

# Package ‘ic.infer’

April 17, 2009

**Title** Inequality constrained inference in linear normal situations

**Version** 1.0-6

**Depends** R(>= 2.5.0), quadprog, mvtnorm, boot, kappalab

**Date** 2008-09-19

**Author** Ulrike Groemping

**Maintainer** Ulrike Groemping <groemping@tfh-berlin.de>

**Description** This package implements parameter estimation in normal (linear) models under linear equality and inequality constraints and implements normal likelihood ratio tests involving inequality-constrained hypotheses. For inequality-constrained linear models, averaging over R-squared for different orderings of regressors is also included.

**License** GPL (>= 2)

**Encoding** latin1

**URL** <http://prof.tfh-berlin.de/groemping/>

**Repository** CRAN

**Date/Publication** 2008-09-21 19:48:34

## R topics documented:

ic.est . . . . .	2
ic.infer . . . . .	4
ic.test . . . . .	6
ic.weights . . . . .	10
internal.functions . . . . .	12
or.relimp . . . . .	13
orlm . . . . .	15

<b>Index</b>	<b>20</b>
--------------	-----------

ic.est

*Functions for order-restricted estimates and printing thereof***Description**

Function `ic.est` estimates a mean vector under linear inequality constraints, functions `print.orest` and `summary.orest` provide printed results in different degrees of detail.

**Usage**

```
ic.est(x, Sigma, ui, ci = NULL, index = 1:nrow(Sigma), meq = 0,
      tol = sqrt(.Machine$double.eps))
## S3 method for class 'orest':
print(x, digits = max(3, getOption("digits") - 3), scientific = FALSE, ...)
## S3 method for class 'orest':
summary(object, display.unrestr = FALSE, brief = FALSE,
      digits = max(3, getOption("digits") - 3), scientific = FALSE, ...)
```

**Arguments**

<code>x</code>	for <code>ic.est</code> : unrestricted vector (e.g. mean of a sample of random vectors), from which the expected value under linear inequality (and perhaps equality) restrictions is to be estimated for <code>print.orest</code> : object of class <code>orest</code> (normally produced by <code>ic.est</code> or <code>orlm</code> )
<code>object</code>	for <code>summary.orest</code> : object of class <code>orest</code> (normally produced by <code>ic.est</code> or <code>orlm</code> )
<code>Sigma</code>	covariance or correlation matrix (or any multiple thereof) of <code>x</code>
<code>ui</code>	matrix (or vector in case of one single restriction only) defining the left-hand side of the restriction $ui \%*\% \mu \geq ci$ , where $\mu$ is the expectation vector of <code>x</code> ; the first few of these restrictions can be declared equality- instead of inequality restrictions (cf. argument <code>meq</code> ); if only part of the elements of $\mu$ are subject to restrictions, the columns of <code>ui</code> can be restricted to these elements, if their index numbers are provided in <code>index</code> Rows of <code>ui</code> must be linearly independent; in case of linearly dependent rows the function gives an error message with a hint which subset of rows is independent. Note that the restrictions must define a (possibly translated) cone, i.e. e.g. interval restrictions on a parameter are not permitted.
<code>ci</code>	vector on the right-hand side of the restriction (cf. <code>ui</code> ), defaults to a vector of zeroes
<code>index</code>	index numbers of the components of $\mu$ , which are subject to the specified constraints as $ui \%*\% \mu[index] \geq ci$
<code>meq</code>	integer number (default 0) giving the number of rows of <code>ui</code> that are used for equality restrictions instead of inequality restrictions.

tol	numerical tolerance value; estimates closer to 0 than tol are set to exactly 0
digits	number of digits to be used in printing
scientific	if FALSE, suppresses scientific representation of numbers (default: FALSE)
...	further arguments to print
display.unrestr	if TRUE, unrestricted estimate (i.e. object) is also displayed
brief	if TRUE, suppress printing of restrictions; default: FALSE

## Details

Function `ic.est` heavily relies on package **quadprog** for determining the optimizer. It is a convenience wrapper for `solve.QP` from that package. The function is guaranteed to work appropriately if the specified restrictions determine a (translated) cone. In that case, the estimate is the projection along matrix `Sigma` onto one of the faces of that cone (including the interior as the face of the highest dimension); this means that it minimizes the quadratic form  $t(x-b) \% \% \text{solve}(Sigma, x-b)$  among all `b` that satisfy the restrictions  $ui \% \% b \geq ci$  (or, if specified by `meq`, with the first `meq` restrictions equality instead of inequality restrictions).

## Value

Function `ic.est` outputs a list with the following elements:

<code>b.unrestr</code>	<code>x</code>
<code>b.restr</code>	restricted estimate
<code>Sigma</code>	as input
<code>ui</code>	as input
<code>ci</code>	as input
<code>restr.index</code>	index of components of <code>mu</code> , which are subject to the specified constraints as in input index
<code>meq</code>	as input
<code>iact</code>	active restrictions, i.e. restrictions that are satisfied with equality in the solution, as output by <code>solve.QP</code>

## Author(s)

Ulrike Groemping, TFH Berlin

## See Also

See also [ic.test](#), [ic.weights](#), [orlm](#), `solve.QP`

## Examples

```
## different correlation structures
corr.plus <- matrix(c(1,0.9,0.9,1),2,2)
corr.null <- matrix(c(1,0,0,1),2,2)
corr.minus <- matrix(c(1,-0.9,-0.9,1),2,2)
## unrestricted vectors
x1 <- c(1, -1)
x2 <- c(-1, -1)
x3 <- c(10, -1)
## estimation under restriction non-negative orthant
## or first element equal to 0, second non-negative
ice <- ic.est(x1, corr.plus, ui=diag(c(1,1)), ci=c(0,0))
ice
summary(ice)
ice2 <- ic.est(x1, corr.plus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
summary(ice2)
ic.est(x2, corr.plus, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x2, corr.plus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x3, corr.plus, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x3, corr.plus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x1, corr.null, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x1, corr.null, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x2, corr.null, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x2, corr.null, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x3, corr.null, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x3, corr.null, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x1, corr.minus, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x1, corr.minus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x2, corr.minus, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x2, corr.minus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
ic.est(x3, corr.minus, ui=diag(c(1,1)), ci=c(0,0))
ic.est(x3, corr.minus, ui=diag(c(1,1)), ci=c(0,0), meq=1)
## estimation under one element restricted to being non-negative
ic.est(x3, corr.plus, ui=1, ci=0, index=1)
ic.est(x3, corr.plus, ui=1, ci=0, index=2)
```

---

ic.infer

*Package for inequality-constrained estimation and testing*

---

## Description

Package `ic.infer` implements estimation and testing for multivariate normal expectations with linear equality- and inequality constraints. This also includes inference on linear models with linear equality- and inequality constraints on the parameters. Decomposition of R-squared is also included for these models.

## Details

Function `ic.est` estimates the constrained expectation of a multivariate normal random vector, function `ic.test` conducts related tests.

Function `orlm` estimates constrained parameters in normal linear models based on a linear model object or a covariance matrix. The function offers the possibility of bootstrapping the estimates. Tests and confidence intervals are provided by a summary function.

Function `or.relimp` decomposes the  $R^2$ -values analogously to metric `lmg` in package **relaimpo** for unconstrained linear models. However, `or.relimp` is far less comfortable to use and subject to severe limitations, since automatic selection of restrictions for sub models is not in all cases trivial.

The package makes use of various other R packages: **quadprog** is used for constrained estimation, **mvtnorm** in calculation of weights for null distributions of test statistics, **kappalab** for averaging over orderings in function `or.relimp`, and **boot** for bootstrapping.

### Value

The output of function `ic.est` belongs to S3 class `orest`.

The output of function `ic.test` belongs to S3 class `ict`.

The output of function `orlm` belongs to S3 classes `orlm` and `orest`.

All these classes offer print and summary methods.

The output of function `or.relimp` is a named vector.

### Acknowledgements

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

It also uses modifications of numerical routines that were provided by John Fox in R-help.

### Author(s)

Ulrike Groemping, TFH Berlin

### References

- Kudo, A. (1963) A multivariate analogue of the one-sided test. *Biometrika* **50**, 403–418
- Sasabuchi, S. (1980) A test of a multivariate normal mean with composite hypotheses determined by linear inequalities. *Biometrika* **67**, 429–429
- 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. (2004) *Constrained Statistical Inference*. Wiley, New York

### See Also

See also `ic.est`, `ic.test`, `orlm`, `or.relimp`, packages **boot**, **kappalab**, **mvtnorm**, **quadprog**, and **relaimpo**

---

<code>ic.test</code>	<i>Function for testing inequality-related hypotheses for multivariate normal random variables</i>
----------------------	--

---

## Description

`ic.test` tests linear inequality hypotheses for multivariate normal means by likelihood ratio tests. `print` and `summary` functions display results in different degrees of detail.

## Usage

```
ic.test(obj, TP = 1, s2 = 1, df.error = Inf,
        ui0.11 = diag(rep(1, length(obj$b.restr))),
        ci0.11 = NULL, meq.alt = 0,
        df = NULL, wt = NULL, tol=sqrt(.Machine$double.eps), ...)
## S3 method for class 'ict':
print(x, digits = max(3, getOption("digits") - 3), scientific = FALSE, ...)
## S3 method for class 'ict':
summary(object, brief = TRUE, digits = max(3, getOption("digits") - 3),
        scientific = FALSE, tol=sqrt(.Machine$double.eps), ...)
```

## Arguments

<code>obj</code>	Object of class <code>orest</code> that contains unrestricted and restricted estimate, covariance structure, and restriction; for objects of class <code>orlm</code> (that inherit from class <code>orest</code> ) information on <code>s2</code> and <code>df.error</code> is taken from <code>obj</code> (i.e. specifications of <code>s2</code> and <code>df.error</code> in the call to <code>ic.test</code> are ignored)
<code>TP</code>	type of test problem, cf. details
<code>s2</code>	multiplier that modifies the matrix <code>obj\$Sigma</code> into the (estimated) covariance matrix of the unrestricted estimate; <code>obj\$Sigma</code> may be a covariance matrix ( <code>s2=1</code> , default), a correlation matrix or an otherwise rescaled covariance matrix (e.g. <code>cov.unscaled</code> from a linear model)
<code>df.error</code>	error degrees of freedom connected with estimation of <code>s2</code> (e.g. residual <code>df</code> from linear model); if <code>df.error &lt; Inf</code> , the test is based on a mixture of beta-distributions with parameters <code>df/2</code> and <code>df.error/2</code> , otherwise the test is based on a mixture of chi-square distributions with degrees of freedom in <code>df</code> .
<code>ui0.11</code>	matrix (or vector in case of one restriction only) for defining (additional) equality restrictions for TP 11 (in addition to restrictions in <code>obj</code> ); note that there must be as many columns as there are elements of vector <code>b.restr</code> (no extra index vector taken); if there is overlap between restrictions in <code>ui0.11</code> and restrictions already present in <code>obj</code> , restrictions already present in <code>obj</code> are projected out for <code>ui0.11</code> : for example, the default choice for <code>ui0.11</code> means that all elements of the expectation are 0; some of these restrictions may already be present in <code>obj</code> and are projected out of <code>ui0.11</code> by <code>ic.test</code>

<code>ci0.11</code>	right-hand-side vector for equality restrictions defined by <code>ui0.11</code> ; so far, these should be 0!
<code>meq.alt</code>	number of equality restrictions (from beginning) that are maintained under the alternative hypothesis (for TP21)
<code>df</code>	optional vector of degrees of freedom for mixed chibar- or beta- distributions; if omitted, degrees of freedom and weights are calculated; if given, must be accompanied by corresponding <code>wt</code>
<code>wt</code>	optional vector of weights for mixed chibar- or beta- distributions; if omitted, weights are calculated using function <code>ic.weights</code> ; if given, must be accompanied by corresponding <code>df</code> (can be obtained from call to <code>ic.weights</code> or from previous runs of <code>ic.test</code> )
<code>x</code>	output object from <code>ict.test</code> (of class <code>ict</code> )
<code>tol</code>	numerical tolerance value; estimates closer to 0 than <code>tol</code> are set to exactly 0
<code>...</code>	Further options, e.g. algorithm for <code>ic.weights</code>
<code>digits</code>	number of digits to display
<code>scientific</code>	if FALSE, suppresses scientific format; default: FALSE
<code>object</code>	output object from <code>ict.test</code> (of class <code>ict</code> )
<code>brief</code>	if TRUE, requests brief output without restrictions (default), otherwise restrictions are shown with indication, which are active

## Details

The following test problems are implemented:

TP=1: H0: restrictions valid with equality vs. H1: at least one inequality

TP=2: H0: all restrictions true vs. H1: at least one restriction false

TP=3: H0: restrictions false vs. H1: restrictions true (with inequality)

TP=11: H0: restriction valid with equality and further linear equalities vs. H1: at least one equality from H0 violated, restriction valid

TP=21: H0: restrictions valid (including some equality restrictions) vs. H1: at least one restriction from H0 violated, some equality restrictions are maintained

Note that TPs 1 and 11 can reject H0 even if H1 is violated by the data. Rejection of H0 does not provide evidence for H1 (but only against H0) in these TPs because H1 is not the opposite of H0. The tests concentrate their power in H1, but are only guaranteed to observe their level for the stated H0.

Also note that TP 3 does not make sense if `obj` involves equality restrictions (`obj$meq>0`).

Under TPs 1, 2, 11, and 21, the distributions of test statistics are mixtures of chi-square distributions (`df.error=Inf`) or beta-distributions (`df.error` finite) with different degrees of freedom (chi-square) or parameter combinations (beta). Shapiro (1988) gives detailed information on the mixing weights for the different scenarios. Basically, there are two different situations:

If `meq=0`, the weights are probabilities that a random variable with covariance matrix `ui%%cov%%t(ui)` is realized in the positive orthant or its lower-dimensional faces, respectively (if `ui` has too few columns, blow up by columns of 0s in appropriate positions) (Shapiro, formulae (5.5) or (5.10), respectively).

If  $meq > 0$  (but not all restrictions are equality restrictions), the weights are probabilities that a random variable with covariance matrix the inverse of the lower right corner of `solve(ui***cov***t(ui))` is realized in the positive orthant or its lower-dimensional faces, respectively (Shapiro, formula (5.9)).

These weights must then be combined with the appropriate degrees of freedom - these can be worked out by realizing that either the null hypothesis or the alternative hypothesis has fixed dimension and the respective mixing degrees of freedom are obtained by taking the difference to the dimension of the respective other hypothesis, which is correct because - given a certain dimension of the inequality-restricted estimate, the inequality-restricted estimate is a projection onto a linear space of that dimension.

The test for TP 3 (cf. e.g. Sasabuchi 1980) is based on the intersection-union principle and simply obtains its p-value as the maximum p-value from testing the individual restrictions.

### Value

object of class `ict`, which is a list containing elements

<code>TP</code>	test problem identifier (cf. argument <code>TP</code> )
<code>b.unrestr</code>	unrestricted estimate
<code>b.restr</code>	restricted estimate
<code>ui</code>	restriction matrix, LHS
<code>ci</code>	restriction vector, RHS
<code>restr.index</code>	elements of mean referred to by <code>ui</code> and <code>ci</code>
<code>meq</code>	number of equality restrictions (first <code>meq</code> rows of <code>ui</code> ), <code>meq</code> must not exceed <code>nrow(ui)-1</code>
<code>iact</code>	row numbers of active restrictions (all equality restrictions plus inequality restrictions that are met with equality by the solution <code>b.restr</code> )
<code>ui.extra</code>	additional restrictions for <code>TP=11</code> , calculated from input parameter <code>ui0.11</code> by projecting out restrictions present in <code>ui</code> and - if necessary - omitting linearly dependent rows
<code>b.eqrestr</code>	equality-restricted estimate for <code>TP=1</code>
<code>b.extra.restr</code>	estimate for null hypothesis of <code>TP=11</code>
<code>T</code>	test statistic
<code>p.value</code>	p-value
<code>s2</code>	input parameter
<code>cov</code>	matrix with <code>s2*cov</code> equal to covariance matrix of unrestricted estimate
<code>df.error</code>	input parameter
<code>df.bar</code>	vector of degrees of freedom for test statistic distribution, cf. also input parameter <code>df</code>
<code>wt.bar</code>	vector of weights for test statistic distribution, cf. also input parameter <code>wt</code>

**Author(s)**

Ulrike Groemping, TFH Berlin

**References**

- Sasabuchi, S. (1980) A test of a multivariate normal mean with composite hypotheses determined by linear inequalities. *Biometrika* **67**, 429–429
- Shapiro, A. (1988) Towards a unified theory of inequality-constrained testing in multivariate analysis. *International Statistical Review* **56**, 49–62

**See Also**

See also [ic.est](#), [ic.weights](#)

**Examples**

```

corr.plus <- matrix(c(1,0.5,0.5,1),2,2)
corr.null <- matrix(c(1,0,0,1),2,2)
corr.minus <- matrix(c(1,-0.5,-0.5,1),2,2)
## unrestricted vectors
x1 <- c(1, 1)
x2 <- c(-1, 1)
ict1 <- ic.test(ic.est(x1, corr.plus, ui=diag(c(1,1)), ci=c(0,0)))
ict1
summary(ict1)
ic.test(ic.est(x1, corr.plus, ui=diag(c(1,1)), ci=c(0,0)), s2=1, df.error=10)
ic.test(ic.est(x1, corr.minus, ui=diag(c(1,1)), ci=c(0,0)))
ic.test(ic.est(x1, corr.minus, ui=diag(c(1,1)), ci=c(0,0)), s2=1, df.error=10)
ic.test(ic.est(x2, corr.plus, ui=diag(c(1,1)), ci=c(0,0)))
ic.test(ic.est(x2, corr.plus, ui=diag(c(1,1)), ci=c(0,0)), s2=1, df.error=10)
ic.test(ic.est(x2, corr.minus, ui=diag(c(1,1)), ci=c(0,0)))
ic.test(ic.est(x2, corr.minus, ui=diag(c(1,1)), ci=c(0,0)), s2=1, df.error=10)

ict2 <- ic.test(ic.est(x2, corr.plus, ui=diag(c(1,1)), ci=c(0,0)),TP=2)
summary(ict2)
ict3 <- ic.test(ic.est(x1, corr.plus, ui=diag(c(1,1)), ci=c(0,0)),TP=3)
summary(ict3)

ict11 <- ic.test(ic.est(x1, corr.plus, ui=c(1,1), ci=0),TP=11, ui0.11 =c(1,0))
summary(ict11)

## larger example
corr.plus <- diag(1,8)
for (i in 1:7)
  for (j in (i+1):8)
    corr.plus[i,j] <- corr.plus[j,i] <- 0.5
u <- rbind(rep(1,6), c(-1,-1,-1,1,1,1), c(-1,0,1,0,0,0), c(0,0,0,-1,0,1))
ice <- ic.est(c(rep(1,4),rep(4,4)), corr.plus, ui=u, ci=rep(0,4), index=2:7, meq = 1)
ict1 <- ic.test(ice,TP=1)
summary(ict1)
ict2 <- ic.test(ice,TP=2)

```

```
summary(ict2)
ict11 <- ic.test(ice,TP=11)
summary(ict11,digits=3)
ice <- ic.est(c(rep(1,4),rep(4,4)), corr.plus, ui=u, ci=rep(0,4), index=2:7)
ict3 <- ic.test(ice, TP=3)
summary(ict3)
```

---

ic.weights                    *functions for calculating the distributions of normal distribution  
order-related likelihood ratio tests*

---

### Description

Test statistics of normal distribution-based order-related likelihood ratio tests are often distributed as mixtures of chi-square or beta-distributions with different parameters. These functions determine the mixing weights and the cumulative distribution functions based on these. They can be directly used and are called by function `ic.test`.

### Usage

```
ic.weights(corr, ...)
pchibar(x, df, wt)
pbetabar(x, df1, df2, wt)
```

### Arguments

<code>corr</code>	<code>corr</code> is the correlation or covariance matrix (or any multiple thereof) of the data or coefficients for which weights are to be calculated
<code>...</code>	<code>...</code> contains further arguments to be given to function <code>pmvnorm</code> of package <code>mvtnorm</code> for calculating multivariate normal rectangle probabilities; it is possible to select an algorithm (default in current version of <code>mvtnorm</code> : <code>algorithm = GenzBretz()</code> ) and to tune weight accuracy by modifying including additional parameters into the algorithm specification, cf. <code>help</code> for <code>GenzBretz</code>
<code>x</code>	<code>x</code> is the quantile for which the distribution function is to be calculated
<code>df</code>	is the vector of the degrees of freedom for the chi-square distributions that are mixed into the chibar-square-distribution with the proportions given in <code>wt</code>
<code>wt</code>	each element of <code>wt</code> is the mixing weight of the chi-square distribution with <code>df</code> as in the corresponding element of <code>df</code> ; such weights can be calculated with function <code>ic.weights</code>
<code>df1</code>	vector of first parameters of the beta-distributions to be mixed into the betabar-distribution
<code>df2</code>	second parameter of the beta-distributions to be mixed into the betabar-distribution; error degrees of freedom in the tests implemented for linear models in <code>summary.orlm</code>

## Details

Function `ic.weights` uses results by Kudo (1963) regarding the calculation of the weights. The weights are the probabilities that the projection along its covariance onto the non-negative orthant of a multivariate normal random vector with expectation 0 and correlation `corr` lies in faces of dimensions `nrow(corr) : 1` (in this order). It is known that these probabilities coincide with various other useful probabilities related to order-related hypothesis testing, cf. e.g. Shapiro (1988). Calculation of the weights involves various calls to function `pmvnorm` from package `mvtnorm`.

Functions `pchibar` (taken from package `ibdreg`) and `pbetabar` calculate cumulative probabilities from mixtures of chi-square and beta-distributions, respectively.

## Value

`ic.weights` returns the vector of weights, `pchibar` and `pbetabar` return the cumulative probability of the respective distribution. Function `ic.weights` relies on package `mvtnorm` for determining multivariate normal rectangle probabilities. Note that these calculations involve Monte Carlo steps so that these weights are not completely repeatable.

## Author(s)

Ulrike Groemping, TFH Berlin

## References

- Kudo, A. (1963) A multivariate analogue of the one-sided test. *Biometrika* **50**, 403–418
- 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. (2004) *Constrained Statistical Inference*. Wiley, New York

## See Also

`ic.test`, `orlm`, `pmvnorm`, `GenzBretz`

## Examples

```
z <- 0.5
corr <- matrix(c(1, 0.9, 0.9, 1), 2, 2)
print(wt.plus <- ic.weights(corr))
T <- c(z, z)
1-pchibar(T, 2:0, wt.plus)
1-pbetabar(T/(T+10), 2:0, 10, wt.plus)
corr <- matrix(c(1, 0, 0, 1), 2, 2)
print(wt.0 <- ic.weights(corr))
T <- c(z, z)
1-pchibar(T, 2:0, wt.0)
1-pbetabar(T/(T+10), 2:0, 10, wt.0)
corr <- matrix(c(1, -0.9, -0.9, 1), 2, 2)
print(wt.minus <- ic.weights(corr))
T <- c(z, z)
1-pchibar(T, 2:0, wt.minus)
1-pbetabar(T/(T+10), 2:0, 10, wt.minus)
```

---

internal.functions *internal functions not intended for the user*

---

### Description

nchoosek is originally taken from package vsn by Wolfgang Huber, GaussianElimination and RREF have been provided by John Fox in R-help and have been modified by the author to provide more output

### Usage

```
nchoosek(n, k)    ## not exported, calculates all combinations
GaussianElimination(A, B, tol=sqrt(.Machine$double.eps),
  verbose=FALSE) ## not exported
RREF(X, ...)     ## not exported, calculates reduced Echelon form
```

### Arguments

n	number of elements to choose from
k	number of elements to choose
A	argument to GaussianElimination
B	argument to GaussianElimination
tol	argument to GaussianElimination
verbose	argument to GaussianElimination
X	matrix to be reduced to reduced Echelon form
...	further arguments to GaussianElimination

### Value

nchoosek returns all subsets of size k, for GaussianElimination and RREF cf. comments in code. The latter are used for reducing a matrix with less than full row rank to a set of linearly independent rows.

### Author(s)

Ulrike Groemping, TFH Berlin, based on code by John Fox and Wolfgang Huber

### See Also

[ic.test](#), [orlm](#)

**Examples**

```

z <- 0.5
corr <- matrix(c(1,0.9,0.9,1),2,2)
print(wt.plus <- ic.weights(corr))
T <- c(z,z)
l-pchibar(T,2:0,wt.plus)
l-pbetabar(T/(T+10),2:0,10,wt.plus)
corr <- matrix(c(1,0,0,1),2,2)
print(wt.0 <- ic.weights(corr))
T <- c(z,z)
l-pchibar(T,2:0,wt.0)
l-pbetabar(T/(T+10),2:0,10,wt.0)
corr <- matrix(c(1,-0.9,-0.9,1),2,2)
print(wt.minus <- ic.weights(corr))
T <- c(z,z)
l-pchibar(T,2:0,wt.minus)
l-pbetabar(T/(T+10),2:0,10,wt.minus)

```

---

or.relimp

---

*Function to calculate relative importance for order-restricted linear models*


---

**Description**

The function calculates relative importance by averaging over the variables R-squared contributions from all orderings of variables for linear models with inequality restrictions on the parameters. NOTE: only useful if each restriction refers to exactly one variable, or if it is adequate to reduce multi-variable restrictions by omitting the affected variables but leaving the restriction otherwise intact.

**Usage**

```

or.relimp(model, ui, ci = NULL, ...)
## S3 method for class 'lm':
or.relimp(model, ui, ci = NULL, index = 2:length(coef(model)), meq = 0,
          tol = sqrt(.Machine$double.eps), ...)

## Default S3 method:
or.relimp(model, ui, ci = NULL, index = 2:ncol(model), meq = 0,
          tol = sqrt(.Machine$double.eps), ...)

all.R2(covmat, ui, ci = NULL, index = 2:ncol(covmat), meq = 0,
       tol = sqrt(.Machine$double.eps), ...)
## user does not need to call this function

```

**Arguments**

<code>model</code>	a linear model object of class <code>lm</code> with data included; for function <code>or.relimp</code> , all explanatory variables must be numeric (i.e. no factors), and higher-order terms (e.g. interactions) are not permitted.
	OR
	the covariance matrix of the response (first position) and all regressors
<code>covmat</code>	the covariance matrix of the response (first position) and all regressors
<code>ui</code>	cf. explanation in <code>link{orlm}</code> ; cf. also details below
<code>ci</code>	cf. explanation in <code>link{orlm}</code>
<code>index</code>	cf. explanation in <code>link{orlm}</code>
<code>meq</code>	cf. explanation in <code>link{orlm}</code>
<code>tol</code>	cf. explanation in <code>link{orlm}</code>
<code>...</code>	Further options

**Details**

Function `or.relimp` uses function `all.R2` for calculating the R-squared values of all subsets that are subsequently handed to function `Shapley.value` (from package `kappalab`), which takes care of the averaging over ordering.

**WARNING:** In models with subsets of the regressors, the columns of the matrix `ui` referring to regressors outside the current subset are simply deleted for the sub model. This is only reasonable if either the individual constraints refer to individual parameters only (e.g. all parameters restricted to be non-negative) or if the constraints are still reasonable in the sub model with some variables deleted, e.g. perhaps (depending on the application) sum of all parameters less or equal to 1.

**WARNING:** If the number of regressors ( $p$ ) is large, the functions quickly becomes unmanageable (a vector of size  $2^p$  is returned or handled in the process).

**Value**

`all.R2` returns a vector ( $2^p$  elements) with all R-squared values ( $p$  is the number of regressors, vector is ordered from empty to full model in natural order (cf. `ic.infer:::nchoosek` for the order within one model size).

`or.relimp` returns a vector ( $p$  elements) with average R-squared contributions from all models with respective subset of restrictions `ui %*% beta >= ci` enforced.

**Author(s)**

Ulrike Groemping, TFH Berlin

**See Also**

See also `orlm` for order-restricted linear models and `calc.relimp` from R-package `relaimpo` for a much more comfortable and much faster routine for unrestricted linear models

**Examples**

```

covswiss <- cov(swiss)
## all R2-values for restricted linear model with restrictions that
## Catholic and Infant.Mortality have non-negative coefficients
R2s <- all.R2(covswiss, ui=rbind(c(0,0,0,1,0),c(0,0,0,0,1)))
R2s

Shapley.value(set.func(R2s)) ## directly using package kappalab

### with convenience wrapper from this package
or.relimp(covswiss, ui=rbind(c(0,0,0,1,0),c(0,0,0,0,1)))

### also works on linear models
limo <- lm(swiss)
#or.relimp(limo, ui=rbind(c(0,0,0,1,0),c(0,0,0,0,1)))

## same model using index vector
or.relimp(limo, ui=rbind(c(1,0),c(0,1)), index=5:6)

```

---

orlm

---

*Functions for order restricted linear regression estimation and testing*


---

**Description**

Function `orlm` calculates order-restricted linear models (linear equality and inequality constraints). It uses the internal function `boot.orlm` for bootstrapping, which in turn uses the internal functions `orlm.forboot...`. The remaining functions extract coefficients, provide a residual plot, give a short printout or a more extensive summary.

**Usage**

```

orlm(model, ui, ci, ...)
## S3 method for class 'lm':
orlm(model, ui, ci, index = 2:length(coef(model)), meq = 0,
      orig.out = FALSE, boot = FALSE, B = 1000, fixed = FALSE,
      tol = sqrt(.Machine$double.eps), ...)
## Default S3 method:
orlm(model, ui, ci, index = NULL, meq = 0,
      tol = sqrt(.Machine$double.eps), df.error = NULL, ...)
boot.orlm(model, B = 1000, fixed = FALSE, ui, ci, index, meq)
orlm.forboot.fixed(data, indices, ...)
orlm.forboot(data, indices, index = index, ...)
## S3 method for class 'orlm':
coef(object, ...)
## S3 method for class 'orlm':
plot(x, caption = "Residuals vs Fitted",
      panel = if (add.smooth) panel.smooth else points, sub.caption = NULL,

```

```

main = "", ..., id.n = 3, labels.id = names(x$residuals), cex.id = 0.75,
add.smooth = getOption("add.smooth"), label.pos = c(4, 2),
cex.caption = 1)
## S3 method for class 'orlm':
print(x, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'orlm':
summary(object, display.unrestr = FALSE, brief = FALSE,
digits = max(3, getOption("digits") - 3),
scientific = FALSE, overall.tests = TRUE,
bootCIs = TRUE, bty = "perc", level = 0.95, ...)

```

## Arguments

model	a linear model object (class <code>lm</code> ) with data included OR a covariance matrix of $Y$ and all regressors (in this order)
ui	matrix (or vector in case of one single restriction only) defining the left-hand side of the restriction $ui \%*\% beta \geq ci$ , where $beta$ is the parameter vector; the first few of these restrictions can be declared equality- instead of inequality restrictions (cf. argument <code>meq</code> ); if only part of the elements of $beta$ are subject to restrictions, the columns of $ui$ can be restricted to these elements, if their index numbers are provided in <code>index</code> ; by default, <code>index</code> excludes the intercept, i.e. the columns of $ui$ refer to the non-intercept elements of <code>coef(model)</code> Rows of $ui$ must be linearly independent; in case of linearly dependent rows the function gives an error message with a hint which subset of rows is independent. Note that the restrictions must define a (possibly translated) cone, i.e. e.g. interval restrictions on a parameter are not permitted.
ci	vector on the right-hand side of the restriction (cf. <code>ui</code> )
index	index numbers of the components of $beta$ , which are subject to the specified constraints as $ui \%*\% beta[index] \geq ci$ , default is <code>index = 2:length(coef(model))</code> , i.e. $ui$ is supposed to have columns for all coefficients except the intercept; CAUTIONS: - <code>index</code> refers to the position of the coefficient in the model. The first coefficient is usually the intercept (which is therefore per default excluded from restrictions). - If the intercept is included into restrictions (model with intercept, <code>index</code> containing the element 1, intercept-related column of $ui$ not consisting of zeroes only), R-squared values may become unreasonable, if the restriction on the intercept is active.
meq	integer number (default 0) giving the number of rows of $ui$ that are used for equality restrictions instead of inequality restrictions.
orig.out	should the original model be included in the output list ? (default: FALSE)
boot	should bootstrapping be conducted ? (default: FALSE)
B	number of bootstrap samples (default: 1000)

<code>fixed</code>	should bootstrapping consider the sample as fixed and bootstrap residuals ? (default: FALSE)
<code>data</code>	data handed to bootstrap sampling routine
<code>indices</code>	indices for sampling
<code>tol</code>	numerical tolerance value; estimates closer to 0 than <code>tol</code> are set to exactly 0
<code>df.error</code>	error degrees of freedom (number of observations minus number of columns of covariance matrix) for <code>orlm.default</code> ; required in order to calculate adequate covariance matrix and tests; valid coefficient estimates can also be obtained for arbitrary values of <code>df.error</code>
<code>...</code>	Further options
<code>object</code>	object of class <code>orlm</code> (created by function <code>orlm</code> )
<code>x</code>	object of class <code>orlm</code> (created by function <code>orlm</code> )
<code>caption</code>	like in function <code>plot.lm</code>
<code>panel</code>	like in function <code>plot.lm</code>
<code>sub.caption</code>	like in function <code>plot.lm</code>
<code>main</code>	like in function <code>plot.lm</code>
<code>id.n</code>	like in function <code>plot.lm</code>
<code>labels.id</code>	like in function <code>plot.lm</code>
<code>cex.id</code>	like in function <code>plot.lm</code>
<code>add.smooth</code>	like in function <code>plot.lm</code>
<code>label.pos</code>	like in function <code>plot.lm</code>
<code>cex.caption</code>	like in function <code>plot.lm</code>
<code>digits</code>	number of digits to display
<code>display.unrestr</code>	if TRUE, also display unrestricted model; default: FALSE
<code>brief</code>	if TRUE, suppress printing of restrictions; default: FALSE
<code>scientific</code>	if FALSE, suppresses scientific format; default: FALSE
<code>overall.tests</code>	if FALSE, suppresses output of overall model tests; default: TRUE; for models with large sets of restrictions, tests can take up substantial time because of weight calculation
<code>bootCIs</code>	if FALSE, suppresses bootstrap confidence intervals, even though the <code>obj</code> contains a <code>bootout</code> element; default: TRUE
<code>bty</code>	type of bootstrap confidence interval; any of <code>"perc"</code> , <code>"bca"</code> , <code>"norm"</code> or <code>"basic"</code> , cf. function <code>boot.ci</code> from package <code>boot</code> , default: <code>"perc"</code>
<code>level</code>	confidence level for bootstrap confidence intervals, default: 0.95

## Details

Function `orlm` performs order restricted linear model analysis. Functions `coef.orlm`, `plot.orlm`, `print.orlm`, and `summary.orlm` provide methods for reporting the results on an object of S3 class `orlm`. The functions directly referring to bootstrapping are internal and should not be called by the user but are called from within function `orlm` if option `boot` is set to `TRUE`.

Of course, bootstrapping is not possible, if function `orlm` is applied to a covariance matrix, since the raw data are not available in this case. Also note that the intercept is not estimated in this case but can easily be estimated from the resulting estimate if the variable means are known (cf. `example`).

The output from `summary.orlm` provides information about the restrictions, a comparison of  $R^2$ -values for unrestricted and restricted model, restricted estimates, and

- if requested (option `boot` set to `TRUE` in function `orlm` and option `bootCIs` set to `TRUE` in the summary function) with bootstrap confidence intervals,

- if requested (option `overall.tests` set to `TRUE`) several restriction-related tests (implemented by calls to `ic.test`): The analogue to the overall F-Test in the ordinary linear model is the test of all coefficients but intercept equal to 0 within the restricted parameter space. In addition, three tests related to the restriction are reported:

Test 1: H0: Restriction valid with equality vs. H1: at least one inequality

Test 2: H0: Restriction valid vs. H1: restriction violated

Test 3: H0: Restriction violated or valid with equality vs. H1: all restrictions valid with inequality

Test 3 is conducted in case of no equality-restrictions only.

## Value

The output of function `orlm` belongs to S3 classes `orlm` and `orest`. It is a list with the following items:

<code>b.restr</code>	restricted estimate
<code>b.unrestr</code>	unrestricted estimate
<code>R2</code>	R-squared
<code>residuals</code>	residuals of restricted model
<code>fitted.values</code>	fitted values of restricted model
<code>weights</code>	observation weights
<code>orig.R2</code>	R-squared of unrestricted model
<code>df.error</code>	error degrees of freedom of unrestricted model
<code>s2</code>	MSE of unrestricted model
<code>Sigma</code>	variance covariance matrix of beta-hat in unrestricted model
<code>origmodel</code>	unrestricted model itself (NULL, if <code>orig.out=FALSE</code> )
<code>ui</code>	as input
<code>ci</code>	as input
<code>restr.index</code>	the input vector index
<code>meq</code>	as input

`iact` active restrictions, i.e. restrictions that are satisfied with equality in the solution, as output by `solve.QP`

`bootout` object of class `boot` obtained by bootstrapping, will be used by `summary.orlm` for calculating bootstrap confidence intervals; NULL if `boot=FALSE`

**Author(s)**

Ulrike Groemping, TFH Berlin

**References**

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

**See Also**

See also `ic.est`, `ic.test`, `or.relimp`, `solve.QP`

**Examples**

```
limo <- lm(swiss)
## restricted linear model with restrictions that
## - Education and Examination have same coefficient
## - Catholic and Infant.Mortality have non-negative coefficients
orlimo <- orlm(limo, ui=rbind(c(0,1,-1,0,0),c(0,0,0,1,0),c(0,0,0,0,1)), meq=1)
orlimo
plot(orlimo)
summary(orlimo)
## same model using index vector
orlimo <- orlm(limo, ui=rbind(c(1,-1,0,0),c(0,0,1,0),c(0,0,0,1)), index=3:6, meq=1)

## reduced number of bootstrap samples below reasonable size for example run time
orlimo <- orlm(limo, ui=rbind(c(1,-1,0,0),c(0,0,1,0),c(0,0,0,1)),
              index=3:6, meq=1, boot=TRUE, B=100)
summary(orlimo)

## bootstrap considering data as fixed
orlimof <- orlm(limo, ui=rbind(c(1,-1,0,0),c(0,0,1,0),c(0,0,0,1)),
              index=3:6, meq=1, boot=TRUE, B=100, fixed=TRUE)
summary(orlimof, brief=TRUE)
```

# Index

## \*Topic **h**test

ic.infer, 4  
ic.test, 6  
orlm, 15

## \*Topic **m**odels

ic.infer, 4  
or.relimp, 13  
orlm, 15

## \*Topic **m**ultivariate

ic.est, 2  
ic.infer, 4  
ic.test, 6  
ic.weights, 10  
internal.functions, 12  
or.relimp, 13  
orlm, 15

## \*Topic **o**ptimize

ic.est, 2

## \*Topic **r**egression

ic.infer, 4  
or.relimp, 13  
orlm, 15

all.R2(*or.relimp*), 13

boot.orlm(*orlm*), 15

coef.orlm(*orlm*), 15

GaussianElimination

(*internal.functions*), 12

ic.est, 2, 5, 9, 19

ic.infer, 4

ic.infer-package(*ic.infer*), 4

ic.test, 3, 5, 6, 11, 12, 19

ic.weights, 3, 7, 9, 10

internal.functions, 12

nchoosek(*internal.functions*), 12

or.relimp, 5, 13, 19

orlm, 3, 5, 11, 12, 14, 15

orlm.forboot.fixed(*orlm*), 15

orlm.lm(*orlm*), 15

pbetabar(*ic.weights*), 10

pchibar(*ic.weights*), 10

plot.lm, 17

plot.orlm(*orlm*), 15

print.ict(*ic.test*), 6

print.orest(*ic.est*), 2

print.orlm(*orlm*), 15

RREF(*internal.functions*), 12

summary.ict(*ic.test*), 6

summary.orest(*ic.est*), 2

summary.orlm(*orlm*), 15