifying the type of interval. Possible values are • "none" No interval, this is the default • "confidence" Confidence intervals for the estimators • "prediction" Prediction intervals
`level`	Numeric value between 0 and 1, specifying the confidence level
`...`	For compatibility with `fitted` generic.

Details

If `log = FALSE`, `fitted.lmvar` returns estimators and intervals for the observations Y stored in `object`.

If `log = TRUE`, `fitted.lmvar` returns estimators and intervals for e^Y.

Confidence intervals are calculated under the assumption of asymptotic normality. This assumption holds when the number of observations is large. Intervals must be treated cautiously in case of a small number of observations. Intervals can also be unreliable if `object` was created with a constraint on the minimum values of the standard deviations sigma.

This function is identical to the function `predict.lmvar` in which the parameters `X_mu` and `X_sigma` are left unspecified.

Value

In the case `mu = FALSE` and `interval = "none"`: a numeric vector containing the estimators for the standard deviation.

In the case `sigma = FALSE` and `interval = "none"`: a numeric vector containing the estimators for the expected values.

In all other cases: a matrix with one column for each requested feature and one row for each observation. The column names are

- `mu` Estimators for the expected value μ
- `sigma` Estimators for the standard deviation σ
- `mu_lwr` Lower bound of the confidence interval for μ
- `mu_upr` Upper bound of the confidence interval for μ
- `sigma_lwr` Lower bound of the confidence interval for σ
- `sigma_upr` Upper bound of the confidence interval for σ
- `lwr` Lower bound of the prediction interval
- `upr` Upper bound of the prediction interval

See Also

`predict.lmvar` for expected values, standard deviations and intervals for model matrices different from the ones present in `object`.

`coef.lmvar` and `confint` for maximum likelihood estimators and confidence intervals for β_μ and β_σ.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
y = attenu$accel
fit = lmvar(y, X, X_s)

# Calculate the expected value of each observation
fitted(fit, sigma = FALSE)

# Calculate the standard deviation of each observation
fitted(fit, mu = FALSE)

# Calculate the expected values and their 95% confidence intervals
fitted(fit, sigma = FALSE, interval = "confidence")

# Calculate the standard deviations and their 80% confidence intervals
fitted(fit, mu = FALSE, interval = "confidence", level = 0.8)

# Calculate both the expected values and the standard deviations
fitted(fit)

# Calculate the expected values, the standard deviations and their 95% confidence intervals
fitted(fit, interval = "confidence")

# Calculate the expected values and the 90% prediction intervals
fitted(fit, interval = "prediction", level = 0.9)

# Fit the log of 'accel'
y = log(attenu$accel)
fit_log = lmvar(y, X, X_s)

# Calculate both the expected values and the standard deviations of the log of 'accel'
fitted(fit_log)

# Calculate the expected values and the standard deviations of 'accel'
fitted(fit_log, log = TRUE)

# Calculate the expected values and the standard deviations of 'accel',
# as well as their 90% confidence intervals
fitted(fit_log, log = TRUE, interval = "confidence", level = 0.9)
```

`fwbw` *Forward / backward-step model selection*

Description

Model selection by a forward / backward-stepping algorithm. The algorithm reduces the degrees of freedom of an existing object containing a model fit. It searches for the subset of degrees of freedom that results in an optimal goodness-of-fit. This is the subset for which a user-specified function reaches its minimum. The search is carried out by alternately attempting to remove and insert degrees of freedom.

Usage

```
fwbw(object, fun, ...)
```

Arguments

`object`	Object containing a fit to a specific model
`fun`	User-specified function which measures the goodness-of-fit.
`...`	Further arguments for specific methods

Value

A list with the following members.

- `object` An object which contains the model for which `fun` is minimized.
- `fun` the minimum value of the user-specified function `fun`.

See Also

`fwbw.lm` and `fwbw.lmvar_no_fit`

`fwbw.lm` *Forward / backward-step model selection for an object of class 'lm'*

Description

Model selection by a forward / backward-stepping algorithm. The algorithm reduces the degrees of freedom of an existing 'lm' object. It searches for the subset of degrees of freedom that results in an optimal goodness-of-fit. The optimal subset is the subset for which a user-specified function reaches its minimum.

Usage

```
## S3 method for class 'lm'
fwbw(object, fun, fw = FALSE, counter = TRUE,
  df_percentage = 0.05, control = list(), ...)
```

Arguments

`object`	Object of class 'lm'
`fun`	User-specified function which measures the goodness-of-fit. See 'Details'.
`fw`	Boolean, if `TRUE` the search will start with a minimum degrees of freedom ('forward search'). If `FALSE` the search will start with the full model ('backward search').
`counter`	Boolean, if `TRUE` and `fw = TRUE`, the algorithm will carry out backward steps (attempts to remove degrees of freedom) while searching for the optimal subset. If `FALSE` and `fw = TRUE`, the algorithm will only carry out forward steps (attempts to insert degrees if freedom). The effect of `counter` is opposite if `fw = FALSE`.
`df_percentage`	Percentage of degrees of freedom that the algorithm attempts to remove at a backward-step, or insert at a forward_step. Must be a number between 0 and 1.
`control`	List of control options. The following options can be set • `monitor` Boolean, if `TRUE` information about the attempted removals and insertions will be printed during the run. Default is `FALSE`. • `plot` Boolean, if `TRUE` a plot will be shown at the end of the run. It shows how the value of `fun` decreases during the run. Default is `FALSE`.
`...`	for compatibility with `fwbw` generic

Details

Description of the algorithm: The function `fwbw.lm` selects the subset of all the degrees of freedom present in `object` for which the user-specified function `fun` is minimized. This function is supposed to be a measure for the foodness-of-fit. Typical examples would be `fun=AIC` or `fun=BIC`. The function `fun` can also be a measure of the prediction error, determined by cross-validation.

This function is intended for situations in which the degrees of freedom in `object` is so large that it is not feasible to go through all possible subsets systematically to find the smallest value of `fun`. Instead, the algorithm generates subsets by removing degrees of freedom from the current-best subset (a 'backward' step) and reinserting degrees of freedom that were previously removed (a 'forward' step). Whenever a backward or forward step results in a subset for which `fun` is smaller than for the current-best subset, the new subset becomes current-best.

The start set depends on the argument `fw`. If `fw = TRUE`, the algorithm starts with only one degree of freedom for the expected values μ. This degree is the intercept term, if the model in `object` contains an intercept term. If `fw = FALSE` (the default), the algorithm starts with all degrees of freedom present in `object`.

At a backward step, the model removes `df_percentage` of the degrees of freedom of the current-best subset (with a minimum of 1 degree of freedom). The degrees that are removed are the ones with the largest p-value (p-values can be seen with the function `summary.lm`). If the removal results in a larger value of `fun`, the algorithm will try again by halving the degrees of freedom it removes.

At a forward step, inserts `df_percentage` of the degrees of freedom that are present in `object` but left out in the current-best subset (with a minimum of 1 degree of freedom). It inserts those degees of freedom which are estimated to increase the likelihood most. If the insertion results in a larger value of `fun`, the algorithm will try again by halving the degrees of freedom it inserts.

If `counter = FALSE`, the algorithm is 'greedy': it will only carry out forward-steps in case `fw = TRUE` or backward-steps in case `fw = FALSE`.

The algorithm stops if neither the backward nor the forward step resulted in a lower value of `fun`. It returns the current-best model and the minimum value of `fun`.

The user-defined function: The function `fun` must be a function which is a measure for the goodness-of-fit. It must take one argument: an object of class 'lm'. Its return value must be a single number. A smaller number (more negative) must represent a better fit. During the run, a fit to the data is carried out for each new subset of degrees of freedom. The result of the fit is an object of class 'lm'. This object is passed on to `fun` to evaluate the goodness-of-fit. Typical examples for `fun` are `AIC` and `BIC`.

Monitor information: When the `control`-option `monitor` is equal to `TRUE`, information is displayed about the progress of the run. The following information is displayed:

- `Iteration` A counter which first value is always `0`, followed by `1`. From then on, the counter is increased whenever the addition or removal of degrees of freedom results in a smaller function value than the smallest so far.
- `attempted removals/insertions` The number of degrees of freedoms that one attempts to remove or insert
- `function value` The value of the user-specified function `fun` after the removal or insertion of the degrees of freedom
- The last column shows the word `insert` when the attempt regards the insertion of degrees of freedom. When nothing is shown, the algorithm attempted to remove degrees of freedom.

Other: If the model matrix present in `object` conatains a column with the name "(Intercept)", the intercept term for the expected values μ will not be removed by `fwbw.lm`.

When a new subset of degrees of freedom is generated by either a backward or a forward step, the response vector in `object` is fitted to the new model. The fit is carried out by `lm`.

Value

A list with the following members.

- `object` An object of class 'lm' which contains the model for which `fun` is minimized.
- `fun` The minimum value of the user-specified function `fun`.

See Also

`fwbw` for the generic method

`fwbw.lmvar_no_fit` for the corresponding function for an 'lmvar_no_fit' (or an 'lmvar') object

Examples

```
# Generate model matrix
set.seed(1820)

n_rows = 1000
n_cols = 4

X = matrix(sample(-9:9, n_rows * n_cols, replace = TRUE), nrow = n_rows, ncol = n_cols)

column_names = sapply(1:n_cols, function(i_column){paste("column", i_column, sep = "_")})
colnames(X) = column_names

# Generate betas
beta = sample(c(-1,-0.5, 0.5, 1), n_cols + 1, replace = TRUE)

# Generate response vector
mu = X %*% beta[-1] + beta[1]
y = rnorm( n_rows, mean = mu, sd = 2.5)

# Add columns for cross-terms to model matrix. They have no predictive power for the response y.
X = model.matrix(~ . + 0 + column_1 * ., data = as.data.frame(X))
colnames(X)

# Create model in which cross-terms in X are unrelated to response vector y.
fit = lm(y ~ ., as.data.frame(X), x = TRUE, y = TRUE)

# Check whether model selection with BIC as criterion manages
# to remove cross-terms. Start with the full model. Monitor the iterations.
fwbw = fwbw(fit, BIC, control = list(monitor = TRUE))
names(coef(fwbw$object))

# The same with AIC as criterion. Plot how the AIC develops.
fwbw = fwbw(fit, AIC, control = list(plot = TRUE))
names(coef(fwbw$object))

# Model selection starting with an intercept term only.
fwbw = fwbw(fit, BIC, fw = TRUE)
names(coef(fwbw$object))
```

`fwbw.lmvar_no_fit` *Forward / backward-step model selection for an 'lmvar' object*

Description

Model selection by a forward / backward-stepping algorithm. The algorithm reduces the degrees of freedom of an existing 'lmvar' object. It searches for the subset of degrees of freedom that results in an optimal goodness-of-fit. This is the subset for which a user-specified function reaches its minimum.

Usage

```
## S3 method for class 'lmvar_no_fit'
fwbw(object, fun, fw = FALSE, counter = TRUE,
  df_percentage = 0.05, control = list(), ...)
```

Arguments

`object`	Object of class 'lmvar_no_fit' (hence it can also be of class 'lmvar')
`fun`	User-specified function which measures the goodness-of-fit. See 'Details'.
`fw`	Boolean, if `TRUE` the search will start with a minimum degrees of freedom ('forward search'). If `FALSE` the search will start with the full model ('backward search').
`counter`	Boolean, if `TRUE` and `fw = TRUE`, the algorithm will carry out backward steps (attempts to remove degrees of freedom) while searching for the optimal subset. If `FALSE` and `fw = TRUE`, the algorithm will only carry out forward steps (attempts to insert degrees if freedom). The effect of `counter` is opposite if `fw = FALSE`.
`df_percentage`	Percentage of degrees of freedom that the algorithm attempts to remove at a backward-step, or insert at a forward-step. Must be a number between 0 and 1.
`control`	List of control options. The following options can be set • `monitor` Boolean, if `TRUE` information about the attempted removals and insertions will be printed during the run. Default is `FALSE`. • `plot` Boolean, if `TRUE` a plot will be shown at the end of the run. It shows how the value of `fun` decreases during the run. Default is `FALSE`.
`...`	for compatibility with `fwbw` generic

Details

Description of the algorithm: The function `fwbw` selects the subset of all the degrees of freedom present in `object` for which the user-specified function `fun` is minimized. This function is supposed to be a measure for the goodness-of-fit. Typical examples would be `fun=AIC` or `fun=BIC`. Another example is where `fun` is a measure of the prediction error, determined by cross-validation or otherwise.

The function `fwbw` is intended for situations in which the degrees of freedom in `object` is so large that it is not feasible to go through all possible subsets systematically to find the smallest value of `fun`. Instead, the algorithm generates subsets by removing degrees of freedom from the current-best subset (a 'backward' step) and reinserting degrees of freedom that were previously removed (a 'forward' step). Whenever a backward or forward step results in a subset for which `fun` is smaller than for the current-best subset, the new subset becomes current-best.

The start set depends on the argument `fw`. If `fw = TRUE`, the algorithm starts with only two degrees of freedom: one for the expected values μ and one for the standard deviations σ. These degrees are the intercept terms, if the model in `object` contains them. If `fw = FALSE` (the default), the algorithm starts with all degrees of freedom present in `object`.

At a backward step, the model removes degrees of freedom of the current-best subset. It removes at least 1 degree of freeedom and at most `df_percentage` of the degrees in the current-best subset. The degrees that are removed are the ones with the largest p-value (p-values can be seen with the function `summary.lmvar`). If the removal results in a larger value of `fun`, the algorithm will try again by halving the degrees of freedom it removes.

At a forward step, the algorithm inserts degrees of freedom that are present in `object` but left out in the current-best subset. It inserts at least 1 degree of freedom and at most `df_percentage` of the current-best subset. It inserts those degees of freedom which are estimated to increase the likelihood most. If the insertion results in a larger value of `fun`, the algorithm will try again by halving the degrees of freedom it inserts.

If `counter = FALSE`, the algorithm is 'greedy': it will only carry out forward-steps in case `fw = TRUE` or backward-steps in case `fw = FALSE`.

The algorithm stops if neither the backward nor the forward step resulted in a lower value of `fun`. It returns the current-best model and the minimum value of `fun`.

The user-defined function: The function `fun` must be a function which is a measure for the goodness-of-fit. It must take one argument: an object of class 'lmvar'. Its return value must be a single number. A smaller (more negative) number must represent a better fit. During the run, a fit to the data is carried out for each new subset of degrees of freedom. The result of the fit is an object of class 'lmvar'. This object is passed on to `fun` to evaluate the goodness-of-fit. Typical examples for `fun` are `AIC.lmvar` and `BIC`.

Monitor information: When the `control`-option `monitor` is equal to `TRUE`, information is displayed about the progress of the run. The following information is displayed:

- `Iteration` A counter which first value is always `0`, followed by `1`. From then on, the counter is increased whenever the addition or removal of degrees of freedom results in a smaller function value than the smallest so far.
- `attempted removals/insertions` The number of degrees of freedoms that one attempts to remove or insert
- `function value` The value of the user-specified function `fun` after the removal or insertion of the degrees of freedom
- The last column shows the word `insert` when the attempt regards the insertion of degrees of freedom. When nothing is shown, the algorithm attempted to remove degrees of freedom.

Other: If `object` was created with `intercept_mu = TRUE`, the intercept term for the expected values μ will not be removed by `fwbw.lmvar`. Likewise for `intercept_sigma`.

When a new subset of degrees of freedom is generated by either a backward or a forward step, the response vector in `object` is fitted to the new model. The fit is carried out by `lmvar`. The arguments used in the call to `lmvar` (other than `X_mu` and `X_sigma`) are the same as used to create `object`, except that the control options `mu_full_rank` and `sigma_full_rank` are both set to TRUE. Setting them to TRUE can be done safely because the model matrices `object$X_mu` and `object$X_sigma` are guaranteed to be full-rank.

Value

A list with the following members.

- `object` An object of class 'lmvar' which contains the model for which `fun` is minimized.
- `fun` the minimum value of the user-specified function `fun`.

See Also

`fwbw` for the S3 generic method

`fwbw.lm` for the corresponding function for an 'lm' object

`lmvar` for the constructor of a 'lmvar' object

`lmvar_no_fit` for the constructor of a 'lmvar_no_fit' object

The number of degrees of freedom is given by `dfree`.

Examples

```
# Generate model matrices
set.seed(1820)

n_rows = 1000
n_cols = 4

X_mu = matrix(sample(-9:9, n_rows * n_cols, replace = TRUE), nrow = n_rows, ncol = n_cols)
X_sigma = matrix(sample(-9:9, n_rows * n_cols, replace = TRUE), nrow = n_rows, ncol = n_cols)

column_names = sapply(1:n_cols, function(i_column){paste("column", i_column, sep = "_")})
colnames(X_mu) = column_names
colnames(X_sigma) = paste(column_names, "_s", sep = "")

# Generate betas
beta_mu = sample(c(-1,-0.5, 0.5, 1), n_cols + 1, replace = TRUE)
beta_sigma = sample(c(-1,-0.5, 0.5, 1), n_cols + 1, replace = TRUE)

# Generate response vector
mu = X_mu %*% beta_mu[-1] + beta_mu[1]
log_sigma = X_sigma %*% beta_sigma[-1] + beta_sigma[1]
y = rnorm( n_rows, mean = mu, sd = exp(log_sigma))

# Add columns for cross-terms to model matrices. They have no predictive power for the response y.
X_mu = model.matrix(~ . + 0 + column_1 * ., data = as.data.frame(X_mu))
X_sigma = model.matrix(~ . + 0 , data = as.data.frame(X_sigma))
c( colnames(X_mu), colnames(X_sigma))

# Create lmvar object
fit = lmvar(y, X_mu, X_sigma)

# Check whether backward- / forward step model selection with BIC as criterion manages
# to remove cross-terms
fwbw = fwbw(fit, BIC, control = list(monitor = TRUE))
names(coef(fwbw$object))

# The same with AIC as criterion
fwbw = fwbw(fit, AIC, control = list(monitor = TRUE))
names(coef(fwbw$object))

# Model selection starting with an intercept term only.
fwbw = fwbw(fit, BIC, fw = TRUE)
```

```
names(coef(fwbw$object))

# It also works on an object of class 'lmvar_no_fit'
no_fit = lmvar_no_fit(y, X_mu, X_sigma)
fwbw( no_fit, AIC, control = list(monitor = TRUE))
```

`lmvar` *Linear regression with non-constant variances*

Description

Performs a Gaussian regression with non-constant variances and outputs an 'lmvar' object.

Usage

```
lmvar(y, X_mu = NULL, X_sigma = NULL, intercept_mu = TRUE,
  intercept_sigma = TRUE, sigma_min = 0, slvr_options = list(method =
  "NR"), control = list(), ...)
```

Arguments

`y`	Vector of observations
`X_mu`	Model matrix for the expected values μ
`X_sigma`	Model matrix for the logarithms of the standard deviations σ
`intercept_mu`	Boolean, if TRUE a column with the name '(Intercept)' is added to the matrix `X_mu`. This column represents the intercept term in the model for μ.
`intercept_sigma`	Boolean, if TRUE a column with the name '(Intercept_s)' is added to the matrix `X_sigma`. This column represents the intercept term in the model for $\log \sigma$.
`sigma_min`	Numeric, the minimum value for the standard deviations sigma. Can be a single number which applies to all observations or a vector containing a separate minimum for each observation.
`slvr_options`	A list with options to be passed on to the function `maxLik` which maximizes the log-liklihood
`control`	A list with further options. The following options can be set. • `check_hessian` Boolean, if TRUE it is checked whether the Hessian is negative-definite, i.e., whether the log-likelihood is at a maximum. The default value is TRUE. • `slvr_log` Boolean, if TRUE the output of `maxLik` is added to the 'lmvar' object. The default value is FALSE. • `monitor` Boolean, if TRUE diagnostic messages about errors that occur will be printed during the run. The default value is FALSE.

- `remove_df_sigma_pre` Warning: this is an experimental option which could cause unexpected issues! Boolean, if TRUE the algorithm removes degrees of freedom of the model for σ to avoid convergence problems. They are removed before carrying out the fit. See 'Details'. The default value is FALSE.
- `remove_df_sigma_post` Boolean, if TRUE the algorithm will remove degrees of freedom of the model for σ if this is necessary to achieve a satisfactory fit. They are removed after a fit has been attempted and turned out to fail. This option has no effect if `sigma_min` (or one of its elements) is larger than zero. See 'Details'. The default value is FALSE.
- `mu_full_rank` Boolean, if TRUE it is assumed that `X_mu` (with the intercept term added in case `intercept_mu = TRUE`) is full-rank. The default value is FALSE.
- `sigma_full_rank` Boolean, if TRUE it is assumed that `X_sigma` (with the intercept term added in case `intercept_sigma = TRUE`) is full-rank. The default value is FALSE.

`...`	Additional arguments, not used in the current implementation

Details

Response vector: The vector `y` must be a numeric vector. It can not contain special values like `NULL`, `NaN`, etc.

Model matrices: If the matrix `X_mu` is not specified, the model for the expected values μ will consist of an intercept term only. The argument `intercept_mu` is ignored in this case. Likewise, the model for $\log \sigma$ will consist of an intercept term only if `X_sigma` is not specified. In the latter case, the model reduces to a standard linear model.

Both model matrices must be numeric matrices. They can not contain special values like `NULL`, `NaN`, etc.

The model matrices can be of class `matrix` or of a class from the package `Matrix`. They can also be of class `numeric` in case they are matrices with one column only.

All columns in the matrix `X_mu` must either have a name, or no column has a name at all. It is not allowed that some colums have a name while others don't. The same is true for `X_sigma`.

If supplied, the column names must be unique within `X_mu`. The same is true for `X_sigma`. A column name can be used in both `X_mu` and `X_sigma` though.

In case the matrix `X_mu` has no columns with a column name, `lmvar` gives the names `v1`, `v2` etc. to the columns. Likewise, if the matrix `X_sigma` has no columns with a column name, `lmvar` gives the names `v1_s`, `v2_s` etc. to the columns.

Matrix `X_mu` can not have a column with the name '(Intercept)'. Matrix `X_sigma` can not have a column with the name '(Intercept_s)'. Both names are reserved names.

Minimum sigma: The argument `sigma_min` allows one to run a regression under the constraint of a minimum standard deviation for each observation. The argument can be a single number, which applies to all observations, or a vector which contains a separate minimum for each observation. In the latter case, all vector elements must be zero (in which case no constraint is applied) or all vector elements must be larger than zero.

The option of a minimum sigma is intended for situations in which an unconstrained regression attempts to make sigma equal to zero for one or more observations. This will cause extreme values for the β_σ and numerical instabilities.Such a situation can be remedied by bounding sigma from below.

In case `sigma_min` has a value (or a vector of values) larger than zero and the solve-method is "NR", the solve method is set to "BFGS". If the argument `constraints` is passed on to `maxlik` (as a list member of `slvr_options`), it is ignored.

Error estimates and confidence intervals (e.g. such as given by `summary.lmvar` and `predict.lmvar`) can be unreliable if minimum sigmas are specified.

Likelihood maximization: The function `maxLik` from the `maxLik` package, is used to maximize the (profile) log-likelihood. `maxLik` returns a termination code which reports whether a maximum was found, see `maxLik`. For the method "NR", a potential problem is reported by a `maxLik` return code different from `1`, `2` or `8`. For other methods, a code different from `0` flags a potential problem. In case the return code flags a potential problem, the message from `maxLik` is output as a warning.

All list elements in `slvr_options` are passed on as arguments to `maxLik`. The name of the list element is the argument name, the value of the list element is the argument value. It is not allowed to pass on the arguments `fn`, `grad` or `hess`. In case the list does not contain an element `method`, it is set to "NR". In case the list does not contain an element `start`, an initial estimate for β_σ is set by `lmvar`.

In case one wants to supply an initial estimate for the coefficients, one has to supply an initial estimate for β_σ. If `beta_sigma_initial` is the initial estimate, one passes on the argument `slvr_options = list(start = beta_sigma_initial)`. The inital estimate `beta_sigma_initial` must be a numeric vector. Its length must be as follows.

- In case `X_sigma` is not specified or has the value `NULL`, the initial estimate must be a single value.
- In case `X_sigma` is specified and `intercept_sigma = TRUE`: the length must be equal to the number of columns of `X_sigma` plus one. The first element of the vector is the initial estimate of the intercept term for $\log \sigma$, the second element is the inital estimate corresponding to the first column in `X_sigma`, the third element is the inital estimate corresponding to the second column in `X_sigma`, etc.
- In case `X_sigma` is specified and `intercept_sigma = FALSE`: the length must be equal to the number of columns of `X_sigma`. The first element of the vector is the initial estimate corresponding to the first column in `X_sigma`, the second element is the inital estimate corresponding to the second column in `X_sigma`, etc.

There is no need to supply an inital estimate for β_μ, see the vignette 'Math' for details.

Reducing the degrees of freedom to improve fit: When `maxLik` exits with return code 3 (and corresponding warning 'Last step could not find a value above the current. Boundary of parameter space?'), it somehow did not succeed to fit an 'lmvar' model properly. The same is true if the the Hessian if the log-likelihood is not negative-definite.

In this situation, a proper fit can sometimes be achieved if one drops a few extra columns (sometimes just one column) from `X_sigma`. See the vignette 'Math' for details. The options `control = list(remove_df_sigma_pre = TRUE, remove_df_sigma_post = TRUE))` do just that. They attempt to achieve a proper fit by dropping columns (i.e., reducing the degrees of freedom of the model for σ) if necessary.

The option `remove_df_sigma_pre` inspects the model matrices and the response vector before carrying out the fit, and drops columns from `X_sigma` if necessary. Warning: this is an experimental option which could cause unexpected issues!

The option `remove_df_sigma_post = TRUE` attempts to achieve a proper fit in the following two cases.

- `maxLik` uses the solve-method "NR" (the default method) or "BFGSR" and exits with return code 3. Note that this not the case when `sigma_min` (or one of its elements) has been set to a value larger than zero because then the method "BFGS" is used.
- The option `check_hessian` is `TRUE` and the Hessian of the log-likelihood is not negative-definite.

I.e., this option drops columns from `X_sigma` based on the results of a failed fit.

Other: When `check_hessian = TRUE`, it is checked whether the fitted log-likelihood is at a maximum. A warning will be issued if that is not the case.

The control options `mu_full_rank` and `sigma_full_rank` are for efficiency purposes. If set to TRUE, the corresponding model matrices will not be made full-rank because it is assumed they are full-rank already. However, setting one of these to TRUE while the corresponding model matrix is not full-rank will cause unpredictable and possibly unnoticed errors. These options should therefore only be changed from their default value with the utmost care.

See the vignettes that come with the `lmvar` package for more info. Run `vignette(package="lmvar")` to list the available vignettes.

Value

An object of class 'lmvar', which is a list. Users are discouraged to access list-members directly. Instead, list-members are to be accessed with the various accessor and utility functions in the package. Exceptions are the following list members for which no accessor functions exist:

- `y` the vector of observations
- `X_mu` the model matrix for μ. In general, it differs from the user-supplied `X_mu` because `lmvar` adds an intercept-column (unless `intercept_mu` is `FALSE`) and makes the matrix full-rank.
- `X_sigma` the model matrix for $\log \sigma$. In general, it differs from the user-supplied `X_sigma` because `lmvar` adds an intercept-column (unless `intercept_sigma` is `FALSE`) and makes the matrix full-rank.
- `intercept_mu` boolean which tells whether or not an intercept column `(Intercept)` has been added to the model matrix X_μ
- `intercept_sigma` boolean which tells whether or not an intercept column `(Intercept_s)` has been added to the model matrix X_σ
- `sigma_min` the value of the argument `sigma_min` in the call of `lmvar`
- `slvr_options` the value of the argument `slvr_options` in the call of `lmvar`
- `control` the value of the argument `control` in the call of `lmvar`
- `slvr_log` the output of `maxLik` (the solver routine used to maximize the likelihood). Included only if the argument `slvr_log` has the value `TRUE`. See `maxLik` for details about this output.

See Also

`lmvar_no_fit` to create an object which is like an 'lmvar' object without carrying out a model fit.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# For more info on the data, study the dataset
help("attenu")

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations. The standard deviation
# is large for small distances and small for large distances. The use of 'dist'
# as explanatory variable makes the beta for the intercept term blow up.
# Therefore we use '1 / dist' as explanatory variable
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Inspect the results
summary(fit)

# Carry out the fit with an inital estimate for beta and ask for
# a report of the solver-routine
beta_sigma_start = c(-4, 0, 0)
fit = lmvar(attenu$accel, X, X_s,
            slvr_options = list(start = beta_sigma_start),
            control = list(slvr_log = TRUE))
fit$slvr_log
```

`lmvar_no_fit` *Create an 'lmvar'-like object without a model fit*

Description

Creates an 'lmvar'-like object without carrying out a model fit. This object is a 'lmvar' object from which all members have been left out that are the result of the fit. Such an object can be used in functions which typically use an 'lmvar' object as input but do not need the fit results. Since no fit is performed, `lmvar_no_fit` is convenient when the fit is time-consuming or, e.g., does not converge.

Usage

```
lmvar_no_fit(y, X_mu = NULL, X_sigma = NULL, intercept_mu = TRUE,
  intercept_sigma = TRUE, sigma_min = 0, slvr_options = list(method =
  "NR"), control = list(), ...)
```

Arguments

`y`	Vector of observations
`X_mu`	Model matrix for the expected values μ
`X_sigma`	Model matrix for the logarithms of the standard deviations σ
`intercept_mu`	see the function `lmvar`
`intercept_sigma`	see the function `lmvar`.
`sigma_min`	see the function `lmvar`.
`slvr_options`	see the function `lmvar`
`control`	see the function `lmvar`.
`...`	Additional arguments, not used in the current implementation

Details

See `lmvar` for the requirements and a further explanation of all the arguments.

The class 'lmvar' is an extension of the class 'lmvar_no_fit'. This means that each object which is of class 'lmvar', is of class 'lmvar_no_fit' as well. Wherever an object of class 'lmvar_no_fit' is required, an object of class 'lmvar' can be used as well.

Accessor and utility functions for a 'lmvar_no_fit' object are: `nobs.lmvar_no_fit`, `alias.lmvar_no_fit` and `dfree`

The function `lmvar_no_fit` is especially useful in combination with `fwbw.lmvar_no_fit`. In case a model with many degrees of freedom does not converge with `lmvar`, one can create an 'lmvar_no_fit' object. This is used as input for `fwbw` with the argument `fw = TRUE`. The `fwbw` algorithm will try to select an optimal subset of all degrees of freedom, starting with the smallest subset possible.

Although `lmvar_no_fit` does not carry out a model fit, it will do the following:

- add an intercept term to the model matrices (unless `intercept_mu` is FALSE and/or `intercept_sigma` is FALSE), and
- make the model matrices full rank.

Value

An object of class 'lmvar_no_fit', which is a list. The list-members are the same as for an object of call 'lmvar', but with members that are the result of the model fit left out.

Users are discouraged to access list-members directly. Instead, list-members are to be accessed with the various accessor and utility functions in the package. Exceptions are the following list members for which no accessor functions exist:

- `call` the function call
- `y` the vector of observations
- `X_mu` the model matrix for μ. In general, it differs from the user-supplied `X_mu` because `lmvar_no_fit` adds an intercept-column (unless `intercept_mu` is FALSE) and makes the matrix full-rank.

- `X_sigma` the model matrix for $\log \sigma$. In general, it differs from the user-supplied `X_sigma` because `lmvar_no_fit` adds an intercept-column (unless `intercept_sigma` is FALSE) and makes the matrix full-rank.
- `intercept_mu` boolean which tells whether or not an intercept column `(Intercept)` has been added to the model matrix X_μ
- `intercept_sigma` boolean which tells whether or not an intercept column `(Intercept_s)` has been added to the model matrix X_σ
- `sigma_min` the value of the argument `sigma_min` in the call of `lmvar_no_fit`
- `slvr_options` the value of the argument `slvr_options` in the call of `lmvar_no_fit`
- `control` the value of the argument `control` in the call of `lmvar_no_fit`

Examples

```
# As example we sue the dataset 'iris' from the library 'datasets'
library(datasets)

# Create the model matrix for both the expected values and the standard deviations
X = model.matrix( ~ Species - 1, data = iris)

# Take as response variabe the variable Sepal.length
y = iris$Sepal.Length

# Construct a 'lmvar_no_fit' object
no_fit = lmvar_no_fit( y, X, X)

# The following functions all work on such an object
nobs(no_fit)
dfree(no_fit)
alias(no_fit)

# You can also supply 'lmvar' arguments
no_fit = lmvar_no_fit( y, X[,-1], X[,-1], intercept_mu = FALSE, intercept_sigma = FALSE)
dfree(no_fit)

# Some (most) arguments have no effect except that they are stored in the 'lmvar_no_fit'
# object
no_fit = lmvar_no_fit( y, X, X, control = list( slvr_log = TRUE, remove_df_sigma = TRUE))
no_fit$control
```

`logLik.lmvar` *Log-likelihood for an object of class 'lmvar'*

Description

Log-likelihood for an object of class 'lmvar'

Usage

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

Arguments

`object`	Object of class 'lmvar'
`...`	For compatibility with `logLik` generic

Value

'logLik' object, a number containing the log-likelihood with an attribute 'df' containing the degrees of freedom

See Also

`dfree` for the degrees of freedom for an object of class 'lmvar'.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Show the log-likelihood and the degrees of freedom of the fit
# using the 'print' method for an object of class 'logLik'
logLik(fit)

# Obtain the log-likelihood itself
logLik(fit)[1]
```

`nobs.lmvar_no_fit` *Number of observations for an object of class 'lmvar'*

Description

The number of observations in an object of class 'lmvar'.

Usage

```
## S3 method for class 'lmvar_no_fit'
nobs(object, ...)
```

Arguments

`object`	Object of class ’lmvar_no_fit’ (hence it can also be of class ’lmvar’)
`...`	For compatibility with `nobs` generic

Value

Integer containing the number of observations in the model in `object`.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Return the number of observations in the fit
nobs(fit)

# Check that this is equal to the number of observations in the dataset
nobs(fit) == nrow(attenu)

# Function also works on object of class 'lmvar_no_fit'
no_fit = lmvar_no_fit(attenu$accel, X, X_s)
nobs(no_fit)
```

`plot.lmvar` *Plot diagnostics for an ’lmvar’ object*

Description

This function produces 5 plots which should help to judge the goodness of an ’lmvar’ fit.

Usage

```
## S3 method for class 'lmvar'
plot(x, which = c(1:3, 5), id.n = 3, cex.id = 0.75,
  show = TRUE, ...)
```

Arguments

`x`	Object of class ’lmvar’
`which`	Integer vector slecting which of the 5 plots is produced
`id.n`	Integer, the number of ’extreme’ observations that are labelled in the plots
`cex.id`	Numeric, scale-factor for the size of the observation labels in the plots
`show`	Boolean, if TRUE the number of the plot is shown in the plot-title and the name of `x` is shown in the label of the x-axis.
`...`	for compatibility with `plot` generic

Details

The plots are intended to be a quick and easy way to get an impression of the goodness-of-fit. The function is intended for an interactive R-session and users must hit <enter> before each plot is deplayed. The following plots can be produced.

1. A plot of the residuals $y - \mu$ versus the fitted values μ.
2. A QQ-plot, showing the z-score $(y - \mu)/\sigma$ resulting from the fit versus the z-score calculated from the sample quantiles. The sample quantiles are calculated as `ppoints`(n) with n the number of obervations in `x`.
3. A histogram of the distribution of the quantiles of the response values. The quantiles are calculated under the assumption that the response values are normally distributed with expected values μ and standard deviations σ.
4. A plot of the z-scores versus the fitted values.
5. A scale-location plot showing the square root of the absolute z-scores versus the fitted values.

If relevant, plots show the average y-value as a red line. This line is created by the function `panel.smooth`. If relevant, plots show the expected average y-value as a dotted gray line.

To suppress labelling of observations in the plots, set `id.n` to zero or a negative value. If `id.n` is set to a value equal to or larger than the number of observations in `x`, all points in the plots are labelled.

Value

There is no return value. The function only shows plots in the graphics output device.

Examples

```
if (interactive()){

# As example we use the dataset 'cats' from the library 'MASS'.
library(MASS)
```

```
# We regress the cats heart weight 'Hwt' onto its body weight 'Bwt'
X = model.matrix(~ Bwt - 1, cats)
fit = lmvar(cats$Hwt, X_mu = X, X_sigma = X)

# Display all plots
plot(fit)

# Display two plots that focus on the shape of the distribution
plot(fit, which = c(2, 3))

# Suppress plot number and name of the 'lmvar' object being plot in  plot 3
plot(fit, which = 3, show = FALSE)

# Label the 5 observations with the most extreme residuals in plot 1
plot(fit, which = 1, id.n = 5)

}
```

`plot_lm_loglik` *Plot of the log-likelihood surface of a lineair model*

Description

Creates a 3-d plot of the maximum log-likelihood of a lineair model. The maximum log-likelihood is plotted as a function of two elements of the parameter vector β. Optionally, the maximum of a quadratic approximation to the log-likelihood surface is plotted.

This function is intended for development purposes only.

Usage

```
plot_lm_loglik(y, X, beta_or, beta_x, beta_y, add_qa = FALSE,
  plot_width = 3, plot_points = 20)
```

Arguments

`y`	Vector of observations
`X`	Model matrix
`beta_or`	Vector of beta values around which the plot is centered. Also the origin of the quadratic appriximation.
`beta_x`	Component of beta to be plotted at the x-axis. Can be an index of `beta_or` or a name in case `beta_or` is a named vector
`beta_y`	Component of beta to be plotted at the y-axis. Can be an index of `beta_or` or a name in case `beta_or` is a named vector
`add_qa`	Boolean, specifies whether or not a quadratic approximation of the maximum log-likelihood surface is plotted
`plot_width`	Single numeric value, half the range of x- and y-values
`plot_points`	Integer, number of points in the x- and y-dimension at which the maximum-likelihood surface is calculated.

Details

The function plots the maximum log-likelihood of linear model as a function of two components of the vector β. Optionally, it also plots a quadratic approximation to the log-likelihood and plot its maximum.

The quadratic approximation is defined as

$logL_q(\beta) = logL(\beta_o) + g(\beta - \beta_o) + 0.5(\beta - \beta_o)H(\beta - \beta_o)$

where $logL_q$ is the quadratic approximation of the log-likelihood, β_o the value of β at the origin, g the gradient and H the Hessian of the log-likelihood at β_o.

For each point (β_x, β_y), the other components of the vector β are chosen such that the log-likelihood is at its maximum. This maximum is plotted.

The same is true for a quadratic approximation of the log-likelihood, except when it has no maximum (i.e. the Hessian H is not negative-definite at β_o). In that case, a waning is issued and the other components of β are set to the value they have in `beta_or`. The resulting quadratic approximation is plotted. This does not affect the plot of the maximum of the true log-likelihood.

To create the plot, the package `plotly` needs to be installed. A warning is issued if that is not the case.

Examples

```
## Not run:

# Carry out a linear regression with the 'iris' data set
fit = lm( Petal.Length ~ Species, data = iris, x = TRUE, y = TRUE)
X = fit$x
y = fit$y

# We center the plot at the maximum-likelihood
beta_or = coef(fit)

# Plot the maximum log-likelihood
lmvar:::plot_lm_loglik( y, X, beta_or = beta_or, beta_x = "(Intercept)",
                        beta_y = "Speciesversicolor")

# Plot against the two species
lmvar:::plot_lm_loglik( y, X, beta_or = beta_or, beta_x = "Speciesversicolor",
                        beta_y = "Speciesvirginica")

# Increase the resolution
lmvar:::plot_lm_loglik( y, X, beta_or = beta_or, beta_x = "Speciesversicolor",
                        beta_y = "Speciesvirginica", plot_points = 40)

# Remove the intercept term from the model matrix and fit again
XX = X[,-1]
fit = lm( y ~ . - 1, data = as.data.frame(XX))

# Estimate the effect of adding an intercept term in a quadratic approximation and compare
# with exact result
beta_or = c( 0, coef(fit))
lmvar:::plot_lm_loglik( y, X, beta_or = beta_or, beta_x = 1, beta_y = "Speciesversicolor",
```

```
                          add_qa = TRUE, plot_points = 40, plot_width = 5)

# Note that in the last case the quadratic approximation has no maximum. Hence the beta for
# "Speciesvirginica" is kept at beta_or[3] in the calculation of the surface of the
# quadratic approximation.

## End(Not run)
```

`plot_qdis` *Plot of the distribution of quantiles*

Description

Function produces plot of the distribution of quantiles for one or more model fits

Usage

```
plot_qdis(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object which contains model fit
`object_2`	Object which contains model fit
`...`	Other arguments.

Details

If `object_2` is specified, a plot for the distribution of quantiles for `object` and one for `object_2` will be combined in the same plot.

See Also

`plot_qdis.lm` and `plot_qdis.lmvar`

`plot_qdis.lm` *Plot of the distribution of quantiles for an object of class ’lm’*

Description

Function produces plot of the distribution of quantiles for an object of class ’lm’ and, optionally, for another object of class ’lm’ or ’lmvar’.

Usage

```
## S3 method for class 'lm'
plot_qdis(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object of class ’lm’
`object_2`	Object of class ’lm’ or class ’lmvar’
`...`	for compatibility with `plot_qdis` generic.

Details

If `object_2` is specified, a plot for `object_1` and one for `object_2` will be combined in the same plot.

All inputs of class ’lm’ must contain the response vector y. I.e., one must run `lm` with argument `y = TRUE`.

See Also

`plot_qdis`

Examples

```
if(interactive()){

library(lmvar)

# create a linear model using the 'iris' data set
fit_lm = lm(Petal.Length ~ Species, data = iris, y = TRUE)

plot_qdis(fit_lm)

# compare 'lm' with 'lmvar' fit
X = model.matrix(~ Species - 1, data = iris)
fit_lmvar = lmvar(iris$Petal.Length, X, X)

plot_qdis(fit_lm, fit_lmvar)

# check whether inclustion of petal in model improves distribution of quantiles
fit_lm_width = lm(Petal.Length ~ Species + Petal.Width, data = iris, y = TRUE)

plot_qdis(fit_lm, fit_lm_width)

}
```

`plot_qdis.lmvar` *Plot of the distribution of quantiles for an object of class ’lmvar’*

Description

Function produces plot of the distribution of quantiles for an object of class ’lmvar’ and, optionally, for another object of class ’lm’ or ’lmvar’.

Usage

```
## S3 method for class 'lmvar'
plot_qdis(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object of class 'lmvar'
`object_2`	Object of class 'lm' or class 'lmvar'
`...`	for compatibility with `plot_qdis` generic.

Details

If `object_2` is specified, a plot for `object_1` and one for `object_2` will be combined in the same plot.

If `object_2` is of class 'lm', it must contain the response vector y. I.e., one must run `lm` with argument y = TRUE.

See Also

`plot_qdis`

Examples

```
if (interactive()){

library(lmvar)

# create a 'lmvar' model using the 'iris' data set
X = model.matrix(~ Species - 1, data = iris)
fit_lmvar = lmvar(iris$Petal.Length, X, X)

plot_qdis(fit_lmvar)

# compare 'lmvar' model with linear model
fit_lm = lm( Petal.Length ~ Species, data = iris, y = TRUE)

plot_qdis(fit_lmvar, fit_lm)

# check whether inclustion of petal in model improves distribution of quantiles
X = model.matrix(~ Species + Petal.Width - 1, data = iris)
fit_lmvar_width = lmvar(iris$Petal.Length, X, X)

plot_qdis(fit_lmvar, fit_lmvar_width)

}
```

`plot_qdis_lmlike` *Plot of the distribution of quantiles for objects of class 'lm' or 'lmvar'*

Description

Function produces a histogram of quantiles for objects of class 'lm' or 'lmvar'. This function is called by `plot_qdis.lm` and `plot_qdis.lmvar`. It is not intended to be called directly.

Usage

```
plot_qdis_lmlike(object_1, object_2 = NULL, name_1, name_2 = NULL)
```

Arguments

`object_1`	Object of class 'lm' or 'lmvar'
`object_2`	Object of class 'lm' or class 'lmvar'
`name_1`	Character string, the name of `object_1`
`name_2`	Character string, the name of `object_2`

Details

If `object_2` is specified, a plot for `object_1` and one for `object_2` will be combined in the same plot.

The string `name_1` and (optionally) `name_2` are used in the legend of the plot as names for `object_1` and (optionally) `object_2`.

All inputs of class 'lm' must contain the response vector y. I.e., one must run `lm` with argument `y = TRUE`.

`plot_qq` *QQ-plot*

Description

Function produces QQ-plots for one or more model fits

Usage

```
plot_qq(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object which contains model fit
`object_2`	Object which contains model fit
`...`	Other arguments.

Details

If `object_2` is specified, a QQ-plot for `object_1` and one for `object_2` will be combined in the same plot.

See Also

`plot_qq.lm` and `plot_qq.lmvar`

`plot_qq.lm` *QQ-plot for an object of class 'lm'*

Description

Function produces QQ-plots for an object of class 'lm' and, optionally, for another object of class 'lm' or 'lmvar'.

Usage

```
## S3 method for class 'lm'
plot_qq(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object of class 'lm'
`object_2`	Object of class 'lm' or class 'lmvar'
`...`	for compatibility with `plot_qq` generic.

Details

If `object_2` is specified, a QQ-plot for `object_1` and one for `object_2` will be combined in the same plot.

All inputs of class 'lm' must contain the response vector y. I.e., one must run `lm` with argument `y = TRUE`.

See Also

`plot_qq`

Examples

```
if (interactive()){

library(lmvar)

# create a linear model using the 'iris' data set
fit_lm = lm( Petal.Length ~ Species, data = iris, y = TRUE)
```

```
plot_qq(fit_lm)

# compare 'lm' with 'lmvar' fit
X = model.matrix(~ Species - 1, data = iris)
fit_lmvar = lmvar(iris$Petal.Length, X, X)

plot_qq(fit_lm, fit_lmvar)

# check whether inclustion of petal in model improves QQ-plot
fit_lm_width = lm( Petal.Length ~ Species + Petal.Width, data = iris, y = TRUE)

plot_qq(fit_lm, fit_lm_width)

}
```

`plot_qq.lmvar` *QQ-plot for an object of class ’lmvar’*

Description

Function produces QQ-plots for an object of class ’lmvar’ and, optionally, for another object of class ’lm’ or ’lmvar’.

Usage

```
## S3 method for class 'lmvar'
plot_qq(object_1, object_2 = NULL, ...)
```

Arguments

`object_1`	Object of class ’lmvar’
`object_2`	Object of class ’lm’ or class ’lmvar’
`...`	for compatibility with `plot_qq` generic.

Details

If `object_2` is specified, a QQ-plot for `object_1` and one for `object_2` will be combined in the same plot.

If `object_2` id of class ’lm’, it must contain the response vector y. I.e., one must run `lm` with argument y = `TRUE`.

See Also

`plot_qq`

Examples

```
if (interactive()){

library(lmvar)

# create a 'lmvar' model using the 'iris' data set
X = model.matrix(~ Species - 1, data = iris)
fit_lmvar = lmvar(iris$Petal.Length, X, X)

plot_qq(fit_lmvar)

# compare the 'lmvar' model with a lineair model
fit_lm = lm( Petal.Length ~ Species, data = iris, y = TRUE)

plot_qq(fit_lmvar, fit_lm)

# check whether inclustion of petal in model improves QQ-plot
X = model.matrix(~ Species + Petal.Width - 1, data = iris)
fit_lmvar_width = lmvar(iris$Petal.Length, X, X)

plot_qq(fit_lmvar, fit_lmvar_width)

}
```

`plot_qq_lmlike` *QQ-plot for an object of class 'lm' or class 'lmvar'*

Description

Function produces QQ-plots for an object of class 'lm' or 'lmvar'. This function is called by `plot_qq.lm` and `plot_qq.lmvar`. It is not intended to be called directly.

Usage

```
plot_qq_lmlike(object_1, object_2 = NULL, name_1, name_2 = NULL)
```

Arguments

`object_1`	Object of class 'lm' or class 'lmvar'
`object_2`	Object of class 'lm' or class 'lmvar'
`name_1`	Character string, the name of `object_1`
`name_2`	Character string, the name of `object_2`

Details

If `object_2` is specified, a QQ-plot for `object_1` and one for `object_2` will be combined in the same plot.

The string `name_1` and (optionally) `name_2` are used in the legend of the plot as names for `object_1` and (optionally) `object_2`.

All inputs of class 'lm' must contain the response vector y. I.e., one must run `lm` with argument `y = TRUE`.

`predict.lmvar` *Predictions for model matrices*

Description

Estimators and confidence intervals for the expected values and standard deviations of the response vector Y, given model matrices `X_mu` and `X_sigma`. Prediction intervals for Y. Alternatively, estimators and intervals can be for e^Y.

The estimators and intervals are based on the maximum likelihood-estimators for β_μ and β_σ and their covariance matrix present in an 'lmvar' object.

Usage

```
## S3 method for class 'lmvar'
predict(object, X_mu = NULL, X_sigma = NULL,
  mu = TRUE, sigma = TRUE, log = FALSE, interval = c("none",
  "confidence", "prediction"), level = 0.95, ...)
```

Arguments

`object`	Object of class 'lmvar'
`X_mu`	Model matrix for the expected values
`X_sigma`	Model matrix for the logarithm of the standard deviations
`mu`	Boolean, specifies whether or not to include the estimators and intervals for the expected values
`sigma`	Boolean, specifies whether or not to include the estimators and intervals for the standard deviations
`log`	Boolean, specifies whether estimators and intervals should be for Y (`log = FALSE`) or for e^Y (`log = TRUE`).
`interval`	Character string, specifying the type of interval. Possible values are • "none" No interval, this is the default • "confidence" Confidence intervals for the estimators • "prediction" Prediction intervals
`level`	Numeric value between 0 and 1, specifying the confidence level
`...`	For compatibility with `predict` generic

Details

When `X_mu = NULL`, the model matrix X_μ is taken from `object`. Likewise, when `X_sigma = NULL`, X_σ is taken from `object`.

Both `X_mu` and `X_sigma` must have column names. Column names are matched with the names of the elements of β_μ and β_σ in `object`. Columns with non-matching names are ignored. In case not all names in β_μ can be matched with a column in `X_mu`, a warning is given. The same is true for β_σ and `X_sigma`.

`X_mu` can not have a column with the name "(Intercept)". This column is added by `predict.lmvar` in case it is present in `object`. Likewise, `X_sigma` can not have a column with the name "(Intercept_s)". It is added by `predict.lmvar` in case it is present in `object`

Both matrices must be numeric and can not contain special values like `NULL`, `NaN`, etc.

If `log = FALSE`, `predict.lmvar` returns expected values and standard deviations for the observations Y corresponding to the model matrices X_μ and X_σ.

If `log = TRUE`, `predict.lmvar` returns expected values and standard deviations for e^Y.

The fit in `object` can be obtained under the constraint that the standard deviations σ are larger than a minimum value (see the documentation of `lmvar`). However, there is no guarantee that the values of σ given by `predict`, satisfy the same constraint. This depends entirely on `X_sigma`.

Confidence intervals are calculated under the asumption of asymptotic normality. This asumption holds when the number of observations is large. Intervals must be treated cautiously in case of a small number of observations. Intervals can also be unreliable if `object` was created with a constraint on the minimum values of the standard deviations σ.

`predict.lmvar` with `X_mu = NULL` and `X_sigma = NULL` is equivalent to the function `fitted.lmvar`.

Value

In the case `mu = FALSE` and `interval = "none"`: a numeric vector containing the estimators for the standard deviation.

In the case `sigma = FALSE` and `interval = "none"`: a numeric vector containing the estimators for the expected values.

In all other cases: a matrix with one column for each requested feature and one row for each observation. The column names are

- `mu` Estimators for the expected value μ
- `sigma` Estimators for the standard deviation σ
- `mu_lwr` Lower bound of the confidence interval for μ
- `mu_upr` Upper bound of the confidence interval for μ
- `sigma_lwr` Lower bound of the confidence interval for σ
- `sigma_upr` Upper bound of the confidence interval for σ
- `lwr` Lower bound of the prediction interval
- `upr` Upper bound of the prediction interval

See Also

`coef.lmvar` and `confint` for maximum likelihood estimators and confidence intervals for β_μ and β_σ.

`fitted.lmvar` is equivalent to `predict.lmvar` with `X_mu = NULL` and `X_sigma = NULL`.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Create the response vector
y = attenu$accel

# Carry out the fit
fit = lmvar(y, X, X_s)

# Calculate the expected values and standard deviations of 'accel'
# for the current magnitudes and distances
predict(fit)

# Calculate the expected values and standard deviations of 'accel' for earthquakes
# with a 10% larger magnitude, at the current distances
XP = cbind(1.1 * attenu$mag, attenu$dist)
colnames(XP) = c("mag", "dist")

XP_s = cbind(1.1 * attenu$mag, 1 / attenu$dist)
colnames(XP_s) = c("mag", "dist_inv")

predict(fit, XP, XP_s)

# Calculate only the expected values
predict(fit, XP, XP_s, sigma = FALSE)

# Calculate only the standard deviations
predict(fit, XP, XP_s, mu = FALSE)

# Calculate the expected values and their 95% confidence intervals
predict(fit, XP, XP_s, sigma = FALSE, interval = "confidence")

# Calculate the standard deviations and their 90% confidence intervals
predict(fit, XP, XP_s, mu = FALSE, interval = "confidence", level = 0.9)

# Calculate the expected values and the 90% prediction intervals of 'accel'
```

```
predict(fit, XP, XP_s, sigma = FALSE, interval = "prediction", level = 0.9)

# Change the model and fit the log of 'accel'
y = log(attenu$accel)
fit_log = lmvar(y, X, X_s)

# Calculate the expected values and standard deviations of the log of 'accel'
predict(fit_log, XP, XP_s)

# Calculate the expected values and standard deviations of 'accel'
predict(fit_log, XP, XP_s, log = TRUE)

# Calculate the expected values and standard deviations of 'accel',
# as well as their 99% confidence intervals
predict(fit_log, XP, XP_s, log = TRUE, interval = "confidence", level = 0.99)
```

`print.cvlmvar` *Print method for an object of class 'cvlmvar'*

Description

Print method for an object of class 'cvlmvar'. This object is created by the functions `cv.lm` and `cv.lmvar`.

Usage

```
## S3 method for class 'cvlmvar'
print(x, digits = NULL, ...)
```

Arguments

`x`	Object of class 'cvlmvar'
`digits`	Integer, number of significant digits of standard deviations in printed output. Must be larger than zero, or `NULL`.
`...`	For compatibility with `print` generic.

Details

If `digits` = `NULL`, printed values are not rounded. Otherwise, all standard deviations are rounded to `digits` significant digits. All corresponding mean values are rounded to a precision equal to the maximum precision of the rounded value of the standard deviation.

Examples of `print.cvlmvar` are provided in the examples of the functions `cv.lm` and `cv.lmvar`.

`print.summary_lmvar` *Print method for the summary of an 'lmvar' object.*

Description

Print method for an object of the class 'summary_lmvar'. This object is created by `summary.lmvar`.

Usage

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

Arguments

`x`	Object of class 'summary_lmvar'
`...`	For compatibility with `print` generic.

See Also

`summary.lmvar` for a summary of the fit present in an object of class 'lmvar'.

`residuals.lmvar` *Residuals from an 'lmvar' object*

Description

Calculates residuals from an 'lmvar' object. This object can be a fit to either a response vector or the logarithm of the response vector.

Usage

```
## S3 method for class 'lmvar'
residuals(object, log = FALSE, ...)
```

Arguments

`object`	Object of class 'lmvar'
`log`	Boolean, specifies whether `object` is a fit to a response-variable Y or to its logarithm $\log Y$ In both cases, `residuals.lmvar` returns residuals for Y itself.
`...`	For compatibility with `residuals` generic

Details

In case `log = FALSE`, the residual of an observation is defined as $y - \mu$, where y is the value of the observation and μ its expected value.

In case `log = TRUE`, the residual of an observation is defined as $e^y - \mu$, where μ is the expected value of e^y.

Value

A numeric vector with the residual for each observation in `object`.

See Also

`fitted.lmvar` for the expected values in an object of class ’lmvar’.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Calculate the residuals
residuals(fit)
```

`summary.lmvar` *Summary overview for an object of class ’lmvar’*

Description

Summary overview for an object of class ’lmvar’.

Usage

```
## S3 method for class 'lmvar'
summary(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class 'lmvar'
`mu`	Boolean, specifies whether or not to include the coefficients β_μ in the table of coefficients
`sigma`	Boolean, specifies whether or not to include the coefficients β_σ in the table of coefficients
`...`	For compatibility with `summary` generic

Details

Standard errors and z-statistics are calculated under the assumption of asymptotic normality for maximum likelihood estimators. They may not be reliable when the number of observations in `object` is small.

Value

An object of class 'summary_lmvar'. This is a list with the following members:

- `call` Call that created `object`
- `coefficients` Data frame with one row for each element of β_μ and β_σ and the following variables.
  - `Estimate` maximum-likelihood estimate
  - `Std. Error` standard error, defined as $\sqrt(var(\beta))$ with $var(\beta)$ the estimated variance of β.
  - `z value` z-statistic, defined as $\beta/\sqrt(var(\beta))$
  - `Pr(>|z|)` p-value of the z-statistic, calculated from the standard normal distribution.
- `residuals` A numeric vector with the minimum, the 25% quartile, the median, the 75% quartile and the maximum standardized residual. The standardized residual of an observation is defined as $(y - \mu)/\sigma$ where y is the value of the observation, μ the expectation value and σ the standard deviation of the observation.
- `sigma` A numeric vector with the minimum, the 25% quartile, the median, the 75% quartile and the maximum standard deviation σ of all observations.
- `aliased_mu` A named logical vector. The names are the column names of the user-supplied model matrix X_μ. The values (`TRUE` or `FALSE`) tell whether or not the column has been removed by `lmvar` to make the matrix full-rank.
- `aliased_sigma` As `aliased_mu` but for the user-supplied model matrix X_σ.
- `logLik_ratio` The difference in log-likelihood between the model in `object` and a classical linear model with model matrix X_μ and a constant variance for all observations.
- `df_additional` The difference in degrees in freedom between the model in `object` and a classical linear model with model matrix X_μ and a constant variance for all observations. Is equal to `NULL` if X_σ does not contain an intercept term.
- `p_value` The p-value of `2 loglik_ratio`, calculated from a chi-squared distribution with `df` degrees of freedom. Is equal to `NULL` if there are no additional degrees of freedom.
- `nobs` The number of observations in `object`.
- `df` The degrees of freedom of the fit in `object`.
- `options` A list of argument-values of the function call.

See Also

`coef` to extract the matrix with estimates, standard-errors, t-statistics and p-values for β_μ and β_σ from a 'summary_lmvar' object.

`vcov.lmvar` for the covariance matrix of the β_μ and β_σ in an object of class 'lmvar'.

`print.summary_lmvar` for a print method for a 'summary_lmvar' object.

`fitted.lmvar` for the expected values and standard deviations of the observations in an object of class 'lmvar'.

`logLik.lmvar` for the log-likelihood of a fit in an object of class 'lmvar'.

`alias.lmvar_no_fit` to obtain the aliased columns of the user-supplied model matrices in the call of `lmvar`.

Examples

```
# As example we use the dataset 'attenu' from the library 'datasets'. The dataset contains
# the response variable 'accel' and two explanatory variables 'mag'  and 'dist'.
library(datasets)

# Create the model matrix for the expected values
X = cbind(attenu$mag, attenu$dist)
colnames(X) = c("mag", "dist")

# Create the model matrix for the standard deviations.
X_s = cbind(attenu$mag, 1 / attenu$dist)
colnames(X_s) = c("mag", "dist_inv")

# Carry out the fit
fit = lmvar(attenu$accel, X, X_s)

# Print a summary of the fit
summary(fit)

# Include only the coefficients beta for the expected values
summary(fit, sigma = FALSE)

# Include only the coefficients beta for the standard deviations
summary(fit, mu = FALSE)

# Extract the matrix of coefficients from the summary
coef(summary(fit))
```

`vcov.lmvar` *Variance-covarience matrix of the coefficients beta for an object of class 'lmvar'*

Description

Variance-covarience matrix (also simply called the 'covariance matrix') for the maximum-likelihood estimators of β_μ and β_σ. The matrix is calculated with the assumption of asymptotic normality of maximum likelihood estimators. This assumption is only valid in the limit of a large number of observations.

Usage

```
## S3 method for class 'lmvar'
vcov(object, mu = TRUE, sigma = TRUE, ...)
```

Arguments

`object`	Object of class 'lmvar'
`mu`	Specifies whether or not the covariance matrix for β_μ is included in the returned matrix
`sigma`	Specifies whether or not the covariance matrix for β_σ is included in the returned matrix
`...`	For compatibility with `vcov` generic

Details

The variance-covariance matrix is calculated as I^{-1}/n where I is the Fisher information matrix and n the number of observations.

When `mu = TRUE` and `sigma = TRUE`, the full covariance matrix for the combined vector $(\beta_\mu, \beta_\sigma)$ is returned.

When `mu = TRUE` and `sigma = FALSE`, only the covariance matrix for β_μ is returned.

When `mu = FALSE` and `sigma = TRUE`, only the covariance matrix for β_σ is returned.

Value

A 'matrix' object containing the (approximate) variance-covariance matrix of the maximum-likelihood estimators of β_μ and β_σ in `object`.

See Also

`summary.lmvar` for standard errors for β_μ and β_μ.

`nobs.lmvar_no_fit` for the number of observations in an object of class 'lmvar'.

`fisher` for the Fisher information matrix of an object of class 'lmvar'.

See the vignette "Math" (to be viewed with `vignette("Math", "lmvar")`) for details.

Index