Package 'greybox'

February 22, 2026

Type Package

Title Toolbox for Model Building and Forecasting

Version 2.0.8

Date 2026-02-20

URL https://github.com/config-i1/greybox

BugReports https://github.com/config-i1/greybox/issues

Language en-GB

Description Implements functions and instruments for regression model building and its application to forecasting. The main scope of the package is in variables selection and models specification for cases of time series data. This includes promotional modelling, selection between different dynamic regressions with non-standard distributions of errors, selection based on cross validation, solutions to the fat regression model problem and more. Models developed in the package are tailored specifically for forecasting purposes. So as a results there are several methods that allow producing forecasts from these models and visualising them.

License LGPL-2.1

Depends R (>= 3.5.0)

Imports stats, generics (>= 0.1.2), graphics, utils, pracma, nloptr, statmod, zoo, texreg, xtable, methods

LinkingTo Rcpp

Suggests smooth (>= 3.1.0), doMC, doParallel, foreach, testthat, rmarkdown, knitr

Enhances vars, forecast

RoxygenNote 7.3.3

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation yes

Author Ivan Svetunkov [aut, cre] (Senior Lecturer at Centre for Marketing Analytics and Forecasting, Lancaster University, UK), Yves R. Sagaert [ctb] (Visiting Research at Centre for Marketing Analytics and Forecasting, Lancaster University, UK)

Maintainer Ivan Svetunkov <ivan@svetunkov.com>

Repository CRAN

Date/Publication 2026-02-22 15:40:09 UTC

Contents

`accuracy.greybox` *Error measures for an estimated model*

Description

Function produces error measures for the provided object and the holdout values of the response variable. Note that instead of parameters `x`, `test`, the function accepts the vector of values in `holdout`. Also, the parameters `d` and `D` are not supported - MASE is always calculated via division by first differences.

Usage

```
## S3 method for class 'greybox'
accuracy(object, holdout = NULL, ...)

## S3 method for class 'predict.greybox'
accuracy(object, holdout = NULL, ...)
```

Arguments

`object`	The estimated model or a forecast from the estimated model generated via either `predict()` or `forecast()` functions.
`holdout`	The vector of values of the response variable in the holdout (test) set. If not provided, then the function will return the in-sample error measures.
`...`	Other variables passed to the `forecast()` function (e.g. `newdata`).

Details

The function is a wrapper for the measures function and is implemented for convenience.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
xreg <- cbind(rlaplace(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rlaplace(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

ourModel <- alm(y~x1+x2+trend, xreg, subset=c(1:80), distribution="dlaplace")
 accuracy(predict(ourModel,xreg[-c(1:80),]), xreg[-c(1:80),"y"])
```

---

actuals *Function extracts the actual values from the function*

Description

This is a simple method that returns the values of the response variable of the model

Usage

```
actuals(object, all = TRUE, ...)

## Default S3 method:
actuals(object, all = TRUE, ...)

## S3 method for class 'lm'
actuals(object, all = TRUE, ...)

## S3 method for class 'alm'
actuals(object, all = TRUE, ...)

## S3 method for class 'predict.greybox'
actuals(object, all = TRUE, ...)
```

Arguments

object	Model estimated using one of the functions of smooth package.
all	If FALSE, then in the case of the occurrence model, only demand sizes will be returned.
...	Other parameters to pass to the method. Currently nothing is supported here.

Value

The vector of the response variable.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

ourModel <- stepwise(xreg)

actuals(ourModel)
```

AICc *Corrected Akaike's Information Criterion and Bayesian Information Criterion*

Description

This function extracts AICc / BICc from models. It can be applied to wide variety of models that use logLik() and nobs() methods (including the popular lm, forecast, smooth classes).

Usage

```
AICc(object, ...)

BICc(object, ...)
```

Arguments

object	Time series model.
...	Some stuff.

Details

AICc was proposed by Nariaki Sugiura in 1978 and is used on small samples for the models with normally distributed residuals. BICc was derived in McQuarrie (1999) and is used in similar circumstances.

IMPORTANT NOTE: both of the criteria can only be used for univariate models (regression models, ARIMA, ETS etc) with normally distributed residuals! In case of multivariate models, both criteria need to be modified. See Bedrick & Tsai (1994) for details.

Value

This function returns numeric value.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Burnham Kenneth P. and Anderson David R. (2002). Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach. Springer-Verlag New York. DOI: [10.1007/b97636](http://dx.doi.org/10.1007/b97636).
- McQuarrie, A. D. (1999). A small-sample correction for the Schwarz SIC model selection criterion. Statistics & Probability Letters, 44(1), 79–86. [10.1016/S0167-7152(98)00294-6](https://doi.org/10.1016/S0167-7152(98)00294-6).
- McQuarrie A.D., A small-sample correction for the Schwarz SIC model selection criterion, Statistics & Probability Letters 44 (1999) pp.79-86. doi:10.1016/S01677152(98)002946
- Sugiura Nariaki (1978) Further analysts of the data by Akaike's information criterion and the finite corrections, Communications in Statistics - Theory and Methods, 7:1, 13-26, doi:10.1080/03610927808827599
- Bedrick, E. J., & Tsai, C.-L. (1994). Model Selection for Multivariate Regression in Small Samples. Biometrics, 50(1), 226. doi:10.2307/2533213

See Also

AIC, BIC

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

ourModel <- stepwise(xreg)

AICc(ourModel)
BICc(ourModel)
```

---

aid *Automatic Identification of Demand*

Description

The function applies several models on the provided time series and identifies what type of demand it is based on an information criterion.

Usage

```
aid(y, ic = c("AICc", "AIC", "BICc", "BIC"), level = 0.99,
  loss = "likelihood", ...)

aidCat(data, ...)
```

Arguments

y	The vector of the data.
ic	Information criterion to use.
level	The confidence level used in stockouts identification.
loss	The type of loss function to use in model estimation. See alm for possible options.
...	Other parameters passed to the alm() function. In case of the aidCat() function, the aid() parameters can be passed here.
data	The dataframe or a matrix to which the aid function should be applied

Details

In the first step, function creates inter-demand intervals and fits a model with LOWESS of it assuming Geometric distribution. The outliers from this model are treated as potential stock outs.

In the second step, the function creates explanatory variables based on LOWESS of the original data, then applies Normal, Normal + Bernoulli models and selects the one that has the lowest IC. Based on that, it decides what type of demand the data corresponds to: regular or intermittent. Finally, if the data is count, the function will identify that.

Value

aid() returns an object of class "aid", which contains:

- y - The original data;
- models - All fitted models;
- ICs - Values of information criteria;
- type - The list with the specific type categories (count/fractional, intermittent/regular, smooth/lumpy);
- name - The name of the identified demand;
- stockouts - List with start and end ids of potential stockouts;
- new - Binary showing whether the data start with the abnormal number of zeroes. Must be a new product then;
- obsolete - Binary showing whether the data ends with the abnormal number of zeroes. Must be product that was discontinued (obsolete).

aidCat() returns an object of class "aidCat", which contains:

- categories - the vector with the names of categories for each time series;
- types - the vector with the count of series in each category;
- anomalies - the vector that contains counts of cases where product was flagged as new, obsolete and having stockouts.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
# Data from Poisson distribution
y <- rpois(120, 0.7)
aid(y)

xreg <- cbind(x1=rpois(100,1), x2=rpois(100,2),
              x3=rpois(100,5), x4=rnorm(100,10,2))
plot(aidCat(xreg))
```

`alm` *Augmented Linear Model*

Description

Function estimates model based on the selected distribution

Usage

```
alm(formula, data, subset, na.action, distribution = c("dnorm", "dlaplace",
  "ds", "dgnorm", "dlogis", "dt", "dalaplace", "dlnorm", "dllaplace", "dls",
  "dlgnorm", "dbcnorm", "dinvgauss", "dgamma", "dexp", "dfnorm", "drectnorm",
  "dpois", "dnbinom", "dbinom", "dgeom", "dbeta", "dlogitnorm", "plogis",
  "pnorm"), loss = c("likelihood", "MSE", "MAE", "HAM", "LASSO", "RIDGE",
  "ROLE", "QUALE"), occurrence = c("none", "plogis", "pnorm"),
  scale = NULL, orders = c(0, 0, 0), parameters = NULL, fast = FALSE,
  ...)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. Can also include `trend`, which would add the global trend. Also accepts backshift operator to introduce lags/leads of variables, e.g. 'y~x+B(x,1)'.
`data`	a data frame or a matrix, containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The factory-fresh default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.

`distribution` what density function to use in the process. The full name of the distribution should be provided here. Values with "d" in the beginning of the name refer to the density function, while "p" stands for "probability" (cumulative distribution function). The names align with the names of distribution functions in R. For example, see dnorm.

`loss` The type of Loss Function used in optimization. `loss` can be:

- `"likelihood"` - the model is estimated via the maximisation of the likelihood of the function specified in `distribution`;
- `"MSE"` (Mean Squared Error),
- `"MAE"` (Mean Absolute Error),
- `"HAM"` (Half Absolute Moment),
- `"LASSO"` - use LASSO to shrink the parameters of the model;
- `"RIDGE"` - use RIDGE to shrink the parameters of the model;
- `"ROLE"` - "RObust Likelihood Estimator", which uses trimmed mean in the calculation of scale and likelihood. A separate parameter `trim` can be provided in ellipsis. If not provided, the default value is 0.05;
- `"QUALE"` - QUAntile Likelihood Estimator. Minimises a specified quantile of the likelihood (instead of taking the sum). Requires parameter `lambda` to be specified.

In case of LASSO / RIDGE, the variables are not normalised prior to the estimation, but the parameters are divided by the standard deviations of explanatory variables inside the optimisation. As the result the parameters of the final model have the same interpretation as in the case of classical linear regression. Note that the user is expected to provide the parameter `lambda`.

A user can also provide their own function here as well, making sure that it accepts parameters `actual`, `fitted` and `B`. Here is an example:

```
lossFunction <- function(actual, fitted, B, xreg) return(mean(abs(actual-fitted)))
loss=lossFunction
```

See `vignette("alm","greybox")` for some details on losses and distributions.

`occurrence` what distribution to use for occurrence variable. Can be `"none"`, then nothing happens; `"plogis"` - then the logistic regression using `alm()` is estimated for the occurrence part; `"pnorm"` - then probit is constructed via `alm()` for the occurrence part. In both of the latter cases, the formula used is the same as the formula for the sizes. Alternatively, you can provide the formula here, and `alm` will estimate logistic occurrence model with that formula. Finally, an "alm" model can be provided and its estimates will be used in the model construction.

If this is not `"none"`, then the model is estimated in two steps: 1. Occurrence part of the model; 2. Sizes part of the model (excluding zeroes from the data).

`scale` formula for scale parameter of the model. If `NULL`, then it is assumed that the scale is constant. This might be useful if you need a model with changing variance (i.e. in case of heteroscedasticity). The log-link is used for the scale (i.e. take exponent of obtained fitted value for the scale, so that it is always positive).

`orders` the orders of ARIMA to include in the model. Only non-seasonal orders are accepted.

`parameters` vector of parameters of the linear model. When `NULL`, it is estimated.

`fast` if `TRUE`, then the function won't check whether the data has variability and whether the regressors are correlated. Might cause trouble, especially in cases of multicollinearity.

`...` additional parameters to pass to distribution functions. This includes:

- `alpha` - value for Asymmetric Laplace distribution;
- `size` - the size for the Negative Binomial distribution;
- `nu` - the number of degrees of freedom for Chi-Squared and Student's t;
- `shape` - the shape parameter for Generalised Normal distribution;
- `lambda` - the meta parameter for LASSO / RIDGE. Should be between 0 and 1, regulating the strength of shrinkage, where 0 means don't shrink parameters (use MSE) and 1 means shrink everything (ignore MSE);
- `lambdaBC` - lambda for Box-Cox transform parameter in case of Box-Cox Normal Distribution.
- `FI=TRUE` will make the function also produce Fisher Information matrix, which then can be used to calculated variances of smoothing parameters and initial states of the model. This is used in the vcov method;

You can also pass parameters to the optimiser:

1. `B` - the vector of starting values of parameters for the optimiser, should correspond to the explanatory variables. If formula for scale was provided, the parameters for that part should follow the parameters for location;
2. `algorithm` - the algorithm to use in optimisation (`"NLOPT_LN_NELDERMEAD"` by default);
3. `maxeval` - maximum number of evaluations to carry out. Default is 40 per estimated parameter. In case of LASSO / RIDGE the default is 80 per estimated parameter;
4. `maxtime` - stop, when the optimisation time (in seconds) exceeds this;
5. `xtol_rel` - the precision of the optimiser (the default is 1E-6);
6. `xtol_abs` - the absolute precision of the optimiser (the default is 1E-8);
7. `ftol_rel` - the stopping criterion in case of the relative change in the loss function (the default is 1E-4);
8. `ftol_abs` - the stopping criterion in case of the absolute change in the loss function (the default is 0 - not used);
9. `print_level` - the level of output for the optimiser (0 by default). If equal to 41, then the detailed results of the optimisation are returned.

You can read more about these parameters by running the function nloptr.print.options.

Details

This is a function, similar to lm, but using likelihood for the cases of several non-normal distributions. These include:

1. dnorm - Normal distribution,
2. dlaplace - Laplace distribution,
3. ds - S-distribution,
4. dgnorm - Generalised Normal distribution,

5. dlogis - Logistic Distribution,
6. dt - T-distribution,
7. dalaplace - Asymmetric Laplace distribution,
8. dlnorm - Log-Normal distribution,
9. dllaplace - Log-Laplace distribution,
10. dls - Log-S distribution,
11. dlgnorm - Log-Generalised Normal distribution,
12. dfnorm - Folded normal distribution,
13. drectnorm - Rectified normal distribution,
14. dbcnorm - Box-Cox normal distribution,
15. dinvgauss - Inverse Gaussian distribution,
16. dgamma - Gamma distribution,
17. dexp - Exponential distribution,
18. dlogitnorm - Logit-normal distribution,
19. dbeta - Beta distribution,
20. dpois - Poisson Distribution,
21. dnbinom - Negative Binomial Distribution,
22. dbinom - Binomial Distribution,
23. dgeom - Geometric Distribution,
24. plogis - Cumulative Logistic Distribution,
25. pnorm - Cumulative Normal distribution.

This function can be considered as an analogue of glm, but with the focus on time series. This is why, for example, the function has `orders` parameter for ARIMA and produces time series analysis plots with `plot(alm(...))`.

This function is slower than `lm`, because it relies on likelihood estimation of parameters, hessian calculation and matrix multiplication. So think twice when using `distribution="dnorm"` here.

The estimation is done via the maximisation of likelihood of a selected distribution, so the number of estimated parameters always includes the scale. Thus the number of degrees of freedom of the model in case of `alm` will typically be lower than in the case of `lm`.

See more details and examples in the vignette for "ALM": `vignette("alm","greybox")`

Value

Function returns `model` - the final model of the class "alm", which contains:

- coefficients - estimated parameters of the model,
- FI - Fisher Information of parameters of the model. Returned only when `FI=TRUE`,
- fitted - fitted values,
- residuals - residuals of the model,
- mu - the estimated location parameter of the distribution,

- scale - the estimated scale parameter of the distribution. If a formula was provided for scale, then an object of class "scale" will be returned.
- distribution - distribution used in the estimation,
- logLik - log-likelihood of the model. Only returned, when `loss="likelihood"` or `loss="ROLE"` and in several special cases of distribution and loss combinations (e.g. `loss="MSE"`, distribution="dnorm"),
- loss - the type of the loss function used in the estimation,
- lossFunction - the loss function, if the custom is provided by the user,
- lossValue - the value of the loss function,
- res - the output of the optimisation (nloptr function),
- df.residual - number of degrees of freedom of the residuals of the model,
- df - number of degrees of freedom of the model,
- call - how the model was called,
- rank - rank of the model,
- data - data used for the model construction,
- terms - terms of the data. Needed for some additional methods to work,
- occurrence - the occurrence model used in the estimation,
- B - the value of the optimised parameters. Typically, this is a duplicate of coefficients,
- other - the list of all the other parameters either passed to the function or estimated in the process, but not included in the standard output (e.g. `alpha` for Asymmetric Laplace),
- timeElapsed - the time elapsed for the estimation of the model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

`stepwise`, `lmCombine`, `xregTransformer`

Examples

```
### An example with mtcars data and factors
mtcars2 <- within(mtcars, {
   vs <- factor(vs, labels = c("V", "S"))
   am <- factor(am, labels = c("automatic", "manual"))
   cyl  <- factor(cyl)
   gear <- factor(gear)
   carb <- factor(carb)
})
# The standard model with Log-Normal distribution
ourModel <- alm(mpg~., mtcars2[1:30,], distribution="dlnorm")
summary(ourModel)
plot(ourModel)
# Produce table based on the output for LaTeX
```

```
xtable(summary(ourModel))

# Produce predictions with the one sided interval (upper bound)
predict(ourModel, mtcars2[-c(1:30),], interval="p", side="u")

# Model with heteroscedasticity (scale changes with the change of qsec)
ourModel <- alm(mpg~., mtcars2[1:30,], scale=~qsec)

### Artificial data for the other examples
xreg <- cbind(rlaplace(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rlaplace(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

# An example with Laplace distribution
ourModel <- alm(y~x1+x2+trend, xreg, subset=c(1:80), distribution="dlaplace")
summary(ourModel)
plot(predict(ourModel,xreg[-c(1:80),]))

# And another one with Asymmetric Laplace distribution (quantile regression)
# with optimised alpha
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), distribution="dalaplace")

# An example with AR(1) order
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), orders=c(1,0,0))
summary(ourModel)
plot(predict(ourModel,xreg[-c(1:80),]))

# AR(1) with distributed lags for x1 (ARDL)
ourModel <- alm(y~x1+x2+B(x1,1), xreg, subset=c(1:80), orders=c(1,0,0))
summary(ourModel)
plot(predict(ourModel,xreg[-c(1:80),]))
# Get dynamic multipliers
multipliers(ourModel, "x1", h=5)

### Examples with the count data
xreg[,1] <- round(exp(xreg[,1]-70),0)

# Negative Binomial distribution
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), distribution="dnbinom")
summary(ourModel)
predict(ourModel,xreg[-c(1:80),],interval="p",side="u")

# Poisson distribution
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), distribution="dpois")
summary(ourModel)
predict(ourModel,xreg[-c(1:80),],interval="p",side="u")


### Examples with binary response variable
xreg[,1] <- round(xreg[,1] / (1 + xreg[,1]),0)

# Logistic distribution (logit regression)
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), distribution="plogis")
```

```
summary(ourModel)
plot(predict(ourModel,xreg[-c(1:80),],interval="c"))

# Normal distribution (probit regression)
ourModel <- alm(y~x1+x2, xreg, subset=c(1:80), distribution="pnorm")
summary(ourModel)
plot(predict(ourModel,xreg[-c(1:80),],interval="p"))
```

`association` *Measures of association*

Description

Function returns the matrix of measures of association for different types of variables.

Usage

```
association(x, y = NULL, use = c("na.or.complete", "complete.obs",
  "everything", "all.obs"), method = c("auto", "pearson", "spearman",
  "kendall", "cramer"))

assoc(x, y = NULL, use = c("na.or.complete", "complete.obs", "everything",
  "all.obs"), method = c("auto", "pearson", "spearman", "kendall", "cramer"))
```

Arguments

`x`	Either data.frame or a matrix
`y`	The numerical variable.
`use`	What observations to use. See cor function for details. The only option that is not available here is `"pairwise.complete.obs"`.
`method`	Which method to use for the calculation of measures of association. By default this is `"auto"`, which means that the function will use: cor, mcor or cramer - depending on the scales of variables. The other options force the function to use one and the same method for all the variables: • `"pearson"` - Pearson's correlation coefficient using cor; • `"spearman"` - Spearman's correlation coefficient based on cor; • `"kendall"` - Kendall's correlation coefficient via cor; • `"cramer"` - Cramer's V using cramer; Be aware that the wrong usage of measures of association might give misleading results.

Details

The function looks at the types of the variables and calculates different measures depending on the result:

- If both variables are numeric, then Pearson's correlation is calculated;
- If both variables are categorical, then Cramer's V is calculated;
- Finally, if one of the variables is categorical, and the other is numeric, then multiple correlation is returned.

After that the measures are wrapped up in a matrix.

Function also calculates the p-values associated with the respective measures (see the return).

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

`assoc()` is just a short name for the `association{}`.

Value

The following list of values is returned:

- value - Matrix of the coefficients of association;
- p.value - The p-values for the parameters;
- type - The matrix of the types of measures of association.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

`table`, `tableplot`, `spread`,`cramer`, `mcor`

Examples

```
association(mtcars)
```

B *Create Lagged or Lead Variables for Time Series Regression*

Description

'B()' acts as a backshift operator and creates lagged (past values) or lead (future values) versions of a variable for use in regression formulas. This function is designed to work within R formula syntax, similar to how 'I()' or 'log()' work.

Usage

```
B(x, k, ...)
```

Arguments

x	A numeric vector or time series variable to be lagged or lead
k	An integer specifying the lag order: • Positive values (e.g., ‘k = 1‘) create lags (past values) • Negative values (e.g., ‘k = -1‘) create leads (future values) • Zero (‘k = 0‘) returns the original variable unchanged
...	Parameters passed to ‘xregExpander()‘.

Details

The function calls for the ‘xregExpander()‘ to create lags/leads. So, you can pass additional parameters to it via ellipsis.

When ‘k > 0‘ (lag), the function shifts values forward in time, so ‘B(x, 1)‘ at time ‘t‘ contains the value of ‘x‘ at time ‘t-1‘.

When ‘k < 0‘ (lead), the function shifts values backward in time, so ‘B(x, -1)‘ at time ‘t‘ contains the value of ‘x‘ at time ‘t+1‘.

Value

A numeric vector of the same length as ‘x‘. The missing values are treated by the ‘xregExpander()‘. By default they are extrapolated (`gaps="auto"`).

See Also

, `xregExpander` for data frame lag operations `lag` for time series lag (different behavior)

Examples

```
# Create sample time series data
y = rnorm(10)

# Create lags
B(y, 1)

# Create leads
B(y, -1)
```

calm *Combine regressions based on information criteria*

Description

Function combines parameters of linear regressions of the first variable on all the other provided data. This is done based on the alm, so the function Combines ALM.

Usage

```
calm(data, ic = c("AICc", "AIC", "BIC", "BICc"), bruteforce = FALSE,
  silent = TRUE, formula = NULL, subset = NULL,
  distribution = c("dnorm", "dlaplace", "ds", "dgnorm", "dlogis", "dt",
  "dalaplace", "dlnorm", "dllaplace", "dls", "dlgnorm", "dbcnorm", "dinvgauss",
  "dgamma", "dexp", "dfnorm", "drectnorm", "dpois", "dnbinom", "dbeta",
  "dlogitnorm", "plogis", "pnorm"), parallel = FALSE, ...)

lmCombine(...)
```

Arguments

data	Data frame containing dependent variable in the first column and the others in the rest.
ic	Information criterion to use.
bruteforce	If TRUE, then all the possible models are generated and combined. Otherwise the best model is found and then models around that one are produced and then combined.
silent	If FALSE, then nothing is silent, everything is printed out. TRUE means that nothing is produced.
formula	If provided, then the selection will be done from the listed variables in the formula after all the necessary transformations.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
distribution	Distribution to pass to alm(). See alm for details.
parallel	If TRUE, then the model fitting is done in parallel. WARNING! Packages foreach and either doMC (Linux and Mac only) or doParallel are needed in order to run the function in parallel.
...	Other parameters passed to alm().

Details

The algorithm uses alm() to fit different models and then combines the models based on the selected IC. The parameters are combined so that if they are not present in some of models, it is assumed that they are equal to zero. Thus, there is a shrinkage effect in the combination.

Some details and examples of application are also given in the vignette "Greybox": vignette("greybox","greybox")

Value

Function returns `model` - the final model of the class "greyboxC". The list of variables:

- coefficients - combined parameters of the model,
- vcov - combined covariance matrix of the model,
- fitted - the fitted values,
- residuals - residual of the model,
- distribution - distribution used in the estimation,
- logLik - combined log-likelihood of the model,
- IC - the values of the combined information criterion,
- ICType - the type of information criterion used,
- df.residual - number of degrees of freedom of the residuals of the combined model,
- df - number of degrees of freedom of the combined model,
- importance - importance of the parameters,
- combination - the table, indicating which variables were used in every model construction and what were the weights for each model,
- timeElapsed - the time elapsed for the estimation of the model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Burnham Kenneth P. and Anderson David R. (2002). Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach. Springer-Verlag New York. DOI: [10.1007/b97636](http://dx.doi.org/10.1007/b97636).
- McQuarrie, A. D. (1999). A small-sample correction for the Schwarz SIC model selection criterion. Statistics & Probability Letters, 44(1), 79–86. [10.1016/S0167-7152(98)00294-6](https://doi.org/10.1016/S0167-7152(98)00294-6).

See Also

`step`, `xregExpander`, `stepwise`

Examples

```
### Simple example
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
inSample <- xreg[1:80,]
outSample <- xreg[-c(1:80),]
# Combine all the possible models
ourModel <- calm(inSample,bruteforce=TRUE)
predict(ourModel,outSample)
```

```
plot(predict(ourModel,outSample))

### Fat regression example
xreg <- matrix(rnorm(5000,10,3),50,100)
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(50,0,3),xreg,rnorm(50,300,10))
colnames(xreg) <- c("y",paste0("x",c(1:100)),"Noise")
inSample <- xreg[1:40,]
outSample <- xreg[-c(1:40),]
# Combine only the models close to the optimal
ourModel <- calm(inSample, ic="BICc",bruteforce=FALSE)
summary(ourModel)
plot(predict(ourModel, outSample))

# Combine in parallel - should increase speed in case of big data
## Not run: ourModel <- calm(inSample, ic="BICc", bruteforce=TRUE, parallel=TRUE)
summary(ourModel)
plot(predict(ourModel, outSample))
## End(Not run)
```

`coef.greybox` *Coefficients of the model and their statistics*

Description

These are the basic methods for the alm and greybox models that extract coefficients, their covariance matrix, confidence intervals or generating the summary of the model. If the non-likelihood related loss was used in the process, then it is recommended to use bootstrap (which is slow, but more reliable).

Usage

```
## S3 method for class 'greybox'
coef(object, bootstrap = FALSE, ...)

## S3 method for class 'alm'
confint(object, parm, level = 0.95, bootstrap = FALSE, ...)

## S3 method for class 'scale'
confint(object, parm, level = 0.95, bootstrap = FALSE, ...)

## S3 method for class 'alm'
vcov(object, bootstrap = FALSE, ...)

## S3 method for class 'scale'
vcov(object, bootstrap = FALSE, ...)

## S3 method for class 'alm'
summary(object, level = 0.95, bootstrap = FALSE, ...)
```

Arguments

object	The model estimated using alm or other greybox function.
bootstrap	The logical, which determines, whether to use bootstrap in the process or not.
...	Parameters passed to coefbootstrap function.
parm	The parameters that need to be extracted.
level	The confidence level for the construction of the interval.

Details

The coef() method returns the vector of parameters of the model. If bootstrap=TRUE, then the coefficients are calculated as the mean values of the bootstrapped ones.

The vcov() method returns the covariance matrix of parameters. If bootstrap=TRUE, then the bootstrap is done using coefbootstrap function

The confint() constructs the confidence intervals for parameters. Once again, this can be done using bootstrap=TRUE.

Finally, the summary() returns the table with parameters, their standard errors, confidence intervals and general information about the model.

Value

Depending on the used method, different values are returned.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

alm, coefbootstrap

Examples

```
# An example with ALM
ourModel <- alm(mpg~., mtcars, distribution="dlnorm")
coef(ourModel)
vcov(ourModel)
confint(ourModel)
summary(ourModel)
```

coefbootstrap *Bootstrap for parameters of models*

Description

The function does the bootstrap for parameters of models and returns covariance matrix together with the original bootstrapped data.

Usage

```
coefbootstrap(object, nsim = 1000, size = floor(0.75 * nobs(object)),
  replace = FALSE, prob = NULL, parallel = FALSE, method = c("cr",
  "dsr"), ...)

## S3 method for class 'lm'
coefbootstrap(object, nsim = 1000, size = floor(0.75 *
  nobs(object)), replace = FALSE, prob = NULL, parallel = FALSE,
  method = c("cr", "dsr"), ...)

## S3 method for class 'alm'
coefbootstrap(object, nsim = 1000, size = floor(0.75 *
  nobs(object)), replace = FALSE, prob = NULL, parallel = FALSE,
  method = c("cr", "dsr"), ...)
```

Arguments

object	The model estimated using either lm, or alm, or glm.
nsim	Number of iterations (simulations) to run.
size	A non-negative integer giving the number of items to choose (the sample size), passed to sample function in R. If not provided and model contains ARIMA components, this value will be selected at random on each iteration. This is only used for method="cr".
replace	Should sampling be with replacement? Also, passed to sample function in R. Only used in method="cr".
prob	A vector of probability weights for obtaining the elements of the vector being sampled. This is passed to the sample as well. Only used with method="cr".
parallel	Either a logical, specifying whether to do the calculations in parallel, or the number, specifying the number of cores to use for the parallel calculation.
method	Which bootstrap method to use. Currently two options are supported: "dsr" - "Data Shape Replication, implemented in dsrboot; "cr" - "Case Resampling", basic bootstrap that assumes that observations are independent (not suitable for models with ARIMA elements).
...	Parameters passed to the dsrboot function.

Details

The function applies the same model as in the provided object on a smaller sample in order to get the estimates of parameters and capture the uncertainty about them. This is a simple implementation of the case resampling, which assumes that the observations are independent.

Value

Class "bootstrap" is returned, which contains:

- vcov - the covariance matrix of parameters;
- coefficients - the matrix with the bootstrapped coefficients.
- nsim - number of runs done;
- size - the sample size used in the bootstrap;
- replace - whether the sampling was done with replacement;
- prob - a vector of probability weights used in the process;
- parallel - whether the calculations were done in parallel;
- model - the name of the model used (the name of the function);
- timeElapsed - the time that was spend on the calculations.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

alm

Examples

```
# An example with ALM
ourModel <- alm(mpg~., mtcars, distribution="dlnorm", loss="HAM")
# A fast example with 10 iterations. Use at least 100 to get better results
coefbootstrap(ourModel, nsim=10)
```

---

cramer *Calculate Cramer's V for categorical variables*

Description

Function calculates Cramer's V for two categorical variables based on the table function

Usage

```
cramer(x, y, use = c("na.or.complete", "complete.obs", "everything",
  "all.obs"), unbiased = TRUE)
```

Arguments

x	First categorical variable.
y	Second categorical variable.
use	What observations to use. See cor function for details. The only option that is not available here is "pairwise.complete.obs".
unbiased	Determines whether to calculate the biased version of Cramer's V or the one with the small sample correction.

Details

The function calculates Cramer's V and also returns the associated statistics from Chi-Squared test with the null hypothesis of independence of the two variables.

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

Value

The following list of values is returned:

- value - The value of Cramer's V;
- statistic - The value of Chi squared statistic associated with the Cramer's V;
- p.value - The p-value of Chi squared test associated with the Cramer's V;
- df - The number of degrees of freedom from the test.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

Wicher Bergsma (2013), A bias-correction for Cramér's V and Tschuprow's T. Journal of the Korean Statistical Society, 42, pp. 323-328. doi:10.1016/j.jkss.2012.10.002.

See Also

table, tableplot, spread, mcor, association

Examples

```
cramer(mtcars$am, mtcars$gear)
```

`dalaplace` *Asymmetric Laplace Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the Asymmetric Laplace distribution with the location parameter mu, scale and the asymmetry parameter alpha.

Usage

```
dalaplace(q, mu = 0, scale = 1, alpha = 0.5, log = FALSE)

palaplace(q, mu = 0, scale = 1, alpha = 0.5)

qalaplace(p, mu = 0, scale = 1, alpha = 0.5)

ralaplace(n = 1, mu = 0, scale = 1, alpha = 0.5)
```

Arguments

`q`	vector of quantiles.
`mu`	vector of location parameters (means).
`scale`	vector of scale parameters.
`alpha`	value of asymmetry parameter. Varies from 0 to 1.
`log`	if `TRUE`, then probabilities are returned in logarithms.
`p`	vector of probabilities.
`n`	number of observations. Should be a single number.

Details

When mu=0 and scale=1, the Laplace distribution becomes standardized. The distribution has the following density function:

f(x) = alpha (1-alpha) / scale exp(-(x-mu)/scale (alpha - I(x<=mu))),

where I(.) is the indicator function (equal to 1 if the condition is satisfied and zero otherwise).

When alpha=0.5, then the distribution becomes Symmetric Laplace, where scale = 1/2 MAE.

This distribution function aligns with the quantile estimates of parameters (Geraci & Bottai, 2007).

Finally, both `palaplace` and `qalaplace` are returned for the lower tail of the distribution.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- `dalaplace` returns the density function value for the provided parameters.
- `palaplace` returns the value of the cumulative function for the provided parameters.
- `qalaplace` returns quantiles of the distribution. Depending on what was provided in `p`, `mu` and `scale`, this can be either a vector or a matrix, or an array.
- `ralaplace` returns a vector of random variables generated from the Laplace distribution. Depending on what was provided in `mu` and `scale`, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, `<ivan@svetunkov.com>`

References

- Geraci Marco, Bottai Matteo (2007). Quantile regression for longitudinal data using the asymmetric Laplace distribution. Biostatistics (2007), 8, 1, pp. 140-154 doi:10.1093/biostatistics/kxj039
- Yu, K., & Zhang, J. (2005). A three-parameter asymmetric laplace distribution and its extension. Communications in Statistics - Theory and Methods, 34, 1867-1879. doi:10.1080/03610920500199018

See Also

Distributions

Examples

```
x <- dalaplace(c(-100:100)/10, 0, 1, 0.2)
plot(x, type="l")

x <- palaplace(c(-100:100)/10, 0, 1, 0.2)
plot(x, type="l")

qalaplace(c(0.025,0.975), 0, c(1,2), c(0.2,0.3))

x <- ralaplace(1000, 0, 1, 0.2)
hist(x)
```

dbcnorm *Box-Cox Normal Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the distribution that becomes normal after the Box-Cox transformation. Note that this is based on the original Box-Cox paper.

Usage

```
dbcnorm(q, mu = 0, sigma = 1, lambda = 0, log = FALSE)

pbcnorm(q, mu = 0, sigma = 1, lambda = 0)

qbcnorm(p, mu = 0, sigma = 1, lambda = 0)

rbcnorm(n = 1, mu = 0, sigma = 1, lambda = 0)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
sigma	vector of scale parameters.
lambda	the value of the Box-Cox transform parameter.
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

The distribution has the following density function:

f(y) = y^(lambda-1) 1/sqrt(2 pi) exp(-((y^lambda-1)/lambda -mu)^2 / (2 sigma^2))

Both pbcnorm and qbcnorm are returned for the lower tail of the distribution.

In case of lambda=0, the values of the log normal distribution are returned. In case of lambda=1, the values of the normal distribution are returned with mu=mu+1.

All the functions are defined for non-negative values only.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- dbcnorm returns the density function value for the provided parameters.
- pbcnorm returns the value of the cumulative function for the provided parameters.

- qbcnorm returns quantiles of the distribution. Depending on what was provided in p, mu and sigma, this can be either a vector or a matrix, or an array.
- rbcnorm returns a vector of random variables generated from the bcnorm distribution. Depending on what was provided in mu and sigma, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Box, G. E., & Cox, D. R. (1964). An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211–252. Retrieved from https://www.jstor.org/stable/2984418

See Also

Distributions

Examples

```
x <- dbcnorm(c(-1000:1000)/200, 0, 1, 1)
plot(c(-1000:1000)/200, x, type="l")

x <- pbcnorm(c(-1000:1000)/200, 0, 1, 1)
plot(c(-1000:1000)/200, x, type="l")

qbcnorm(c(0.025,0.975), 0, c(1,2), 1)

x <- rbcnorm(1000, 0, 1, 1)
hist(x)
```

---

detectdst *DST and Leap year detector functions*

Description

Functions to detect, when Daylight Saving Time and leap year start and finish

Usage

```
detectdst(object)

detectleap(object)
```

Arguments

object	Either a zoo / xts object or a vector of dates / times in POSIXt / Date class. Note that in order for detectdst() to work correctly, your data should not have missing observations. Otherwise it might not be possible to locate, when DST happens.

Details

The detectdst function detects, when the change for the DST starts and ends. It assumes that the frequency of data is not lower than hourly. The detectleap function does similar for the leap year, but flagging the 29th of February as a starting and to the 28th of February next year as the ending dates.

In order for the methods to work, the object needs to be of either zoo / xts or POSIXt class and should contain valid dates.

Value

List containing:

- start - data frame with id (number of observation) and the respective dates, when the DST / leap year start;
- end - data frame with id and dates, when DST / leap year end.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

xregExpander, temporaldummy,outlierdummy

Examples

```
# Generate matrix with monthly dummies for a zoo object
x <- as.POSIXct("2004-01-01")+0:(365*24*8)*60*60
detectdst(x)
detectleap(x)
```

---

determination *Coefficients of determination*

Description

Function produces coefficients of determination for the provided data

Usage

```
determination(xreg, bruteforce = TRUE, ...)

determ(object, ...)
```

Arguments

xreg	Data frame or a matrix, containing the exogenous variables.
bruteforce	If TRUE, then all the variables will be used for the regression construction (sink regression). If the number of observations is smaller than the number of series, the function will use stepwise function and select only meaningful variables. So the reported values will be based on stepwise regressions for each variable.
...	Other values passed to cor function.
object	The object, for which to calculate the coefficients of determination.

Details

The function calculates coefficients of determination (aka R^2) between all the provided variables. The higher the coefficient for a variable is, the higher the potential multicollinearity effect in the model with the variable will be. Coefficients of determination are connected directly to Variance Inflation Factor (VIF): VIF = 1 / (1 - determination). Arguably it is easier to interpret, because it is restricted with (0, 1) bounds. The multicollinearity can be considered as serious, when determination > 0.9 (which corresponds to VIF > 10).

The method `determ` can be applied to wide variety of classes, including `lm`, `glm` and `alm`.

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

Value

Function returns the vector of determination coefficients.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

cor, mcor, stepwise

Examples

```
### Simple example
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("x1","x2","x3","Noise")
determination(xreg)
```

dfnorm *Folded Normal Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the folded normal distribution with the location parameter mu and the scale sigma (which corresponds to standard deviation in normal distribution).

Usage

```
dfnorm(q, mu = 0, sigma = 1, log = FALSE)

pfnorm(q, mu = 0, sigma = 1)

qfnorm(p, mu = 0, sigma = 1)

rfnorm(n = 1, mu = 0, sigma = 1)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
sigma	vector of scale parameters.
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

The distribution has the following density function:

f(x) = 1/sqrt(2 pi) (exp(-(x-mu)^2 / (2 sigma^2)) + exp(-(x+mu)^2 / (2 sigma^2)))

Both pfnorm and qfnorm are returned for the lower tail of the distribution.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- dfnorm returns the density function value for the provided parameters.
- pfnorm returns the value of the cumulative function for the provided parameters.
- qfnorm returns quantiles of the distribution. Depending on what was provided in p, mu and sigma, this can be either a vector or a matrix, or an array.
- rfnorm returns a vector of random variables generated from the fnorm distribution. Depending on what was provided in mu and sigma, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Wikipedia page on folded normal distribution: https://en.wikipedia.org/wiki/Folded_normal_distribution.

See Also

Distributions

Examples

```
x <- dfnorm(c(-1000:1000)/200, 0, 1)
plot(x, type="l")

x <- pfnorm(c(-1000:1000)/200, 0, 1)
plot(x, type="l")

qfnorm(c(0.025,0.975), 0, c(1,2))

x <- rfnorm(1000, 0, 1)
hist(x)
```

---

dgnorm *The generalized normal distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the Generalised Normal distribution with the location mu, a scale and a shape parameters.

Usage

```
dgnorm(q, mu = 0, scale = 1, shape = 1, log = FALSE)

pgnorm(q, mu = 0, scale = 1, shape = 1, lower.tail = TRUE,
  log.p = FALSE)

qgnorm(p, mu = 0, scale = 1, shape = 1, lower.tail = TRUE,
  log.p = FALSE)

rgnorm(n, mu = 0, scale = 1, shape = 1)
```

Arguments

q	vector of quantiles
mu	location parameter
scale	scale parameter
shape	shape parameter
log, log.p	logical; if TRUE, probabilities p are given as log(p)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise $P[X > x]$
p	vector of probabilities
n	number of observations

Details

A generalized normal random variable x with parameters location μ, scale $s > 0$ and shape $\beta > 0$ has density:

$$p(x) = \beta exp{-(|x - \mu|/s)^{\beta}}/(2s\Gamma(1/\beta)).$$

The mean and variance of x are μ and $s^2\Gamma(3/\beta)/\Gamma(1/\beta)$, respectively.

The function are based on the functions from gnorm package of Maryclare Griffin (package has been abandoned since 2018).

The quantile and cumulative functions use uniform approximation for cases `shape>100`. This is needed, because otherwise it is not possible to calculate the values correctly due to `scale^(shape)=Inf` in R.

Author(s)

Maryclare Griffin and Ivan Svetunkov

Source

`dgnorm`, `pgnorm`, `qgnorm` and`rgnorm` are all parametrized as in Version 1 of the Generalized Normal Distribution Wikipedia page, which uses the parametrization given by in Nadarajah (2005). The same distribution was described much earlier by Subbotin (1923) and named the exponential power distribution by Box and Tiao (1973).

References

- Box, G. E. P. and G. C. Tiao. "Bayesian inference in Statistical Analysis." Addison-Wesley Pub. Co., Reading, Mass (1973).
- Nadarajah, Saralees. "A generalized normal distribution." Journal of Applied Statistics 32.7 (2005): 685-694.
- Subbotin, M. T. "On the Law of Frequency of Error." Matematicheskii Sbornik 31.2 (1923): 206-301.

See Also

Distributions

Examples

```
# Density function values for standard normal distribution
x <- dgnorm(seq(-1, 1, length.out = 100), 0, sqrt(2), 2)
plot(x, type="l")

#CDF of standard Laplace
x <- pgnorm(c(-100:100), 0, 1, 1)
plot(x, type="l")

# Quantiles of S distribution
qgnorm(c(0.025,0.975), 0, 1, 0.5)

# Random numbers from a distribution with shape=10000 (approximately uniform)
x <- rgnorm(1000, 0, 1, 1000)
hist(x)
```

Distributions *Distribution functions of the greybox package*

Description

The greybox package implements several distribution functions. In this document, I list the main functions and provide links to related resources.

Details

- Generalised normal distribution (with a kurtosis parameter by Nadarajah, 2005): qgnorm, dgnorm, pgnorm, rgnorm.
- S distribution (a special case of Generalised Normal with shape=0.5): qs, ds, ps, rs.
- Laplace distribution (special case of Generalised Normal with shape=1): qlaplace, dlaplace, plaplace, rlaplace.
- Asymmetric Laplace distribution (Yu & Zhang, 2005): qalaplace, dalaplace, palaplace, ralaplace.
- Logit Normal distribution (Mead, 1965): qlogitnorm, dlogitnorm, plogitnorm, rlogitnorm.
- Box-Cox Normal distribution (Box & Cox, 1964): qbcnorm, dbcnorm, pbcnorm, rbcnorm.
- Folded Normal distribution: qfnorm, dfnorm, pfnorm, rfnorm.
- Rectified Normal distribution: qrectnorm, drectnorm, prectnorm, rrectnorm.
- Three parameter Log Normal distribution (Sangal & Biswas, 1970): qtplnorm, dtplnorm, ptplnorm, rtplnorm.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.ru>

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Nadarajah, Saralees. "A generalized normal distribution." Journal of Applied Statistics 32.7 (2005): 685-694.
- Wikipedia page on Laplace distribution: https://en.wikipedia.org/wiki/Laplace_distribution.
- Yu, K., & Zhang, J. (2005). A three-parameter asymmetric laplace distribution and its extension. Communications in Statistics - Theory and Methods, 34, 1867-1879. doi:10.1080/03610920500199018
- Mead, R. (1965). A Generalised Logit-Normal Distribution. Biometrics, 21 (3), 721–732. doi: 10.2307/2528553
- Box, G. E., & Cox, D. R. (1964). An Analysis of Transformations. Journal of the Royal Statistical Society. Series B (Methodological), 26(2), 211–252. Retrieved from https://www.jstor.org/stable/2984418
- Sangal, B. P., & Biswas, A. K. (1970). The 3-Parameter Distribution Applications in Hydrology. Water Resources Research, 6(2), 505–515. doi:10.1029/WR006i002p00505

See Also

Distributions from the stats package.

---

dlaplace *Laplace Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the Laplace distribution with the location parameter mu and the scale parameter (which is equal to Mean Absolute Error, aka Mean Absolute Deviation).

Usage

```
dlaplace(q, mu = 0, scale = 1, log = FALSE)

plaplace(q, mu = 0, scale = 1)

qlaplace(p, mu = 0, scale = 1)

rlaplace(n = 1, mu = 0, scale = 1)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
scale	vector of mean absolute errors.
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

When mu=0 and scale=1, the Laplace distribution becomes standardized. The distribution has the following density function:

f(x) = 1/(2 scale) exp(-abs(x-mu) / scale)

Both plaplace and qlaplace are returned for the lower tail of the distribution.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- dlaplace returns the density function value for the provided parameters.
- plaplace returns the value of the cumulative function for the provided parameters.
- qlaplace returns quantiles of the distribution. Depending on what was provided in p, mu and scale, this can be either a vector or a matrix, or an array.
- rlaplace returns a vector of random variables generated from the Laplace distribution. Depending on what was provided in mu and scale, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Wikipedia page on Laplace distribution: https://en.wikipedia.org/wiki/Laplace_distribution.

See Also

Distributions

Examples

```
x <- dlaplace(c(-100:100)/10, 0, 1)
plot(x, type="l")

x <- plaplace(c(-100:100)/10, 0, 1)
plot(x, type="l")
```

```
qlaplace(c(0.025,0.975), 0, c(1,2))

x <- rlaplace(1000, 0, 1)
hist(x)
```

`dlogitnorm` *Logit Normal Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the distribution that becomes normal after the Logit transformation.

Usage

```
dlogitnorm(q, mu = 0, sigma = 1, log = FALSE)

plogitnorm(q, mu = 0, sigma = 1)

qlogitnorm(p, mu = 0, sigma = 1)

rlogitnorm(n = 1, mu = 0, sigma = 1)
```

Arguments

`q`	vector of quantiles.
`mu`	vector of location parameters (means).
`sigma`	vector of scale parameters.
`log`	if `TRUE`, then probabilities are returned in logarithms.
`p`	vector of probabilities.
`n`	number of observations. Should be a single number.

Details

The distribution has the following density function:

f(y) = 1/(sqrt(2 pi) y (1-y)) exp(-(logit(y) -mu)^2 / (2 sigma^2))

where y is in (0, 1) and logit(y) =log(y/(1-y)).

Both `plogitnorm` and `qlogitnorm` are returned for the lower tail of the distribution.

All the functions are defined for the values between 0 and 1.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- `dlogitnorm` returns the density function value for the provided parameters.
- `plogitnorm` returns the value of the cumulative function for the provided parameters.
- `qlogitnorm` returns quantiles of the distribution. Depending on what was provided in `p`, `mu` and `sigma`, this can be either a vector or a matrix, or an array.
- `rlogitnorm` returns a vector of random variables generated from the logitnorm distribution. Depending on what was provided in `mu` and `sigma`, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Mead, R. (1965). A Generalised Logit-Normal Distribution. Biometrics, 21 (3), 721–732. doi: 10.2307/2528553

See Also

Distributions

Examples

```
x <- dlogitnorm(c(-1000:1000)/200, 0, 1)
plot(c(-1000:1000)/200, x, type="l")

x <- plogitnorm(c(-1000:1000)/200, 0, 1)
plot(c(-1000:1000)/200, x, type="l")

qlogitnorm(c(0.025,0.975), 0, c(1,2))

x <- rlogitnorm(1000, 0, 1)
hist(x)
```

---

drectnorm *Rectified Normal Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the Rectified Normal distribution.

Usage

```
drectnorm(q, mu = 0, sigma = 1, log = FALSE)

prectnorm(q, mu = 0, sigma = 1)

qrectnorm(p, mu = 0, sigma = 1)

rrectnorm(n = 1, mu = 0, sigma = 1)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
sigma	vector of scale parameters.
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

If x~N(mu, sigma^2) then y = max(0, x) follows Rectified Normal distribution: y~RectN(mu, sigma^2), which can be written as:

f_y = 1(x<=0) F_x(mu, sigma) + 1(x>0) f_x(x, mu, sigma),

where F_x is the cumulative distribution function and f_x is the probability density function of normal distribution.

Both prectnorm and qrectnorm are returned for the lower tail of the distribution.

All the functions are defined for non-negative values only.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- drectnorm returns the density function value for the provided parameters.
- prectnorm returns the value of the cumulative function for the provided parameters.
- qrectnorm returns quantiles of the distribution. Depending on what was provided in p, mu and sigma, this can be either a vector or a matrix, or an array.
- rrectnorm returns a vector of random variables generated from the RectN distribution. Depending on what was provided in mu and sigma, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

Distributions

Examples

```
x <- drectnorm(c(-1000:1000)/200, 0, 1)
plot(c(-1000:1000)/200, x, type="l")

x <- prectnorm(c(-1000:1000)/200, 0, 1)
plot(c(-1000:1000)/200, x, type="l")

qrectnorm(c(0.025,0.975), 0, c(1,2))

x <- rrectnorm(1000, 0, 1)
hist(x)
```

---

ds *S Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the S distribution with the location parameter mu and a scaling parameter scale.

Usage

```
ds(q, mu = 0, scale = 1, log = FALSE)

ps(q, mu = 0, scale = 1)

qs(p, mu = 0, scale = 1)

rs(n = 1, mu = 0, scale = 1)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
scale	vector of scaling parameter (which are equal to ham/2).
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

When mu=0 and ham=2, the S distribution becomes standardized with scale=1 (this is because scale=ham/2). The distribution has the following density function:

f(x) = 1/(4 scale^2) exp(-sqrt(abs(x-mu)) / scale)

The S distribution has fat tails and large excess.

Both `ps` and `qs` are returned for the lower tail of the distribution.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- `ds` returns the density function value for the provided parameters.
- `ps` returns the value of the cumulative function for the provided parameters.
- `qs` returns quantiles of the distribution. Depending on what was provided in `p`, `mu` and `scale`, this can be either a vector or a matrix, or an array.
- `rs` returns a vector of random variables generated from the S distribution. Depending on what was provided in `mu` and `scale`, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

Distributions

Examples

```
x <- ds(c(-1000:1000)/10, 0, 1)
plot(x, type="l")

x <- ps(c(-1000:1000)/10, 0, 1)
plot(x, type="l")

qs(c(0.025,0.975), 0, 1)

x <- rs(1000, 0, 1)
hist(x)
```

dsrboot *Data Shape Replication Bootstrap*

Description

The function implements a bootstrap inspired by the Maximum Entropy Bootstrap

Usage

```
dsrboot(y, nsim = 100, intermittent = FALSE, type = c("multiplicative",
  "additive"), kind = c("nonparametric", "parametric"), lag = frequency(y),
  sd = NULL, scale = TRUE)

## S3 method for class 'dsrboot'
plot(x, sorted = FALSE, legend = TRUE, ...)
```

Arguments

y	The original time series
nsim	Number of iterations (simulations) to run.
intermittent	Whether to treat the demand as intermittent or not.
type	Type of bootstrap to use. "additive" means that the randomness is added, while "multiplicative" implies the multiplication.
kind	A kind of the bootstrap to do: nonparametric or parametric. The latter relies on the normal distribution, while the former uses the empirical distribution of differences of the data.
lag	The lag to use in the calculation of differences. Should be 1 for non-seasonal data.
sd	Standard deviation to use in the normal distribution. Estimated as mean absolute differences of the data if omitted.
scale	Whether or not to do scaling of time series to the bootstrapped ones to have similar variance to the original data.
x	The object of the class dsrboot.
sorted	Whether the sorted (TRUE) or the original (FALSE) data should be used.
legend	Whether to produce the legend on the plot.
...	Other parameters passed to the plot function.

Details

The "Data Shape Replication" bootstrap reproduces the shape of the original time series by creating randomness around it. It is done in the following steps:

1. Sort the data in the ascending order, recording the original order of elements; 2. Take first differences of the original data and sort them; 3. Generate random numbers from the uniform distribution between 0 and 1; 4. Get the smoothed differences that correspond to the random numbers

(randomly extract empirical quantiles). This way we take the empirical density into account when selecting the differences; 5. Add the random differences to the sorted series from (1) to get a new time series; 6. Sort the new time series in the ascending order; 7. Reorder (6) based on the initial order of series; 8. Centre the data around the original series; 9. Scale the data to make sure that the variance is constant over time.

If the multiplicative bootstrap is used then logarithms of the sorted series are used and at the very end, the exponent of the resulting data is taken. This way the discrepancies in the data have similar scale no matter what the level of the original series is. In case of the additive bootstrap, the trended series will be noisier when the level of series is low.

Value

The function returns:

- `call` - the call that was used originally;
- `data` - the original data used in the function;
- `boot` - the matrix with the new series in columns and observations in rows.
- `type` - type of the bootstrap used.
- `sd` - the value of sd used in case of parameteric bootstrap.
- `scale` - whether the scaling was needed.
- `smooth` - the smoothed ordered actual data.

Author(s)

Ivan Svetunkov, `<ivan@svetunkov.com>`

References

Vinod HD, López-de-Lacalle J (2009). "Maximum Entropy Bootstrap for Time Series: The meboot R Package." Journal of Statistical Software, 29(5), 1–19. doi:10.18637/jss.v029.i05.

Examples

```
plot(dsrboot(AirPassengers))
```

`dtplnorm` *Three Parameter Log Normal Distribution*

Description

Density, cumulative distribution, quantile functions and random number generation for the 3 parameter log normal distribution with the location parameter mu, scale sigma (which corresponds to standard deviation in normal distribution) and shifting parameter shift.

Usage

```
dtplnorm(q, mu = 0, sigma = 1, shift = 0, log = FALSE)

ptplnorm(q, mu = 0, sigma = 1, shift = 0)

qtplnorm(p, mu = 0, sigma = 1, shift = 0)

rtplnorm(n = 1, mu = 0, sigma = 1, shift = 0)
```

Arguments

q	vector of quantiles.
mu	vector of location parameters (means).
sigma	vector of scale parameters.
shift	vector of shift parameters.
log	if TRUE, then probabilities are returned in logarithms.
p	vector of probabilities.
n	number of observations. Should be a single number.

Details

The distribution has the following density function:

f(x) = 1/(x-a) 1/sqrt(2 pi) exp(-(log(x-a)-mu)^2 / (2 sigma^2))

Both `ptplnorm` and `qtplnorm` are returned for the lower tail of the distribution.

The function is based on the lnorm functions from stats package, introducing the shift parameter.

Value

Depending on the function, various things are returned (usually either vector or scalar):

- `dtplnorm` returns the density function value for the provided parameters.
- `ptplnorm` returns the value of the cumulative function for the provided parameters.
- `qtplnorm` returns quantiles of the distribution. Depending on what was provided in `p`, `mu` and `sigma`, this can be either a vector or a matrix, or an array.
- `rtplnorm` returns a vector of random variables generated from the tplnorm distribution. Depending on what was provided in `mu` and `sigma`, this can be either a vector or a matrix or an array.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Sangal, B. P., & Biswas, A. K. (1970). The 3-Parameter Distribution Applications in Hydrology. Water Resources Research, 6(2), 505–515. doi:10.1029/WR006i002p00505

See Also

Distributions

Examples

```
x <- dtplnorm(c(-1000:1000)/200, 0, 1, 1)
plot(c(-1000:1000)/200, x, type="l")

x <- ptplnorm(c(-1000:1000)/200, 0, 1, 1)
plot(c(-1000:1000)/200, x, type="l")

qtplnorm(c(0.025,0.975), 0, c(1,2), 1)

x <- rtplnorm(1000, 0, 1, 1)
hist(x)
```

---

errorType *Functions that extracts type of error from the model*

Description

This function allows extracting error type from any model.

Usage

```
errorType(object, ...)
```

Arguments

object	Model estimated using one of the functions of smooth package.
...	Currently nothing is accepted via ellipsis.

Details

errorType extracts the type of error from the model (either additive or multiplicative).

Value

Either "A" for additive error or "M" for multiplicative. All the other functions return strings of character.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
ourModel <- alm(y~x1+x2,as.data.frame(xreg))

errorType(ourModel)
```

`extractScale` *Functions to extract scale and standard error from a model*

Description

Functions extract scale and the standard error of the residuals. Mainly needed for the work with the model estimated via sm.

Usage

```
extractScale(object, ...)

## Default S3 method:
extractScale(object, ...)

## S3 method for class 'greybox'
extractScale(object, ...)

extractSigma(object, ...)

## Default S3 method:
extractSigma(object, ...)

## S3 method for class 'greybox'
extractSigma(object, ...)
```

Arguments

`object`	The model estimated using lm / alm / etc.
`...`	Other parameters (currently nothing).

Details

In case of a simpler model, the functions will return the scalar using `sigma()` method. If the scale model was estimated, `extractScale()` and `extractSigma()` will return the conditional scale and the conditional standard error of the residuals respectively.

Value

One of the two is returned, depending on the type of estimated model:

- Scalar from `sigma()` method if the variance is assumed to be constant.
- Vector of values if the scale model was estimated

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

sm

Examples

```
# Generate the data
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+sqrt(exp(0.8+0.2*xreg[,1]))*rnorm(100,0,1),
              xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

# Estimate the location and scale model
ourModel <- alm(y~., xreg, scale=~x1+x2)

# Extract scale
extractScale(ourModel)
# Extract standard error
extractSigma(ourModel)
```

---

graphmaker *Linear graph construction function*

Description

The function makes a standard linear graph using the provided actuals and forecasts.

Usage

```
graphmaker(actuals, forecast, fitted = NULL, lower = NULL, upper = NULL,
  level = NULL, legend = TRUE, vline = TRUE, parReset = TRUE, ...)
```

Arguments

actuals	The vector of actual values
forecast	The vector of forecasts. Should be ts object that starts at the end of fitted values.
fitted	The vector of fitted values.
lower	The vector of lower bound values of a prediction interval. Should be ts object that start at the end of fitted values.
upper	The vector of upper bound values of a prediction interval. Should be ts object that start at the end of fitted values.
level	The width of the prediction interval.
legend	If TRUE, the legend is drawn.
vline	Whether to draw the vertical line, splitting the in-sample and the holdout sample.
parReset	Whether to reset par() after plotting things or not. If FALSE then you can add elements to the plot (e.g. additional lines).
...	Other parameters passed to plot() function.

Details

Function uses the provided data to construct a linear graph. It is strongly advised to use ts objects to define the start of each of the vectors. Otherwise the data may be plotted incorrectly. The colours can be changed by defining a different palette via the palette() function. The function then would use colours 1 - 6 in the palette.

Value

Function does not return anything.

Author(s)

Ivan Svetunkov

See Also

ts

Examples

```
xreg <- cbind(y=rnorm(100,100,10),x=rnorm(100,10,10))
almModel <- alm(y~x, xreg, subset=c(1:90))
values <- predict(almModel, newdata=xreg[-c(1:90),], interval="prediction")

graphmaker(xreg[,1],values$mean,fitted(values))
graphmaker(xreg[,1],values$mean,fitted(values),legend=FALSE)
graphmaker(xreg[,1],values$mean,fitted(values),legend=FALSE,lower=values$lower,upper=values$upper)

# Produce the necessary ts objects from an arbitrary vectors
actuals <- ts(c(1:10), start=c(2000,1), frequency=4)
```

```
forecast <- ts(c(11:15),start=end(actuals)[1]+end(actuals)[2]*deltat(actuals),
               frequency=frequency(actuals))
graphmaker(actuals,forecast)

# This should work as well
graphmaker(c(1:10),c(11:15))

# This way you can add additional elements to the plot
graphmaker(c(1:10),c(11:15), parReset=FALSE)
points(c(1:15))
# But don't forget to do dev.off() in order to reset the plotting area afterwards
```

`greybox` *Grey box*

Description

Toolbox for working with univariate models for purposes of analysis and forecasting

Details

Package:	greybox
Type:	Package
Date:	2018-02-13 - Inf
License:	GPL-2

The following functions are included in the package:

- AICc and BICc - AIC / BIC corrected for the sample size.
- pointLik - point likelihood of the function.
- pAIC, pAICc, pBIC, pBICc - point versions of respective information criteria.
- coefbootstrap - Method that uses a simple implementation of the case resampling to get bootstrapped estimates of parameters of the model.
- dsrboot - Bootstrap inspired by the meboot package, that creates bootstraped series based on the provided one.
- determination - Coefficients of determination between different exogenous variables.
- temporaldummy - Matrix with seasonal dummy variables.
- outlierdummy - Matrix with dummies for outliers.
- alm - Advanced Linear Model - regression, estimated using likelihood with specified distribution (e.g. Laplace or Chi-Squared).
- sm - Scale Model - Regression model for scale of distribution (e.g. for Variance of Normal distribution). Requires an `lm()` or `alm()` model.

- stepwise - Stepwise based on information criteria and partial correlations. Efficient and fast.
- xregExpander - Function that expands the provided data into the data with lags and leads.
- xregTransformer - Function produces mathematical transformations of the variables, such as taking logarithms, square roots etc.
- xregMultiplier - Function produces cross-products of the matrix of the provided variables.
- lmCombine - Function combines lm models from the estimated based on information criteria weights.
- lmDynamic - Dynamic regression based on point AIC.
- ro - Rolling origin evaluation.
- Distributions - document explaining the distribution functions of the greybox package.
- spread - function that produces scatterplots / boxplots / tableplots, depending on the types of variables.
- assoc - function that calculates measures of association, depending on the types of variables.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

stepwise, lmCombine

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

stepwise(xreg)
```

---

hm *Half moment of a distribution and its derivatives.*

Description

hm() function estimates half moment from some predefined constant C. ham() estimates the Half Absolute Moment. asymmetry() function returns Asymmetry coefficient, while extremity() returns the coefficient of Extremity, both based on hm(). Finally, cextremity() returns the Complex Extremity coefficient, based on hm().

Usage

```
hm(x, C = mean(x, na.rm = TRUE), ...)

ham(x, C = mean(x, na.rm = TRUE), ...)

asymmetry(x, C = mean(x, na.rm = TRUE), ...)

extremity(x, C = mean(x, na.rm = TRUE), ...)

cextremity(x, C = mean(x, na.rm = TRUE), ...)
```

Arguments

x	A variable based on which HM is estimated.
C	Centring parameter.
...	Other parameters passed to mean function.

Details

NA values of x are excluded on the first step of calculation.

Value

A complex variable is returned for the hm() and cextremity() functions, and real values are returned for ham(), asymmetry() and extremity().

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov I., Kourentzes N., Svetunkov S. "Half Central Moment for Data Analysis". Working Paper of Department of Management Science, Lancaster University, 2023:3, 1–21.

Examples

```
x <- rnorm(100,0,1)
hm(x)
ham(x)
asymmetry(x)
extremity(x)
cextremity(x)
```

implant *Implant the scale model in the location model*

Description

The function implants the scale model into the location model. It currently works with alm / adam and sm() method.

Usage

```
implant(location, scale, ...)
```

Arguments

location	Model estimated using either alm or adam.
scale	The scale model, estimate with sm method.
...	Currently nothing. Implemented for flexibility.

Details

The function is needed in order to treat the scale of model correctly in the methods like forecast().

Value

The model of the same class as the location model, but with scale from the estimated model via sm(). This is needed to produce appropriate forecasts in case of scale model and to take into account the correct number of estimated parameters.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

alm, adam, sm

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+sqrt(exp(0.8+0.2*xreg[,1]))*rnorm(100,0,1),
              xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

# Estimate the location model
ourModel <- alm(y~.,xreg)
# Estimate the scale model
ourScale <- sm(ourModel,formula=~x1+x2)
# Implant scale into location
```

```
ourModel <- implant(ourModel, ourScale)
```

`is.greybox` *Greybox classes checkers*

Description

Functions to check if an object is of the specified class

Usage

```
is.greybox(x)

is.alm(x)

is.occurrence(x)

is.greyboxC(x)

is.greyboxD(x)

is.rollingOrigin(x)

is.rmc(x)

is.scale(x)
```

Arguments

x	The object to check.

Details

The list of functions includes:

- `is.greybox()` tests if the object was produced by a greybox function (e.g. alm / stepwise / lmCombine / lmDynamic);
- `is.alm()` tests if the object was produced by `alm()` function;
- `is.occurrence()` tests if an occurrence part of the model was produced;
- `is.greyboxC()` tests if the object was produced by `lmCombine()` function;
- `is.greyboxD()` tests if the object was produced by `lmDynamic()` function;
- `is.rmc()` tests if the object was produced by `rmc()` function;
- `is.rollingOrigin()` tests if the object was produced by `ro()` function;
- `is.scale()` tests if the object is of the class "scale" (produced by alm or sm in case of heteroscedastic model);

Value

TRUE if this is the specified class and FALSE otherwise.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
xreg <- cbind(rlaplace(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rlaplace(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

ourModel <- alm(y~x1+x2, xreg, distribution="dnorm")

is.alm(ourModel)
is.greybox(ourModel)
is.greyboxC(ourModel)
is.greyboxD(ourModel)
```

---

lmDynamic *Combine regressions based on point information criteria*

Description

Function combines parameters of linear regressions of the first variable on all the other provided data using pAIC weights. This is an extension of the lmCombine function, which relies upon the idea that the combination weights might change over time.

Usage

```
lmDynamic(data, ic = c("AICc", "AIC", "BIC", "BICc"), bruteforce = FALSE,
  silent = TRUE, formula = NULL, subset = NULL,
  distribution = c("dnorm", "dlaplace", "ds", "dgnorm", "dlogis", "dt",
  "dalaplace", "dlnorm", "dllaplace", "dls", "dlgnorm", "dbcnorm", "dfnorm",
  "dinvgauss", "dgamma", "dpois", "dnbinom", "dlogitnorm", "plogis", "pnorm"),
  parallel = FALSE, lowess = TRUE, f = NULL, ...)
```

Arguments

data	Data frame containing dependent variable in the first column and the others in the rest.
ic	Information criterion to use.
bruteforce	If TRUE, then all the possible models are generated and combined. Otherwise the best model is found and then models around that one are produced and then combined.

silent	If FALSE, then nothing is silent, everything is printed out. TRUE means that nothing is produced.
formula	If provided, then the selection will be done from the listed variables in the formula after all the necessary transformations.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
distribution	Distribution to pass to alm(). See alm for details.
parallel	If TRUE, then the model fitting is done in parallel. WARNING! Packages foreach and either doMC (Linux and Mac only) or doParallel are needed in order to run the function in parallel.
lowess	Logical defining, whether LOWESS should be used to smooth the dynamic weights. By default it is TRUE.
f	the smoother span for LOWESS. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. If NULL the parameter will be optimised by minimising ic.
...	Other parameters passed to alm().

Details

The algorithm uses alm() to fit different models and then combines the models based on the selected point IC. The combination weights are calculated for each observation based on the point IC and then smoothed via LOWESS if the respective parameter (lowess) is set to TRUE.

Some details and examples of application are also given in the vignette "Greybox": vignette("greybox","greybox")

Value

Function returns model - the final model of the class "greyboxD", which includes time varying parameters and dynamic importance of each variable. The list of variables:

- coefficients - the mean (over time) parameters of the model,
- vcov - the combined covariance matrix of the model,
- fitted - the fitted values,
- residuals - the residuals of the model,
- distribution - the distribution used in the estimation,
- logLik - the mean (over time) log-likelihood of the model,
- IC - dynamic values of the information criterion (pIC),
- ICType - the type of information criterion used,
- df.residual - mean number of degrees of freedom of the residuals of the model,
- df - mean number of degrees of freedom of the model,
- importance - dynamic importance of the parameters,
- call - call used in the function,
- rank - rank of the combined model,
- data - the data used in the model,

- mu - the location value of the distribution,
- scale - the scale parameter if alm() was used,
- coefficientsDynamic - table with parameters of the model, varying over the time,
- df.residualDynamic - dynamic df.residual,
- dfDynamic - dynamic df,
- weights - the dynamic weights for each model under consideration,
- timeElapsed - the time elapsed for the estimation of the model.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Burnham Kenneth P. and Anderson David R. (2002). Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach. Springer-Verlag New York. DOI: [10.1007/b97636](http://dx.doi.org/10.1007/b97636).
- McQuarrie, A. D. (1999). A small-sample correction for the Schwarz SIC model selection criterion. Statistics & Probability Letters, 44(1), 79–86. [10.1016/S0167-7152(98)00294-6](https://doi.org/10.1016/S0167-7152(98)00294-6).

See Also

stepwise, lmCombine

Examples

```
### Simple example
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
inSample <- xreg[1:80,]
outSample <- xreg[-c(1:80),]
# Combine all the possible models
ourModel <- lmDynamic(inSample,bruteforce=TRUE)
predict(ourModel,outSample)
plot(predict(ourModel,outSample))
```

---

mcor *Multiple correlation*

Description

Function calculates multiple correlation between y and x, constructing a linear regression model

Usage

```
mcor(x, y, use = c("na.or.complete", "complete.obs", "everything",
  "all.obs"))
```

Arguments

x	Either data.frame or a matrix
y	The numerical variable.
use	What observations to use. See cor function for details. The only option that is not available here is "pairwise.complete.obs".

Details

This is based on the linear regression model with the set of variables in x. The returned value is just a coefficient of multiple correlation from regression, the F-statistics of the model (thus testing the null hypothesis that all the parameters are equal to zero), the associated p-value and the degrees of freedom.

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

Value

The following list of values is returned:

- value - The value of the coefficient;
- statistic - The value of F-statistics associated with the parameter;
- p.value - The p-value of F-statistics associated with the parameter;
- df.residual - The number of degrees of freedom for the residuals;
- df - The number of degrees of freedom for the data.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

table, tableplot, spread,cramer, association

Examples

```
mcor(mtcars$am, mtcars$mpg)
```

ME *Error measures*

Description

Functions allow to calculate different types of error metrics for point and interval forecasts:

1. ME - Mean Error,
2. MAE - Mean Absolute Error,
3. MSE - Mean Squared Error,
4. MRE - Mean Root Error (Kourentzes, 2014),
5. MIS - Mean Interval Score (Gneiting & Raftery, 2007),
6. MPE - Mean Percentage Error,
7. MAPE - Mean Absolute Percentage Error (See Svetunkov, 2017 for the critique),
8. MASE - Mean Absolute Scaled Error (Hyndman & Koehler, 2006),
9. RMSSE - Root Mean Squared Scaled Error (used in M5 Competition),
10. SAME - Scaled Absolute Mean Error (similar to MASE, but measures bias),
11. rMAE - Relative Mean Absolute Error (Davydenko & Fildes, 2013),
12. rRMSE - Relative Root Mean Squared Error,
13. rAME - Relative Absolute Mean Error,
14. rMIS - Relative Mean Interval Score,
15. sMSE - Scaled Mean Squared Error (Petropoulos & Kourentzes, 2015),
16. sPIS- Scaled Periods-In-Stock (Wallstrom & Segerstedt, 2010),
17. sCE - Scaled Cumulative Error,
18. sMIS - Scaled Mean Interval Score,
19. GMRAE - Geometric Mean Relative Absolute Error.

Usage

```
ME(holdout, forecast, na.rm = TRUE)

MAE(holdout, forecast, na.rm = TRUE)

MSE(holdout, forecast, na.rm = TRUE)

MRE(holdout, forecast, na.rm = TRUE)

MIS(holdout, lower, upper, level = 0.95, na.rm = TRUE)

MPE(holdout, forecast, na.rm = TRUE)

MAPE(holdout, forecast, na.rm = TRUE)
```

```
MASE(holdout, forecast, scale, na.rm = TRUE)

RMSSE(holdout, forecast, scale, na.rm = TRUE)

rMAE(holdout, forecast, benchmark, na.rm = TRUE)

rRMSE(holdout, forecast, benchmark, na.rm = TRUE)

rAME(holdout, forecast, benchmark, na.rm = TRUE)

rMIS(holdout, lower, upper, benchmarkLower, benchmarkUpper, level = 0.95,
  na.rm = TRUE)

sMSE(holdout, forecast, scale, na.rm = TRUE)

sPIS(holdout, forecast, scale, na.rm = TRUE)

sCE(holdout, forecast, scale, na.rm = TRUE)

sMIS(holdout, lower, upper, scale, level = 0.95, na.rm = TRUE)

GMRAE(holdout, forecast, benchmark, na.rm = TRUE)

SAME(holdout, forecast, scale, na.rm = TRUE)
```

Arguments

holdout	The vector or matrix of holdout values.
forecast	The vector or matrix of forecasts values.
na.rm	Logical, defining whether to remove the NAs from the provided data or not.
lower	The lower bound of the prediction interval.
upper	The upper bound of the prediction interval.
level	The confidence level of the constructed interval.
scale	The value that should be used in the denominator of MASE. Can be anything but advised values are: mean absolute deviation of in-sample one step ahead Naive error or mean absolute value of the in-sample actuals.
benchmark	The vector or matrix of the forecasts of the benchmark model.
benchmarkLower	The lower bound of the prediction interval of the benchmark model.
benchmarkUpper	The upper bound of the prediction interval of the benchmark model.

Details

In case of sMSE, scale needs to be a squared value. Typical one – squared mean value of in-sample actuals.

If all the measures are needed, then measures function can help.

There are several other measures, see details of pinball and hm.

Value

All the functions return the scalar value.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Kourentzes N. (2014). The Bias Coefficient: a new metric for forecast bias https://kourentzes.com/forecasting/2014/12/17/the-bias-coefficient-a-new-metric-for-forecast-bias/
- Svetunkov, I. (2017). Naughty APEs and the quest for the holy grail. https://openforecast.org/2017/07/29/naughty-apes-and-the-quest-for-the-holy-grail/
- Fildes R. (1992). The evaluation of extrapolative forecasting methods. International Journal of Forecasting, 8, pp.81-98.
- Hyndman R.J., Koehler A.B. (2006). Another look at measures of forecast accuracy. International Journal of Forecasting, 22, pp.679-688.
- Petropoulos F., Kourentzes N. (2015). Forecast combinations for intermittent demand. Journal of the Operational Research Society, 66, pp.914-924.
- Wallstrom P., Segerstedt A. (2010). Evaluation of forecasting error measurements and techniques for intermittent demand. International Journal of Production Economics, 128, pp.625-636.
- Davydenko, A., Fildes, R. (2013). Measuring Forecasting Accuracy: The Case Of Judgmental Adjustments To Sku-Level Demand Forecasts. International Journal of Forecasting, 29(3), 510-522. doi:10.1016/j.ijforecast.2012.09.002
- Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association, 102(477), 359–378. doi:10.1198/016214506000001437

See Also

pinball, hm, measures

Examples

```
y <- rnorm(100,10,2)
testForecast <- rep(mean(y[1:90]),10)

MAE(y[91:100],testForecast)
MSE(y[91:100],testForecast)

MPE(y[91:100],testForecast)
MAPE(y[91:100],testForecast)

# Measures from Petropoulos & Kourentzes (2015)
MASE(y[91:100],testForecast,mean(abs(y[1:90])))
sMSE(y[91:100],testForecast,mean(abs(y[1:90]))^2)
```

```
sPIS(y[91:100],testForecast,mean(abs(y[1:90])))
sCE(y[91:100],testForecast,mean(abs(y[1:90])))

# Original MASE from Hyndman & Koehler (2006)
MASE(y[91:100],testForecast,mean(abs(diff(y[1:90]))))

testForecast2 <- rep(y[91],10)
# Relative measures, from and inspired by Davydenko & Fildes (2013)
rMAE(y[91:100],testForecast2,testForecast)
rRMSE(y[91:100],testForecast2,testForecast)
rAME(y[91:100],testForecast2,testForecast)
GMRAE(y[91:100],testForecast2,testForecast)

#### Measures for the prediction intervals
# An example with mtcars data
ourModel <- alm(mpg~., mtcars[1:30,], distribution="dnorm")
ourBenchmark <- alm(mpg~1, mtcars[1:30,], distribution="dnorm")

# Produce predictions with the interval
ourForecast <- predict(ourModel, mtcars[-c(1:30),], interval="p")
ourBenchmarkForecast <- predict(ourBenchmark, mtcars[-c(1:30),], interval="p")

MIS(mtcars$mpg[-c(1:30)],ourForecast$lower,ourForecast$upper,0.95)
sMIS(mtcars$mpg[-c(1:30)],ourForecast$lower,ourForecast$upper,mean(mtcars$mpg[1:30]),0.95)
rMIS(mtcars$mpg[-c(1:30)],ourForecast$lower,ourForecast$upper,
       ourBenchmarkForecast$lower,ourBenchmarkForecast$upper,0.95)

### Also, see pinball function for other measures for the intervals
```

`measures` *Error measures for the provided forecasts*

Description

Function calculates several error measures using the provided forecasts and the data for the holdout sample.

Usage

```
measures(holdout, forecast, actual, digits = NULL, benchmark = c("naive",
  "mean"))
```

Arguments

`holdout`	The vector of the holdout values.
`forecast`	The vector of forecasts produced by a model.
`actual`	The vector of actual in-sample values.
`digits`	Number of digits of the output. If `NULL` then no rounding is done.

benchmark — The character variable, defining what to use as benchmark for relative measures. Can be either "naive" or "mean" (arithmetic mean of the whole series. The latter can be useful when dealing with intermittent data.

Value

The functions returns the named vector of errors:

- ME,
- MAE,
- MSE
- MPE,
- MAPE,
- MASE,
- sMAE,
- RMSSE,
- SAME,
- sMSE,
- sCE,
- rMAE,
- rRMSE,
- rAME,
- asymmetry,
- sPIS.

For the details on these errors, see Errors.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Svetunkov, I. (2017). Naughty APEs and the quest for the holy grail. https://openforecast.org/2017/07/29/naughty-apes-and-the-quest-for-the-holy-grail/
- Fildes R. (1992). The evaluation of extrapolative forecasting methods. International Journal of Forecasting, 8, pp.81-98.
- Hyndman R.J., Koehler A.B. (2006). Another look at measures of forecast accuracy. International Journal of Forecasting, 22, pp.679-688.
- Petropoulos F., Kourentzes N. (2015). Forecast combinations for intermittent demand. Journal of the Operational Research Society, 66, pp.914-924.
- Wallstrom P., Segerstedt A. (2010). Evaluation of forecasting error measurements and techniques for intermittent demand. International Journal of Production Economics, 128, pp.625-636.
- Davydenko, A., Fildes, R. (2013). Measuring Forecasting Accuracy: The Case Of Judgmental Adjustments To Sku-Level Demand Forecasts. International Journal of Forecasting, 29(3), 510-522. doi:10.1016/j.ijforecast.2012.09.002

Examples

```
y <- rnorm(100,10,2)
ourForecast <- rep(mean(y[1:90]),10)

measures(y[91:100],ourForecast,y[1:90],digits=5)
```

`multipliers` *Dynamic multipliers from an ARDL model*

Description

This function extracts the beta (distributed lag) coefficients for a specific variable from an estimated ALM model and ARIMA(p,d,0) polynomials. It then uses them to calculate dynamic multipliers for that variable for the horizon `h`.

Usage

```
multipliers(object, parm, h = 10)
```

Arguments

`object`	An estimated ALM model object (with coefficients from `coef()`).
`parm`	Character string of the variable name.
`h`	Horizon for which to produce the dynamic multipliers.

Value

Numeric vector of dynamic multipliers over time

See Also

, `B` for creating lagged variables

Examples

```
## Not run:
  # Fit a model with lagged variables
  test <- alm(drivers ~ kms + law + B(kms, 1) + B(kms, 2),
              Seatbelts, orders = c(1, 0, 0))
  multipliers(test, "kms", h=10)

## End(Not run)
```

nparam *Number of parameters and number of variates in the model*

Description

`nparam()` returns the number of estimated parameters in the model, while `nvariate()` returns number of variates for the response variable.

Usage

```
nparam(object, ...)

nvariate(object, ...)
```

Arguments

`object`	Time series model.
`...`	Some other parameters passed to the method.

Details

`nparam()` is a very basic and a simple function which does what it says: extracts number of estimated parameters in the model. `nvariate()` returns number of variates (dimensions, columns) for the response variable (1 for the univariate regression).

Value

Both functions return numeric values.

Author(s)

Ivan Svetunkov, `<ivan@svetunkov.com>`

See Also

nobs, logLik

Examples

```
### Simple example
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
ourModel <- lm(y~.,data=as.data.frame(xreg))

nparam(ourModel)
nvariate(ourModel)
```

`outlierdummy` *Outlier detection and matrix creation*

Description

Function detects outliers and creates a matrix with dummy variables. Only point outliers are considered (no level shifts).

Usage

```
outlierdummy(object, ...)

## Default S3 method:
outlierdummy(object, level = 0.999, type = c("rstandard",
  "rstudent"), ...)

## S3 method for class 'alm'
outlierdummy(object, level = 0.999, type = c("rstandard",
  "rstudent"), ...)
```

Arguments

`object`	Model estimated using one of the functions of smooth package.
`...`	Other parameters. Not used yet.
`level`	Confidence level to use. Everything that is outside the constructed bounds based on that is flagged as outliers.
`type`	Type of residuals to use: either standardised or studentised. Ignored if count distributions used.

Details

The detection is done based on the type of distribution used and confidence level specified by user.

Value

The class "outlierdummy", which contains the list:

- outliers - the matrix with the dummy variables, flagging outliers;
- statistic - the value of the statistic for the normalised variable;
- id - the ids of the outliers (which observations have them);
- level - the confidence level used in the process;
- type - the type of the residuals used;
- errors - the errors used in the detection. In case of count distributions, probabilities are returned.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

influence.measures

Examples

```
# Generate the data with S distribution
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rs(100,0,3),xreg)
colnames(xreg) <- c("y","x1","x2")

# Fit the normal distribution model
ourModel <- alm(y~x1+x2, xreg, distribution="dnorm")

# Detect outliers
xregOutlierDummy <- outlierdummy(ourModel)
```

---

pAIC *Point AIC*

Description

This function returns a vector of AIC values for the in-sample observations

Usage

```
pAIC(object, ...)

pAICc(object, ...)

pBIC(object, ...)

pBICc(object, ...)
```

Arguments

object	Time series model.
...	Some stuff.

Details

This is based on pointLik function. The formula for this is: pAIC_t = 2 * k - 2 * T * l_t , where k is the number of parameters, T is the number of observations and l_t is the point likelihood. This way we preserve the property that AIC = mean(pAIC).

Value

The function returns the vector of point AIC values.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

pointLik

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
ourModel <- alm(y~x1+x2,as.data.frame(xreg))

pAICValues <- pAIC(ourModel)

mean(pAICValues)
AIC(ourModel)
```

---

pcor *Partial correlations*

Description

Function calculates partial correlations between the provided variables

Usage

```
pcor(x, y = NULL, use = c("na.or.complete", "complete.obs", "everything",
  "all.obs"), method = c("pearson", "spearman", "kendall"))
```

Arguments

x	Either data.frame or a matrix with numeric values.
y	The numerical variable.
use	What observations to use. See cor function for details. The only option that is not available here is "pairwise.complete.obs".
method	Which method to use for the calculation of the partial correlations. This can be either Pearson's, Spearman's or Kendall's coefficient. See cor for details.

Details

The calculation is done based on multiple linear regressions. The function calculates them for each pair of variables based on the residuals of linear models of those variables from the other variables in the dataset.

Value

The following list of values is returned:

- value - Matrix of the coefficients of partial correlations;
- p.value - The p-values for the parameters;
- method - The method used in the calculations.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

mcor, cramer, association

Examples

```
pcor(mtcars)
```

---

pinball *Pinball function*

Description

The function returns the value from the pinball function for the specified level and the type of loss

Usage

```
pinball(holdout, forecast, level, loss = 1, na.rm = TRUE)
```

Arguments

holdout	The vector or matrix of the holdout values.
forecast	The forecast of a distribution (e.g. quantile or expectile). It should be the same length as the holdout.
level	The level associated with the forecast (e.g. level of quantile).
loss	The type of loss to use. The number which corresponds to L1, L2 etc. L1 implies the loss for quantiles, while L2 is for the expectile.
na.rm	Logical, defining whether to remove the NAs from the provided data or not.

Value

The function returns the scalar value.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
# An example with mtcars data
ourModel <- alm(mpg~., mtcars[1:30,], distribution="dnorm")

# Produce predictions with the interval
ourForecast <- predict(ourModel, mtcars[-c(1:30),], interval="p")

# Pinball with the L1 (quantile value)
pinball(mtcars$mpg[-c(1:30)],ourForecast$upper,level=0.975,loss=1)
pinball(mtcars$mpg[-c(1:30)],ourForecast$lower,level=0.025,loss=1)

# Pinball with the L2 (expectile value)
pinball(mtcars$mpg[-c(1:30)],ourForecast$upper,level=0.975,loss=2)
pinball(mtcars$mpg[-c(1:30)],ourForecast$lower,level=0.025,loss=2)
```

---

plot.greybox *Plots of the fit and residuals*

Description

The function produces diagnostics plots for a greybox model

Usage

```
## S3 method for class 'greybox'
plot(x, which = c(1, 2, 4, 6), level = 0.95,
  legend = FALSE, ask = prod(par("mfcol")) < length(which) &&
  dev.interactive(), lowess = TRUE, ...)
```

Arguments

x	Estimated greybox model.
which	Which of the plots to produce. The possible options (see details for explanations):

1. Actuals vs Fitted values;
2. Standardised residuals vs Fitted;
3. Studentised residuals vs Fitted;
4. Absolute residuals vs Fitted;

5. Squared residuals vs Fitted;
6. Q-Q plot with the specified distribution;
7. Fitted over time;
8. Standardised residuals over observations;
9. Studentised residuals over observations;
10. ACF of the residuals;
11. PACF of the residuals;
12. Cook's distance over observations with 0.5, 0.75 and 0.95 quantile lines from Fisher's distribution;
13. Absolute standardised residuals vs Fitted;
14. Squared standardised residuals vs Fitted;
15. ACF of the squared residuals;
16. PACF of the squared residuals.

`level` Confidence level. Defines width of confidence interval. Used in plots (2), (3), (7), (8), (9), (10) and (11).

`legend` If `TRUE`, then the legend is produced on plots (2), (3) and (7).

`ask` Logical; if `TRUE`, the user is asked to press Enter before each plot.

`lowess` Logical; if `TRUE`, LOWESS lines are drawn on scatterplots, see lowess.

`...` The parameters passed to the plot functions. Recommended to use with separate plots.

Details

The list of produced plots includes:

1. Actuals vs Fitted values. Allows analysing, whether there are any issues in the fit. Does the variability of actuals increase with the increase of fitted values? Is the relation well captured? They grey line on the plot corresponds to the perfect fit of the model.
2. Standardised residuals vs Fitted. Plots the points and the confidence bounds (red lines) for the specified confidence `level`. Useful for the analysis of outliers;
3. Studentised residuals vs Fitted. This is similar to the previous plot, but with the residuals divided by the scales with the leave-one-out approach. Should be more sensitive to outliers;
4. Absolute residuals vs Fitted. Useful for the analysis of heteroscedasticity;
5. Squared residuals vs Fitted - similar to (3), but with squared values;
6. Q-Q plot with the specified distribution. Can be used in order to see if the residuals follow the assumed distribution. The type of distribution depends on the one used in the estimation (see `distribution` parameter in alm);
7. Fitted over time. Plots actuals (black line), fitted values (purple line) and prediction interval (red lines) of width `level`, but only in the case, when there are some values lying outside of it. Can be used in order to make sure that the model did not miss any important events over time;
8. Standardised residuals vs Time. Useful if you want to see, if there is autocorrelation or if there is heteroscedasticity in time. This also shows, when the outliers happen;

9. Studentised residuals vs Time. Similar to previous, but with studentised residuals;
10. ACF of the residuals. Are the residuals autocorrelated? See acf for details;
11. PACF of the residuals. No, really, are they autocorrelated? See pacf for details;
12. Cook's distance over time. Shows influential observations. 0.5, 0.75 and 0.95 quantile lines from Fisher's distribution are also plotted. If the value is above them then the observation is influencial. This does not work well for non-normal distributions;
13. Absolute standardised residuals vs Fitted. Similar to the previous, but with absolute values. This is more relevant to the models where scale is calculated as an absolute value of something (e.g. Laplace);
14. Squared standardised residuals vs Fitted. This is an additional plot needed to diagnose heteroscedasticity in a model with varying scale. The variance on this plot will be constant if the adequate model for `scale` was constructed. This is more appropriate for normal and the related distributions.

Which of the plots to produce, is specified via the `which` parameter. The plots 2, 3, 7, 8 and 9 also use the parameters `level`, which specifies the confidence level for the intervals.

Value

The function produces the number of plots, specified in the parameter `which`.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

plot.lm, rstandard, rstudent

Examples

```
xreg <- cbind(rlaplace(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rlaplace(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")

ourModel <- alm(y~x1+x2, xreg, distribution="dnorm")

par(mfcol=c(4,4))
plot(ourModel, c(1:14))
```

pointLik *Point likelihood values*

Description

This function returns a vector of logarithms of likelihoods for each observation

Usage

```
pointLik(object, log = TRUE, ...)
```

Arguments

object	Time series model.
log	Whether to take logarithm of likelihoods.
...	Some stuff.

Details

Instead of taking the expected log-likelihood for the whole series, this function calculates the individual value for each separate observation. Note that these values are biased, so you would possibly need to take number of degrees of freedom into account in order to have an unbiased estimator.

This value is based on the general likelihood (not its concentrated version), so the sum of these values may slightly differ from the output of logLik.

Value

This function returns a vector.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

AIC, BIC

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
ourModel <- alm(y~x1+x2,as.data.frame(xreg))

pointLik(ourModel)

# Bias correction
pointLik(ourModel) - nparam(ourModel)
```

```
# Bias correction in AIC style
2*(nparam(ourModel)/nobs(ourModel) - pointLik(ourModel))

# BIC calculation based on pointLik
log(nobs(ourModel))*nparam(ourModel) - 2*sum(pointLik(ourModel))
```

`polyprod` *This function calculates parameters for the polynomials*

Description

The function accepts two vectors with the parameters for the polynomials and returns the vector of parameters after their multiplication. This can be especially useful, when working with ARIMA models.

Usage

```
polyprod(x, y)
```

Arguments

`x`	The vector of parameters of the first polynomial.
`y`	The vector of parameters of the second polynomial.

Value

The function returns a matrix with one column with the parameters for the polynomial, starting from the 0-order.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

convolve

Examples

```
## Not run: polyprod(c(1,-2,-1),c(1,0.5,0.3))
```

predict.alm *Forecasting using greybox functions*

Description

The functions allow producing forecasts based on the provided model and newdata.

Usage

```
## S3 method for class 'alm'
predict(object, newdata = NULL, interval = c("none",
  "confidence", "prediction"), level = 0.95, side = c("both", "upper",
  "lower"), occurrence = NULL, ...)

## S3 method for class 'greybox'
predict(object, newdata = NULL, interval = c("none",
  "confidence", "prediction"), level = 0.95, side = c("both", "upper",
  "lower"), ...)

## S3 method for class 'scale'
predict(object, newdata = NULL, interval = c("none",
  "confidence", "prediction"), level = 0.95, side = c("both", "upper",
  "lower"), ...)

## S3 method for class 'greybox'
forecast(object, newdata = NULL, h = NULL, ...)

## S3 method for class 'alm'
forecast(object, newdata = NULL, h = NULL, ...)
```

Arguments

object	Time series model for which forecasts are required.
newdata	The new data needed in order to produce forecasts.
interval	Type of intervals to construct: either "confidence" or "prediction". Can be abbreviated
level	Confidence level. Defines width of prediction interval.
side	What type of interval to produce: "both" - produces both lower and upper bounds of the interval, "upper" - upper only, "lower" - respectively lower only. In the "both" case the probability is split into two parts: ((1-level)/2, (1+level)/2). When "upper" is specified, then the intervals for (0, level) are constructed Finally, with "lower" the interval for (1-level, 1) is returned.
occurrence	If occurrence was provided, then a user can provide a vector of future values via this variable.
...	Other arguments passed to vcov function (see coef.alm for details).
h	The forecast horizon.

Details

`predict` produces predictions for the provided model and `newdata`. If `newdata` is not provided, then the data from the model is extracted and the fitted values are reproduced. This might be useful when confidence / prediction intervals are needed for the in-sample values.

`forecast` function produces forecasts for `h` steps ahead. There are four scenarios in this function:

1. If the `newdata` is not provided, then it will produce forecasts of the explanatory variables to the horizon `h` (using `es` from smooth package or using Naive if `smooth` is not installed) and use them as `newdata`.
2. If `h` and `newdata` are provided, then the number of rows to use will be regulated by `h`.
3. If `h` is `NULL`, then it is set equal to the number of rows in `newdata`.
4. If both `h` and `newdata` are not provided, then it will use the data from the model itself, reproducing the fitted values.

After forming the `newdata` the `forecast` function calls for `predict`, so you can provide parameters `interval`, `level` and `side` in the call for `forecast`.

Value

`predict.greybox()` returns object of class "predict.greybox", which contains:

- `model` - the estimated model.
- `mean` - the expected values.
- `fitted` - fitted values of the model.
- `lower` - lower bound of prediction / confidence intervals.
- `upper` - upper bound of prediction / confidence intervals.
- `level` - confidence level.
- `newdata` - the data provided in the call to the function.
- `variances` - conditional variance for the holdout sample. In case of `interval="prediction"` includes variance of the error.

`predict.alm()` is based on `predict.greybox()` and returns object of class "predict.alm", which in addition contains:

- `location` - the location parameter of the distribution.
- `scale` - the scale parameter of the distribution.
- `distribution` - name of the fitted distribution.

`forecast()` functions return the same "predict.alm" and "predict.greybox" classes, with the same set of output variables.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

predict.lm

Examples

```
xreg <- cbind(rlaplace(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rlaplace(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
inSample <- xreg[1:80,]
outSample <- xreg[-c(1:80),]

ourModel <- alm(y~x1+x2, inSample, distribution="dlaplace")

predict(ourModel,outSample)
predict(ourModel,outSample,interval="c")

plot(predict(ourModel,outSample,interval="p"))
plot(forecast(ourModel,h=10,interval="p"))
```

---

rmcb *Regression for Multiple Comparison with the Best*

Description

RMCB stands for "Regression for Multiple Comparison with the Best", referring to the comparison of forecasting methods. This is a regression-based version of the Nemenyi / MCB test relies on the ranks of variables. This test is based on Nemenyi / MCB test (Demsar, 2006). It transforms the data into ranks and then constructs a regression on them of the type:

Usage

```
rmcb(data, level = 0.95, outplot = c("mcb", "lines", "none"),
  select = NULL, ...)

## S3 method for class 'rmcb'
plot(x, outplot = c("mcb", "lines"), select = NULL, ...)
```

Arguments

`data`	Matrix or data frame with observations in rows and variables in columns.
`level`	The width of the confidence interval. Default is 0.95.
`outplot`	What type of plot to use after the calculations. This can be either "MCB" (`"mcb"`), or "Vertical lines" (`"lines"`), or nothing (`"none"`). You can also use plot method on the produced object in order to get the same effect.
`select`	What column of data to highlight on the plot. If NULL, then the method with the lowest value is selected.
`...`	Other parameters passed to rank function. `distribution` parameter can also be passed here to alter the behaviour of the function.
`x`	The produced rmcb model.

Details

y = b' X + e,

where y is the vector of the ranks of provided data (as.vector(data)), X is the matrix of dummy variables for each column of the data (forecasting method), b is the vector of coefficients for the dummies and e is the error term of the model. Given that the data is ranked, it test the differences in medians between the methods and then produces plots based on that.

The critical distances by default are calculated based on the Studentised range statistics, so the test gives exactly the same result as the Nemenyi test. However, there are several options with the "distribution" parameter, which would fit a regression and extract standard errors from it to produce critical distances. e.g. `distribution="dnorm"` would rely on the Student's distribution and the covariance matrix of the parameters. On large datasets, this performs similar to the default option of `distribution="tukey"`. If any other distribution is used, the function would call `alm()` and fit a model on the original data with the assumed distribution (e.g. comparing the locations for each of the methods).

There is also a `plot()` method that allows producing either "mcb" or "lines" style of plot. This can be regulated via `plot(x, outplot="lines")`.

Value

If `outplot!="none"`, then the function plots the results after all the calculations using plot.rmcb() function.

Function returns a list of a class "rmcb", which contains the following variables:

- mean Mean values for each method.
- interval Confidence intervals for each method.
- vlines Coordinates used for outplot="l", marking the groups of methods.
- groups The table containing the groups. `TRUE` - methods are in the same group, `FALSE` - they are not.
- methods Similar to `group` parameter, but with a slightly different presentation.
- p.value p-value for the test of the significance of the model. This is the value from the F test of the linear regression.
- level Confidence level.
- model lm model produced for the calculation of the intervals.
- outplot Style of the plot to produce.
- select The selected variable to highlight.

Author(s)

Ivan Svetunkov, `<ivan@svetunkov.com>`

References

- Demsar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research, 7, 1-30. https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf

Examples

```
N <- 50
M <- 4
ourData <- matrix(rnorm(N*M,mean=0,sd=1), N, M)
ourData[,2] <- ourData[,2]+4
ourData[,3] <- ourData[,3]+3
ourData[,4] <- ourData[,4]+2
colnames(ourData) <- c("Method A","Method B","Method C - long name","Method D")
ourTest <- rmcb(ourData, level=0.95)

# See the mean ranks:
ourTest$mean
# The same is for the intervals:
ourTest$interval

# You can also reproduce plots in different styles:
plot(ourTest, outplot="lines")

# Or you can use the default "mcb" style and set additional parameters for the plot():
par(mar=c(2,2,4,0)+0.1)
plot(ourTest, main="Four methods")
```

---

ro *Rolling Origin*

Description

The function does rolling origin for any forecasting function

Usage

```
ro(data, h = 10, origins = 10, call, value = NULL, step = 1,
  ci = FALSE, co = TRUE, silent = TRUE, parallel = FALSE, ...)
```

Arguments

data	Data vector or ts object with the response variable passed to the function.
h	The forecasting horizon.
origins	The number of rolling origins.
call	The call that is passed to the function. The call must be in quotes. Example: "forecast(ets(data),h)". Here data shows where the data is and h defines where the horizon should be passed in the call. Some hidden parameters can also be specified in the call. For example, parameters counti, counto and countf are used in the inner loop and can be used for the regulation of exogenous variables sizes. See examples for the details.

value	The variable or set of variables returned by the call. For example, mean for functions of forecast package. This can also be a vector of variables. See examples for the details. If the parameter is NULL, then all the values from the call are returned (could be really messy!). Note that if your function returns a list with matrices, then ro will return an array. If your function returns a list, then you will have a list of lists in the end. So it makes sense to understand what you want to get before running the function.
step	Number of observations to skip between origins when moving to the next one. For example, could be useful if forecast should be produced every week on Sunday, in which case, for daily data, step=7.
ci	The parameter defines if the in-sample window size should be constant. If TRUE, then with each origin one observation is added at the end of series and another one is removed from the beginning.
co	The parameter defines whether the holdout sample window size should be constant. If TRUE, the rolling origin will stop when less than h observations are left in the holdout.
silent	If TRUE, nothing is printed out in the console.
parallel	If TRUE, then the model fitting is done in parallel. WARNING! Packages foreach and either doMC (Linux and Mac only) or doParallel are needed in order to run the function in parallel.
...	This is temporary and is needed in order to capture "silent" parameter if it is provided.

Details

This function produces rolling origin forecasts using the data and a call passed as parameters. The function can do all of that either in serial or in parallel, but it needs foreach and either doMC (Linux only) or doParallel packages installed in order to do the latter.

This is a dangerous function, so be careful with the call that you pass to it, and make sure that it is well formulated before the execution. Also, do not forget to provide the value that needs to be returned or you might end up with very messy results.

For more details and more examples of usage, please see vignette for the function. In order to do that, just run the command: vignette("ro","greybox")

Value

Function returns the following variables:

- actuals - the data provided to the function.
- holdout - the matrix of actual values corresponding to the produced forecasts from each origin.
- value - the matrices / array / lists with the produced data from each origin. Name of each object corresponds to the names in the parameter value.

Author(s)

Yves Sagaert

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Tashman, (2000) Out-of-sample tests of forecasting accuracy: an analysis and review International Journal of Forecasting, 16, pp. 437-450. doi:10.1016/S01692070(00)000650.

Examples

```
y <- rnorm(100,0,1)
ourCall <- "predict(arima(x=data,order=c(0,1,1)),n.ahead=h)"
# NOTE that the "data" needs to be used in the call, not "y".
# This way we tell the function, where "y" should be used in the call of the function.

# The default call and values
ourValue <- "pred"
ourRO <- ro(y, h=5, origins=5, ourCall, ourValue)

# We can now plot the results of this evaluation:
plot(ourRO)

# You can also use dollar sign
ourValue <- "$pred"
# And you can have constant in-sample size
ro(y, h=5, origins=5, ourCall, ourValue, ci=TRUE)

# You can ask for several values
ourValue <- c("pred","se")
# And you can have constant holdout size
ro(y, h=5, origins=20, ourCall, ourValue, ci=TRUE, co=TRUE)

#### The following code will give exactly the same result as above,
#### but computed in parallel using all but 1 core of CPU:
## Not run: ro(y, h=5, origins=20, ourCall, ourValue, ci=TRUE, co=TRUE, parallel=TRUE)

#### If you want to use functions from forecast package, please note that you need to
#### set the values that need to be returned explicitly. There are two options for this.
# Example 1:
## Not run: ourCall <- "forecast(ets(data), h=h, level=95)"
ourValue <- c("mean", "lower", "upper")
ro(y,h=5,origins=5,ourCall,ourValue)
## End(Not run)

# Example 2:
## Not run: ourCall <- "forecast(ets(data), h=h, level=c(80,95))"
ourValue <- c("mean", "lower[,1]", "upper[,1]", "lower[,2]", "upper[,2]")
ro(y,h=5,origins=5,ourCall,ourValue)
## End(Not run)

#### A more complicated example using the for loop and
#### several time series
x <- matrix(rnorm(120*3,0,1), 120, 3)

## Form an array for the forecasts we will produce
## We will have 4 origins with 6-steps ahead forecasts
```

```
ourForecasts <- array(NA,c(6,4,3))

## Define models that need to be used for each series
ourModels <- list(c(0,1,1), c(0,0,1), c(0,1,0))

## This call uses specific models for each time series
ourCall <- "predict(arima(data, order=ourModels[[i]]), n.ahead=h)"
ourValue <- "pred"

## Start the loop. The important thing here is to use the same variable 'i' as in ourCall.
for(i in 1:3){
    ourData <- x[,i]
    ourForecasts[,,i] <- ro(data=ourData,h=6,origins=4,call=ourCall,
                            value=ourValue,co=TRUE,silent=TRUE)$pred
}

## ourForecasts array now contains rolling origin forecasts from specific
## models.

##### An example with exogenous variables
x <- rnorm(100,0,1)
xreg <- matrix(rnorm(200,0,1),100,2,dimnames=list(NULL,c("x1","x2")))

## 'counti' is used to define in-sample size of xreg,
## 'counto' - the size of the holdout sample of xreg

ourCall <- "predict(arima(x=data, order=c(0,1,1), xreg=xreg[counti,,drop=FALSE]),
            n.ahead=h, newxreg=xreg[counto,,drop=FALSE])"
ourValue <- "pred"
ro(x,h=5,origins=5,ourCall,ourValue)

##### Poisson regression with alm
x <- rpois(100,2)
xreg <- cbind(x,matrix(rnorm(200,0,1),100,2,dimnames=list(NULL,c("x1","x2"))))
ourCall <- "predict(alm(x~., data=xreg[counti,,drop=FALSE], distribution='dpois'),
                    newdata=xreg[counto,,drop=FALSE])"
ourValue <- "mean"
testRO <- ro(xreg[,1],h=5,origins=5,ourCall,ourValue,co=TRUE)
plot(testRO)

## 'countf' is used to take xreg of the size corresponding to the whole
## sample on each iteration
## This is useful when working with functions from smooth package.
## The following call will return the forecasts from es() function of smooth.
## Not run: ourCall <- "es(data=data, h=h, xreg=xreg[countf,,drop=FALSE])"
ourValue <- "forecast"
ro(x,h=5,origins=5,ourCall,ourValue)
## End(Not run)
```

---

sm *Scale Model*

Description

This method produces a model for scale of distribution for the provided pre-estimated model. The model can be estimated either via `lm` or `alm`.

Usage

```
sm(object, ...)

## Default S3 method:
sm(object, formula = NULL, data = NULL,
  parameters = NULL, ...)

## S3 method for class 'lm'
sm(object, formula = NULL, data = NULL, parameters = NULL,
  ...)

## S3 method for class 'alm'
sm(object, formula = NULL, data = NULL, parameters = NULL,
  ...)
```

Arguments

`object`	The pre-estimated `alm` or `lm` model.
`...`	Other parameters to pass to the method, including those explained in alm (e.g. parameters for optimiser).
`formula`	The formula for scale. It should start with ~ and contain all variables that should impact the scale.
`data`	The data, on which the scale model needs to be estimated. If not provided, then the one used in the `object` is used.
`parameters`	The parameters to use in the model. Only needed if you know the parameters in advance or want to test yours.

Details

This function is useful, when you suspect a heteroscedasticity in your model and want to fit a model for the scale of the pre-specified distribution. This function is complementary for `lm` or `alm`.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

Examples

```
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+sqrt(exp(0.8+0.2*xreg[,1]))*rnorm(100,0,1),
              xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
```

```
# Estimate the location model
ourModel <- alm(y~.,xreg)
# Estimate the scale model
ourScale <- sm(ourModel,formula=~x1+x2)
# Summary of the scale model
summary(ourScale)
```

---

spread *Construct scatterplot / boxplots for the data*

Description

Function constructs the plots depending on the types of variables in the provided matrix / data frame.

Usage

```
spread(data, histograms = FALSE, log = FALSE, lowess = FALSE, ...)
```

Arguments

data	Either matrix or data frame with the data.
histograms	If TRUE, then the histograms and barplots are produced on the diagonal of the matrix. Otherwise the names of the variables are written there.
log	If TRUE, then the logarithms of all numerical variables are taken.
lowess	If TRUE, then LOWESS lines are added to scatterplots and means are connected with lines on boxplots, see lowess for details.
...	Other parameters passed to the plot function. Currently only "main" parameter is accepted.

Details

If both variables are in metric scale, then the classical scatterplot is constructed. If one of them is either integer (up to 10 values) or categorical (aka 'factor'), then boxplots (with grey dots corresponding to mean values) are constructed. Finally, for the two categorical variables the tableplot is returned (see tableplot function for the details). All of this is packed in a matrix. The colours in the plot can be changed by defining a different palette via the `palette()` function, in which case spread() will use the first four colours in the pallete.

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

Value

Function does not return anything. It just plots things.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

plot, table, tableplot

Examples

```
### Simple example
spread(mtcars)
spread(mtcars,log=TRUE)
```

`stepwise` *Stepwise selection of regressors*

Description

Function selects variables that give linear regression with the lowest information criteria. The selection is done stepwise (forward) based on partial correlations. This should be a simpler and faster implementation than step() function from 'stats' package.

Usage

```
stepwise(data, ic = c("AICc", "AIC", "BIC", "BICc"), silent = TRUE,
  df = NULL, formula = NULL, subset = NULL, method = c("pearson",
  "kendall", "spearman"), distribution = c("dnorm", "dlaplace", "ds",
  "dgnorm", "dlogis", "dt", "dalaplace", "dlnorm", "dllaplace", "dls",
  "dlgnorm", "dbcnorm", "dinvgauss", "dgamma", "dexp", "dfnorm", "drectnorm",
  "dpois", "dnbinom", "dbeta", "dlogitnorm", "plogis", "pnorm"),
  occurrence = c("none", "plogis", "pnorm"), ...)
```

Arguments

`data`	Data frame containing dependant variable in the first column and the others in the rest.
`ic`	Information criterion to use.
`silent`	If `silent=FALSE`, then nothing is silent, everything is printed out. `silent=TRUE` means that nothing is produced.
`df`	Number of degrees of freedom to add (should be used if stepwise is used on residuals).
`formula`	If provided, then the selection will be done from the listed variables in the formula after all the necessary transformations.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`method`	Method of correlations calculation. The default is Pearson's correlation, which should be applicable to a wide range of data in different scales.
`distribution`	Distribution to pass to `alm()`. See alm for details.

occurrence	what distribution to use for occurrence part. See alm for details.
...	This is temporary and is needed in order to capture "silent" parameter if it is provided.

Details

The algorithm uses alm() to fit different models and cor() to select the next regressor in the sequence.

Some details and examples of application are also given in the vignette "Greybox": `vignette("greybox","greybox")`

Value

Function returns `model` - the final model of the class "alm". See alm for details of the output.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

References

- Burnham Kenneth P. and Anderson David R. (2002). Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach. Springer-Verlag New York. DOI: [10.1007/b97636](http://dx.doi.org/10.1007/b97636).
- McQuarrie, A. D. (1999). A small-sample correction for the Schwarz SIC model selection criterion. Statistics & Probability Letters, 44(1), 79–86. [10.1016/S0167-7152(98)00294-6](https://doi.org/10.1016/S0167-7152(98)00294-6).

See Also

step, xregExpander, lmCombine

Examples

```
### Simple example
xreg <- cbind(rnorm(100,10,3),rnorm(100,50,5))
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y","x1","x2","Noise")
stepwise(xreg)

### Mixture distribution of Log Normal and Cumulative Logit
xreg[,1] <- xreg[,1] * round(exp(xreg[,1]-70) / (1 + exp(xreg[,1]-70)),0)
colnames(xreg) <- c("y","x1","x2","Noise")
ourModel <- stepwise(xreg, distribution="dlnorm",
                     occurrence=stepwise(xreg, distribution="plogis"))
summary(ourModel)

### Fat regression example
xreg <- matrix(rnorm(20000,10,3),100,200)
xreg <- cbind(100+0.5*xreg[,1]-0.75*xreg[,2]+rnorm(100,0,3),xreg,rnorm(100,300,10))
colnames(xreg) <- c("y",paste0("x",c(1:200)),"Noise")
ourModel <- stepwise(xreg,ic="AICc")
```

```
plot(ourModel$ICs,type="l",ylim=range(min(ourModel$ICs),max(ourModel$ICs)+5))
points(ourModel$ICs)
text(c(1:length(ourModel$ICs))+0.1,ourModel$ICs+5,names(ourModel$ICs))
```

`tableplot` *Construct a plot for categorical variable*

Description

Function constructs a plot for two categorical variables based on table function

Usage

```
tableplot(x, y = NULL, labels = TRUE, legend = FALSE, points = TRUE,
  ...)
```

Arguments

`x`	First categorical variable. Can be either vector, factor, matrix or a data frame. If `y` is NULL and `x` is either matrix of a data frame, then the first two variables of the data will be plotted against each other.
`y`	Second categorical variable. If not provided, then only `x` will be plotted.
`labels`	Whether to print table labels inside the plot or not.
`legend`	If `TRUE`, then the legend for the tableplot is drawn. The plot is then produced on a separate canvas (new `par()`).
`points`	Whether to plot points in the areas. They help in understanding how many values lie in specific categories.
`...`	Other parameters passed to the plot function.

Details

The function produces the plot of the `table()` function with colour densities corresponding to the respective frequencies of appearance. If the value appears more often than the other (e.g. 0.5 vs 0.15), then it will be darker. By default, the frequency of 0 corresponds to the white colour, the frequency of 1 corresponds to the black. The colours can be changed by defining a different palette via the `palette()` function. In that case only two first colours are used, where the colour intensity changes from the first one to the second one. The function also adds the number of dots (points) proportional to the number of values in each category to simplify the reading of plots.

See details in the vignette "Marketing analytics with greybox": `vignette("maUsingGreybox","greybox")`

Value

Function does not return anything. It just plots things.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

plot, table, spread

Examples

```
palette("Tableau")
tableplot(mtcars$am, mtcars$gear)
```

---

temporaldummy *Dummy variables for provided seasonality type*

Description

Function generates the matrix of dummy variables for the months / weeks / days / hours / minutes / seconds of year / month / week / day / hour / minute.

Usage

```
temporaldummy(object, ...)

## Default S3 method:
temporaldummy(object, type = c("month", "quarter", "week",
  "day", "hour", "halfhour", "minute", "second"), of = c("year", "quarter",
  "month", "week", "day", "hour", "minute"), factors = FALSE, h = 0, ...)

## S3 method for class 'ts'
temporaldummy(object, type = c("month", "quarter", "week",
  "day", "hour", "halfhour", "minute", "second"), of = c("year", "quarter",
  "month", "week", "day", "hour", "minute"), factors = FALSE, h = 0, ...)

## S3 method for class 'Date'
temporaldummy(object, type = c("month", "quarter", "week",
  "day", "hour", "halfhour", "minute", "second"), of = c("year", "quarter",
  "month", "week", "day", "hour", "minute"), factors = FALSE, h = 0, ...)

## S3 method for class 'POSIXt'
temporaldummy(object, type = c("month", "quarter", "week",
  "day", "hour", "halfhour", "minute", "second"), of = c("year", "quarter",
  "month", "week", "day", "hour", "minute"), factors = FALSE, h = 0, ...)

## S3 method for class 'zoo'
temporaldummy(object, type = c("month", "quarter", "week",
```

```
"day", "hour", "halfhour", "minute", "second"), of = c("year", "quarter",
"month", "week", "day", "hour", "minute"), factors = FALSE, h = 0, ...)
```

Arguments

object	Either a ts / msts / zoo / xts / tsibble object or a vector of dates.
...	Other parameters.
type	Specifies what type of frequency to produce. For example, if type="month", then the matrix with dummies for months of the year will be created.
of	Specifies the frequency of what is needed. Used together with type e.g. type="day" and of="month" will produce a matrix with dummies for days of month (31 dummies).
factors	If TRUE, the function will return the categorical variable instead of the matrix with dummies.
h	If not NULL, then the function will produce dummies for this set of observations ahead as well, binding them to the original matrix.

Details

The function extracts dates from the provided object and returns a matrix with dummy variables for the specified frequency type, with the number of rows equal to the length of the object + the specified horizon. If a numeric vector is provided then it will produce dummies based on typical values (e.g. 30 days in month). So it is recommended to use proper classes with this method.

Several notes on how the dummies are calculated in some special cases:

- In case of weeks of years, the first week is defined according to ISO 8601.

Note that not all the combinations of type and of are supported. For example, there is no such thing as dummies for months of week. Also note that some combinations are not very useful and would take a lot of memory (e.g. minutes of year).

The function will return all the dummy variables. If you want to avoid the dummy variables trap, you will need to exclude one of them manually.

If you want to have a different type of dummy variables, let me know, I will implement it.

Value

One of the two is returned, depending on the value of factors variable:

- factors=FALSE: Class "dgCMatrix" with all the dummy variables is returned in case of numeric variable. Feel free to drop one (making it a reference variable) or convert the object into matrix (this will consume more memory than the returned class). In other cases the object of the same class as the provided is returned.
- factors=TRUE: The categorical variable (factor) containing specific values for each observation.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

xregExpander, xregMultiplier,outlierdummy

Examples

```
# Generate matrix with dummies for a ts object
x <- ts(rnorm(100,100,1),frequency=12)
temporaldummy(x)

# Generate matrix with monthly dummies for a zoo object
x <- as.Date("2003-01-01")+0:99
temporaldummy(x, type="month", of="year", h=10)
```

---

xregExpander *Exogenous variables expander*

Description

Function expands the provided matrix or vector of variables, producing values with lags and leads specified by `lags` variable.

Usage

```
xregExpander(xreg, lags = c(-frequency(xreg):frequency(xreg)),
  silent = TRUE, gaps = c("auto", "NAs", "zero", "naive", "extrapolate"))
```

Arguments

`xreg`	Vector / matrix / data.frame, containing variables that need to be expanded. In case of vector / matrix it is recommended to provide `ts` object, so the frequency of the data is taken into account.
`lags`	Vector of lags / leads that we need to have. Negative values mean lags, positive ones mean leads.
`silent`	If `silent=FALSE`, then the progress is printed out. Otherwise the function won't print anything in the console.
`gaps`	Defines how to fill in the gaps in the data. `"NAs"` will leave missing values, `"zero"` will substitute them by zeroes, `"naive"` will use the last / the first actual value, while `"extrapolate"` will use es function from smooth package (if present, otherwise - naive) in order to fill in values. Finally, `"auto"` will let the function select between `"extrapolate"` and `"naive"` depending on the length of series.

Details

This function could be handy when you want to check if lags and leads of a variable influence the dependent variable. Can be used together with `xregDo="select"` in adam, es, ces and ssarima. All the missing values in the beginning and at the end of lagged series are substituted by mean forecasts produced using adam.

Value

ts matrix with the expanded variables is returned.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

es, stepwise

Examples

```
# Create matrix of two variables, make it ts object and expand it
x <- cbind(rnorm(100,100,1),rnorm(100,50,3))
x <- ts(x,frequency=12)
xregExpander(x)
```

---

xregMultiplier *Exogenous variables cross-products*

Description

Function generates the cross-products of the provided exogenous variables.

Usage

```
xregMultiplier(xreg, silent = TRUE)
```

Arguments

xreg	matrix or data.frame, containing variables that need to be expanded. This matrix needs to contain at least two columns.
silent	If silent=FALSE, then the progress is printed out. Otherwise the function won't print anything in the console.

Details

This function might be useful if you have several variables and want to introduce their cross-products. This might be useful when introducing the interactions between dummy and continuous variables.

Value

ts matrix with the transformed and the original variables is returned.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

es, stepwise,xregExpander, xregTransformer

Examples

```
# Create matrix of two variables and expand it
x <- cbind(rnorm(100,100,1),rnorm(100,50,3))
xregMultiplier(x)
```

---

xregTransformer *Exogenous variables transformer*

Description

Function transforms each variable in the provided matrix or vector, producing non-linear values, depending on the selected pool of functions.

Usage

```
xregTransformer(xreg, functions = c("log", "exp", "inv", "sqrt", "square"),
  silent = TRUE)
```

Arguments

xreg	Vector / matrix / data.frame, containing variables that need to be expanded. In case of vector / matrix it is recommended to provide ts object, so the frequency of the data is taken into account.
functions	Vector of names for functions used.
silent	If silent=FALSE, then the progress is printed out. Otherwise the function won't print anything in the console.

Details

This function could be useful when you want to automatically select the necessary transformations of the variables. This can be used together with xregDo="select" in es, ces, gum and ssarima. However, this might be dangerous, as it might lead to the overfitting the data. So be reasonable when you produce the transformed variables.

Value

ts matrix with the transformed and the original variables is returned.

Author(s)

Ivan Svetunkov, <ivan@svetunkov.com>

See Also

es, stepwise,xregExpander

Examples

```
# Create matrix of two variables and expand it
x <- cbind(rnorm(100,100,1),rnorm(100,50,3))
xregTransformer(x)
```

Index