Package ‘restriktor’

February 5, 2026

Title Restricted Statistical Estimation and Inference for Linear Models

Version 0.6-30

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

Depends R(>= 4.0.0)

Imports boot, lavaan(>= 0.6-10), MASS, mvtnorm, tmvtnorm, quadprog, norm, ggplot2, future, future.apply, progressr, scales, gridExtra

License GPL (>= 2)

LazyData yes

URL `https://restriktor.org`

NeedsCompilation no

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

Maintainer Leonard Vanbrabant `<info@restriktor.org>`

Suggests knitr, rmarkdown, bain, testthat (>= 3.0.0), metadat

VignetteBuilder knitr

Repository CRAN

Date/Publication 2026-02-05 14:40:02 UTC

Contents

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

Description

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

Details

Package:	restriktor
Type:	Package
Version:	0.6-30
Date:	2026-02-04
License:	GPL (>=2)
LazyLoad:	yes

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

The function `restriktor` offers the possibility to compute (model robust) standard errors under the restriktions. The parameter estimates can also be bootstrapped, where bootstrapped standard errors and confidence intervals are available via the summary function. Moreover, the function computes the Generalized Order-restricted Information Criterion (GORIC), which is a modification of the AIC and a generalization of the ORIC.

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

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

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

Value

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

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

These classes offer print and summary methods.

Acknowledgements

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

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

Author(s)

Leonard Vanbrabant and Yves Rosseel - Ghent University

References

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

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

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

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

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

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

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

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

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

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

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

See Also

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

Examples

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

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

## restricted linear model with 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

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

Examples

```
head(AngerManagement)
```

`benchmark_functions` *Benchmark Functions for GORIC(A) Analysis*

Description

The ‘benchmark‘ functions perform benchmarking for models using the Generalized Order-Restricted Information Criterion (Approximation) (GORIC(A)).

Usage

```
  benchmark(object, model_type = c("asymp", "means"), ...)

  benchmark_means(object, pop_es = NULL, ratio_pop_means = NULL,
                  group_size = NULL, alt_group_size = NULL,
                  quant = NULL, iter = 2000,
                  control = list(),
                  ncpus = 1, seed = NULL, ...)

  benchmark_asymp(object, pop_est = NULL, sample_size = NULL,
                  alt_sample_size = NULL, quant = NULL, iter = 2000,
                  control = list(),
                  ncpus = 1, seed = NULL, ...)

  ## S3 method for class 'benchmark'
print(x, output_type = c("rgw", "gw", "rlw", "ld", "all"),
                            hypo_rate_threshold = 1, color = TRUE, ...)

  ## S3 method for class 'benchmark'
plot(x, output_type = c("rgw", "rlw", "gw", "ld"),
                           percentiles = NULL, x_lim = c(), log_scale = FALSE,
                           alpha = 0.50, nrow_grid = NULL, ncol_grid = 1,
                           distr_grid = FALSE, ...)
```

Arguments

`object`	An object of class `con_goric` (a GORIC(A) object from the `goric` function).
`model_type`	If "means", the model parameters reflect (adjusted) means, else model_type = "asymp" (default). See details for more information about `asymp`.
`x`	An object of class `benchmark` or `benchmark`.
`pop_es`	A scalar or a vector of population Cohen's f (effect-size) values. By default, it benchmarks ES = 0 (no-effect) and the observed Cohen's f.
`pop_est`	A 1 x k vector or an n x k matrix of population estimates to benchmark. By default, all estimates are set to zero (no-effect) and the observed estimates from the sample are used.
`ratio_pop_means`	A 1 x k vector denoting the relative difference between the k group means. Note that a ratio of `c(3, 2, 1)` gives the same as `c(1, 0, -1)`, as the consecutive relative differences are 1 in both ratios. By default, the relative differences from the data are used.
`group_size`	If the GORICA object is based on estimates and their covariance matrix (instead of on a model/fit object), this should be a 1 x k vector or a scalar to denote the group sizes. If a scalar is specified, it is assumed that each group is of that size.
`alt_group_size`	An 1 x k vector or a scalar to denote alternative group sizes, if you want to use sizes different from those in the data. This can be used, for example, to see the values to which the GORIC(A) weights will converge (and thus to see the

	maximum value of the weights). If a scalar is specified, it is assumed that each group is of that size. By default, the group sizes from the data are used.
`sample_size`	A scalar to denote the (total) sample sizes. Only used if the GORIC object is based on estimates and their covariance matrix (instead of on a model/fit object) or `alt_sample_size` is not `NULL`.
`alt_sample_size`	A scalar to denote an alternative sample size if you want to use a different sample size from the one in the data. This can be used, for example, to see the values to which the GORIC(A) weights will converge (and thus to see the maximum value of the weights).
`quant`	Quantiles for benchmarking results. Defaults 5%, 35%, 50%, 65%, 95%.
`iter`	The number of iterations for benchmarking. Defaults to `2000`.
`hypo_rate_threshold`	A numeric value specifying the threshold for the hypothesis rate. The function calculates the proportion of ratio-of-goric-weights that exceeds this threshold. Defaults to 1.
`control`	A list of control parameters.For more information, see details goric.
`ncpus`	Number of CPUs to use for parallel processing. Defaults to `1`. See details for more information.
`seed`	A seed for random number generation.
`output_type`	A character vector specifying the type of output to print or plot. Options are `"all"`, `"gw"` (goric(a) weights), `"rgw"` (ratio of goric(a) weights), `"rlw"` (ratio of log-likelihood values), and `"ld"` (log-likelihood difference). Defaults to `"rgw"` for print and `"rgw"` for plot.
`color`	If TRUE, the output will include ANSI color coding. Set `color = FALSE` when using this function in R Markdown documents to avoid rendering issues with color codes.
`alpha`	Alpha refers to the opacity of a geom. Values of alpha range from 0 to 1, with lower values corresponding to more transparent colors.
`nrow_grid`	An integer value representing the number of rows in the grid layout.
`ncol_grid`	An integer value representing the number of columns in the grid layout.
`distr_grid`	If TRUE, the facet_grid function is used to create a grid of separate plots for each effect-size (estimates).
`percentiles`	A numeric vector specifying the percentiles to be shown. By default the percentiles are inherited from the quantiles used for benchmarking, see `quant`.
`x_lim`	A numeric vector of length 2 specifying the x-axis limits. Defaults to `c()`.
`log_scale`	logical, If TRUE, The x-axis is transformed using a base-10 logarithmic scale. This transformation adjusts the way the data is visualized on the x-axis, but does not alter the underlying data values themselves.
`...`	See goric.

Details

The function `benchmark_asymp` is named as such because it generates data from a multivariate normal distribution with means equal to the population parameter estimates and a covariance matrix derived from the original data. This is based on the assumption that parameter estimates are asymptotically normally distributed. This assumption is valid for many statistical models, including parameters from a generalized linear model (GLM). In such models, as the sample size increases, the distribution of the parameter estimates tends to a normal distribution, allowing us to utilize the multivariate normal distribution for benchmarking.

`benchmark_means` benchmarks the group means of a given GORIC(A) object by evaluating various population effect sizes and comparing the observed group means against these benchmarks. `benchmark_asymp` benchmarks the population estimates of a given GORIC(A) object by evaluating various population estimates and comparing them against the observed estimates.

`print.benchmark` prints the results of benchmark analyses performed on objects of class `benchmark`.

`plot.benchmark` generates density plots for benchmark analyses of objects of class `benchmark`.

The benchmark function leverages the **future** package for parallel processing, allowing users to speed up computations by distributing tasks across multiple cores or machines. If the user does not specify a parallelization plan using `future::plan()`, the package will choose an appropriate strategy based on the user's operating system. Specifically, on Windows, the package defaults to using `multisession`, which creates separate R sessions for each parallel task. On Unix-like systems (such as Linux and macOS), the package defaults to `multicore`, which uses forked R processes to avoid the overhead of setting up separate R sessions.

The `plan()` must be specified before running the benchmark function, e.g., `future::plan(future::multisession, workers = ncpus)`

Value

`benchmark_means` and `benchmark_asymp` return a list of class `benchmark_means`, `benchmark`, and `list` or `benchmark_asymp`, `benchmark`, and `list` containing the results of the benchmark analysis.

`print.benchmark` does not return a value. It prints formatted benchmark analysis results to the console.

`plot.benchmark` returns a gtable object that can be displayed or further customized using various functions from the gridExtra and grid packages. This allows for flexible and detailed adjustments to the appearance and layout of the plot.

Author(s)

Leonard Vanbrabant and Rebecca Kuiper

Examples

```
set.seed(1234)
# Generate data for 4 groups with different group sizes
group1 <- rnorm(10, mean = 5, sd = 0.1)
group2 <- rnorm(20, mean = 5.5, sd = 1)
group3 <- rnorm(30, mean = 6, sd = 0.5)
group4 <- rnorm(40, mean = 6.5, sd = 0.8)
```

```
# Combine data into a data frame
data <- data.frame(
  value = c(group1, group2, group3, group4),
  group = factor(rep(1:4, times = c(10,20,30,40)))
)

# Perform ANOVA
anova_result <- aov(value ~ -1 + group, data = data)

# model/hypothesis
h1 <- 'group1 < group2 < group3 < group4'
h2 <- 'group1 > group2 < group3 < group4'

# fit h1 and h2 model against the unconstrained model (i.e., failsafe to avoid
# selecting a weak hypothesis)
fit_goric <- goric(anova_result, hypotheses = list(H1 = h1, H2 = h2),
                   comparison = "unconstrained", type = "goric")

# by default: ES = 0 \& ES = observed ES
# In practice you want to increase the number of iterations (default = 1000).

# multisession supports windows machines
# future::plan(future::multisession, workers = 8)
# multicore supports unix machines
# future::plan(future::multicore, workers = 8)
benchmark_results_mean <- benchmark(fit_goric, iter = 10, model_type = "means")
print(benchmark_results_mean)


# by default the ratio of GORIC weights for the preferred hypothesis (here h1) is
# plotted against its competitors (i.e., h2 and the unconstrained). To improve
# the readability of the plot, the argument hypothesis_comparison can be used to
# focus on a specif competitor. Further readability can be achieved by setting
# the x_lim option.
plot(benchmark_results_mean, output_type = "rgw")

# specify custom effect-sizes

benchmark_results_mean_es <- benchmark(fit_goric, iter = 10,
                                       pop_es = c(0, 0.1),
                                       model_type = "means")
print(benchmark_results_mean_es)


# Benchmark asymptotic estimates

fit_gorica <- goric(anova_result, hypotheses = list(h1=h1),
                    comparison = "complement", type = "gorica")

# by default: no-effect \& estimates from the sample are used
benchmark_results_asymp <- benchmark(fit_gorica, sample_size = 30, iter = 5,
                                     model_type = "asymp")
print(benchmark_results_asymp)
```

```
# specify custom population estimates
my_pop_est <- rbind("no" = c(0,0,0,0), "observed"= coef(anova_result))

benchmark_results_asymp <- benchmark(fit_gorica, sample_size = 30,
                                     iter = 5, pop_est = my_pop_est,
                                     model_type = "asymp")
print(benchmark_results_asymp)
plot(benchmark_results_asymp, x_lim = c(0, 75))
```

`bootstrapD` *Bootstrapping a Lavaan Model*

Description

Bootstrap the D statistic.

Usage

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

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

Arguments

`h0`	An object of class `lavaan`. The restricted model.
`h1`	An object of class `lavaan`. The unrestricted model.
`x`	an object of class `conTestLavaan`.
`constraints`	The imposed (in)equality constraints on the model.
`type`	hypothesis test type "A", "B".
`bootstrap.type`	If `"parametric"`, the parametric bootstrap is used. If `"bollen.stine"`, the semi-nonparametric Bollen-Stine bootstrap is used. The default is set to `"bollen.stine"`.
`R`	Integer. The number of bootstrap draws.
`return.D`	Logical; if `TRUE`, the function returns bootstrapped D-values.

`double.bootstrap`	If `"standard"` (default) the genuine double bootstrap is used to compute an additional set of plug-in p-values for each bootstrap sample. If `"no"`, no double bootstrap is used. If `"FDB"`, the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. Note that the `"FDB"` is experimental and should not be used by inexperienced users.
`double.bootstrap.R`	Integer; number of double bootstrap draws. The default value is set to 249.
`double.bootstrap.alpha`	The significance level to compute the adjusted alpha based on the plugin p-values. Only used if `double.bootstrap = "standard"`. The default value is set to 0.05.
`verbose`	If `TRUE`, show information for each bootstrap draw.
`warn`	Sets the handling of warning messages. See `options`.
`parallel`	The type of parallel operation to be used (if any). If missing, the default is `"no"`.
`ncpus`	Integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	An optional **parallel** or **snow** cluster for use if `parallel = "snow"`. If not supplied, a cluster on the local machine is created for the duration of the `bootstrapLavaan` or `bootstrapLRT` call.
`digits`	the number of significant digits to use when printing.
`...`	no additional arguments for now.
`seed`	An integer to set the seed. Or NULL if no reproducible seeds are needed.

Value

A bootstrap p value, calculated as the proportion of bootstrap samples with a D statistic at least as large as the D statistic for the original data.

Author(s)

Leonard Vanbrabant

References

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

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

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

Examples

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

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


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

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

example1

##########################
### artificial example ###
##########################

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

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

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

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

# fit without constraints
```

```
fit <- lavaan::sem(model, data = Data.new)

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

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

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

Description

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

Usage

```
data(Burns)
```

Format

A data frame of 278 observations of 4 variables.

`PTSS` post-traumatic stress symptoms

`gender` gender

`age` age in years

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

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

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

References

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

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

Examples

```
head(Burns)
```

`calculate_IC_weights` *Calculating IC weights based on IC values (AIC, ORIC, GORIC(A), BIC, SIC, ...)*

Description

This function transforms IC values into IC weights: IC values denote the ordering of hypotheses/models, while IC weights quantify the relative strength of hypotheses/models.

Usage

```
calculate_IC_weights(IC, hypo_names = NULL)
calc_ICweights(IC, hypo_names = NULL)

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

Arguments

`IC`	A vector or one-column matrix with information criteria (AIC, ORIC, GORIC(A), BIC, SIC, ...) values of length 'NrHypos', where 'NrHypos' stands for the number of hypotheses/ models.
`x`	an object of class `con_goric`.
`hypo_names`	Optional. Vector containing 'NrHypos' characters which will be used for labeling the hypothesis. Default: H1, H2, ...
`use_scientific`	If TRUE (default), the IC weights and ratio of IC weights will be formatted using scientific notation. If FALSE, standard numeric formatting is used.
`digits`	the number of significant digits to use when printing.
`...`	no additional arguments for now.

Value

IC weights, which quantify the relative strength of hypotheses/models.

Examples

```
IC <- c(1,2,3)
calculate_IC_weights(IC)


## PT weights
# This examples shows how to calculate PT weights.
```

```
# Notably, one is interested in PT weights when the log-likelihood for two or more
# hypotheses are (approximately) equal.
# Then, the comparison between those hypotheses is solely based on the PT values.
# The IC weights will then equal the PT weights.
# In that case, there is support for the overlap (boundary) of these hypotheses.
# Thus, when the IC weights equal the PT weights for a (sub)set of hypotheses,
# then there is support for the overlap (boundary) of these hypotheses.

y <- rnorm(30)
group <- factor(rep(c("A","B","C"), each = 10))
fit.lm <- lm(y ~ -1 + group)
est <- coef(fit.lm)
VCOV_est <- vcov(fit.lm)
H1 <- "groupA < groupB < groupC"
results <- goric(est, VCOV = VCOV_est, hypotheses = list(H1),
                 comparison = "complement", type = "gorica")
calculate_IC_weights(results$result[,3])
```

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

Description

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

Usage

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

Arguments

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

Details

Hypothesis test Type C:

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

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

Value

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

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

See Also

quadprog, `iht`

Examples

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

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

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

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

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

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

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


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

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


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0

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

`conTestF` *F-bar test for iht*

Description

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

Usage

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

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

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

Arguments

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

- `maxit` the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported).
- `tol` numerical tolerance value. Estimates smaller than `tol` are set to 0.
- `chunk_size` the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., mix_weights_bootstrap_limit.
- `convergence_crit` the convergence criterion for the iterative process. The default is 1e-03.

`...` Additional arguments that can be passed to the p.distr function, or arguments for the restriktor or iht function. Consider, for example, the mix_weights_bootstrap_limit control argument, which specifies the maximum number of bootstrap draws (default is 100.000) used to compute the chi-bar-square weights. If mix_weights_bootstrap_limit is set to 100.000, then in each iteration, a sample of size 5000 is added until the weights converge, or the maximum limit is reached.

Details

The following hypothesis tests are available:

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

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

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

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

Value

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

`CON` a list with useful information about the constraints.

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

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

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

See Also

quadprog, `iht`

Examples

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

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

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

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

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

iht(fit1.lm, myConstraints1)


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

iht(fit.restr1)


  # Or in matrix notation.
  Amat1 <- rbind(c(-1, 0,  1),
                 c( 0, 1, -1))
  myRhs1 <- rep(0L, nrow(Amat1))
  myNeq1 <- 0

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


#########################
## 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 < group3 < group4 '

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


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

iht(fit2.con)

# increasing means in matrix notation.
Amat2 <- rbind(c(-1, 1, 0, 0),
               c( 0,-1, 1, 0),
               c( 0, 0,-1, 1))
myRhs2 <- rep(0L, nrow(Amat2))
myNeq2 <- 0

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

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

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

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


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

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

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

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

iht(fit4.lm, constraints = myConstraints4)

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

`conTestLRT` *Likelihood-ratio-bar test for iht*

Description

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

Usage

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

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

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

Arguments

`object`	an object of class `conLM`, `conMLM` or `conGLM`.
`type`	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
`neq.alt`	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
`boot`	the null-distribution of these test-statistics (except under type "C", see details) takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if `"parametric"`, the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See `p.distr` for other distributional options. If `"model.based"`, a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If `"no"`, the p-value is computed based on the weights obtained via simulation (`mix_weights = "boot"`) or using the multivariate normal distribution function (`mix_weights = "pmvnorm"`). Note that, these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class `conRLM`, namely for computing the p-value for the robust `test = "Wald"` and the robust `"score"`. In these cases the weights need to be recalculated.
`R`	integer; number of bootstrap draws for `boot`. The default value is set to 9999.
`p.distr`	random generation distribution for the parametric bootstrap. For all available distributions see `?distributions`. For example, if `rnorm`, samples are drawn from the normal distribution (default) with mean zero and variance one. If `rt`, samples are drawn from a t-distribution. If `rchisq`, samples are drawn from a chi-square distribution. The random generation distributional parameters will be passed in via `...`.
`parallel`	the type of parallel operation to be used (if any). If missing, the default is set "no".
`ncpus`	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
`seed`	seed value. The default value is set to 1234.
`verbose`	logical; if TRUE, information is shown at each bootstrap draw.
`control`	a list of control arguments: • `absval` tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm. • `maxit` the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported). • `tol` numerical tolerance value. Estimates smaller than `tol` are set to 0. • `chunk_size` the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., mix_weights_bootstrap_limit.

	- `convergence_crit` the convergence criterion for the iterative process. The default is 1e-03.
`...`	Additional arguments that can be passed to the p.distr function, or arguments for the restriktor or iht function. Consider, for example, the mix_weights_bootstrap_limit argument, which specifies the maximum number of bootstrap draws (default is 100.000) used to compute the chi-bar-square weights. If mix_weights_bootstrap_limit is set to 100.000, then in each iteration, a sample of size 5000 is added until the weights converge, or the maximum limit is reached.

Details

The following hypothesis tests are available:

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

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

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

Value

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

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

`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b_eqrestr`	equality restricted regression coefficients. Only available for `type = "A"` and `type = "global"`, else `b.eqrestr = NULL`.
`b_unrestr`	unrestricted regression coefficients.
`b_restr`	restricted regression coefficients.
`b_restr_alt`	restricted regression coefficients under HA if some equality constraints are maintained. Only available for `type = "B"` else `b_restr_alt = NULL`.
`Sigma`	variance-covariance matrix of unrestricted model.
`R2_org`	unrestricted R-squared, not available for objects of class `conGLM`.
`R2_reduced`	restricted R-squared, not available for objects of class `conGLM`.
`boot`	same as input.
`model_org`	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

See Also

quadprog, `conTest`

Examples

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

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

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

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

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

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


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

iht(fit_restr1, test = "LRT")


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0

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

#########################
## 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 < group3 < group4 '

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

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

iht(fit2_con, test = "LRT")


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

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

iht(fit2_con, constraints = Amat2, rhs = myRhs2, neq = myNeq2,
    type = "A", test = "LRT", boot = "parametric", R = 9)

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

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

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


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

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

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

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

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

`conTestScore` *Score-bar test for iht*

Description

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

Usage

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

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

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

Arguments

`object`	an object of class `conLM`, `conRLM` or `conGLM`.
`type`	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
`neq.alt`	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B"), see example 3.
`boot`	the null-distribution of these test-statistics (except under type "C", see details) takes the form of a mixture of F-distributions. The tail probabilities can be computed directly via bootstrapping; if `"parametric"`, the p-value is computed based on the parametric bootstrap. By default, samples are drawn from a normal distribution with mean zero and varance one. See `p.distr` for other distributional options. If `"model.based"`, a model-based bootstrap method is used. Instead of computing the p-value via simulation, the p-value can also be computed using the chi-bar-square weights. If `"no"`, the p-value is computed based on the weights obtained via simulation (`mix_weights = "boot"`) or using the multivariate normal distribution function (`mix_weights = "pmvnorm"`). Note that,

	these weights are already available in the restriktor objected and do not need to be estimated again. However, there are two exception for objects of class `conRLM`, namely for computing the p-value for the robust `test = "Wald"` and the robust `"score"`. In these cases the weights need to be recalculated.
`R`	integer; number of bootstrap draws for `boot`. The default value is set to 9999.
`p.distr`	random generation distribution for the parametric bootstrap. For all available distributions see `?distributions`. For example, if `rnorm`, samples are drawn from the normal distribution (default) with mean zero and variance one. If `rt`, samples are drawn from a t-distribution. If `rchisq`, samples are drawn from a chi-square distribution. The random generation distributional parameters will be passed in via `...`.
`parallel`	the type of parallel operation to be used (if any). If missing, the default is set "no".
`ncpus`	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
`seed`	seed value. The default value is set to 1234.
`verbose`	logical; if TRUE, information is shown at each bootstrap draw.
`control`	a list of control arguments: • `absval` tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). Only used for model of class lm. • `maxit` the maximum number of iterations for the optimizer (default = 10000). Only used for model of class mlm (not yet supported). • `tol` numerical tolerance value. Estimates smaller than `tol` are set to 0. • `chunk_size` the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., mix_weights_bootstrap_limit. • `convergence_crit` the convergence criterion for the iterative process. The default is 1e-03.
`...`	Additional arguments that can be passed to the p.distr function, or arguments for the restriktor or iht function. Consider, for example, the mix_weights_bootstrap_limit argument, which specifies the maximum number of bootstrap draws (default is 100.000) used to compute the chi-bar-square weights. If mix_weights_bootstrap_limit is set to 100.000, then in each iteration, a sample of size 5000 is added until the weights converge, or the maximum limit is reached.

Details

The following hypothesis tests are available:

- Type A: Test H0: all constraints with equalities ("=") active against HA: at least one inequality restriction (">") strictly true.
- Type B: Test H0: all constraints with inequalities (">") (including some equalities ("=")) active against HA: at least one restriction false (some equality constraints may be maintained).

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

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

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

Value

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

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

See Also

quadprog, `conTest`

Examples

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

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

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

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

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

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


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

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


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0
```

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


#########################
## 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 < group3 < group4 '

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


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

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


# increasing means in matrix notation.
Amat2 <- rbind(c(-1, 1, 0, 0),
               c( 0,-1, 1, 0),
               c( 0, 0,-1, 1))
myRhs2 <- rep(0L, nrow(Amat2))
myNeq2 <- 0

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

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

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

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


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

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

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

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

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

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

Description

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

Usage

```
## S3 method for class 'conRLM'
conTestWald(object, type = "A", neq.alt = 0,
            boot = "no", R = 9999, p.distr = rnorm,
```

```
          parallel = "no", ncpus = 1L, cl = NULL, seed = 1234,
          verbose = FALSE, control = NULL, ...)
```

Arguments

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

- `chunk_size` the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., mix_weights_bootstrap_limit.
- `convergence_crit` the convergence criterion for the iterative process. The default is 1e-03.

`...`	Additional arguments that can be passed to the p.distr function, or arguments for the restriktor or iht function. Consider, for example, the mix_weights_bootstrap_limit argument, which specifies the maximum number of bootstrap draws (default is 100.000) used to compute the chi-bar-square weights. If mix_weights_bootstrap_limit is set to 100.000, then in each iteration, a sample of size 5000 is added until the weights converge, or the maximum limit is reached.

Details

The following hypothesis tests are available:

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

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

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

Value

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

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

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

See Also

quadprog, `conTest`

Examples

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

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

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

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

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

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


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

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


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0

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


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

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

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

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

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

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

# increasing means in matrix notation.
Amat2 <- rbind(c(-1, 1, 0, 0),
               c( 0,-1, 1, 0),
               c( 0, 0,-1, 1))
myRhs2 <- rep(0L, nrow(Amat2))
myNeq2 <- 0

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


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

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

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


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

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

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

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

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

`conTest_ceq` *Tests for iht with equality constraints only*

Description

`conTest_ceq` tests linear equality restricted hypotheses for (robust) linear models by F-, Wald-, and score-tests. It can be used directly and is called by the `conTest` function if all restrictions are equalities.

Usage

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

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

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

Arguments

`object`	an object of class `conLM`, `conRLM` or `conGLM`.
`test`	test statistic; for information about the null-distribution see details. • for object of class lm and glm; if "F" (default), the classical F-statistic is computed. If "Wald", the classical Wald-statistic is computed. If "score", the classical score test statistic is computed. • for object of class rlm; if "F" (default), a robust likelihood ratio type test statistic (Silvapulle, 1992a) is computed. If "Wald", a robust Wald test statistic (Silvapulle, 1992b) is computed. If "score", a score test statistic (Silvapulle, 1996) is computed.

`boot`	if `"parametric"`, the p-value is computed based on the parametric bootstrap. See `p.distr` for available distributions. If `"model.based"`, a model-based bootstrap method is used. Model-based bootstrapping is not supported for the `conGLM` object yet.
`R`	integer; number of bootstrap draws for `boot`. The default value is set to 9999.
`p.distr`	the p.distr function is specified by this function. For all available distributions see `?distributions`. For example, if `rnorm`, samples are drawn from the normal distribution (default) with mean zero and variance one. If `rt`, samples are drawn from a t-distribution. If `rchisq`, samples are drawn from a chi-square distribution. The distributional parameters will be passed in via ....
`parallel`	the type of parallel operation to be used (if any). If missing, the default is set "no".
`ncpus`	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the conTest call.
`seed`	seed value. The default value is set to 1234.
`verbose`	logical; if TRUE, information is shown at each bootstrap draw.
`...`	additional arguments to be passed to the p.distr function.

Value

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

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

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

See Also

quadprog, `iht`

Examples

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

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

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

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

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

iht(fit1.lm, myConstraints1)


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

iht(fit_restr1)


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 2
```

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

`conTest_summary` *function for computing all available hypothesis tests*

Description

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

Usage

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

Arguments

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

Value

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

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

`Ts`	test-statistic value.
`df.residual`	the residual degrees of freedom.
`pvalue`	tail probability for `Ts`.
`b.eqrestr`	equality restricted regression coefficients. Only available for `type = "A"` and `type = "global"`, else `b.eqrestr = NULL`.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`b.restr.alt`	restricted regression coefficients under HA if some equality constraints are maintained.
`Sigma`	variance-covariance matrix of unrestricted model.
`R2.org`	unrestricted R-squared.
`R2.reduced`	restricted R-squared.
`boot`	same as input.
`model.org`	original model.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

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

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

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

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

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

See Also

quadprog, `iht`

Examples

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

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

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

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

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

iht(fit1.lm, myConstraints1)


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

iht(fit.restr1)

# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0

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

`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).

Usage

```
con_weights_boot(VCOV, Amat, meq, R = 1e5L,
                 chunk_size = 5000L, convergence_crit = 1e-03,
                 seed = NULL, verbose = FALSE, ...)
```

Arguments

`VCOV`	variance-covariance matrix of the data for which the weights are to be calculated.
`Amat`	constraints matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ). The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see `restriktor`.
`meq`	integer (default = 0) treating the number of constraints rows as equality constraints instead of inequality constraints. For example, if `meq = 2`, this means that the first two rows of the constraints matrix R are treated as equality constraints.
`R`	integer; the maximum number of bootstrap draws for `mix_weights_bootstrap_limit`. The default value is set to 1e5. See details for more information.
`chunk_size`	integer; the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., mix_weights_bootstrap_limit.
`convergence_crit`	the convergence criterion for the iterative process. The default is 1e-03. See details for more information.
`seed`	seed value.
`verbose`	logical; if TRUE, information is shown at each bootstrap draw.
`...`	additional parameters for the `rtmvnorm` function.

Details

Iterative Weight Updating and Convergence Checking ## The function adds in each run chunks of 5000 samples (default) to compute the chi-bar-square weights. After each iteration, the function checks if the weights have converged. This is determined by the convergence_crit parameter.

Convergence is assessed by comparing the absolute difference between the current and previous iteration's weights against the convergence_crit. If the change in weights is smaller than the convergence criterion, it indicates that the weights have stabilized, suggesting convergence.

If the weights have not converged and the `mix_weights_bootstrap_limit` has not been reached, the function proceeds with adding another set of 5000 samples and updates the weights accordingly.If the maximum number of iterations is reached without convergence, the function returns the (non-converged) weights. In this situation, it is advisible to increase the number of `mix_weights_bootstrap_limit`.

Value

If convergence is reached, the function returns a vector with the mixing weights with the following attributes:

`total_bootstrap_draws`	total number of bootstrap draws
`converged`	have the chi-bar-square weights converged
`convergence_crit`	convergence criterium
`wt_bar_chunk`	matrix with the chi-bar-square weights for each iteration
`chunk_size`	how many samples are added in each iteration
`total_chunks`	what is the maximum number of chunks based on `mix_weights_bootstrap_limit` and `chunk_size`
`chunk_iter`	number of iterations run
`error.idx`	which bootstrap samples were not succesful
`mix_weights_bootstrap_limit`	the maximum number of bootstrap draws

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

Examples

```
W <- matrix(c(1,0.5,0.5,1),2,2)
Amat <- rbind(c(0,1))
meq <- 0L
wt.bar <- con_weights_boot(W, Amat, meq, R = 99999)
wt.bar

# in practice you want to use are more conservative convergence criterion
wt.bar2 <- con_weights_boot(W, Amat, meq, R = 99999, convergence_crit = 1e-02)
wt.bar2
```

---

evSyn *GORIC(A) Evidence synthesis*

Description

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

Usage

```
# I try to guess the input type.
evSyn(object, input_type = NULL, ...)

# input: Parameter estimates and covariance matrix
evSyn_est(object, ..., VCOV = list(), hypotheses = list(),
          type_ev = c("added", "equal", "average"),
          comparison = c("unconstrained", "complement", "none"),
          hypo_names = c(), type = c("gorica", "goricac"),
          order_studies = c("input_order", "ascending", "descending"),
          study_names = c(), study_sample_nobs = NULL)

# input: Log-likelihood and penalty values
evSyn_LL(object, ..., PT = list(), type_ev = c("added", "equal", "average"),
         hypo_names = c(),
         order_studies = c("input_order", "ascending", "descending"),
         study_names = c())

# input: GORIC(A), ORIC, AIC values
evSyn_ICvalues(object, ..., type_ev = c("added", "average"), hypo_names = c(),
               order_studies = c("input_order", "ascending", "descending"),
               study_names = c())

# input: AIC or ORIC or GORIC or GORICA weights or (Bayesian) posterior
# model probabilities
evSyn_ICweights(object, ..., type_ev = c("added", "average"),
                priorWeights = NULL, hypo_names = c(),
                order_studies = c("input_order", "ascending", "descending"),
                study_names = c())

# input: Ratio of AIC or ORIC or GORIC or GORICA weights or
# (Bayesian) posterior model probabilities
evSyn_ICratios(object, ..., type_ev = c("added", "average"),
               priorWeights = NULL, hypo_names = c(),
               order_studies = c("input_order", "ascending", "descending"),
               study_names = c())

# input: result from the goric() function
```

```
# Note that this is a wrapper function for evSyn_LL.
evSyn_gorica(object, ..., type_ev = c("added", "equal", "average"),
             hypo_names = c(),
             order_studies = c("input_order", "ascending", "descending"),
             study_names = c())

# input: Result from the escalc() function from the metafor package.
# Note that this is a wrapper function for evSyn_est.
evSyn_escalc(data, yi_col = "yi", vi_cols = "vi", cluster_col, outcome_col, ...)

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

## S3 method for class 'evSyn'
summary(object, ...)

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

## S3 method for class 'evSyn'
plot(x, output_type = "gorica_weights", xlab = NULL,
xlab_unordered = NULL, angle_x = 30, ...)
```

Arguments

`object` Currently, the following objects can be processed:

- a list of vectors with (standardized) parameter estimates (the `VCOV` argument is required);
- a list of vectors with log-likelihood values (the `PT` argument is required);
- a list of vectors with GORIC(A) weights;
- a list of vectors with ratio of GORIC(A) weights;
- a list of vectors with GORIC(A) values;
- a list of objects of class goric;
- a data.frame of class escalc from the **metafor** package;

`input_type` `character` Specifies the type of input provided to the function. Valid options are:

`"est_vcov"` Indicates that the input consists of estimates and covariance matrices. Invokes the `evSyn_est()` function.

`"ll_pt"` Indicates that the input consists of log-likelihoods and penalty values. Invokes the `evSyn_LL()` function.

`"icweights"` Indicates that the input consists of information criterion (IC) weights. Invokes the `evSyn_ICweights()` function.

`"icratios"` Indicates that the input consists of IC weights ratios. Invokes the `evSyn_ICratios()` function.

`"icvalues"` Indicates that the input consists of IC values. Invokes the `evSyn_ICvalues()` function.

	`"gorica"` Indicates that the input is of class goric from the `goric` function. Invokes the `evSyn_gorica()` function. `"escalc"` Indicates that the input is of class escalc from the **metafor** package. Invokes the `evSyn_escalc()` function. If `input_type` is `NULL`, the function attempts to infer the input type based on the structure of the `object` and other arguments.
`VCOV`	a list of covariance matrices of the (standardized) parameter estimates of interest.
`PT`	a list of vectors with penalty values.
`x`	an object of class `evSyn`
`type_ev`	type of evidence-synthesis approach: Equal-evidence approach (`type_ev = "equal"`), Added-evidence approach (`type_ev = "added"`), or Average-evidence approach (`type_ev = "average"`). See details for more information.
`type`	If `"gorica"`, a proxi for the log-likihood is calculated by assuming that the parameter estimates (of interest) are multivariate normally distributed. Then, either the unbiased covariance matrix of the estimates (based on 'N'; e.g., for an lm object) is used or the biased one obtained via the vcov function (e.g., for a lavaan object). In the latter case, a corresponding message will be printed. If `"goricac"`, then the small-sample-size correction is applied.
`hypotheses`	When applying the same set of hypotheses to each study, the syntax structure should be as follows: "hypotheses = list(H1, H2, ...)". However, if a different set of hypotheses is applied to each study, the syntax structure should be as follows: hypotheses = list(set1 = list(H11, H12), set2 = list(H21, H22)). See `goric` how to specify the hypotheses syntax or see the example section below.
`comparison`	denotes the failsafe hypothesis; one can choose from "unconstrained", "complement", and "none". If `"none"`, the model is only compared against the models provided by the user. By default, if one user-specified hypothesis, `"comparison = 'complement'"`; if multiple user-specified hypotheses, `"comparison = 'unconstrained'"`. Note that the complement can only be computed for one model/hypothesis at a time.
`priorWeights`	vector that represents the prior belief for this model. By default, equal prior weights are used (i.e., 1/(#hypotheses)). Notably, in case the prior weights do not sum to 1, it will be rescaled such that it does; which implies that relative importance can be used and not per se weights.
`hypo_names`	character vector for labelling the hypotheses. By default the names are set to H1, H2, ...
`output_type`	a plot options. If `"gorica_weights"` (default), a plot with the cumulative goric(a) weights and goric(a) weights per study is displayed. If `"ll_weights"`, a plot with the cumulative log-likelihood weights and log-likelihood weights per study is displayed.
`xlab_unordered`	A character vector specifying the original (unordered) study labels. This argument should be used when studies have been re-ordered using `order_studies`. The labels will be reordered internally to match the plotting order.
`angle_x`	Numeric value specifying the angle (in degrees) of the x-axis labels. Useful when labels are long and overlap.

`study_names`	A character vector specifying the names or labels of the individual studies. When provided, these names will be used throughout the output, including in tables, summaries, and plots (in place of numeric study indices). If `NULL` (default), studies are automatically numbered (1, 2, ..., K) following the order determined by `order_studies`.
`study_sample_nobs`	A numeric vector specifying the sample size (`n`) for each study. When provided, this information will be included in summaries and can be used for weighted averaging or diagnostic purposes in the evidence synthesis. If `NULL` (default), all studies are assumed to have equal weight with respect to sample size.
`order_studies`	A character string that determines how studies are ordered in the output and plots. If `"input_order"` (default), studies are displayed in the order they were provided. If `"ascending"` or `"descending"`, studies are ordered based on the ascending or descending cumulative support for the preferred hypothesis, respectively.
`xlab`	a vector specifying custom labels for the x-axis (be aware that study ordering may be changed based on `order_studies`). The length of the vector must match the number of studies in the dataset. If not provided, the x-axis labels default to a sequence from 1 to the number of studies (possibly re-ordered based on `order_studies`).
`data`	an object of class "escalc" from the **metafor** package.
`cluster_col`	a character string specifying the column in `data` that contains the cluster/study identifier. Observations within the same cluster/study should share the same value in this column. When not specified, the function looks for a column with one of the following names: "trial", "study", "author", "authors", "Trial", "Study", "Author", or "Authors".
`outcome_col`	a column name in `data` containing the outcome identifiers/labels for each observation within a cluster/study. The hypothesis/-es should be based on this labelling. By default (i.e., if 'outcome_col' is null), the function assumes that the parameter label used in the hypothesis is "theta" (one outcome variable); when there is additionally a column called 'outcome', it will use that as the outcome identifier, employing its unique levels/values as the labels (more outcome variables). If 'outcome_col' is not null but user-specified, then the labeling should be based on the unique level(s)/value(s) in the corresponding data column. Notably, outcome_col is related to the labeling; the column with estimates is referred to as 'yi_col' and is, by default, set to "yi". For more information, see Example 5 below.
`yi_col`	a character string specifying the column in `data` that contains the outcome values / estimates for each observation. The default is `"yi"`. For more information, see Example 5 below.
`vi_cols`	a character vector specifying the columns in `data` that contain the variance and covariance values for each observation. The default is, in the case of one outcome variable, `"vi"`; in the case of multiple (namely, O) variables, `c("v1i", ..., "vOi"`. For more information, see Example 5 below.
`digits`	the number of significant digits to use when printing.
`...`	This depends on the class of the object.

Details

In the **added-evidence** approach, evidence from each study or dataset is cumulatively aggregated. This means that for every new study, the log-likelihood and the penalty term are added to the cumulative totals. The strength of the aggregated evidence in this approach depends on the nature of the evidence itself. Simply having more studies doesn't necessarily mean stronger evidence if those studies provide weak or contradictory evidence.
Opt for this approach when you anticipate each new piece of evidence to provide an incremental contribution to the overall evidence, without the need to normalize or average across datasets. It's especially suitable when you believe that the aggregated evidence from multiple studies is stronger than if the data were combined into a single study.

The **equal-evidence** approach aggregates the cumulative evidence in the same manner as the added-evidence approach. However, when calculating the GORICA, the cumulative evidence is divided by the number of studies. This ensures that the contribution from each study or dataset remains equal, regardless of the total count. Conceptually, aggregating evidence from multiple studies in this approach can be likened to obtaining evidence from a single larger study, similar to how a meta-analysis treats combined evidence.
Choose this method when you want each study to contribute equally to the overall evidence, irrespective of the size or scope of each individual dataset. It's ideal for situations where you view the combined evidence from multiple studies as equivalent to that from a single, larger study.

The **average-evidence** method can be conceptualized as a form of multiverse analysis. When faced with a single dataset, there are often numerous analytical choices available, such as handling missing data, selecting variables, or choosing statistical methods. Each choice can lead to a different analysis or model, creating a "multiverse" of possible outcomes.
For each of these analyses, an "evidence" score can be calculated, indicating how well the model fits the data. Some models might offer a superior fit, while others might not align as closely with the data. The average-evidence method aggregates these scores, providing an average measure of fit across all considered models. This approach offers an overarching perspective on the general trend across all analyses. If the average evidence suggests a good fit, it indicates that the majority of the chosen analyses align well with the data. This method is invaluable for assessing the robustness of results, ensuring that findings are not merely artifacts of a specific analytical choice but are consistent across various model specifications on the same dataset.
Opt for the average-evidence approach when you wish to gauge the central tendency of evidence across multiple analytical choices. It's especially beneficial when aiming to determine the robustness of results across various model specifications applied to the same dataset.

Value

An object of class evSyn for which a print, summary and plot function is available. The output comprises, among other things, the cumulative and final evidence for the theory-based hypotheses.

Author(s)

Leonard Vanbrabant and Rebecca M. Kuiper

Examples

```
## By following these examples, you can appropriately specify hypotheses based on
## your research questions and analytical framework.
```

```
# The hypotheses (i.e., constraints) have to be in a list. It is recommended to name
# each hypothesis in the list. Otherwise, the hypotheses are named accordingly 'H1', 'H2', \ldots.

# Example using text-based syntax (the labels x1, x2, and x3 are the names of
# coef(model) or names(vector))
H1 <- '(x1, x2, x3) > 0'
H2 <- '(x1, x3) > 0; x2 < 0'
H3 <- 'x1 > 0; x2 < 0; x3 = 0'
hypotheses = list(hypo1 = H1, hypo2 = H2, hypo3 = H3)
# Here, it is assumed that each study uses the same hypothesis specification

# Example using mixed syntax:
H1 <- 'x1 > x2 > x3 > 0'
hypotheses = list(Ha = H1, Hb = 'x1 = x2 > x3')
# Here, it is assumed that each study uses the same hypothesis specification

# Example using a different set of hypotheses for each study:
# Set for Study 1
H11 <- 'group1 = group2 > group3'
H12 <- 'group2 > group1 > group3'
# Set for Study 2
H21 <- 'gr1 = gr2 > gr3'
H22 <- 'gr2 > gr1 > gr3'
#
# correct
hypotheses = list(set1 = list(H11, H12), set2 = list(H21, H22))
# note that the list names set1 and set2 are redundant and can be left out.
# It is crucial to ensure that the hypotheses across each set are ordered in a similar manner.
#
# NOT correct
#hypotheses = list(set1 = list(H12, H11), set2 = list(H21, H22))


## Example 1: based on one estimate and its variance
# 4 studies

# estimates (beta) (from the 4 primary studies)
est_1 <- c(beta1 = 0.09)
est_2 <- c(beta1 = 0.14)
est_3 <- c(beta1 = 1.09)
est_4 <- c(beta1 = 1.781)
Param_studies <- list(est_1, est_2, est_3, est_4)

# standard error of the beta's (from the 4 primary studies)
vcov_est_1 <- matrix(c(0.029^2), nrow = 1)
vcov_est_2 <- matrix(c(0.054^2), nrow = 1)
vcov_est_3 <- matrix(c(0.093^2), nrow = 1)
vcov_est_4 <- matrix(c(0.179^2), nrow = 1)
CovMx_studies <- list(vcov_est_1, vcov_est_2, vcov_est_3, vcov_est_4)

# Set of hypotheses for each study
# Note: in this case the same for each study
```

```
H0   <- "beta1 = 0"
Hpos <- "beta1 > 0"
Hneg <- "beta1 < 0"
hypotheses <- list(H0 = H0, Hpos = Hpos, Hneg = Hneg)

# Since this covers the whole space / covers all theories, we do not need a safeguard-hypothesis:
comparison <- "none"

evS4_added <- evSyn(object = Param_studies, VCOV = CovMx_studies,
                    hypotheses = hypotheses,
                    type_ev = "added",
                    comparison = "none")
evS4_added
summary(evS4_added)
plot(evS4_added)

evS4_equal <- evSyn(object = Param_studies, VCOV = CovMx_studies,
                    hypotheses = hypotheses,
                    type_ev = "equal",
                    comparison = "none")
evS4_equal
summary(evS4_equal)
plot(evS4_equal)


## Example 2: based on (multiple) estimates and their covariance matrices
# 2 studies

# estimates and their covariance matrices (from the 2 primary studies)
est_1 <- c(1.88, 2.54, 0.02)
names(est_1) <- c("group1", "group2", "group3")
vcov_est_1 <- diag(c(0.2149074, 0.2149074, 0.1408014))

est_2 <- c(0.98, 0.02, 0.27)
names(est_2) <- c("gr1", "gr2", "gr3")
vcov_est_2 <- diag(c(0.1382856, 0.1024337, 0.0987754))

# list of estimates
object <- list(est_1, est_2)
# list of covariance matrices of the estimates
VCOV <- CovMx_studies <- list(vcov_est_1, vcov_est_2)

# Hypotheses (study-specific)
# names(est_1) # Specify restrictions using those names
H11 <- 'group1 = group2 > group3'
H12 <- 'group2 > group1 > group3'
# names(est_2) # Specify restrictions using those names
H21 <- 'gr1 = gr2 > gr3'
H22 <- 'gr2 > gr1 > gr3'

# list of (competing) hypotheses
hypotheses <- list(H1 = list(H11, H12), H2 = list(H21, H22))
```

```
# Evidence synthesis (added-evidence)
evS2_added <- evSyn(object, VCOV = VCOV, hypotheses = hypotheses,
                    type_ev = "added", comparison = "unconstrained")
evS2_added
plot(evS2_added)


## Example 3: based on (multiple) estimates and their covariance matrices
# 3 studies

# generate data
ratio <- c(1,1.1,1.2)
n <- c(30, 50, 100)

# Generate data1
n1 <- n[1]
x11 <- rnorm(n1)
x12 <- rnorm(n1)
x13 <- rnorm(n1)
data <- cbind(x11, x12, x13)
# Standardize data - since parameters for continuous variables will be compared
data1 <- as.data.frame(scale(data))
y1 <- ratio[1]*data1$x11 + ratio[2]*data1$x12 + ratio[3]*data1$x13 + rnorm(n1)
# Note: since there is one outcome, the outcome does not need to be standardized.
fit.lm1 <- lm(y1 ~ 1 + x11 + x12 + x13, data = data1)

# Generate data2
n2 <- n[2]
x21 <- rnorm(n2)
x22 <- rnorm(n2)
x23 <- rnorm(n2)
data <- cbind(x21, x22, x23)
data2 <- as.data.frame(scale(data))
y2 <- ratio[1]*data2$x21 + ratio[2]*data2$x22 + ratio[3]*data2$x23 + rnorm(n2)
fit.lm2 <- lm(y2 ~ 1 + x21 + x22 + x23, data = data2)

# Generate data3
n3 <- n[3]
x31 <- rnorm(n3)
x32 <- rnorm(n3)
x33 <- rnorm(n3)
data <- cbind(x31, x32, x33)
data3 <- as.data.frame(scale(data))
y3 <- ratio[1]*data3$x31 + ratio[2]*data3$x32 + ratio[3]*data3$x33 + rnorm(n3)
fit.lm3 <- lm(y3 ~ 1 + x31 + x32 + x33, data = data3)

# Extract estimates and their covariance matrix (per study)
est_1 <- coef(fit.lm1)
est_2 <- coef(fit.lm2)
est_3 <- coef(fit.lm3)
vcov_est_1 <- vcov(fit.lm1)
vcov_est_2 <- vcov(fit.lm2)
vcov_est_3 <- vcov(fit.lm3)
```

```
names(est_1) <- names(est_2) <- names(est_3) <- c("intercept", "x1", "x2", "x3")

# Parameter estimate values from the primary studies
Param_studies <- list(est_1, est_2, est_3)

# standard error of the beta's
CovMx_studies <- list(vcov_est_1, vcov_est_2, vcov_est_3)

# Set of hypotheses for each study. Note: in this case the same for each study
hypothesis <- 'x1 < x2 < x3'

# In our theory, we compare estimates of continuous variables, so we standardized
# the data beforehand to ensure comparability. In 'Param_studies' and 'CovMx_studies',
# the intercept can be omitted without affecting the GORIC(A) weights, as there are
# no restrictions on it. Since we have only one theory-based hypothesis, we will
# utilize the more powerful complement of the hypothesis (Vanbrabant, Van Loey, Kuiper, 2019).
# The complement represents the remaining 11 theories, while the unconstrained
# scenario includes all 12 possible theories, including H1.

# Evidence synthesis (added-evidence)
evS3 <- evSyn(object = Param_studies, VCOV = CovMx_studies,
              hypotheses = list(H1 = hypothesis),
              type_ev = "added",
              comparison = "complement")
evS3
plot(evS3)


## Example 4: based on loglikelihood values and penalty values

# Make LL and PT values a list
LL <- as.list(data.frame(t(myLLs)))
penalty.values <- as.list(data.frame(t(myPTs)))

# Evidence synthesis
#
# Added-evidence
evS_LL_added <- evSyn(object = LL, PT = penalty.values,
                      hypo_names = colnames(myLLs),
                      type_ev = "added")
evS_LL_added
#
# Equal-evidence
evS_LL_equal <- evSyn(object = LL, PT = penalty.values,
                      hypo_names = colnames(myLLs),
                      type_ev = "equal")
evS_LL_equal


## Example 5: based on AIC, ORIC, GORIC(A) values

# Make GORIC values a list
```

```
goric.values <- as.list(data.frame(t(myGORICs)))

# Evidence synthesis (added-evidence)
evS_Gv <- evSyn(goric.values,
                hypo_names = colnames(myGORICs))
evS_Gv


## Example 6: based on an escalc object

## Example 6a: one outcome, default labeling
if (requireNamespace("metadat", quietly = TRUE)) {
  # access data within metafor
 data_a <- metadat::dat.berkey1998[metadat::dat.berkey1998$outcome == "PD", ][, c(1:4, 6:7)]

  # yi gives the estimates and vi their variances.
  # So, here:
  # cluster_col = "trial" using the first variable with a cluster variable
  # name (year and author are other possibilities)
  # outcome_col = NULL # No 'outcome' variable, so 'theta' is used as labeling
  # yi_col = "yi"
  # vi_cols = "vi"
  #

  # The hypothesis should thus now be based on the default labelling theta:
  H1 <- "abs(theta) > 0.5"
  evS_escalc_a <- evSyn(
    data_a,
    hypotheses = list(H1 = H1),
    type_ev = "equal",
    comparison = "complement"
  )
  evS_escalc_a
}


## Example 6b: one outcome, default labeling coming from 'outcome' variable
if (requireNamespace("metadat", quietly = TRUE)) {
  data_b <- metadat::dat.berkey1998[metadat::dat.berkey1998$outcome == "PD", ][, c(1:7)]
  # yi gives the estimates and vi their variances.
  # So, here:
  # cluster_col = "trial" # using the first variable with a cluster variable
  # name (year and author are other possibilities)
  # outcome_col = "outcome" # was NULL, but found 'outcome' variable, so use that
  # labeling, that is, "PD"
  # yi_col = "yi"
  # vi_cols = "vi"
  #
 # The hypothesis should thus now be based on the labeling based on the 'outcome' variable:
  H1 <- "abs(PD) > 0.5"
  evS_escalc_b <- evSyn(data_b,
                        hypotheses = list(H1 = H1),
                        type_ev = "equal",
```

```
                      comparison = "complement")
  evS_escalc_b
}


## Example 6c: 2 outcomes, default labeling coming from 'outcome' variable
if (requireNamespace("metadat", quietly = TRUE)) {
  data_c <- metadat::dat.berkey1998
  # yi gives the estimates (for both outcomes), vi their variances,
  # and v1i & v2i combined their covariance matrices.
  # So, here:
  # cluster_col = "trial" # using the first variable with a cluster variable
  # name (year and author are other possibilities)
  # outcome_col = "outcome" # was NULL, but found 'outcome' variable, so use
  # that labeling, that is, "PD"
  # yi_col = "yi"
  # vi_cols = c("v1i", "v2i")
  #
 # The hypothesis should thus now be based on the labeling based on the 'outcome' variable:
  H1 <- "abs(AL) > 0.5, abs(PD) > 0.5"
  evS_escalc_c <- evSyn(data_c,
                      hypotheses = list(H1 = H1),
                      type_ev = "equal",
                      comparison = "complement")
  evS_escalc_c
}
```

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

Description

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

Usage

`data(Exam)`

Format

A data frame of 20 observations of 4 variables.

`Scores`	exam scores
`Hours`	study hours
`Anxiety`	anxiety scores
`APS`	average point score

References

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

Examples

```
head(Exam)
```

`FacialBurns` *Dataset for illustrating the conTest_conLavaan function.*

Description

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

Usage

```
data(FacialBurns)
```

Format

A data frame of 77 observations of 6 variables.

`Selfesteem` Rosenberg's self-esteem scale

`HADS` Anxiety and depression scale

`Age` Age measured in years, control variable

`TBSA` Total Burned Surface Area

`RUM` Rumination, control variable

`Sex` Gender, grouping variable

Examples

```
head(FacialBurns)
```

---

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

Description

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

Usage

```
goric(object, ...)

## Default S3 method:
goric(object, ..., hypotheses = NULL,
      comparison = NULL, type = "goric",
      VCOV = NULL, sample_nobs = NULL,
      penalty_factor = 2, Heq = FALSE, control = list(), debug = FALSE)

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

## S3 method for class 'numeric'
goric(object, ..., hypotheses = NULL,
      comparison = NULL, type = "gorica",
      VCOV = NULL, sample_nobs = NULL,
      debug = FALSE)

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

## S3 method for class 'CTmeta'
goric(object, ..., hypotheses = NULL,
      comparison = NULL, type = "gorica",
      sample_nobs = NULL, debug = FALSE)

## S3 method for class 'rma'
goric(object, ..., hypotheses = NULL,
      comparison = NULL, type = "gorica",
      VCOV = NULL, sample_nobs = NULL,
      debug = FALSE)
```

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

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

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

Arguments

`object`	an object containing the outcome of an unconstrained statistical analysis. Currently, the following objects can be processed: • a fitted unconstrained object of class `aov`, `lm`, `glm`, `rlm` or `mlm`. • a numeric (named) vector containing the unconstrained estimates resulting from any statistical analysis. • a fitted object of class `lavaan`. See examples on how to specify the hypotheses. • a fitted object of class `CTmeta`. • a fitted object of class `rma`. • a fitted object of class `nlmerMod`. • a fitted object of class `glmerMod`. • a fitted object of class `lmerMod`.
`x`	an object of class `con_goric`.
`...`	this depends on the class of the object. If object is of class `lavaan`, the standardized or unstandardized vcov can be used, using setting `standardized` = `TRUE`. See details for more information. *Options for calculating the chi-bar-square weights:* Parameters passed to the truncated multivariate normal distribution. By default, restriktor (i.e. `con_weights_boot` function) uses no truncation points for calculating the chi-bar-square weights, which renders to the multivariate normal distribution. See the manual page of the `rtmvnorm` function from the **rtmvnorm** to see how to specify a truncated mvnorm distribution and the possible arguments.
`hypotheses`	a named list; Please note that the hypotheses argument in the given context serves the same purpose as the constraints argument utilized in the restriktor function. The distinction between them is solely semantic. There are two ways to constrain parameters. First, the hypothesis syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of `coef(model)` or `names(vector)` can be used as names. See details for more information. Note that objects of class "mlm" do not (yet) support this method. Second, the hypothesis syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see details.

`comparison`	The default behavior depends on the number of user-specified order-restricted hypotheses. If a single hypothesis is specified, it is compared against its complement by default. When multiple order-restricted hypotheses are specified, the unconstrained model is added as a safeguard to the set of hypotheses. These default settings can be overridden. Use `"unconstrained"` to include the unconstrained model in the set of models. Use `"complement"` to compare the order-restricted object against its complement; note that the complement can only be computed for one model/hypothesis at a time for now. If `"none"` is chosen, the model is compared only against those provided by the user.
`type`	if `"goric"`, the generalized order-restricted information criterion value is computed. If `"gorica"`, a proxi for the log-likihood is calculated by assuming that the parameter estimates (of interest) are multivariate normally distributed. Then, either the unbiased covariance matrix of the estimates (based on 'N'; e.g., for an lm object) is used or the biased one obtained via the vcov function (e.g., for a lavaan object). In the latter case, a corresponding message will be printed. If `"goricc"` or `"goricac"`, then the small-sample-size correction is applied.
`VCOV`	variance-covariance matrix. Only needed if object is of class 'numeric' (and, therefore, `type = "gorica"` or `type = "goricac"`). For many fit objects, one can use the function vcov() to obtain the VCOV. This renders the (biased) restricted sample covariance matrix and not the unbiased sample covariance matrix (based on 'N'). To obtain the unbiased one, one should make an adjustment based on samples size and the rank of the model (i.e., number of predictors including intercept); as shwon in an example below.
`sample_nobs`	the number of observations, only needed if `type = "goricac"`. Note that, for many fit objects and/or if `type = "goricc"`, the number of observations are inherited from the fitted object. In case a vector is given, it is assumed that those are group sizes; then, their sum is used as the number of observations.
`penalty_factor`	The penalty factor adjusts the penalty in the GORIC(A) formula (GORIC(A) = -2 x log-likelihood + penalty_factor x penalty). By default, GORIC(A) uses a penalty factor of 2, but penalty factor allows this to be customized. Higher values of penalty factor place a stronger emphasis on model simplicity, helping to prevent overfitting by penalizing complex models more heavily.
`Heq`	If `TRUE`, the null hypothesis is added to the set of models. This means that all inequality constraints are replaced with equality constraints, effectively testing the hypothesis that the parameters satisfy exact equality rather than inequality. Defaults to `FALSE`.
`missing`	the default setting for objects of class "lm" is listwise: all cases with missing values are removed from the data before the analysis. This is only valid if the data are missing completely at random (MCAR). Another option is to use "two.stage". In this approach, missing data are imputed using an EM algorithm. However, we cannot use the complete data as input for futher analyses, because the resulting complete data variance-covariance matrix will not be correct. Therefore, we compute the correct aymptotic covariance (Savalei and Bentler, 2009) and use it as input for the `goric.numeric` function to compute a GORICA(C) value. Note that, the parameter estimates are also recomputed using the complete data.

`auxiliary`	Vector. The inclusion of auxiliary variables can improve the imputation model. These auxiliary variables are not part of the target model.
`emControl`	a list of control arguments for the `emnorm` function from the **norm** package.
`standardized`	if TRUE, standardized parameter estimates are used.
`digits`	the number of significant digits to use when printing.
`debug`	if TRUE, debugging information is printed out.

Control options for calculating the chi-bar-square weights:

`control`	• `chunk_size` integer; the chi-bar-square weights are computed for samples of size `chunk_size = 5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., `mix_weights_bootstrap_limit`. • `mix_weights_bootstrap_limit` integer; maximum number of bootstrap draws. The default value is set to 1e5. • `convergence_crit` the convergence criterion for the iterative bootstrap process. Default is 1e-03.
`brief`	if TRUE, a short overview is printed.

Details

The GORIC(A) can be used to evaluate theory-based, informative hypotheses, that is, quantify the relative support of theory-based, informative hypotheses. The GORIC(A) values themselves are not interpretable and the interest lie in their differences. The GORIC(A) weights reflect the support of each hypothesis in the set. To compare two hypotheses (and not one to the whole set), one can examine the ratio of the two corresponding GORIC(A) weights. To avoid selecting a weakly supported hypothesis as the best one, the unconstrained hypothesis is usually included as safeguard.

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

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

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

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

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

Note on not full row-rank

If the restriction matrix is not of full row-rank, this means one of the following:

- There is at least one redundant restriction specified in the hypothesis. Then, either
  - [a] Leave the redundant one out
  - [b] Use another (more time-consuming) way of obtaining the level probabilities for the penalty term (goric function does this by default): Bootstrapping, as discussed above.
- There is at least one range restriction (e.g., -2 < group1 < 2). Such a restriction can be evaluated but there is a sensitivity (of a scaling factor in the covariance matrix, like with a prior in a Bayes factor) which currently cannot be checked for.
- There is at least one conflicting restriction (e.g., 2 < group1 < -2).

Such a restriction can evidently never hold and is thus impossible to evaluate. To prevent this type of error delete the one that is incorrect.

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 M. Kuiper

References

Kuiper, R.M., Hoijtink, H., and Silvapulle, M.J. (2011). An Akaike-type information criterion for model selection under inequality constraints. *Biometrika*, 98, 2, 495–501. doi:10.1093/biomet/asr002

Vanbrabant, L. and Kuiper, R. (2020). Evaluating a theory-based hypothesis against its complement using an AIC-type information criterion with an application to facial burn injury. Psychological Methods. Vanbrabant, L., Van Loey, N., and Kuiper, R. M. (2020). Evaluating a Theory-Based Hypothesis Against Its Complement Using an AIC-Type Information Criterion With an Application to Facial Burn Injury. *Psychological Methods*, 25(2), 129-142. doi:10.1037/met0000238

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

Examples

```
## By following these examples, you can appropriately specify hypotheses based on
## your research questions and analytical framework.

# The hypotheses (i.e., constraints) have to be in a list. It is recommended to name
```

```
# each hypothesis in the list. Otherwise the hypotheses are named accordingly 'H1', 'H2', \ldots.
# Another option is to use the \code{llist()} function from the \pkg{Hmisc} package, where.

# text-based syntax (the labels x1, x2, and x2 are the names of coef(model) or names(vector))
h1 <- '(x1, x2, x3) > 0'
h2 <- '(x1, x3) > 0; x2 = 0'
h3 <- 'x1 > 0; x2 < 0; x3 = 0'
hypotheses = list(hypo1 = h1, hypo2 = h2, hypo3 = h3)

# define constraints matrix directly (note that the constraints have to be specified pairwise).
# the element names (i.e., constraints, rhs, neq) must be used.
h1 <- list(constraints = c(0,1,0))
h2 <- list(constraints = rbind(c(0,1,0), c(0,0,1)), rhs = c(0.5, 1), neq = 0)
hypotheses = list(H1 = h1, H2 = h2)

# mixed syntax:
hypotheses = list(Ha = h1, Hb = 'x1 = x2 > x3')

# lavaan object syntax:
# the recommended option for objects of class lavaan is to use labels (here a, b and c)
# to define our hypothesis.
model.lav <- "y ~ 1 + a*x1 + b*x2 + c*x3 + x4"
# fit lavaan model, for example
# library(lavaan)
# fit.lav <- sem(model, data = DATA)
# define hypothesis syntax
hypotheses = list(h1 = 'a > b > c')
# if needed absolute values can be used.
hypotheses = list(h1 = 'abs(a) > abs(b) > abs(c)')

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

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

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

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


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

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

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


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

# extract unconstrained estimates
est <- coef(fit.lm)
# unconstrained variance-covariance matrix
#biased VCOV
VCOV <- vcov(fit.lm)
#unbiased VCOV
N_min_k <- fit.lm$df.residual
N <- N_min_k + fit.lm$rank
VCOV_unbiased <- vcov(fit.lm) * N_min_k / N

## constraint syntax (character)
h1 <- "x1 > 0"
h2 <- "x1 > 0; x2 > 0"
# use fitted unconstrained linear model
goric(fit.lm, hypotheses = list(h1 = h1, h2 = h2), type = "gorica")
# use unconstrained estimates
# unbiased VCOV - then same result as with fit object above
goric(est, VCOV = VCOV_unbiased, hypotheses = list(h1 = h1, h2 = h2), type = "gorica")
# biased VCOV
goric(est, VCOV = VCOV, hypotheses = list(h1 = h1, h2 = 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)), rhs = c(0.5, 1), neq = 0)
goric(fit.lm, hypotheses = list(h1 = h1, h2 = h2), type = "gorica")
goric(est, VCOV = VCOV, hypotheses = list(h1 = h1, h2 = h2), type = "gorica")
```

`Hurricanes` *The Hurricanes Dataset*

Description

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

Usage

```
data(Hurricanes)
```

Format

A data frame of 46 observations of 3 variables.

`Year`

`Hurricanes` Number of Hurricanes

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

References

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

Examples

```
head(Hurricanes)
```

---

iht *function for informative hypothesis testing (iht)*

Description

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

Usage

```
iht(...)

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

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

Arguments

`object`	an object of class `lm` or `rlm`. In this case, the constraint syntax needs to be specified OR an object of class `restriktor`. The constraints are inherited from the fitted restriktor object and do not to be specified again.
`model`	lavaan model syntax specifying the model. See `model.syntax` for more information.
`constraints`	there are two ways to constrain parameters. First, the constraint syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of `coef(model)` can be used as names. See details `restriktor` for more information. Second, the constraint syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see the details in the `restriktor` function.
`data`	the data frame containing the observed variables being used to fit the lavaan model.
`type`	hypothesis test type "A", "B", "C", "global", or "summary" (default). See details for more information.
`test`	test statistic; for information about the null-distribution see details. • for object of class lm; if "F" (default), the F-bar statistic (Silvapulle, 1996) is computed. If "LRT", a likelihood ratio test statistic (Silvapulle and Sen, 2005, chp 3.) is computed. If "score", a global score test statistic (Silvapulle and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, the usual unconstrained F-, Wald-, LR- and score-test statistic is computed. • for object of class rlm; if "F" (default), a robust likelihood ratio type test statistic (Silvapulle, 1992a) is computed. If "Wald", a robust Wald test statistic (Silvapulle, 1992b) is computed. If "score", a global score test statistic (Silvapulle, and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, unconstrained robust F-, Wald-, score-test statistics are computed. • for object of class glm; if "F" (default), the F-bar statistic (Silvapulle, 1996) is computed. If "LRT", a likelihood ratio test statistic (Silvapulle and Sen, 2005, chp 4.) is computed. If "score", a global score test statistic (Silvapulle and Silvapulle, 1995) is computed. Note that, in case of equality constraints only, the usual unconstrained F-, Wald-, LR- and score-test statistic is computed.
`rhs`	vector on the right-hand side of the constraints; $R\theta \geq rhs$. The length of this vector equals the number of rows of the constraints matrix R and consists of zeros by default. Note: only used if constraints input is a matrix or vector.

`neq`	integer (default = 0) treating the number of constraints rows as equality constraints instead of inequality constraints. For example, if `neq = 2`, this means that the first two rows of the constraints matrix R are treated as equality constraints. Note: only used if constraints input is a matrix or vector.
`neq.alt`	integer: number of equality constraints that are maintained under the alternative hypothesis (for hypothesis test type "B").
`R`	Integer; number of bootstrap draws. The default value is set to 1000.
`bootstrap.type`	If `"parametric"`, the parametric bootstrap is used. If `"bollen.stine"`, the semi-nonparametric Bollen-Stine bootstrap is used. The default is set to `"bollen.stine"`.
`return.test`	Logical; if `TRUE`, the function returns bootstrapped test-values.
`double.bootstrap`	If `"standard"` (default) the genuine double bootstrap is used to compute an additional set of plug-in p-values for each bootstrap sample. If `"no"`, no double bootstrap is used. If `"FDB"`, the fast double bootstrap is used to compute second level LRT-values for each bootstrap sample. Note that the `"FDB"` is experimental and should not be used by inexperienced users.
`double.bootstrap.R`	Integer; number of double bootstrap draws. The default value is set to 249.
`double.bootstrap.alpha`	The significance level to compute the adjusted alpha based on the plugin p-values. Only used if `double.bootstrap = "standard"`. The default value is set to 0.05.
`parallel`	The type of parallel operation to be used (if any). If missing, the default is set "no".
`ncpus`	Integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	An optional parallel or snow cluster for use if `parallel = "snow"`. If not supplied, a cluster on the local machine is created for the duration of the `InformativeTesting` call.
`verbose`	Logical; if `TRUE`, information is shown at each bootstrap draw.
`...`	futher options for the `iht` and/or `restriktor` function. See details for more information.

Details

The following hypothesis tests are available:

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

- Type global: equal to Type A but H0 contains additional equality constraints. This test is analogue to the global F-test in lm, where all coefficients but the intercept equal 0.

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

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

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

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

Value

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

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

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

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

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

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

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

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

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

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

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

See Also

quadprog, `conTest`

Examples

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

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

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

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

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

iht(fit1.lm, myConstraints1)


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

iht(fit.restr1)


# Or in matrix notation.
Amat1 <- rbind(c(-1, 0,  1),
               c( 0, 1, -1))
myRhs1 <- rep(0L, nrow(Amat1))
myNeq1 <- 0

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

#########################
```

```
## 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 < group3 < group4 '

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


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

iht(fit2.con)

# increasing means in matrix notation.
Amat2 <- rbind(c(-1, 1, 0, 0),
               c( 0,-1, 1, 0),
               c( 0, 0,-1, 1))
myRhs2 <- rep(0L, nrow(Amat2))
myNeq2 <- 0

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

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

iht(fit2.lm, constraints = myConstraints3)

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


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

```
                    group3 < group4 '

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

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


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

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

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

iht(fit5.lm, constraints = myConstraints5)

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


# testing equality and/or inequality restrictions in SEM:

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

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

```
                    start(.10, .20)*TBSA '


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

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

example6

##########################
### artificial example ###
##########################

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

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

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

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

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

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

# we only generate 10 bootstrap samples in this example; in practice
# you may wish to use a much higher number, say > 1000. The double
# bootstrap is not necessary in case of an univariate ANOVA model.
example7 <- iht(model = model, data = Data.new,
                start = lavaan::parTable(fit),
```

```
                R = 10L, double.bootstrap = "no",
                constraints = constraints)
example7
```

`iht-methods` *Methods for iht*

Description

Print function for objects of class `conTest`.

Usage

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

Arguments

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

Value

No return value, just the result of the print function

Examples

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

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

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

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

iht(fit.con)
```

`Kuiper2012estimates` *Estimates and standard errors from four studies on past experience and buyer trust*

Description

This dataset contains effect estimates for the association between previous experience and buyer trust from four independent studies. The studies used different statistical models (linear, probit, and three-level logistic regression), making the raw parameter estimates not directly comparable across studies. Therefore, GORICA evidence aggregation can be performed using only the study-specific estimates of β_{past} and their uncertainty (standard errors, or equivalently variances).

Usage

```
data(Kuiper2012estimates)
```

Format

An object of class `data.frame` with 4 rows and 4 columns.

study Character. Study identifier (e.g., authors).

year Integer. Publication year of the study.

estimate Numeric. Estimate of β_{past} (effect of previous experience on trust).

se Numeric. Standard error of `estimate`.

Details

Each row corresponds to one study. For each study, the hypothesis of interest is $H_+ : \beta_{past} > 0$. Study-specific GORICA weights quantify support for this hypothesis, and GORICA evidence aggregation combines this support across the four studies to synthesize evidence for the central theory that previous experience positively relates to trust.

Examples

```
data(Kuiper2012estimates)

# Inspect the data
Kuiper2012estimates

# Optional: compute variances from standard errors
# Kuiper2012estimates$var <- Kuiper2012estimates$se^2
```

`myGORICs` *An example of IC values*

Description

An example of IC: A matrix with information criteria (AIC, ORIC, GORIC, or GORICA) values of size 4 x 3 (in general: S x 'NrHypos+1', where 'NrHypos+1' stands for the number of theory-based hypotheses plus a safeguard hypothesis (the complement or unconstrained)).

Usage

```
data(myGORICs)
```

Format

An object of class `matrix` (inherits from `array`) with 4 rows and 3 columns.

Examples

```
data(myGORICs)
```

`myLLs` *An example of log-likelihood (LL) values*

Description

An example of LL: A matrix with log-likelihood values of size 4 x 3 (in general: S x 'NrHypos+1', where 'NrHypos+1' stands for the number of theory-based hypotheses plus a safeguard hypothesis (the complement or unconstrained)).

Usage

```
data(myLLs)
```

Format

An object of class `matrix` (inherits from `array`) with 4 rows and 3 columns.

Examples

```
data(myLLs)
```

`myPTs` *An example of penalty (PT) values*

Description

An example of PT: A matrix with penalty values of size 4 x 3 (in general: S x 'NrHypos+1', where 'NrHypos+1' stands for the number of theory-based hypotheses plus a safeguard hypothesis (the complement or unconstrained)).

Usage

```
data(myPTs)
```

Format

An object of class `matrix` (inherits from `array`) with 4 rows and 3 columns.

Examples

```
data(myPTs)
```

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

Description

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

Usage

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

## S3 method for class 'lm'
conLM(object, constraints = NULL, se = "standard",
      B = 999, rhs = NULL, neq = 0L, mix_weights = "pmvnorm",
      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",
       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",
        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",
        parallel = "no", ncpus = 1L, cl = NULL, seed = NULL,
        control = list(), verbose = FALSE, debug = FALSE, ...)
```

Arguments

`object`	a fitted linear model object of class "lm", "mlm", "rlm" or "glm". For class "rlm" only the loss function `bisquare` is supported for now, otherwise the function gives an error.
`constraints`	there are two ways to constrain parameters. First, the constraint syntax consists of one or more text-based descriptions, where the syntax can be specified as a literal string enclosed by single quotes. Only the names of `coef(model)` can be used as names. See details for more information. Note that objects of class "mlm" do not (yet) support this method. Second, the constraint syntax consists of a matrix R (or a vector in case of one constraint) and defines the left-hand side of the constraint $R\theta \geq rhs$, where each row represents one constraint. The number of columns needs to correspond to the number of parameters estimated (θ) by model. The rows should be linear independent, otherwise the function gives an error. For more information about constructing the matrix R and rhs see details.
`se`	if "`standard`" (default), conventional standard errors are computed based on inverting the observed augmented information matrix. If "const", homoskedastic standard errors are computed. If "`HC0`" or just "`HC`", heteroskedastic robust standard errors are computed (a.k.a Huber White). The options "`HC1`", "`HC2`", "`HC3`", "`HC4`", "`HC4m`", and "`HC5`" are refinements of "`HC0`". For more details about heteroskedastic robust standard errors see the **sandwich** 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.
`parallel`	the type of parallel operation to be used (if any). If missing, the default is set "no".
`ncpus`	integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs.
`cl`	an optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the `restriktor` call.
`seed`	seed value.
`control`	a list of control arguments: • `absval` tolerance criterion for convergence (default = sqrt(.Machine$double.eps)). • `maxit` the maximum number of iterations for the optimizer (default = 10000). • `tol` numerical tolerance value. Estimates smaller than `tol` are set to 0. Control options for calculating the chi-bar-square weights: • `chunk_size` integer; the chi-bar-square weights are computed for samples of size `chunk_size` = `5000L`. This process is repeated iteratively until the weights converges (see `convergenge_crit`) or the maximum is reached, i.e., `mix_weights_bootstrap_limit`. • `mix_weights_bootstrap_limit` integer; maximum number of bootstrap draws. The default value is set to 1e5. • `convergence_crit` the convergence criterion for the iterative bootstrap process. Default is 1e-03.
`verbose`	logical; if TRUE, information is shown at each bootstrap draw.
`debug`	if TRUE, debugging information about the constraints is printed out.

Options for calculating the chi-bar-square weights:

`...`	parameters passed to the truncated multivariate normal distribution. By default, restriktor (i.e. `con_weights_boot` function) uses no truncation points for calculating the chi-bar-square weights, which renders to the multivariate normal distribution. See the manual page of the `rtmvnorm` function from the **rtmvnorm** to see how to specify a truncated mvnorm distribution and the possible arguments.

Details

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note on not full row-rank

If the restriction matrix is not of full row-rank, this means one of the following:

- There is at least one redundant restriction. Then, either
  - [a] Leave the redundant one out
  - [b] Use another (more time-consuming) way of obtaining the level probabilities for the penalty term (goric function does this by default): Bootstrapping, as discussed above.
- There is at least one range restriction (e.g., -2 < group1 < 2). Such a restriction can be evaluated but there is a sensitivity (of a scaling factor in the covariance matrix, like with a prior in a Bayes factor) which currently cannot be checked for.
- There is at least one conflicting restriction (e.g., 2 < group1 < -2).

Such a restriction can evidently never hold and is thus impossible to evaluate. To prevent this type of error delete the one that is incorrect.

Value

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

`CON`	a list with useful information about the restrictions.
`call`	the matched call.
`timing`	how much time several tasks take.
`parTable`	a parameter table with information about the observed variables in the model and the imposed restrictions.
`b.unrestr`	unrestricted regression coefficients.
`b.restr`	restricted regression coefficients.
`residuals`	restricted residuals.

`wresid`	a working residual, weighted for "inv.var" weights only (rlm only)
`fitted`	restricted fitted mean values.
`weights`	(only for weighted fits) the specified weights.
`wgt`	the weights used in the IWLS process (rlm only).
`scale`	the robust scale estimate used (rlm only).
`stddev`	a scale estimate used for the standard errors.
`R2.org`	unrestricted R-squared.
`R2.reduced`	restricted R-squared.
`df.residual`	the residual degrees of freedom
`s2.unrestr`	mean squared error of unrestricted model.
`s2.restr`	mean squared error of restricted model.
`loglik`	restricted log-likelihood.
`Sigma`	variance-covariance matrix of unrestricted model.
`constraints`	matrix with restrictions.
`rhs`	vector of right-hand side elements.
`neq`	number of equality restrictions.
`wt.bar`	chi-bar-square mixing weights or a.k.a. level probabilities.
`iact`	active restrictions.
`converged`	did the IWLS converge (rlm only)?
`iter`	number of iteration needed for convergence (rlm only).
`bootout`	object of class boot. Only available if bootstrapped standard errors are requested, else bootout = NULL.
`control`	list with control options.
`model.org`	original model.
`se`	as input. This information is needed in the summary function.
`information`	observed information matrix with the inverted information matrix and the augmented information matrix as attributes.

Author(s)

Leonard Vanbrabant and Yves Rosseel

References

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

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

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

See Also

iht, goric

Examples

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

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

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

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

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

# Or in matrix notation.
myConstraints1 <- rbind(c(-1, 1, 0),
                        c( 0,-1, 1))
myRhs1 <- rep(0L, nrow(myConstraints1))
myNeq1 <- 0

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

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

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

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

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

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

summary(fit2.con)


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

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

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

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


## increasing means in matrix notation.
myConstraints3 <- rbind(c(-1, 1, 0, 0),
                        c( 0,-1, 1, 0),
                        c( 0, 0,-1, 1))
myRhs3 <- rep(0L, nrow(myConstraints3))
myNeq3 <- 0

fit3.con <- restriktor(fit3.rlm, constraints = myConstraints3,
                       rhs = myRhs3, neq = myNeq3, se = "HC0")
summary(fit3.con)

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

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


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

```
                        group3  < group4 '

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

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

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

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

---

restriktor-methods *Methods for restriktor*

Description

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

Usage

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

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

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

```
## S3 method for class 'restriktor'
coef(object, ..., which = c("restr", "unrestr"))

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

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

Arguments

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

Details

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

Value

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

`se.type`	type of standard error computed, equal to input `se` in the `restriktor` function.

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

Examples

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

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

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

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

summary(fit.con)
```

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

Description

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

Usage

```
data(ZelazoKolb1972)
```

Format

A data frame of 23 observations of 4 treatment variables.

`Age` Age in months

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

References

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

Examples

```
head(ZelazoKolb1972)
```

Index