Package ‘restriktor’

May 24, 2019

Title Restricted Statistical Estimation and Inference for Linear Models

Version 0.2-15

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 (>= 3.0.0)

Imports boot, ic.infer, lavaan, MASS, mvtnorm, quadprog

License GPL (>= 2)

LazyData yes

URL http://restriktor.org

NeedsCompilation no

Author Leonard Vanbrabant [aut, cre], Yves Rosseel [ctb]

Maintainer Leonard Vanbrabant <Leonard.Vanbrabant@Ugent.be>

Suggests

Repository CRAN

Date/Publication 2019-05-24 07:20:02 UTC

R topics documented:

restriktor-package .................................................. 2
AngerManagement .................................................... 4
Burns ........................................................................ 5
conTest ................................................................. 6
conTest-methods ..................................................... 12
conTestC ............................................................... 13
conTestF ............................................................... 15
conTestLRT ........................................................... 21
conTestScore ......................................................... 26
conTestWald .......................................................... 32
conTest_ceq .......................................................... 37
restriktor-package

Package for equality and inequality restricted estimation and hypothesis testing

Description

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

Details

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 restrictions. 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 restrictions. 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 criterion 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


Vanbrabant, L. and Kuiper, R. (n.d.). Giving the complement a compliment: Evaluating a theory-based hypothesis against its complement using the GORIC.

**See Also**

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

**Examples**

```r
# Data preparation
# Ages (in months) at which an infant starts to walk alone.
DATA <- ZelazoKolb1972
idx <- which(DATA$Group == "Control")
DATA <- DATA[-idx, ]

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


Examples

head(AngerManagement)

<table>
<thead>
<tr>
<th>Burns</th>
<th>Relation between the response variable PTSS and gender, age, TBSA, guilt and anger.</th>
</tr>
</thead>
</table>

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


Examples

head(burns)

---

conTest function for informative hypothesis testing (iht)

Description

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

Usage

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

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.

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 ($\theta$) 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.

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.

... further options for the conTest 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 strictly 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 inequality 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`, 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


See Also

quadprog, conTest

Examples

```r
## 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 '

conTest(fit1.lm, myConstraints1)
```
another way is to first fit the restricted model

```r
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

conTest(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

conTest(fit1.lm, constraints = Amat1,
         rhs = myRhs1, neq = mYNeq1)
```

## End(Not run)

### Artificial examples ##

```r
# 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 <- c("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.
conTest(fit2.lm, constraints = myConstraints2, type = "A",
        boot = "parametric", R = 9)
```

## or fit restricted linear model

```r
fit2.con <- restriktor(fit2.lm, constraints = myConstraints2)

conTest(fit2.con)
```

## Not run:

```
# increasing means in matrix notation.
Amat2 <- rbind(c(-1, 1, 0, 0),
               c( 0, 1, -1, 0))
```
```r

c( 0,-1, 1, 0),
c( 0, 0,-1, 1))
myRhs2 <- rep(0L, nrow(Amat2))
myNeq2 <- 0

conTest(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'

conTest(fit2.lm, constraints = myConstraints3)

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

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

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

# 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.
# 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)
conTest(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
```
Methods for `conTest`

Description

Print function for objects of class `conTest`.

Usage

```r
## 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

```r
# 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")
```
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

```r
## 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 inequality 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

See Also
quadprog, conTest

Examples

```r
## 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'

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

# another way is to first fit the restricted model
```
conTestF

```r
fit.restr1 <- restriktor(fit1.lm, constraints = myConstraints1)

conTest(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 <- restrriktor(fit1.lm, constraints = Amat1,
                        rhs = myRhs1, neq = myNeq1)
conTest(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

```r
## S3 method for class 'conLM'
conTestF(object, type = "A", neq.alt = 0,
         boot = "no", R = 99999, 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 = 99999, 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 = 99999, 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 variance 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 object 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:

• absv1 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.
conTestF

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 strictly 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 inequality 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

See Also
quadprog, conTest

Examples

```r
## 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 '

conTest(fit1.lm, myConstraints1)

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

conTest(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

      conTest(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.
conTest(fit2.lm, constraints = myConstraints2, type = "A",
       boot = "parametric", R = 9)

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

conTest(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

conTest(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 '

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

# 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.
# 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)
conTest(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 '
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

```r
# 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 variance one. See \texttt{p.distr} for other distribu-
tional 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 com-
puted using the chi-bar-square weights. If "no", the p-value is computed based
on the weights obtained via simulation (\texttt{mix.weights = "boot"}) or using the
multivariate normal distribution function (\texttt{mix.weights = "pmvnorm"}). Note
that, these weights are already available in the restrktor objected and do not
need to be estimated again. However, there are two exception for objects of
class \texttt{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.

\textbf{R}

integer; number of bootstrap draws for boot. The default value is set to 9999.

\textbf{p.distr}

random generation distribution for the parametric bootstrap. For all available
distributions see \texttt{?distributions}. For example, if \texttt{rnorm}, samples are drawn
from the normal distribution (default) with mean zero and variance one. If \texttt{rt},
samples are drawn from a t-distribution. If \texttt{rchisq}, samples are drawn from a
chi-square distribution. The random generation distributional parameters will
be passed in via ....

\textbf{parallel}

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

\textbf{ncpus}

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

\textbf{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.

\textbf{seed}

seed value. The default value is set to 1234.

\textbf{verbose}

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

\textbf{control}

a list of control arguments:

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

\textbf{...}

additional arguments to be passed to the \texttt{p.distr} function.

\section*{Details}

The following hypothesis tests are available:

\begin{itemize}
  \item \textbf{Type A}: Test H0: all constraints with equalities ("=") active against HA: at least one inequality
    restriction (">") strictly true.
  \item \textbf{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).
  \item \textbf{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.
\end{itemize}
• 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 inequality 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
- amat
- bvec
- meq
- meq_alt
- iact
- type
- test
- ts
- df.residual
- pvalue
- b_eqrestr
- b_unrestr
- b_restr
- b_restr_alt
- Sigma
- R2.org
- R2_reduced
- boot
- model_org

Author(s)

Leonard Vanbrabant and Yves Rosseel
References


See Also

quadprog, conTest

Examples

```r
## 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 '

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

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

conTest(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

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

## End(Not run)

```

# Artificial examples #

```

# Not run
```
# 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 <- c('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.
conTest(fit2_lm, constraints = myconstraints2, type = "A", test = "LRT",
        boot = "parametric", R = 9)

# or fit restricted linear model
fit2_con <- restriktor(fit2_lm, constraints = myconstraints2)

conTest(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
conTest(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 <- c('group1 == group2,
                     group3 < group4')

conTest(fit2_lm, constraints = myconstraints3, type = "B",
        test = "LRT", neq.alt = 1)

# 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.
# 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)
conTest(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 '

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

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

```r
## 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,
```
conTestScore

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 variance one. See p.distr for other distributional options. If "modelbased", 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 object 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.
conTestScore

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 strictly 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 inequality 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

See Also
quadprog, conTest

Examples
```r
## 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'

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

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

conTest(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

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

## End(Not run)

#############################################################
## Artificial examples ##
#############################################################

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

# bootstrap samples in this example; in practice you may wish to
# use a much higher number.
conTest(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

conTest(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'

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

# 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.
# 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)
conTest(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 '

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

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

---

**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**

```R
## 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 variance 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 object 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. Ifrchisq, 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 strictly 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 inequality 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

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 '

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

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

conTest(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

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

## End(Not run)

# Artifical 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.
conTest(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)

cutTest(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
conTest(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 '

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

# fit robust 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.
# 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)
conTest(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 '

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

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

```r
## 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, ...)
```

```r
## 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 choose this to the number of available CPUs.

c1 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

See Also
quadprog, conTest

Examples
```r
## 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'

conTest(fit1.lm, myConstraints1)

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

conTest(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

conTest(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 an 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**

```r
## 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 conTest 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


See Also

`quadprog, conTest`

Examples

```r
## 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"

conTest(fit1.lm, myConstraints1)

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

conTest(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)
conTest(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 = 9999L, 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 (\( \theta \)). 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 support is available.
Exam 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

Examples
```r
W <- matrix(c(1,0.5,0.5,1),2,2)
Amat <- rbind(c(0,1))
meq <- 0
# we only generate 999 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 = 999)
wt.bar
```

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
```r
data(Exam)
```
**goric**

**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**

```r
head(Exam)
```

---

<table>
<thead>
<tr>
<th>goric</th>
<th>Generalized Order-Restrikted Information Criterion (Approximation)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Weights</td>
</tr>
</tbody>
</table>

**Description**

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

**Usage**

```r
goric(object, ..., comparison = c("unconstrained", "complement", "none"),
   VCOV = NULL, type = "goric", bound = NULL, debug = FALSE)
```

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

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

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

**Arguments**

<table>
<thead>
<tr>
<th>object</th>
<th>an object containing the outcome of a statistical analysis. Currently, the following objects can be processed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• a fitted object of class <code>restriktor</code>.</td>
</tr>
<tr>
<td></td>
<td>• a fitted unconstrained object of class <code>lm</code>, <code>rlm</code> or <code>glm</code>.</td>
</tr>
<tr>
<td></td>
<td>• a numeric vector containing the unconstrained estimates resulting from any statistical analysis.</td>
</tr>
</tbody>
</table>
x  
an object of class goric.

...  
this depends on the class of the object. If object is of class restriktor, further objects of class restriktor can be passed. If object is of class glm, rlm or glm, the constraints can be passed. If object is of class numeric, the constraints can be passed. See details for more information.

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-covariance matrix. Only needed if object is of class numeric and type = "gorica".

type  
if "gorica", the log-likelihood is computed using the multivariate normal distribution function.

bound  
not used yet.

digits  
the number of significant digits to use when printing.

debug  
if TRUE, debugging information is printed out.

brief  
if FALSE, an extended 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 $H_1$, the complement $H_c$ can be computed as competing hypothesis. The complement is defined as $H_c = \neg H_1$.

If the object(s) is of class restriktor the constraints are automatically extracted. Otherwise, the constraint syntax can be parsed via the ... If the object is an unconstrained model of class glm, 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. For example, \texttt{h1 <- list(constraints = "x1 > 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.

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


Vanbrabant, L. and Kuiper, R. (n.d.). Giving the complement a compliment: Evaluating a theory-based hypothesis against its complement using the GORIC.

Examples

```r
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 unrestricted linear model
fit1.lm <- lm(Age ~ Group, data = DATA)

# some artificial restrictions
fit1.con <- restrktor(fit1.lm, constraints = "GroupPassive > 0; GroupPassive < GroupNo")
fit2.con <- restrktor(fit1.lm, constraints = "GroupPassive > 0; GroupPassive > GroupNo")
fit3.con <- restrktor(fit1.lm, constraints = "GroupPassive == 0; GroupPassive < GroupNo")
fit4.con <- restrktor(fit1.lm) # unrestricted model

goric(fit1.con, fit2.con, fit3.con, fit4.con)

# fit1.con versus the complement
goric(fit1.con, 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
out <- goric(fit.lm, h1, h2, type = "gorica")
# use unconstrained estimates
```

```
out <- goric(est, VCOV = VCOV, h1, h2, type = "gorica")

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

## mlm
# generate data
n <- 30
mu <- c(1,2,3,4)
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
fit2.mlm <- lm(Y ~ 1)

# constraints
myConstraints2 <- rbind(c(-1,1,0,0), c(0,-1,1,0), c(0,0,-1,1))

# fit restricted multivariate linear model
fit5.con <- restriktor(fit2.mlm, constraints = myConstraints2)
```

---

**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)

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,
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 ($\theta$) 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 package. 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.

c1  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

  ! 2. equality constraints
  x3 == x4; 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
(;
).

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

1. Equality constraints: The "==" operator can be used to define equality constraints (e.g., \(x_1 = 1\)
or \(x_1 = x_2\)).
2. Inequality constraints: The "<" or ">" operator can be used to define inequality constraints (e.g., \(x_1 > 1\) or \(x_1 < x_2\)).
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 \(\text{coef(model)}\) (e.g., \(\text{new} := x_1 + 2 \times x_2\)).

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 \text{lm}, \text{rlm} and \text{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., \(x_3 \times x_4\) becomes \(x_3 \cdot x_4\)). In addition, the intercept variable
names is shown as "\(\text{Intercept}\)". To impose restrictions on the intercept both parentheses must
be replaced by a dot "\(..\)" (e.g., \(\text{Intercept} \) > 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 \(x_2\) is expected to be twice
as large as \(x_1\), then "\(2 \times x_2 = x_1\)".

If constraints = \text{NULL}, the unrestricted model is fitted.

### Value

An object of class \text{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 (\text{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

See Also
iht, goric
Examples

```r
## 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 <- restrktor(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 <- restrktor(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.mlmg, constraints = myConstraints2)
summary(fit2.con)

## rlm
# generate data
n <- 10
means <- c(1, 2, 1, 3)
mm <- length(means)
group <- as.factor(rep(1:mm, each = n))
y <- rnorm(n * mm, 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 = myConstraints2,
                      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, 0, 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
Methods for `restriktor`

**Description**

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

**Usage**

```r
## 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 = TRUE, ...)

## 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 TRUE (default), the Generalized Order-restricted Information Criterion value is computed, the penalty term and the log-likelihood. The GORIC is a modification of the AIC and the ORIC such that it can be applied to a (in)equality restricted hypotheses in multivariate normal linear models. For more information see the goric package.
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

```r
# 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)
```

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


Examples

head(ZelazoKolb1972)
Index

AngerManagement, 4

boot.ci, 57
Burns, 5

c coef.goric (goric), 45
c coef.restriktor (restriktor-methods), 56
c con_weights.boot, 43
c conGLM glm (restriktor), 49
c conLM lm (restriktor), 49
c conMLM glm (restriktor), 49
c conRLM rlm (restriktor), 49
c conTest, 6, 9, 14, 18, 24, 29, 35, 39, 41, 42
c conTest-methods, 12
c conTest_ceq, 37
c conTest_summary, 40
c conTestC, 13
c conTestF, 8, 15
c conTestLRT, 21
c conTestScore, 26
c conTestWald, 32

Exam, 44
goric, 45, 53

Hurricanes, 48
iht, 4, 50, 53
iht (conTest), 6
iht-methods (conTest-methods), 12

logLik.restriktor (restriktor-methods), 56

model.matrix.restriktor
 (restriktor-methods), 56

print.conTest (conTest-methods), 12
print.goric (goric), 45
print.restriktor (restriktor-methods), 56

quadprog, 9, 14, 18, 24, 29, 35, 39, 42
restriktor, 4, 6, 8, 43, 46, 49
restriktor-methods, 56
restriktor-package, 2

summary.goric (goric), 45
summary.restriktor
 (restriktor-methods), 56

ZelazoKolb1972, 58

60