Package 'restriktor'

March 23, 2023

Title Restricted Statistical Estimation and Inference for Linear Models

Version 0.4-500

Description Allow for easy-to-use testing or evaluating of linear equality and inequality restrictions about parameters and effects in (generalized) linear statistical models.

Depends R(>= 4.0.0)

Imports boot, ic.infer, lavaan(>= 0.6-10), MASS, mvtnorm, quadprog, norm2, ggplot2

License GPL (>= 2)

LazyData yes

URL https://restriktor.org

NeedsCompilation no

Author Leonard Vanbrabant [aut, cre],
Yves Rosseel [ctb],
Rebecca Kuiper [aut],
Aleksandra Dacko [ctb]

Maintainer Leonard Vanbrabant <info@restriktor.org>

Suggests

Repository CRAN

Date/Publication 2023-03-23 09:20:02 UTC

R topics documented:

`restriktor-package` *Package for equality and inequality restricted estimation, model selection and hypothesis testing*

Description

Package `restriktor` implements estimation, testing and evaluating of linear equality and inequality restriktions about parameters and effects for univariate and multivariate normal models and generalized linear models.

Details

Package:	restriktor
Type:	Package
Version:	0.4-500
Date:	2023-03-22
License:	GPL (>=2)
LazyLoad:	yes

Function `restriktor` estimates the parameters of an univariate and multivariate linear model (`lm`), robust estimation of the linear model (`rlm`) or a generalized linear model (`glm`) subject to linear equality and/or inequality restriktions. The real work horses are the `conLM`, `conMLM`, the `conRLM`, and the `conGLM` functions. A major advantage of **restriktor** is that the constraints can be specified by a text-based description. This means that users do not have to specify the complex constraint matrix (comparable with a contrast matrix) themselves.

The function `restriktor` offers the possibility to compute (model robust) standard errors under the restriktions. The parameter estimates can also be bootstrapped, where bootstrapped standard errors and confidence intervals are available via the summary function. Moreover, the function computes

the Generalized Order-restricted Information Criterion (GORIC), which is a modification of the AIC and a generalization of the ORIC.

The function `iht` (alias `conTest`) conducts restricted hypothesis tests. F, Wald/LRT and score test-statistics are available. The null-distribution of these test-statistics takes the form of a mixture of F-distributions. The mixing weights (a.k.a. chi-bar-square weights or level probabilities) can be computed using the multivariate normal distribution function with additional Monte Carlo steps or via a simulation approach. Bootstrap methods are available to calculate the mixing weights and to compute the p-value directly. Parameters estimates under the null- and alternative-hypothesis are available from the summary function.

The function `goric` (generalized order-restricted information criterion) computes GORIC values, weights and relative-weights or GORICA (generalized order-restricted information crittion approximation) values, weights and relative weights. The GORIC(A) values are comparable to the AIC values. The function offers the possibility to evaluate an order-restricted hypothesis against its complement, the unconstrained hypothesis or against a set of hypotheses. For now, only one order-restricted hypothesis can be evaluated against its complement but work is in progress to evaluate a set of order-restricted hypothesis against its complement.

The package makes use of various other R packages: **quadprog** is used for restricted estimation, **boot** for bootstrapping, **ic.infer** for computing the mixing weights based on the multivariate normal distribution, **lavaan** for parsing the constraint syntax.

Value

The output of function `restriktor` belongs to S3 class `conLM`, `conMLM`, `conRLM` or `conGLM`.

The output of function `conTest` belongs to S3 class `conTest`.

These classes offer print and summary methods.

Acknowledgements

This package uses as an internal function the function `nchoosek` from **ic.infer**, which is originally from **vsn**, authored by Wolfgang Huber, available under LGPL.

The output style of the `iht` print function is strongly inspired on the summary of the `ic.test` function from the **ic.infer** package.

Author(s)

Leonard Vanbrabant and Yves Rosseel - Ghent University

References

Groemping, U. (2010). Inference With Linear Equality And Inequality Constraints Using R: The Package ic.infer. *Journal of Statistical Software*, Forthcoming.

Kuiper R.M., Hoijtink H., Silvapulle M.J. (2011). An Akaike-type Information Criterion for Model Selection Under Inequality Constraints. *Biometrika*, **98**, 495–501.

Kuiper R.M., Hoijtink H., Silvapulle M.J. (2012). Generalization of the Order-Restricted Information Criterion for Multivariate Normal Linear Models. *Journal of Statistical Planning and Inference*, **142**, 2454–2463. doi:10.1016/j.jspi.2012.03.007.

Robertson T, Wright F, Dykstra R (1988). *Order-Restricted Inference*. Wiley, New York.

Schoenberg, R. (1997). Constrained Maximum Likelihood. *Computational Economics*, **10**, 251–266.

Shapiro, A. (1988). Towards a unified theory of inequality-constrained testing in multivariate analysis. *International Statistical Review* **56**, 49–62.

Silvapulle, M. (1992a). Robust tests of inequality constraints and one-sided hypotheses in the linear model. *Biometrika*, **79**, 621–630.

Silvapulle, M. (1992b). Robust wald-type tests of one-sided hypotheses in the linear model. *Journal of the American Statistical Association*, **87**, 156–161.

Silvapulle, M. (1996). Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

Vanbrabant, L., Van Loey, N., and Kuiper, R.M. (2020). Evaluating a theory-based hypothesis against its complement using an AIC-type information criterion with an application to facial burn injury. *Psychological methods*, **25(2)**, 129-142. https://doi.org/10.1037/met0000238.

See Also

See also `restriktor`, `iht`, packages **boot**, **goric**, **ic.infer**, **mvtnorm**, and **quadprog**.

Examples

```
## Data preparation
## Ages (in months) at which an infant starts to walk alone.
DATA <- ZelazoKolb1972
DATA <- subset(DATA, Group != "Control")

## unrestricted linear model
fit.lm <- lm(Age ~ -1 + Group, data = DATA)
summary(fit.lm)

## restricted linear model with restriktions that the walking
## exercises would not have a negative effect of increasing the
## mean age at which a child starts to walk.

myConstraints <- ' GroupActive  < GroupPassive;
                   GroupPassive < GroupNo '

fit.con <- restriktor(fit.lm, constraints = myConstraints)
summary(fit.con)
```

`AngerManagement` *Reduction of aggression levels Dataset (4 treatment groups)*

Description

The anger management dataset consists of reduction of aggression levels between week 1 (intake) and week 8 (end of training) from four different treatment groups (No-exercises, Physical-exercises, Behavioral-exercises, combination of physical and behavioral exercises).

Usage

```
data(AngerManagement)
```

Format

A data frame of 40 observations of 4 treatment variables and covariate age.

`Anger` reduction in aggression levels

`Group` No, Physical, Behavioral, Both

`Age` persons' age

References

Hoijtink, H. Informative Hypotheses: Theory and Practice for Behavioral and Social Scientists Boca Raton, FL: Taylor & Francis, 2012.

Examples

```
head(AngerManagement)
```

`bootstrapD` *Bootstrapping a Lavaan Model*

Description

Bootstrap the D statistic.

Usage

```
bootstrapD(h0 = NULL, h1 = NULL, constraints, type = "A",
           bootstrap.type = "bollen.stine", R = 1000L,
           return.D = FALSE, double.bootstrap = "no",
           double.bootstrap.R = 500L, double.bootstrap.alpha = 0.05,
           verbose = FALSE, warn = -1L,
           parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL,
           seed = NULL)

## S3 method for class 'conTestLavaan'
print(x, digits = max(3, getOption("digits") - 2), ...)
```

Arguments

h0	An object of class lavaan. The restricted model.
h1	An object of class lavaan. The unrestricted model.
x	an object of class conTestLavaan.
constraints	The imposed (in)equality constraints on the model.
type	hypothesis test type "A", "B".
bootstrap.type	If "parametric", the parametric bootstrap is used. If "bollen.stine", the semi-nonparametric Bollen-Stine bootstrap is used. The default is set to "bollen.stine".
R	Integer. The number of bootstrap draws.
return.D	Logical; if TRUE, the function returns bootstrapped D-values.
double.bootstrap	If "standard" (default) the genuine double bootstrap is used to compute an additional set of plug-in p-values for each bootstrap sample. If "no", no double bootstrap is used. If "FDB", the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. Note that the "FDB" is experimental and should not be used by inexperienced users.
double.bootstrap.R	Integer; number of double bootstrap draws. The default value is set to 249.
double.bootstrap.alpha	The significance level to compute the adjusted alpha based on the plugin p-values. Only used if double.bootstrap = "standard". The default value is set to 0.05.
verbose	If TRUE, show information for each bootstrap draw.
warn	Sets the handling of warning messages. See options.
parallel	The type of parallel operation to be used (if any). If missing, the default is "no".
ncpus	Integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	An optional **parallel** or **snow** cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the bootstrapLavaan or bootstrapLRT call.
digits	the number of significant digits to use when printing.
...	no additional arguments for now.
seed	An integer to set the seed. Or NULL if no reproducible seeds are needed.

Author(s)

Leonard Vanbrabant

References

Bollen, K. and Stine, R. (1992) Bootstrapping Goodness of Fit Measures in Structural Equation Models. Sociological Methods and Research, 21, 205–229.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

Yuan, K.-H., Hayashi, K., and Yanagihara, H. (2007). A class of population covariance matrices in the bootstrap approach to covariance structure analysis. Multivariate Behavioral Research, 42, 261–281.

Examples

```
## Not run:
   ########################
   ### real data example ###
   ########################
   # Multiple group path model for facial burns example.

   # model syntax with starting values.
   burns.model <- 'Selfesteem ~ Age + c(m1, f1)*TBSA + HADS +
                               start(-.10, -.20)*TBSA
                HADS ~ Age + c(m2, f2)*TBSA + RUM +
                       start(.10, .20)*TBSA '


   # constraints syntax
   burns.constraints <- 'f2 > 0  ; m1 < 0
                          m2 > 0  ; f1 < 0
                          f2 > m2 ; f1 < m1'

   # we only generate 2 bootstrap samples in this example; in practice
   # you may wish to use a much higher number.
   # the double bootstrap was switched off; in practice you probably
   # want to set it to "standard".
   example1 <- conTestD(model = burns.model, data = FacialBurns,
                        R = 2, constraints = burns.constraints,
                        double.bootstrap = "no", group = "Sex")

   example1

   ##########################
   ### artificial example ###
   ##########################
   # Simple ANOVA model with 3 groups (N = 20 per group)
   set.seed(1234)
   Y <- cbind(c(rnorm(20,0,1), rnorm(20,0.5,1), rnorm(20,1,1)))
   grp <- c(rep("1", 20), rep("2", 20), rep("3", 20))
   Data <- data.frame(Y, grp)

   #create model matrix
   fit.lm <- lm(Y ~ grp, data = Data)
   mfit <- fit.lm$model
   mm <- model.matrix(mfit)

   Y <- model.response(mfit)
   X <- data.frame(mm[,2:3])
   names(X) <- c("d1", "d2")
   Data.new <- data.frame(Y, X)
```

```
    # model
    model <- 'Y ~ 1 + a1*d1 + a2*d2'

    # fit without constraints
    fit <- lavaan::sem(model, data = Data.new)

    # constraints syntax: mu1 < mu2 < mu3
    constraints <- ' a1 > 0
                     a1 < a2 '

    # we only generate 10 bootstrap samples in this example; in practice
    # you may wish to use a much higher number, say > 1000. The double
    # bootstrap is not necessary in case of an univariate ANOVA model.
    example2 <- conTestD(model = model, data = Data.new,
                         start = lavaan::parTable(fit),
                         R = 10L, double.bootstrap = "no",
                         constraints = constraints)
    example2

## End(Not run)
```

`Burns` *Relation between the response variable PTSS and gender, age, TBSA, guilt and anger.*

Description

Simulated dataset based on the original model parameters. The original data are based on two cohort studies in children from 0 to 4 and 8 to 18 years old with burns and their mother.

Usage

```
data(Burns)
```

Format

A data frame of 278 observations of 4 variables.

`PTSS` post-traumatic stress symptoms

`gender` gender

`age` age in years

`TBSA` estimated percentage total body surface area affected by second and third degree burns

`guilt` parental guilt feelings in relation to the burn event

`anger` parental anger feelings in relation to the burn event

References

Bakker A, Van der Heijden PG, Van Son MJ, Van Loey NE. Course of traumatic stress reactions in couples after a burn event to their young child. Health Psychology 2013; 10(32):1076-1083, doi:10.1037/a0033983.

Egberts MR, van de Schoot R, Boekelaar A, Hendrickx H, Geenen R, NEE V. Child and adolescent internalizing and externalizing problems 12 months postburn: the potential role of preburn functioning, parental posttraumatic stress, and informant bias. Child and Adolescent Psychiatry 2016; 25:791-803.

Examples

```
head(Burns)
```

`conTestC` *one-sided t-test for iht*

Description

`conTestC` tests linear inequality restricted hypotheses for (robust) linear models by a one-sided t-test. This method is based on the union-intersection principle. It is called by the `conTest` function if all restrictions are equalities. For more information see details.

Usage

```
## S3 method for class 'restriktor'
conTestC(object, ...)
```

Arguments

`object`	an object of class `restriktor`.
`...`	no additional arguments for now.

Details

Hypothesis test Type C:

- Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle. Note that, this test only makes sense in case of no equality constraints.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

`CON`	a list with useful information about the constraints.
`Amat`	constraints matrix.
`bvec`	vector of right-hand side elements.
`meq`	number of equality constraints.
`test`	same as input.
`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`Sigma`	variance-covariance matrix of unrestricted model.
`R2.org`	unrestricted R-squared.
`R2.reduced`	restricted R-squared.
`boot`	"no", not used (yet).
`model.org`	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M.J. and Sen, P.K. (2005, chapter 5.). *Constrained Statistical Inference*. Wiley, New York

See Also

quadprog, `iht`

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
```

```
# the corresponding regression parameters.
coef(fit1.lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.lm, myConstraints1, type = "C")


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit.restr1, type = "C")

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  fit1.con <- restriktor(fit1.lm, constraints = Amat1,
                         rhs = myRhs1, neq = myNeq1)
  iht(fit1.con, type = "C")

## End(Not run)
```

---

conTestF *F-bar test for iht*

Description

conTestF tests linear equality and/or inequality restricted hypotheses for linear models by F-tests. It can be used directly and is called by the conTest function if test = "F".

Usage

```
## S3 method for class 'conLM'
conTestF(object, type = "A", neq.alt = 0,
         boot = "no", R = 9999, p.distr = rnorm,
         parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
         verbose = FALSE, control = NULL, ...)

## S3 method for class 'conRLM'
```

```
conTestF(object, type = "A", neq.alt = 0,
         boot = "no", R = 9999, p.distr = rnorm,
         parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
         verbose = FALSE, control = NULL, ...)

## S3 method for class 'conGLM'
conTestF(object, type = "A", neq.alt = 0,
         boot = "no", R = 9999, p.distr = rnorm,
         parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
         verbose = FALSE, control = NULL, ...)
```

Arguments

object	an object of class conLM, conRLM or conGLM.
type	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
neq.alt	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
boot	the null-distribution of these test-statistics (except under type "C") takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if "parametric", the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See p.distr for other distributional options. If "model.based", a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If "no", the p-value is computed based on the weights obtained via simulation (mix.weights = "boot") or using the multivariate normal distribution function (mix.weights = "pmvnorm"). Note that, these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class conRLM, namely for computing the p-value for the robust test = "Wald" and the robust "score". In these cases the weights need to be recalculated.
R	integer; number of bootstrap draws for boot. The default value is set to 9999.
p.distr	random generation distribution for the parametric bootstrap. For all available distributions see ?distributions. For example, if rnorm, samples are drawn from the normal distribution (default) with mean zero and variance one. If rt, samples are drawn from a t-distribution. If rchisq, samples are drawn from a chi-square distribution. The distributional parameters will be passed in via ....
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
seed	seed value. The default value is set to 1234.

verbose	logical; if TRUE, information is shown at each bootstrap draw.
control	a list of control arguments:

- absval tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm.
- maxit the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported).
- tol numerical tolerance value. Estimates smaller than tol are set to 0.

...	additional arguments to be passed to the p.distr function.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.
- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).
- Type C: Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle (Silvapulle and Sen, 2005, chp 5.3). Note that, this test only makes sense in case of no equality constraints.
- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

The exact finite sample distributions of the non-robust F-, score- and LR-test statistics based on restricted OLS estimates and normally distributed errors, are a mixture of F-distributions under the null hypothesis (Wolak, 1987). In agreement with Silvapulle (1992), we found that the results based on these mixtures of F-distributions approximate the tail probabilities of the robust tests better than their asymptotic distributions. Therefore, all p-values for hypothesis test Type "A", "B" and "global" are computed based on mixtures of F-distributions.

Note that, in case of equality constraints only, the null-distribution of the (robust) F-test statistics is based on an F-distribution. The (robust) Wald- and (robust) score-test statistics are based on chi-square distributions.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

CON	a list with useful information about the constraints.
Amat	constraints matrix.
bvec	vector of right-hand side elements.
meq	number of equality constraints.
meq.alt	same as input neq.alt.

iact	number of active constraints.
type	same as input.
test	same as input.
Ts	test-statistic value.
df.residual	the residual degrees of freedom.
pvalue	tail probability for Ts.
b.eqrestr	equality restricted regression coefficients. Only available for type = "A" and type = "global", else b.eqrestr = NULL.
b.unrestr	unrestricted regression coefficients.
b.restr	restricted regression coefficients.
b.restr.alt	restricted regression coefficients under HA if some equality constraints are maintained. Only available for type = "B" else b.restr.alt = NULL.
Sigma	variance-covariance matrix of unrestricted model.
R2.org	unrestricted R-squared, not available for objects of class conGLM.
R2.reduced	restricted R-squared, not available for objects of class conGLM.
boot	same as input.
model.org	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Kudo, A. (1963) A multivariate analogue of the one-sided test. *Biometrika*, **50**, 403–418.

Silvapulle, M. (1992a). Robust tests of inequality constraints and one-sided hypotheses in the linear model. *Biometrika*, **79**, 621–630.

Silvapulle, M. (1996) On an F-type statistic for testing one-sided hypotheses and computation of chi-bar-squared weights. *Statistics and probability letters*, **28**, 137–141.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

Wolak, F. (1987). An exact test for multiple inequality and equality constraints in the linear regression model. *Journal of the American statistical association*, **82**, 782–793.

See Also

quadprog, iht

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
```

```
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.lm, myConstraints1)


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit.restr1)

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  iht(fit1.lm, constraints = Amat1,
      rhs = myRhs1, neq = myNeq1)

## End(Not run)

#########################
## Artificial examples ##
#########################
# generate data
n <- 10
means <- c(1,2,1,3)
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)

# fit unrestricted linear model
fit2.lm <- lm(y ~ -1 + group, data = DATA2)
coef(fit2.lm)

## example 2: increasing means
myConstraints2 <- ' group1 < group2
                    group2 < group3
```

```
                    group3 < group4 '

# compute F-test for hypothesis test Type A and compute the tail
# probability based on the parametric bootstrap. We only generate 9
# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
iht(fit2.lm, constraints = myConstraints2, type = "A",
    boot = "parametric", R = 9)


# or fit restricted linear model
fit2.con <- restriktor(fit2.lm, constraints = myConstraints2)

iht(fit2.con)

## Not run:
  # increasing means in matrix notation.
  Amat2 <- rbind(c(-1, 1, 0, 0),
                 c( 0,-1, 1, 0),
                 c( 0, 0,-1, 1))
  myRhs2 <- rep(0L, nrow(Amat2))
  myNeq2 <- 0

  iht(fit2.con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
      type = "A", boot = "parametric", R = 9)

## End(Not run)

## example 3:
# combination of equality and inequality constraints.
myConstraints3 <- ' group1  = group2
                    group3  < group4 '

iht(fit2.lm, constraints = myConstraints3, type = "B", neq.alt = 1)

# fit resticted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
# Note that, a warning message may be thrown because the number of
# bootstrap samples is too low.
fit3.con <- restriktor(fit2.lm, constraints = myConstraints3,
                       se = "boot.model.based", B = 9)
iht(fit3.con, type = "B", neq.alt = 1)


## example 4:
# restriktor can also be used to define effects using the := operator
# and impose constraints on them. For example, is the
# average effect (AVE) larger than zero?
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
```

```
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit4.lm <- lm(y ~ X*Z, data = DATA3)

# constraint syntax
myConstraints4 <- ' AVE := X + 16.86137*X.Z;
                    AVE > 0 '

iht(fit4.lm, constraints = myConstraints4)

# or
fit4.con <- restriktor(fit4.lm, constraints = ' AVE := X + 16.86137*X.Z;
                                                AVE > 0 ')
iht(fit4.con)
```

---

conTestLRT *Likelihood-ratio-bar test for iht*

Description

conTestLRT tests linear equality and/or inequality restricted hypotheses for linear models by LR-tests. It can be used directly and is called by the conTest function if test = "LRT".

Usage

```
## S3 method for class 'conLM'
conTestLRT(object, type = "A", neq.alt = 0,
           boot = "no", R = 9999, p.distr = rnorm,
           parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
           verbose = FALSE, control = NULL, ...)

## S3 method for class 'conGLM'
conTestLRT(object, type = "A", neq.alt = 0,
            boot = "no", R = 9999, p.distr = rnorm,
            parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
            verbose = FALSE, control = NULL, ...)

## S3 method for class 'conMLM'
conTestLRT(object, type = "A", neq.alt = 0,
            boot = "no", R = 9999, p.distr = rnorm,
            parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
            verbose = FALSE, control = NULL, ...)
```

Arguments

object	an object of class conLM, conMLM or conGLM.
type	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
neq.alt	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
boot	the null-distribution of these test-statistics (except under type "C", see details) takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if "parametric", the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See p.distr for other distributional options. If "model.based", a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If "no", the p-value is computed based on the weights obtained via simulation (mix.weights = "boot") or using the multivariate normal distribution function (mix.weights = "pmvnorm"). Note that, these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class conRLM, namely for computing the p-value for the robust test = "Wald" and the robust "score". In these cases the weights need to be recalculated.
R	integer; number of bootstrap draws for boot. The default value is set to 9999.
p.distr	random generation distribution for the parametric bootstrap. For all available distributions see ?distributions. For example, if rnorm, samples are drawn from the normal distribution (default) with mean zero and variance one. If rt, samples are drawn from a t-distribution. If rchisq, samples are drawn from a chi-square distribution. The random generation distributional parameters will be passed in via ....
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
seed	seed value. The default value is set to 1234.
verbose	logical; if TRUE, information is shown at each bootstrap draw.
control	a list of control arguments: • absval tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm. • maxit the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported). • tol numerical tolerance value. Estimates smaller than tol are set to 0.
...	additional arguments to be passed to the p.distr function.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.
- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).
- Type C: Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle (Silvapulle and Sen, 2005, chp 5.3). Note that, this test only makes sense in case of no equality constraints.
- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

The exact finite sample distributions of the non-robust F-, score- and LR-test statistics based on restricted OLS estimates and normally distributed errors, are a mixture of F-distributions under the null hypothesis (Wolak, 1987). In agreement with Silvapulle (1992), we found that the results based on these mixtures of F-distributions approximate the tail probabilities of the robust tests better than their asymptotic distributions. Therefore, all p-values for hypothesis test Type `"A"`, `"B"` and `"global"` are computed based on mixtures of F-distributions.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

`CON`	a list with useful information about the constraints.
`Amat`	constraints matrix.
`bvec`	vector of right-hand side elements.
`meq`	number of equality constraints.
`meq_alt`	same as input neq.alt.
`iact`	number of active constraints.
`type`	same as input.
`test`	same as input.
`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b_eqrestr`	equality restricted regression coefficients. Only available for `type = "A"` and `type = "global"`, else `b.eqrestr = NULL`.
`b_unrestr`	unrestricted regression coefficients.
`b_restr`	restricted regression coefficients.

b_restr_alt	restricted regression coefficients under HA if some equality constraints are maintained. Only available for type = "B" else b_restr_alt = NULL.
Sigma	variance-covariance matrix of unrestricted model.
R2_org	unrestricted R-squared, not available for objects of class conGLM.
R2_reduced	restricted R-squared, not available for objects of class conGLM.
boot	same as input.
model_org	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

See Also

quadprog, conTest

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1_lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1_lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1_lm, myConstraints1, test = "LRT")


# another way is to first fit the restricted model
fit_restr1 <- restriktor(fit1_lm, constraints = myConstraints1)

iht(fit_restr1, test = "LRT")
```

```
## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  iht(fit1_lm, constraints = Amat1, test = "LRT",
      rhs = myRhs1, neq = myNeq1)

## End(Not run)

#########################
## Artificial examples ##
#########################
# generate data
n <- 10
means <- c(1,2,1,3)
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)

# fit unrestricted linear model
fit2_lm <- lm(y ~ -1 + group, data = DATA2)
coef(fit2_lm)

## example 2: increasing means
myConstraints2 <- ' group1 < group2
                    group2 < group3
                    group3 < group4 '

# compute F-test for hypothesis test Type A and compute the tail
# probability based on the parametric bootstrap. We only generate 9
# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
iht(fit2_lm, constraints = myConstraints2, type = "A", test = "LRT",
    boot = "parametric", R = 9)

# or fit restricted linear model
fit2_con <- restriktor(fit2_lm, constraints = myConstraints2)

iht(fit2_con, test = "LRT")

## Not run:
  # increasing means in matrix notation.
  Amat2 <- rbind(c(-1, 1, 0, 0),
                 c( 0,-1, 1, 0),
                 c( 0, 0,-1, 1))
  myRhs2 <- rep(0L, nrow(Amat2))
  myNeq2 <- 0

  iht(fit2_con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
```

```
      type = "A", test = "LRT", boot = "parametric", R = 9)

## End(Not run)

## example 3:
# combination of equality and inequality constraints.
myConstraints3 <- ' group1  = group2
                    group3  < group4 '

iht(fit2_lm, constraints = myConstraints3, type = "B",
    test = "LRT", neq.alt = 1)

# fit resticted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
# Note that, a warning message may be thrown because the number of
# bootstrap samples is too low.
fit3_con <- restriktor(fit2_lm, constraints = myConstraints3,
                       se = "boot.model.based", B = 9)
iht(fit3_con, type = "B", test = "LRT", neq.alt = 1)


## example 4:
# restriktor can also be used to define effects using the := operator
# and impose constraints on them. For example, is the
# average effect (AVE) larger than zero?
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit4_lm <- lm(y ~ X*Z, data = DATA3)

# constraint syntax
myConstraints4 <- ' AVE := X + 16.86137*X.Z;
                    AVE > 0 '

iht(fit4_lm, constraints = myConstraints4, test = "LRT")

# or
fit4_con <- restriktor(fit4_lm, constraints = ' AVE := X + 16.86137*X.Z;
                                                AVE > 0 ')
iht(fit4_con, test = "LRT")
```

---

conTestScore *Score-bar test for iht*

Description

conTestScore tests linear equality and/or inequality restricted hypotheses for (robust) linear models by score-tests. It can be used directly and is called by the conTest function if test = "score".

Usage

```
## S3 method for class 'conLM'
conTestScore(object, type = "A", neq.alt = 0,
           boot = "no", R = 9999, p.distr = rnorm,
           parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
           verbose = FALSE, control = NULL, ...)

## S3 method for class 'conRLM'
conTestScore(object, type = "A", neq.alt = 0,
           boot = "no", R = 9999, p.distr = rnorm,
           parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
           verbose = FALSE, control = NULL, ...)

## S3 method for class 'conGLM'
conTestScore(object, type = "A", neq.alt = 0,
           boot = "no", R = 9999, p.distr = rnorm,
           parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
           verbose = FALSE, control = NULL, ...)
```

Arguments

object	an object of class conLM, conRLM or conGLM.
type	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
neq.alt	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
boot	the null-distribution of these test-statistics (except under type "C", see details) takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if "parametric", the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See p.distr for other distributional options. If "model.based", a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If "no", the p-value is computed based on the weights obtained via simulation (mix.weights = "boot") or using the multivariate normal distribution function (mix.weights = "pmvnorm"). Note that, these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class conRLM, namely for computing the p-value for the robust test = "Wald" and the robust "score". In these cases the weights need to be recalculated.

R	integer; number of bootstrap draws for boot. The default value is set to 9999.
p.distr	random generation distribution for the parametric bootstrap. For all available distributions see ?distributions. For example, if rnorm, samples are drawn from the normal distribution (default) with mean zero and variance one. If rt, samples are drawn from a t-distribution. If rchisq, samples are drawn from a chi-square distribution. The random generation distributional parameters will be passed in via ....
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
seed	seed value. The default value is set to 1234.
verbose	logical; if TRUE, information is shown at each bootstrap draw.
control	a list of control arguments: • absval tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm. • maxit the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported). • tol numerical tolerance value. Estimates smaller than tol are set to 0.
...	additional arguments to be passed to the p.distr function.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.
- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).
- Type C: Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle (Silvapulle and Sen, 2005, chp 5.3). Note that, this test only makes sense in case of no equality constraints.
- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

The exact finite sample distributions of the non-robust F-, score- and LR-test statistics based on restricted OLS estimates and normally distributed errors, are a mixture of F-distributions under the null hypothesis (Wolak, 1987). In agreement with Silvapulle (1992), we found that the results based on these mixtures of F-distributions approximate the tail probabilities of the robust tests better than their asymptotic distributions. Therefore, all p-values for hypothesis test Type "A", "B" and "global" are computed based on mixtures of F-distributions.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

`CON`	a list with useful information about the constraints.
`Amat`	constraints matrix.
`bvec`	vector of right-hand side elements.
`meq`	number of equality constraints.
`meq.alt`	same as input neq.alt.
`iact`	number of active constraints.
`type`	same as input.
`test`	same as input.
`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b.eqrestr`	equality restricted regression coefficients. Only available for `type = "A"` and `type = "global"`, else `b.eqrestr = NULL`.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`b.restr.alt`	restricted regression coefficients under HA if some equality constraints are maintained. Only available for `type = "B"` else `b.restr.alt = NULL`.
`Sigma`	variance-covariance matrix of unrestricted model.
`R2.org`	unrestricted R-squared, not available for objects of class `conGLM`.
`R2.reduced`	restricted R-squared, not available for objects of class `conGLM`.
`boot`	same as input.
`model.org`	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M. and Silvapulle, P. (1995). A score test against one-sided alternatives. *American statistical association*, **90**, 342–349.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

See Also

quadprog, `conTest`

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.lm, myConstraints1, test = "score")


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit.restr1, test = "score")

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  iht(fit1.lm, constraints = Amat1, test = "score",
      rhs = myRhs1, neq = myNeq1)

## End(Not run)

#########################
## Artificial examples ##
#########################
# generate data
n <- 10
means <- c(1,2,1,3)
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)
```

```
# fit unrestricted linear model
fit2.lm <- lm(y ~ -1 + group, data = DATA2)
coef(fit2.lm)

## example 2: increasing means
myConstraints2 <- ' group1 < group2
                    group2 < group3
                    group3 < group4 '

# compute F-test for hypothesis test Type A and compute the tail
# probability based on the parametric bootstrap. We only generate 9
# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
iht(fit2.lm, constraints = myConstraints2, type = "A", test = "score",
    boot = "parametric", R = 9)


# or fit restricted linear model
fit2.con <- restriktor(fit2.lm, constraints = myConstraints2)

conTest(fit2.con, test = "score")

## Not run:
  # increasing means in matrix notation.
  Amat2 <- rbind(c(-1, 1, 0, 0),
                 c( 0,-1, 1, 0),
                 c( 0, 0,-1, 1))
  myRhs2 <- rep(0L, nrow(Amat2))
  myNeq2 <- 0

  iht(fit2.con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
      type = "A", test = "score", boot = "parametric", R = 9)

## End(Not run)

## example 3:
# combination of equality and inequality constraints.
myConstraints3 <- ' group1  = group2
                    group3  < group4 '

iht(fit2.lm, constraints = myConstraints3, type = "B", test = "score", neq.alt = 1)

# fit resticted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
# Note that, a warning message may be thrown because the number of
# bootstrap samples is too low.
fit3.con <- restriktor(fit2.lm, constraints = myConstraints3,
                       se = "boot.model.based", B = 9)
iht(fit3.con, type = "B", test = "score", neq.alt = 1)


## example 4:
```

```
# restriktor can also be used to define effects using the := operator
# and impose constraints on them. For example, is the
# average effect (AVE) larger than zero?
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit4.lm <- lm(y ~ X*Z, data = DATA3)

# constraint syntax
myConstraints4 <- ' AVE := X + 16.86137*X.Z;
                    AVE > 0 '

iht(fit4.lm, constraints = myConstraints4, test = "score")

# or
fit4.con <- restriktor(fit4.lm, constraints = ' AVE := X + 16.86137*X.Z;
                                                AVE > 0 ')
iht(fit4.con, test = "score")
```

`conTestWald` *Wald-bar test for robust iht*

Description

`conTestWald` tests linear equality and/or inequality restricted hypotheses for linear models by Wald-tests. It can be used directly and is called by the `conTest` function if `test = "Wald"`.

Usage

```
## S3 method for class 'conRLM'
conTestWald(object, type = "A", neq.alt = 0,
            boot = "no", R = 9999, p.distr = rnorm,
            parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
            verbose = FALSE, control = NULL, ...)
```

Arguments

`object`	an object of class `conRLM`.
`type`	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.

neq.alt	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
boot	the null-distribution of these test-statistics (except under type "C", see details) takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if "parametric", the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See p.distr for other distributional options. If "model.based", a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If "no", the p-value is computed based on the weights obtained via simulation (mix.weights = "boot") or using the multivariate normal distribution function (mix.weights = "pmvnorm"). Note that, these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class conRLM, namely for computing the p-value for the robust test = "Wald" and the robust "score". In these cases the weights need to be recalculated.
R	integer; number of bootstrap draws for boot. The default value is set to 9999.
p.distr	random generation distribution for the parametric bootstrap. For all available distributions see ?distributions. For example, if rnorm, samples are drawn from the normal distribution (default) with mean zero and variance one. If rt, samples are drawn from a t-distribution. If rchisq, samples are drawn from a chi-square distribution. The random generation distributional parameters will be passed in via ....
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
seed	seed value. The default value is set to 1234.
verbose	logical; if TRUE, information is shown at each bootstrap draw.
control	a list of control arguments: • absval tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm. • maxit the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported). • tol numerical tolerance value. Estimates smaller than tol are set to 0.
...	additional arguments to be passed to the p.distr function.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.

- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).
- Type C: Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle (Silvapulle and Sen, 2005, chp 5.3). Note that, this test only makes sense in case of no equality constraints.
- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

The exact finite sample distributions of the non-robust F-, score- and LR-test statistics based on restricted OLS estimates and normally distributed errors, are a mixture of F-distributions under the null hypothesis (Wolak, 1987). In agreement with Silvapulle (1992), we found that the results based on these mixtures of F-distributions approximate the tail probabilities of the robust tests better than their asymptotic distributions. Therefore, all p-values for hypothesis test Type `"A"`, `"B"` and `"global"` are computed based on mixtures of F-distributions.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

`CON`	a list with useful information about the constraints.
`Amat`	constraints matrix.
`bvec`	vector of right-hand side elements.
`meq`	number of equality constraints.
`meq.alt`	same as input neq.alt.
`iact`	number of active constraints.
`type`	same as input.
`test`	same as input.
`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b.eqrestr`	equality restricted regression coefficients. Only available for `type = "A"` and `type = "global"`, else `b.eqrestr = NULL`.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`b.restr.alt`	restricted regression coefficients under HA if some equality constraints are maintained. Only available for `type = "B"` else `b.restr.alt = NULL`.
`Sigma`	variance-covariance matrix of unrestricted model.
`R2.org`	unrestricted R-squared, not available for objects of class `conGLM`.
`R2.reduced`	restricted R-squared, not available for objects of class `conGLM`.
`boot`	same as input.
`model.org`	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M. (1992b). Robust Wald-Type Tests of One-Sided Hypotheses in the Linear Model. *Journal of the American Statistical Association*, **87**, 156–161.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

See Also

quadprog, conTest

Examples

```
library(MASS)
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted robust linear model
fit1.rlm <- rlm(Age ~ -1 + Group, data = DATA1, method = "MM")

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.rlm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.rlm, myConstraints1, test = "Wald")


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.rlm, constraints = myConstraints1)

iht(fit.restr1, test = "Wald")

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
```

```
  myNeq1 <- 0

  iht(fit1.rlm, constraints = Amat1, test = "Wald",
      rhs = myRhs1, neq = myNeq1)

## End(Not run)

#########################
## Artificial examples ##
#########################
# generate data
n <- 30
means <- c(1,2,1,3)
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)

# fit unrestricted robust linear model
fit2.rlm <- rlm(y ~ -1 + group, data = DATA2, method = "MM")
coef(fit2.rlm)

## example 2: increasing means
myConstraints2 <- ' group1 < group2
                    group2 < group3
                    group3 < group4 '

# compute Wald-test for hypothesis test Type A and compute the tail
# probability based on the parametric bootstrap. We only generate 9
# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
iht(fit2.rlm, constraints = myConstraints2, type = "A",
    test = "Wald", boot = "parametric", R = 9)


# or fit restricted robust linear model
fit2.con <- restriktor(fit2.rlm, constraints = myConstraints2)

iht(fit2.con, test = "Wald")

## Not run:
  # increasing means in matrix notation.
  Amat2 <- rbind(c(-1, 1, 0, 0),
                 c( 0,-1, 1, 0),
                 c( 0, 0,-1, 1))
  myRhs2 <- rep(0L, nrow(Amat2))
  myNeq2 <- 0

  iht(fit2.con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
      type = "A", test = "Wald", boot = "parametric", R = 9)

## End(Not run)
```

```
## example 3:
# combination of equality and inequality constraints.
myConstraints3 <- ' group1  = group2
                    group3  < group4 '

iht(fit2.rlm, constraints = myConstraints3, type = "B", test = "Wald", neq.alt = 1)

# fit robust resticted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
# Note that, a warning message may be thrown because the number of
# bootstrap samples is too low.
fit3.con <- restriktor(fit2.rlm, constraints = myConstraints3,
                       se = "boot.model.based", B = 9)
iht(fit3.con, type = "B", test = "Wald", neq.alt = 1)


## example 4:
# restriktor can also be used to define effects using the := operator
# and impose constraints on them. For example, is the
# average effect (AVE) larger than zero?
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit3.rlm <- rlm(y ~ X*Z, data = DATA3, method = "MM")

# constraint syntax
myConstraints4 <- ' AVE := X + 16.86137*X.Z;
                    AVE > 0 '

iht(fit3.rlm, constraints = myConstraints4, test = "Wald")

# or
fit3.con <- restriktor(fit3.rlm, constraints = ' AVE := X + 16.86137*X.Z;
                                                 AVE > 0 ')
iht(fit3.con, test = "Wald")
```

---

conTest_ceq *Tests for iht with equality constraints only*

Description

conTest_ceq tests linear equality restricted hypotheses for (robust) linear models by F-, Wald-,

and score-tests. It can be used directly and is called by the conTest function if all restrictions are equalities.

Usage

```
## S3 method for class 'conLM'
conTest_ceq(object, test = "F", boot = "no",
            R = 9999, p.distr = rnorm, parallel = "no",
            ncpus = 1L, cl = NULL, seed = 1234, verbose = FALSE, ...)

## S3 method for class 'conRLM'
conTest_ceq(object, test = "F", boot = "no",
            R = 9999, p.distr = rnorm, parallel = "no",
            ncpus = 1L, cl = NULL, seed = 1234, verbose = FALSE, ...)

## S3 method for class 'conGLM'
conTest_ceq(object, test = "F", boot = "no",
            R = 9999, p.distr = rnorm, parallel = "no",
            ncpus = 1L, cl = NULL, seed = 1234, verbose = FALSE, ...)
```

Arguments

object	an object of class conLM, conRLM or conGLM.
test	test statistic; for information about the null-distribution see details. • for object of class lm and glm; if "F" (default), the classical F-statistic is computed. If "Wald", the classical Wald-statistic is computed. If "score", the classical score test statistic is computed. • for object of class rlm; if "F" (default), a robust likelihood ratio type test statistic (Silvapulle, 1992a) is computed. If "Wald", a robust Wald test statistic (Silvapulle, 1992b) is computed. If "score", a score test statistic (Silvapulle, 1996) is computed.
boot	if "parametric", the p-value is computed based on the parametric bootstrap. See p.distr for available distributions. If "model.based", a model-based bootstrap method is used. Model-based bootstrapping is not supported for the conGLM object yet.
R	integer; number of bootstrap draws for boot. The default value is set to 9999.
p.distr	the p.distr function is specified by this function. For all available distributions see ?distributions. For example, if rnorm, samples are drawn from the normal distribution (default) with mean zero and variance one. If rt, samples are drawn from a t-distribution. If rchisq, samples are drawn from a chi-square distribution. The distributional parameters will be passed in via ....
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.

cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
seed	seed value. The default value is set to 1234.
verbose	logical; if TRUE, information is shown at each bootstrap draw.
...	additional arguments to be passed to the p.distr function.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

CON	a list with useful information about the constraints.
Amat	constraints matrix.
bvec	vector of right-hand side elements.
meq	number of equality constraints.
test	same as input.
Ts	test-statistic value.
df.residual	the residual degrees of freedom.
pvalue	tail probability for Ts.
b_unrestr	unrestricted regression coefficients.
b_restr	restricted regression coefficients.
R2_org	unrestricted R-squared.
R2_reduced	restricted R-squared.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M. (1992a). Robust tests of inequality constraints and one-sided hypotheses in the linear model. *Biometrika*, **79**, 621–630.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M. (1992b). Robust Wald-Type Tests of One-Sided Hypotheses in the Linear Model. *Journal of the American Statistical Association*, **87**, 156–161.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

See Also

quadprog, iht

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  = GroupPassive;
                    GroupPassive = GroupNo '

iht(fit1.lm, myConstraints1)


# another way is to first fit the restricted model
fit_restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit_restr1)

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 2

  iht(fit1.lm, constraints = Amat1,
      rhs = myRhs1, neq = myNeq1)

## End(Not run)
```

---

conTest_summary *function for computing all available hypothesis tests*

Description

conTest_summary computes all available hypothesis tests and returns and object of class conTest for which a print function is available. The conTest_summary can be used directly and is called by the conTest function if type = "summary".

Usage

```
## S3 method for class 'restriktor'
conTest_summary(object, test = "F", ...)
```

Arguments

object	an object of class restriktor.
test	test statistic; for information about the null-distribution see details. • for object of class lm; if "F" (default), the classical F-statistic is computed. If "Wald", the classical Wald-statistic is computed. If "score", the classical score test statistic is computed. • for object of class rlm; if "F" (default), a robust likelihood ratio type test statistic (Silvapulle, 1992a) is computed. If "Wald", a robust Wald test statistic (Silvapulle, 1992b) is computed. If "score", a score test statistic (Silvapulle, 1996) is computed.
...	the same arguments as passed to the iht function, except for type, of course.

Value

An object of class conTest, for which a print is available. More specifically, it is a list with the following items:

CON	a list with useful information about the constraints.
Amat	constraints matrix.
bvec	vector of right-hand side elements.
meq	number of equality constraints.
meq.alt	same as input neq.alt.
iact	number of active constraints.
type	same as input.
test	same as input.
Ts	test-statistic value.
df.residual	the residual degrees of freedom.
pvalue	tail probability for Ts.
b.eqrestr	equality restricted regression coefficients. Only available for type = "A" and type = "global", else b.eqrestr = NULL.
b.unrestr	unrestricted regression coefficients.
b.restr	restricted regression coefficients.
b.restr.alt	restricted regression coefficients under HA if some equality constraints are maintained.
Sigma	variance-covariance matrix of unrestricted model.
R2.org	unrestricted R-squared.

R2.reduced	restricted R-squared.
boot	same as input.
model.org	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Shapiro, A. (1988). Towards a unified theory of inequality-constrained testing in multivariate analysis. *International Statistical Review* **56**, 49–62.

Silvapulle, M. (1992a). Robust tests of inequality constraints and one-sided hypotheses in the linear model. *Biometrika*, **79**, 621–630.

Silvapulle, M. (1992b). Robust Wald-Type Tests of One-Sided Hypotheses in the Linear Model. *Journal of the American Statistical Association*, **87**, 156–161.

Silvapulle, M. and Silvapulle, P. (1995). A score test against one-sided alternatives. *American statistical association*, **90**, 342–349.

Silvapulle, M. (1996) On an F-type statistic for testing one-sided hypotheses and computation of chi-bar-squared weights. *Statistics and probability letters*, **28**, 137–141.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

Wolak, F. (1987). An exact test for multiple inequality and equality constraints in the linear regression model. *Journal of the American statistical association*, **82**, 782–793.

See Also

quadprog, iht

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.lm)

# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
```

```
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.lm, myConstraints1)


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit.restr1)

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  fit1.con <- restriktor(fit1.lm, constraints = Amat1,
                         rhs = myRhs1, neq = myNeq1)
  iht(fit1.con)

## End(Not run)
```

`con_weights_boot` *function for computing the chi-bar-square weights based on Monte Carlo simulation.*

Description

The null-distribution of the test statistics under inequality constraints takes the form of mixtures of F-distributions. This function computes these mixing weights (a.k.a chi-bar-square weights and level probabilities). It can be used directly and is called by the `conTest` function.

Usage

```
con_weights_boot(VCOV, Amat, meq,
                 R = 99999L, parallel = c("no", "multicore", "snow"),
                 ncpus = 1L, cl = NULL, seed = NULL, verbose = FALSE, ...)
```

Arguments

`VCOV`	variance-covariance matrix of the data for which the weights are to be calculated.
`Amat`	constraints matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ). The rows should be linear independent, otherwise the function gives

an error. For more information about constructing the matrix R and rhs see restriktor.

`meq` integer (default = 0) treating the number of constraints rows as equality constraints instead of inequality constraints. For example, if `meq = 2`, this means that the first two rows of the constraints matrix R are treated as equality constraints.

`R` integer; number of bootstrap draws for `mix.bootstrap`. The default value is set to 99999.

`parallel` the type of parallel operation to be used (if any). If missing, the default is set "no".

`ncpus` integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.

`cl` an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.

`seed` seed value.

`verbose` logical; if TRUE, information is shown at each bootstrap draw.

`...` no additional arguments for now.

Value

The function returns a vector with the mixing weights

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Silvapulle, M.J. and Sen, P.K. (2005, p.79). *Constrained Statistical Inference*. Wiley, New York.

Examples

```
W <- matrix(c(1,0.5,0.5,1),2,2)
Amat <- rbind(c(0,1))
meq <- 0L
# we only generate 99 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
wt.bar <- con_weights_boot(W, Amat, meq, R = 99)
wt.bar
```

evSyn *GORIC(A) Evidence synthesis*

Description

GORIC(A) evidence synthesis aggregates the evidence for theory-based hypotheses from multiple studies that may use diverse designs to investigate the same central theory.

Usage

```
evSyn(object, ...)

## S3 method for class 'evSyn'
plot(x, ...)
```

Arguments

object	an object
x	an object of class evSyn
...	This depends on the class of the object.

Details

details

Value

output

Author(s)

Leonard Vanbrabant and Rebecca Kuiper

References

refs

Examples

```
## example
```

Exam — *Relation between exam scores and study hours, anxiety scores and average point scores.*

Description

The data provide information about students' exam scores, average point score, the amount of study hours and anxiety scores.

Usage

```
data(Exam)
```

Format

A data frame of 20 observations of 4 variables.

`Scores` exam scores

`Hours` study hours

`Anxiety` anxiety scores

`APS` average point score

References

The original source of these data is http://staff.bath.ac.uk/pssiw/stats2/examrevision.sav.

Examples

```
head(Exam)
```

FacialBurns — *Dataset for illustrating the conTest_conLavaan function.*

Description

A dataset from the Dutch burn center (http://www.adbc.nl). The data were used to examine psychosocial functioning in patients with facial burn wounds. Psychosocial functioning was measured by Anxiety and depression symptoms (HADS), and self-esteem (Rosenberg's self-esteem scale).

Usage

```
data(FacialBurns)
```

Format

A data frame of 77 observations of 6 variables.

`Selfesteem` Rosenberg's self-esteem scale

`HADS` Anxiety and depression scale

`Age` Age measured in years, control variable

`TBSA` Total Burned Surface Area

`RUM` Rumination, control variable

`Sex` Gender, grouping variable

Examples

```
head(FacialBurns)
```

`goric` *Generalized Order-Restricted Information Criterion (Approximation) Weights*

Description

The `goric` function computes GORIC(A) weights, which are comparable to the Akaike weights.

Usage

```
goric(object, ...)

## Default S3 method:
goric(object, ..., constraints = NULL,
      comparison = c("unconstrained", "complement", "none"),
      VCOV = NULL, sample.nobs = NULL, type = "goric",
      auto.bound = FALSE, debug = FALSE)

## S3 method for class 'restriktor'
goric(object, ..., constraints = NULL,
      comparison = "unconstrained", type = "goric",
      auto.bound = NULL, debug = FALSE)

## S3 method for class 'lm'
goric(object, ..., constraints = NULL,
      comparison = "unconstrained", type = "goric",
      missing = "none", auxiliary = c(), emControl = list(),
      auto.bound = NULL, debug = FALSE)

## S3 method for class 'numeric'
goric(object, ..., constraints = NULL,
```

```
      VCOV = NULL, comparison = "unconstrained",
      type = "gorica", sample.nobs = NULL,
      auto.bound = NULL, debug = FALSE)

## S3 method for class 'lavaan'
goric(object, ..., constraints = NULL,
      comparison = "unconstrained", type = "gorica",
      standardized = FALSE, auto.bound = NULL, debug = FALSE)

## S3 method for class 'CTmeta'
goric(object, ..., constraints = NULL,
      comparison = "unconstrained", type = "gorica",
      sample.nobs = NULL, auto.bound = NULL, debug = FALSE)

## S3 method for class 'rma'
goric(object, ..., constraints = NULL,
      VCOV = NULL, comparison = "unconstrained", type = "gorica",
      sample.nobs = NULL, auto.bound = NULL, debug = FALSE)


## S3 method for class 'con_goric'
print(x, digits = max(3, getOption("digits") - 4), ...)

## S3 method for class 'con_goric'
summary(object, brief = TRUE, digits = max(3, getOption("digits") - 4), ...)

## S3 method for class 'con_goric'
coef(object, ...)
```

Arguments

`object`	an object containing the outcome of an unconstrained statistical analysis. Currently, the following objects can be processed: • a fitted object of class restriktor. • a fitted unconstrained object of class `lm`, `rlm` or `glm`. • a numeric vector containing the unconstrained estimates resulting from any statistical analysis. • a fitted object of class `lavaan`. • a fitted object of class `CTmeta`. • a fitted object of class `rma`.
`x`	an object of class `con_goric`.
`...`	this depends on the class of the object. If object is of class restriktor, further objects of class restriktor can be passed. Note that, the objects have to be of the same class. If object is of class `lavaan`, the standardized or unstandardized vcov can be used, using setting `standardized = TRUE`. See details for more information.

`constraints` list; there are two ways to constrain parameters. First, the constraint syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of `coef(model)` or `names(vector)` can be used as names. See details for more information. Note that objects of class "mlm" do not (yet) support this method.

Second, the constraint syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see details.

`comparison` if "`unconstrained`" (default) the unconstrained model is included in the set of models. If "`complement`" then the restricted object is compared against its complement. Note that the complement can only be computed for one model/hypothesis at a time (for now). If "`none`" the model is only compared against the models provided by the user.

`VCOV` variance-coviance matrix. Only needed if object is of class numeric and `type = "gorica"` or `type = "goricac"`.

`sample.nobs` the number of observations if `type = "goricac"`. Note that, if `type = "goricc"`, the number of observations are inherited from the fitted object.

`type` if `"goric"`, the generalized order-restricted information criterion value is computed. If `"gorica"` the log-likihood is computed using the multivariate normal distribution function.

`missing` the default setting for objects of class "lm" is listwise: all cases with missing values are removed from the data before the analysis. This is only valid if the data are missing completely at random (MCAR). Another option is to use "two.stage". In this approach, missing data are imputed using an EM algorithm. However, we cannot use the complete data as input for futher analyses, because the resulting complete data variance-covariance matrix will not be correct. Therefore, we compute the correct aymptotic covariance (Savalei and Bentler, 2009) and use it as input for the `goric.numeric` function to compute a GORICA(C) value. Note that, the parameter estimates are also recomputed using the complete data.

`auxiliary` Vector. The inclusion of auxiliary variables can improve the imputation model. These auxiliary variables are not part of the target model.

`emControl` a list of control arguments for the `emnorm` function from the **norm2** package.

`standardized` if TRUE, standardized parameter estimates are used.

`auto.bound` not used yet.

`digits` the number of significant digits to use when printing.

`debug` if TRUE, debugging information is printed out.

`brief` if TRUE, a short overview is printed.

Details

The GORIC(A) values themselves are not interpretable and the interest lie in their differences. The GORIC(A) weights reflect the support of each hypothesis in the set. To compare two hypotheses

(and not one to the whole set), one can examine the ratio of the two corresponding GORIC(A) weights. To avoid selecting a weakly supported hypothesis as the best one, the unconstrained hypothesis is usually included as safeguard.

In case of one order-constrained hypothesis, say H1, the complement Hc can be computed as competing hypothesis. The complement is defined as Hc = not H1.

If the object(s) is of class restriktor the constraints are inherited. Otherwise, the constraint syntax can be parsed via the constraint argument. If the object is an unconstrained model of class `lm`, `rlm` or `glm`, then the constraints can be specified in two ways, see `restriktor`. Note that if the constraints are written in matrix notation, then the constraints for each model/hypothesis is put in a named list with specific names amat, rhs, and neq. For example with three parameters x1, x2, x3, and x1 > 0: list(model1 = list(amat = rbind(c(1, 0, 0)), rhs = 0, neq = 0))). The `rhs` and `neq` are not required if they are equal to 0. If `type = "gorica"`, then the object might be a (named) numeric vector. The constraints can again be specified in two ways, see `restriktor`. For examples, see below.

To determine the penalty term values, the chi-bar-square weights (a.k.a. level probabilities) must be computed. If `"mix.weights = "pmvnorm" "` (default), the chi-bar-square weights are computed based on the multivariate normal distribution function with additional Monte Carlo steps. If `"mix.weights = "boot" "`, the chi-bar-square weights are computed using parametric bootstrapping (see `restriktor`).

The "two.stage" approach for missing data uses the EM algorithm from the `norm2` package. The response variables are assumed to be jointly normal. In practice, missing-data procedures designed for variables that are normal are sometimes applied to variables that are not. Binary and ordinal variables are sometimes imputed under a normal model, and the imputed values may be classified or rounded. This is also how restriktor handles (ordered) factors for now.

A better strategy (not implemented yet) would be to convert (ordered) factors into a pair of dummy variables. If the (ordered) factors have missing values, the dummy variables could be included as columns of Y and imputed, but then you have to decide how to convert the continuously distributed imputed values for these dummy codes back into categories.

Value

The function returns a dataframe with the log-likelihood, penalty term, GORIC(A) values and the GORIC(A) weights. Furthermore, a dataframe is returned with the relative GORIC(A) weights.

Author(s)

Leonard Vanbrabant and Rebecca Kuiper

References

Kuiper, R.M., Hoijtink, H., and Silvapulle, M.J. (2011). An Akaike-type information criterion for model selection under inequality constraints. *Biometrika*, **98**, 2, 495–501.

Vanbrabant, L. and Kuiper, R. (2020). Evaluating a theory-based hypothesis against its complement using an AIC-type information criterion with an application to facial burn injury. Psychological Methods.

Victoria Savalei and Peter M. Bentler (2009) A Two-Stage Approach to Missing Data: Theory and Application to Auxiliary Variables, Structural Equation Modeling: A Multidisciplinary Journal, 16:3, 477-497, DOI: 10.1080/10705510903008238

Examples

```
library(MASS)
## lm
## unrestricted linear model for ages (in months) at which an
## infant starts to walk alone.

# prepare data
DATA <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestrikted linear model
fit1.lm <- lm(Age ~ Group, data = DATA)

# some artificial restrictions
H1 <- "GroupPassive > 0; GroupPassive < GroupNo"
H2 <- "GroupPassive > 0; GroupPassive > GroupNo"
H3 <- "GroupPassive = 0; GroupPassive < GroupNo"


# object is of class lm
goric(fit1.lm, constraints = list(H1 = H1, H2 = H2, H3 = H3))

# same result, but using objects of class restriktor
fit1.con <- restriktor(fit1.lm, constraints = H1)
fit2.con <- restriktor(fit1.lm, constraints = H2)
fit3.con <- restriktor(fit1.lm, constraints = H3)

goric(fit1.con, fit2.con, fit3.con)

# same result, but using the parameter estimates and covariance matrix as input
# Note, that in case of a numeric input only the gorica(c) can be computed.
goric(coef(fit1.lm), VCOV = vcov(fit1.lm), constraints = list(H1 = H1, H2 = H2, H3 = H3))


# hypothesis H1 versus the complement (i.e., not H1)
goric(fit1.lm, constraints = list(H1 = H1), comparison = "complement")


## GORICA
# generate data
n <- 10
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 1 + x1 + x2 + rnorm(n)
# fit unconstrained linear model
fit.lm <- lm(y ~ x1 + x2)

# extract unconstrained estimates
est <- coef(fit.lm)
# unconstrained variance-covariance matrix
VCOV <- vcov(fit.lm)

## constraint syntax (character)
```

```
h1 <- "x1 > 0"
h2 <- "x1 > 0; x2 > 0"
# use fitted unconstrained linear model
goric(fit.lm, constraints = list(h1, h2), type = "gorica")
# use unconstrained estimates
goric(est, VCOV = VCOV, constraints = list(h1, h2), type = "gorica")

## constraint syntax (matrix notation)
h1 <- list(amat = c(0,1,0))
h2 <- list(amat = rbind(c(0,1,0), c(0,0,1)), rhs = c(0.5, 1), neq = 0)
goric(fit.lm, constraints = list(h1, h2), type = "gorica")
goric(est, VCOV = VCOV, constraints = list(h1, h2), type = "gorica")


## use parallel processing for computing the level probabilities (i.e., mixing weights)
## Not run:
goric(fit.lm, constraints = list(H1), comparison = "complement",
      mix.weights = "boot", parallel = "snow", ncpus = 8, mix.bootstrap = 99999)

## End(Not run)
```

`Hurricanes` *The Hurricanes Dataset*

Description

The data provide information on the effect of El Nino (Cold, Neutral, Warm) on the number of hurricanes from 1950 to 1995.

Usage

```
data(Hurricanes)
```

Format

A data frame of 46 observations of 3 variables.

`Year`

`Hurricanes` Number of Hurricanes

`ElNino` 1=Cold, 2=Neutral, 3=Warm

References

The original source of these data is the National Hurricane Center in Australia. The dataset was extracted from the table on page 5 in Silvapulle and Sen (2005).

Examples

```
head(Hurricanes)
```

iht *function for informative hypothesis testing (iht)*

Description

iht tests linear equality and/or inequality restricted hypotheses for linear models.

Usage

```
iht(...)

conTest(object, constraints = NULL, type = "summary", test = "F",
        rhs = NULL, neq = 0, ...)

conTestD(model = NULL, data = NULL, constraints = NULL,
         type = c("A","B"), R = 1000L, bootstrap.type = "bollen.stine",
         return.test = TRUE, neq.alt = 0,
         double.bootstrap = "standard", double.bootstrap.R = 249,
         double.bootstrap.alpha = 0.05,
         parallel = c("no", "multicore", "snow"),
         ncpus = 1L, cl = NULL, verbose = FALSE, ...)
```

Arguments

object	an object of class lm or rlm. In this case, the constraint syntax needs to be specified OR an object of class restriktor. The constraints are inherited from the fitted restriktor object and do not to be specified again.
model	lavaan model syntax specifying the model. See model.syntax for more information.
constraints	there are two ways to constrain parameters. First, the constraint syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of coef(model) can be used as names. See details restriktor for more information. Second, the constraint syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see the details in the restriktor function.
data	the data frame containing the observed variables being used to fit the lavaan model.
type	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.

`test` test statistic; for information about the null-distribution see details.

- for object of class lm; if "F" (default), the F-bar statistic (Silvapulle, 1996) is computed. If "LRT", a likelihood ratio test statistic (Silvapulle and Sen, 2005, chp 3.) is computed. If "score", a global score test statistic (Silvapulle and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, the usual unconstrained F-, Wald-, LR- and score-test statistic is computed.
- for object of class rlm; if "F" (default), a robust likelihood ratio type test statistic (Silvapulle, 1992a) is computed. If "Wald", a robust Wald test statistic (Silvapulle, 1992b) is computed. If "score", a global score test statistic (Silvapulle, and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, unconstrained robust F-, Wald-, score-test statistics are computed.
- for object of class glm; if "F" (default), the F-bar statistic (Silvapulle, 1996) is computed. If "LRT", a likelihood ratio test statistic (Silvapulle and Sen, 2005, chp 4.) is computed. If "score", a global score test statistic (Silvapulle and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, the usual unconstrained F-, Wald-, LR- and score-test statistic is computed.

`rhs` vector on the right-hand side of the constraints; $R\theta \geq rhs$. The length of this vector equals the number of rows of the constraints matrix R and consists of zeros by default. Note: only used if constraints input is a matrix or vector.

`neq` integer (default = 0) treating the number of constraints rows as equality constraints instead of inequality constraints. For example, if `neq = 2`, this means that the first two rows of the constraints matrix R are treated as equality constraints. Note: only used if constraints input is a matrix or vector.

`neq.alt` integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B").

`R` Integer; number of bootstrap draws. The default value is set to 1000.

`bootstrap.type` If `"parametric"`, the parametric bootstrap is used. If `"bollen.stine"`, the semi-nonparametric Bollen-Stine bootstrap is used. The default is set to `"bollen.stine"`.

`return.test` Logical; if `TRUE`, the function returns bootstrapped test-values.

`double.bootstrap`
If `"standard"` (default) the genuine double bootstrap is used to compute an additional set of plug-in p-values for each bootstrap sample. If `"no"`, no double bootstrap is used. If `"FDB"`, the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. Note that the `"FDB"` is experimental and should not be used by inexperienced users.

`double.bootstrap.R`
Integer; number of double bootstrap draws. The default value is set to 249.

`double.bootstrap.alpha`
The significance level to compute the adjusted alpha based on the plugin p-values. Only used if `double.bootstrap = "standard"`. The default value is set to 0.05.

`parallel` The type of parallel operation to be used (if any). If missing, the default is set "no".

ncpus	Integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the InformativeTesting call.
verbose	Logical; if TRUE, information is shown at each bootstrap draw.
...	futher options for the iht and/or restriktor function. See details for more information.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.
- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).
- Type C: Test H0: at least one restriction false ("<") against HA: all constraints strikty true (">"). This test is based on the intersection-union principle (Silvapulle and Sen, 2005, chp 5.3). Note that, this test only makes sense in case of no equality constraints.
- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

The null-distribution of hypothesis test Type C is based on a t-distribution (one-sided). Its power can be poor in case of many inequalty constraints. Its main role is to prevent wrong conclusions from significant results from hypothesis test Type A.

The exact finite sample distributions of the non-robust F-, score- and LR-test statistics based on restricted OLS estimates and normally distributed errors, are a mixture of F-distributions under the null hypothesis (Wolak, 1987). For the robust tests, we found that the results based on these mixtures of F-distributions approximate the tail probabilities better than their asymptotic distributions.

Note that, in case of equality constraints only, the null-distribution of the (non-)robust F-test statistics are based on an F-distribution. The (non-)robust Wald- and (non-)robust score-test statistics are based on chi-square distributions.

If object is of class lm or rlm, the conTest function internally calls the restriktor function. Arguments for the restriktor function can be passed on via the .... Additional arguments for the conTest function can also passed on via the .... See for example conTestF for all available arguments.

Value

An object of class conTest or conTestLavaan for which a print is available.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Robertson, T., Wright, F.T. and Dykstra, R.L. (1988). *Order Restricted Statistical Inference* New York: Wiley.

Shapiro, A. (1988). Towards a unified theory of inequality-constrained testing in multivariate analysis. *International Statistical Review* **56**, 49–62.

Silvapulle, M. (1992a). Robust tests of inequality constraints and one-sided hypotheses in the linear model. *Biometrika*, **79**, 621–630.

Silvapulle, M. (1992b). Robust Wald-Type Tests of One-Sided Hypotheses in the Linear Model. *Journal of the American Statistical Association*, **87**, 156–161.

Silvapulle, M. and Silvapulle, P. (1995). A score test against one-sided alternatives. *American statistical association*, **90**, 342–349.

Silvapulle, M. (1996) On an F-type statistic for testing one-sided hypotheses and computation of chi-bar-squared weights. *Statistics and probability letters*, **28**, 137–141.

Silvapulle, M. (1996) Robust bounded influence tests against one-sided hypotheses in general parametric models. *Statistics and probability letters*, **31**, 45–50.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

Vanbrabant, L., Van de Schoot, R., Van Loey, N.E.E. and Rosseel, Y. (2017). A General Procedure for Testing Inequality Constrained Hypotheses in SEM. *Methodology European Journal of Research Methods for the Behavioral and Social Sciences*, **13**, 61-70.

Van de Schoot, R., Hoijtink, H., and Dekovic, M. (2010). Testing inequality constrained hypotheses in SEM models. *Structural Equation Modeling*, **17**, 443-463.

Van de Schoot, R., Strohmeier, D. (2011). Testing informative hypotheses in SEM increases power: An illustration contrasting classical. *International Journal of Behavioral Development*, **35**, 180-190.

Wolak, F. (1987). An exact test for multiple inequality and equality constraints in the linear regression model. *Journal of the American statistical association*, **82**, 782–793.

See Also

quadprog, conTest

Examples

```
## example 1:
# the data consist of ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose constraints on
# the corresponding regression parameters.
coef(fit1.lm)
```

```
# constraint syntax: assuming that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
myConstraints1 <- ' GroupActive  < GroupPassive;
                    GroupPassive < GroupNo '

iht(fit1.lm, myConstraints1)


# another way is to first fit the restricted model
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

iht(fit.restr1)

## Not run:
  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

  iht(fit1.lm, constraints = Amat1,
      rhs = myRhs1, neq = myNeq1)

## End(Not run)

#########################
## Artificial examples ##
#########################
# generate data
n <- 10
means <- c(1,2,1,3)
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)

# fit unrestricted linear model
fit2.lm <- lm(y ~ -1 + group, data = DATA2)
coef(fit2.lm)

## example 2: increasing means
myConstraints2 <- ' group1 < group2
                    group2 < group3
                    group3 < group4 '

# compute F-test for hypothesis test Type A and compute the tail
# probability based on the parametric bootstrap. We only generate 9
# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
iht(fit2.lm, constraints = myConstraints2, type = "A",
    boot = "parametric", R = 9)
```

```
# or fit restricted linear model
fit2.con <- restriktor(fit2.lm, constraints = myConstraints2)

iht(fit2.con)

## Not run:
  # increasing means in matrix notation.
  Amat2 <- rbind(c(-1, 1, 0, 0),
                 c( 0,-1, 1, 0),
                 c( 0, 0,-1, 1))
  myRhs2 <- rep(0L, nrow(Amat2))
  myNeq2 <- 0

  iht(fit2.con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
      type = "A", boot = "parametric", R = 9)

## End(Not run)

## example 3: equality constraints only.
myConstraints3 <- ' group1 = group2
                    group2 = group3
                    group3 = group4 '

iht(fit2.lm, constraints = myConstraints3)

# or
fit3.con <- restriktor(fit2.lm, constraints = myConstraints3)
iht(fit3.con)


## example 4:
# combination of equality and inequality constraints.
myConstraints4 <- ' group1 = group2
                    group3 < group4 '

iht(fit2.lm, constraints = myConstraints4, type = "B", neq.alt = 1)

# fit resticted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
# Note that, a warning message may be thrown because the number of
# bootstrap samples is too low.
fit4.con <- restriktor(fit2.lm, constraints = myConstraints4,
                       se = "boot.model.based", B = 9)
iht(fit4.con, type = "B", neq.alt = 1)


## example 5:
# restriktor can also be used to define effects using the := operator
# and impose constraints on them. For example, is the
# average effect (AVE) larger than zero?
```

```
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit5.lm <- lm(y ~ X*Z, data = DATA3)

# constraint syntax
myConstraints5 <- ' AVE := X + 16.86137*X.Z;
                    AVE > 0 '

iht(fit5.lm, constraints = myConstraints5)

# or
fit5.con <- restriktor(fit5.lm, constraints = ' AVE := X + 16.86137*X.Z;
                                                 AVE > 0 ')
iht(fit5.con)


## Not run:
  # testing equality and/or inequality restrictions in SEM:

  ##########################
  ### real data example ###
  ##########################
  # Multiple group path model for facial burns example.

  # model syntax with starting values.
  burns.model <- 'Selfesteem ~ Age + c(m1, f1)*TBSA + HADS +
                               start(-.10, -.20)*TBSA
              HADS ~ Age + c(m2, f2)*TBSA + RUM +
                     start(.10, .20)*TBSA '


  # constraints syntax
  burns.constraints <- 'f2 > 0  ; m1 < 0
                        m2 > 0  ; f1 < 0
                        f2 > m2 ; f1 < m1'

  # we only generate 2 bootstrap samples in this example; in practice
  # you may wish to use a much higher number.
  # the double bootstrap was switched off; in practice you probably
  # want to set it to "standard".
  example6 <- iht(model = burns.model, data = FacialBurns,
                  R = 2, constraints = burns.constraints,
                  double.bootstrap = "no", group = "Sex")

  example6
```

```
  ##########################
  ### artificial example ###
  ##########################
  # Simple ANOVA model with 3 groups (N = 20 per group)
  set.seed(1234)
  Y <- cbind(c(rnorm(20,0,1), rnorm(20,0.5,1), rnorm(20,1,1)))
  grp <- c(rep("1", 20), rep("2", 20), rep("3", 20))
  Data <- data.frame(Y, grp)

  #create model matrix
  fit.lm <- lm(Y ~ grp, data = Data)
  mfit <- fit.lm$model
  mm <- model.matrix(mfit)

  Y <- model.response(mfit)
  X <- data.frame(mm[,2:3])
  names(X) <- c("d1", "d2")
  Data.new <- data.frame(Y, X)

  # model
  model <- 'Y ~ 1 + a1*d1 + a2*d2'

  # fit without constraints
  fit <- lavaan::sem(model, data = Data.new)

  # constraints syntax: mu1 < mu2 < mu3
  constraints <- ' a1 > 0
                   a1 < a2 '

  # we only generate 10 bootstrap samples in this example; in practice
  # you may wish to use a much higher number, say > 1000. The double
  # bootstrap is not necessary in case of an univariate ANOVA model.
  example7 <- iht(model = model, data = Data.new,
                  start = lavaan::parTable(fit),
                  R = 10L, double.bootstrap = "no",
                  constraints = constraints)
  example7

## End(Not run)
```

---

iht-methods *Methods for iht*

Description

Print function for objects of class conTest.

Usage

```
## S3 method for class 'conTest'
print(x, digits = max(3, getOption("digits") - 2), ...)
```

Arguments

x	an object of class conTest.
digits	the number of significant digits to use when printing.
...	no additional arguments for now.

Examples

```
# unrestricted linear model for ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit.lm <- lm(Age ~ -1 + Group, data = DATA)

# restricted linear model with restrictions that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
fit.con <- restriktor(fit.lm, constraints = "GroupActive  < GroupPassive;
                                             GroupPassive < GroupNo")

iht(fit.con)
```

---

restriktor *Estimating linear regression models with (in)equality restrictions*

Description

Function restriktor estimates the parameters of an univariate and a multivariate linear model (lm), a robust estimation of the linear model (rlm) and a generalized linear model (glm) subject to linear equality and linear inequality restrictions. It is a convenience function. The real work horses are the conLM, conMLM, conRLM and the conGLM functions.

Usage

```
restriktor(object, constraints = NULL, ...)

## S3 method for class 'lm'
conLM(object, constraints = NULL, se = "standard",
```

```
      B = 999, rhs = NULL, neq = 0L, mix.weights = "pmvnorm",
      mix.bootstrap = 99999L, parallel = "no", ncpus = 1L,
      cl = NULL, seed = NULL, control = list(),
      verbose = FALSE, debug = FALSE, ...)

## S3 method for class 'rlm'
conRLM(object, constraints = NULL, se = "standard",
       B = 999, rhs = NULL, neq = 0L, mix.weights = "pmvnorm",
       mix.bootstrap = 99999L, parallel = "no", ncpus = 1L,
       cl = NULL, seed = NULL, control = list(),
       verbose = FALSE, debug = FALSE, ...)

## S3 method for class 'glm'
conGLM(object, constraints = NULL, se = "standard",
       B = 999, rhs = NULL, neq = 0L, mix.weights = "pmvnorm",
       mix.bootstrap = 99999L, parallel = "no", ncpus = 1L,
       cl = NULL, seed = NULL, control = list(),
       verbose = FALSE, debug = FALSE, ...)

## S3 method for class 'mlm'
conMLM(object, constraints = NULL, se = "none",
       B = 999, rhs = NULL, neq = 0L, mix.weights = "pmvnorm",
       mix.bootstrap = 99999L, parallel = "no", ncpus = 1L,
       cl = NULL, seed = NULL, control = list(),
       verbose = FALSE, debug = FALSE, ...)
```

Arguments

`object`	a fitted linear model object of class "lm", "mlm", "rlm" or "glm". For class "rlm" only the loss function `bisquare` is supported for now, otherwise the function gives an error.
`constraints`	there are two ways to constrain parameters. First, the constraint syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of `coef(model)` can be used as names. See details for more information. Note that objects of class "mlm" do not (yet) support this method. Second, the constraint syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see details.
`se`	if "`standard`" (default), conventional standard errors are computed based on inverting the observed augmented information matrix. If "const", homoskedastic standard errors are computed. If "`HC0`" or just "`HC`", heteroskedastic robust standard errors are computed (a.k.a Huber White). The options "`HC1`", "`HC2`", "`HC3`", "`HC4`", "`HC4m`", and "`HC5`" are refinements of "`HC0`". For more details about heteroskedastic robust standard errors see the **sandwich** pack-

	age. If "boot.standard", bootstrapped standard errors are computed using standard bootstrapping. If "boot.model.based" or "boot.residual", bootstrapped standard errors are computed using model-based bootstrapping. If "none", no standard errors are computed. Note that for objects of class "mlm" no standard errors are available (yet).
B	integer; number of bootstrap draws for se. The default value is set to 999. Parallel support is available.
rhs	vector on the right-hand side of the constraints; $R\theta \geq rhs$. The length of this vector equals the number of rows of the constraints matrix R and consists of zeros by default. Note: only used if constraints input is a matrix or vector.
neq	integer (default = 0) treating the number of constraints rows as equality constraints instead of inequality constraints. For example, if neq = 2, this means that the first two rows of the constraints matrix R are treated as equality constraints. Note: only used if constraints input is a matrix or vector.
mix.weights	if "pmvnorm" (default), the chi-bar-square weights are computed based on the multivariate normal distribution function with additional Monte Carlo steps. If "boot", the chi-bar-square weights are computed using parametric bootstrapping. If "none", no chi-bar-square weights are computed. The weights are necessary in the restriktor.summary function for computing the GORIC. Moreover, the weights are re-used in the iht function for computing the p-value for the test-statistic, unless the p-value is computed directly via bootstrapping.
mix.bootstrap	integer; number of bootstrap draws for mix.weights = "boot". The default value is set to 99999. Parallel support is available.
parallel	the type of parallel operation to be used (if any). If missing, the default is set "no".
ncpus	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
cl	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the restriktor call.
seed	seed value.
control	a list of control arguments: • absval tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). • maxit the maximum number of iterations for the optimizer (default = 10000). • tol numerical tolerance value. Estimates smaller than tol are set to 0.
verbose	logical; if TRUE, information is shown at each bootstrap draw.
debug	if TRUE, debugging information about the constraints is printed out.
...	no additional arguments for now.

Details

The constraint syntax can be specified in two ways. First as a literal string enclosed by single quotes as shown below:

```
myConstraints <- '
    # 1. inequality constraints
      x1 > 0
      x1 < x2
    # or
      0 < x1 < x2

    ! 2. equality constraints
      x3 == x4; x4 == x5
    # or
      x3 = x4; x4 = x5
    # or
      x3 = x4 = x5'
```

The variable names x1 to x5 refer to the corresponding regression coefficient. Thus, constraints are impose on regression coefficients and not on the data.

Second, the above constraints syntax can also be written in matrix/vector notation as:

(The first column refers to the intercept, the remaining five columns refer to the regression coefficients x1 to x5.)

```
myConstraints <-
    rbind(c(0, 0, 0,-1, 1, 0), #equality constraint x3 = x4
          c(0, 0, 0, 0,-1, 1), #equality constraint x4 = x5
          c(0, 1, 0, 0, 0, 0), #inequality constraint x1 > rhs
          c(0,-1, 1, 0, 0, 0)) #inequality constraint x1 < x2

# the length of rhs is equal to the number of myConstraints rows.
myRhs <- c(0,0,0,0)

# the first two rows should be considered as equality constraints
myNeq <- 2
```

Blank lines and comments can be used in between the constraints, and constraints can be split over multiple lines. Both the hashtag (#) and the exclamation (!) characters can be used to start a comment. Multiple constraints can be placed on a single line if they are separated by a semicolon (;), a comma (,) or the "&" sign.

In addition compound constraints can be stated via one or more longer equality or inequality sentences e.g., 'x1 > x2 > x3; x3 < 4 < x4' or 'x1 == x2 == x3 & x4 = 1'. Alternatively, the constrains can be specifies as '(x1, x2) > (x3, x4)' which is equivalent to 'x1 > x3; x1 > x4; x2 > x3; x2 > x4'.

There can be three types of text-based descriptions in the constraints syntax:

1. Equality constraints: The "==" or "=" operator can be used to define equality constraints (e.g., x1 = 1 or x1 = x2).
2. Inequality constraints: The "<" or ">" operator can be used to define inequality constraints (e.g., x1 > 1 or x1 < x2).

3. Newly defined parameters: The ":=" operator can be used to define new parameters, which take on values that are an arbitrary function of the original model parameters. The function must be specified in terms of the parameter names in `coef(model)` (e.g., `new := x1 + 2*x2`). By default, the standard errors for these defined parameters are computed by using the so-called Delta method.

Variable names of interaction effects in objects of class lm, rlm and glm contain a semi-colon (:) between the variables. To impose constraints on parameters of interaction effects, the semi-colon must be replaced by a dot (.) (e.g., `x3:x4` becomes `x3.x4`). In addition, the intercept variable names is shown as "`(Intercept)`". To impose restrictions on the intercept both parentheses must be replaced by a dot "`.Intercept.`" (e.g.,`.Intercept. > 10`). Note: in most practical situations we do not impose restrictions on the intercept because we do not have prior knowledge about the intercept. Moreover, the sign of the intercept can be changed arbitrarily by shifting the response variable y.

Each element can be modified using arithmetic operators. For example, if `x2` is expected to be twice as large as `x1`, then "`2*x2 = x1`".

If `constraints = NULL`, the unrestricted model is fitted.

Value

An object of class restriktor, for which a print and a summary method are available. More specifically, it is a list with the following items:

`CON`	a list with useful information about the restrictions.
`call`	the matched call.
`timing`	how much time several tasks take.
`parTable`	a parameter table with information about the observed variables in the model and the imposed restrictions.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`residuals`	restricted residuals.
`wresid`	a working residual, weighted for "inv.var" weights only (rlm only)
`fitted`	restricted fitted mean values.
`weights`	(only for weighted fits) the specified weights.
`wgt`	the weights used in the IWLS process (rlm only).
`scale`	the robust scale estimate used (rlm only).
`stddev`	a scale estimate used for the standard errors.
`R2.org`	unrestricted R-squared.
`R2.reduced`	restricted R-squared.
`df.residual`	the residual degrees of freedom
`s2.unrestr`	mean squared error of unrestricted model.
`s2.restr`	mean squared error of restricted model.
`loglik`	restricted log-likelihood.

Sigma	variance-covariance matrix of unrestricted model.
constraints	matrix with restrictions.
rhs	vector of right-hand side elements.
neq	number of equality restrictions.
wt.bar	chi-bar-square mixing weights or a.k.a. level probabilities.
iact	active restrictions.
converged	did the IWLS converge (rlm only)?
iter	number of iteration needed for convergence (rlm only).
bootout	object of class boot. Only available if bootstrapped standard errors are requested, else bootout = NULL.
control	list with control options.
model.org	original model.
se	as input. This information is needed in the summary function.
information	observed information matrix with the inverted information matrix and the augmented information matrix as attributes.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

Schoenberg, R. (1997). Constrained Maximum Likelihood. *Computational Economics*, **10**, 251–266.

Shapiro, A. (1988). Towards a unified theory of inequality-constrained testing in multivariate analysis. *International Statistical Review* **56**, 49–62.

Silvapulle, M.J. and Sen, P.K. (2005). *Constrained Statistical Inference*. Wiley, New York

See Also

iht, goric

Examples

```
## lm
## unrestricted linear model for ages (in months) at which an
## infant starts to walk alone.

# prepare data
DATA1 <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit1.lm <- lm(Age ~ -1 + Group, data = DATA1)

# the variable names can be used to impose restrictions on
# the corresponding regression parameters.
```

```
coef(fit1.lm)

# restricted linear model with restrictions that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
fit1.con <- restriktor(fit1.lm, constraints = ' GroupActive  < GroupPassive;
                                                GroupPassive < GroupNo ')
summary(fit1.con)

## Not run:
# Or in matrix notation.
myConstraints1 <- rbind(c(-1, 1, 0),
                        c( 0,-1, 1))
myRhs1 <- rep(0L, nrow(R1))
myNeq1 <- 0

fit1.con <- restriktor(fit1.lm, constraints = myConstraints1,
                       rhs = myRhs1, neq = myNeq1)
summary(fit1.con)

## End(Not run)

##########################
## Artificial examples ##
##########################
library(MASS)

## mlm
# generate data
n <- 30
mu <- rep(0, 4)
Sigma <- matrix(5,4,4)
  diag(Sigma) <- c(10,10,10,10)
# 4 Y's.
Y <- mvrnorm(n, mu, Sigma)

# fit unrestricted multivariate linear model
fit.mlm <- lm(Y ~ 1)

# constraints
myConstraints2 <- diag(0,4)
  diag(myConstraints2) <- 1

# fit restricted multivariate linear model
fit2.con <- restriktor(fit.mlm, constraints = myConstraints2)

summary(fit2.con)


## rlm
# generate data
n <- 10
means <- c(1,2,1,3)
```

```
nm <- length(means)
group <- as.factor(rep(1:nm, each = n))
y <- rnorm(n * nm, rep(means, each = n))
DATA2 <- data.frame(y, group)

# fit unrestricted robust linear model
fit3.rlm <- rlm(y ~ -1 + group, data = DATA2, method = "MM")
coef(fit3.rlm)

## increasing means
myConstraints3 <- ' group1 < group2
                    group2 < group3
                    group3 < group4 '

# fit restricted robust linear model and compute
# Huber-White (robust) standard errors.
fit3.con <- restriktor(fit3.rlm, constraints = myConstraints3,
                       se = "HC0")
summary(fit3.con)

## Not run:
## increasing means in matrix notation.
myConstraints3 <- rbind(c(-1, 1, 0, 0),
                        c( 0,-1, 1, 0),
                        c( 0, 0,-1, 1))
myRhs3 <- rep(0L, nrow(myConstraints3))
myNeq2 <- 0

fit3.con <- restriktor(fit3.rlm, constraints = myConstraints3,
                       rhs = myRhs2, neq = myNeq2, se = "HC0")
summary(fit3.con)

## End(Not run)

## equality restrictions only.
myConstraints4 <- ' group1 = group2
                    group2 = group3
                    group3 = group4 '

fit4.con <- restriktor(fit3.rlm, constraints = myConstraints4)
summary(fit4.con)


## combination of equality and inequality restrictions.
myConstraints5 <- ' group1  = group2
                    group3  < group4 '

# fit restricted model and compute model-based bootstrapped
# standard errors. We only generate 9 bootstrap samples in this
# example; in practice you may wish to use a much higher number.
fit5.con <- restriktor(fit3.rlm, constraints = myConstraints4,
                       se = "boot.model.based", B = 9)
# an error is probably thrown, due to a too low number of bootstrap draws.
```

```
summary(fit5.con)

# restriktor can also be used to define effects using the := operator
# and impose restrictions on them. For example, compute the average
# effect (AVE) and impose the restriction AVE > 0.
# generate data
n <- 30
b0 <- 10; b1 = 0.5; b2 = 1; b3 = 1.5
X <- c(rep(c(0), n/2), rep(c(1), n/2))
set.seed(90)
Z <- rnorm(n, 16, 5)
y <- b0 + b1*X + b2*Z + b3*X*Z + rnorm(n, 0, sd = 10)
DATA3 = data.frame(cbind(y, X, Z))

# fit linear model with interaction
fit6.lm <- lm(y ~ X*Z, data = DATA3)

fit6.con <- restriktor(fit6.lm, constraints = ' AVE := X + 16.86137*X.Z;
                                                AVE > 0 ')
summary(fit6.con)
```

`restriktor-methods` *Methods for restriktor*

Description

restricted estimation and confidence intervals for (robust) linear (in)equality restricted hypotheses.

Usage

```
## S3 method for class 'restriktor'
print(x, digits = max(3, getOption("digits") - 2), ...)

## S3 method for class 'restriktor'
summary(object, bootCIs = TRUE,
        bty = "perc", level = 0.95, goric = "goric", ...)

## S3 method for class 'summary.restriktor'
print(x, digits = max(3, getOption("digits") - 2),
              signif.stars = getOption("show.signif.stars"), ...)

## S3 method for class 'restriktor'
coef(object, ...)

## S3 method for class 'restriktor'
model.matrix(object, ...)

## S3 method for class 'restriktor'
```

```
logLik(object, ...)
```

Arguments

object	an object of class restriktor.
x	an object of class restriktor.
bootCIs	if TRUE (default), nonparametric bootstrap confidence intervals are generated. Only available if object contains bootout object.
bty	a character string representing the type of interval required. The value should be any of the values "norm", "basic","perc","bca". The value "stud" is not supported. For more details see boot.ci.
level	the confidence level of the interval (default = 0.95).
goric	if "goric" (default), the generalized order-restricted information criterion value is computed. If "gorica" the log-likihood is computed using the multivariate normal distribution function. If "goricc" or "goricca", a small sample version of the "goric" or "gorica" is computed.
digits	the number of significant digits to use when printing.
signif.stars	If TRUE, "significance stars are printed for each coefficient.
...	no additional arguments for now.

Details

The function print returns the restricted coefficients. The output from the print.summary.conLM function provides information that is comparable with the output from print.summary.lm. Additional information is provided about the unrestricted and restricted R-square and by default the output of the GORIC. If bootstrapped standard errors are requested (e.g., option se = "boot.model.based" in the restriktor function and bootCI = TRUE in the summary function) standard errors and confidence intervals are provided.

Value

The function summary computes and returns a list of summary statistics of the fitted unrestricted and restricted (robust) linear model given in object, plus

se.type	type of standard error computed, equal to input se in the restriktor function.
residuals	the weighted residuals.
coefficients	a p x 4 matrix with columns for the estimated coefficient, its standard error, t-statistic and corresponding p-value. If bootCIs = TRUE and the bootout object is available in the object, bootstrapped standard errors and confidence intervals are produced.
rdf	residual degrees of freedom.
R2.org	unrestricted R-squared.
R2.reduced	restricted R-squared.
goric	goric value and attributed its penalty term and log-likelihood.

Examples

```
# unrestricted linear model for ages (in months) at which an
# infant starts to walk alone.

# prepare data
DATA <- subset(ZelazoKolb1972, Group != "Control")

# fit unrestricted linear model
fit.lm <- lm(Age ~ -1 + Group, data = DATA)

# restricted linear model with restrictions that the walking
# exercises would not have a negative effect of increasing the
# mean age at which a child starts to walk.
fit.con <- restriktor(fit.lm, constraints = ' GroupActive  < GroupPassive;
                                              GroupPassive < GroupNo ')

summary(fit.con)
```

ZelazoKolb1972 *"Walking" in the newborn (4 treatment groups)*

Description

The Zelazo, Zelazo and Kolb (1972) dataset consists of ages (in months) at which an infant starts to walk alone from four different treatment groups (Active-exercise, Passive-exercise, 8 week Control, No-exercise).

Usage

```
data(ZelazoKolb1972)
```

Format

A data frame of 23 observations of 4 treatment variables.

`Age` Age in months

`Group` Active-exercise, Passive-exercise, 8-week Control group, No-exercise

References

Zelazo, P.R., Zelazo, N.A., and Kolb, S. (1972). "Walking in the Newborn". Science, New Series, 176, 314-315

Examples

```
head(ZelazoKolb1972)
```

Index