Package ‘gmodels’

March 6, 2024

Version 2.19.1

Date 2024-03-05

Title Various R Programming Tools for Model Fitting

URL `https://github.com/r-gregmisc/gmodels`

BugReports `https://github.com/r-gregmisc/gmodels/issues`

Maintainer Gregory R. Warnes `<greg@warnes.net>`

Description Various R programming tools for model fitting.

Suggests gplots, gtools, Matrix, nlme, lme4

Imports MASS, gdata, stats

License GPL-2

NeedsCompilation no

RoxygenNote 7.3.1

Encoding UTF-8

Author Gregory R. Warnes [aut, cre],
Ben Bolker [aut],
Thomas Lumley [aut],
Randall C. Johnson [aut] (Contributions from Randall C. Johnson are Copyright (2005) SAIC-Frederick, Inc. Funded by the Intramural Research Program, of the NIH, National Cancer Institute, Center for Cancer Research under NCI Contract NO1-CO-12400),
Airel Muldoon [ctb],
Nitin Jain [aut],
Dirk Enzmann [ctb],
Søren Højsgaards [ctb],
Ulrich Halekoh [ctb],
Mark Schwartz [aut],
Jim Rogers [aut]

Repository CRAN

Date/Publication 2024-03-06 16:00:02 UTC

R topics documented:

`.to.est` *Return a vector for cm in estimable()*

Description

Return a vector for cm in estimable()

Usage

```
.to.est(obj, params)
```

Arguments

`obj`	estimable object
`params`	character vector of names or logical vector with one element per model parameter selecting desrired parameter(s).

Author(s)

Randy Johnson, Laboratory of Genomic Diversity at NCI-Frederick

`ci` *Compute Confidence Intervals*

Description

Compute and display confidence intervals for model estimates. Methods are provided for the mean of a numeric vector `ci.default`, the probability of a binomial vector `ci.binom`, and for `lm`, `lme`, and `mer` objects are provided.

Usage

```
ci(x, confidence = 0.95, alpha = 1 - confidence, ...)

## S3 method for class 'numeric'
ci(x, confidence = 0.95, alpha = 1 - confidence, na.rm = FALSE, ...)
```

Arguments

`x`	object from which to compute confidence intervals.
`confidence`	confidence level. Defaults to 0.95.
`alpha`	type one error rate. Defaults to 1.0-`confidence`
`...`	Arguments for methods
`na.rm`	`logical` indicating whether missing values should be removed.

Value

vector or matrix with one row per model parameter and elements/columns `Estimate`, `CI lower`, `CI upper`, `Std. Error`, `DF` (for lme objects only), and `p-value`.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

`stats::confint()`, `stats::lm()`, `stats::summary.lm()`

Examples

```
# mean and confidence interval
ci( rnorm(10) )

# binomial proportion and exact confidence interval
b <- rbinom( prob=0.75, size=1, n=20 )
ci.binom(b) # direct call
```

```
class(b) <- 'binom'
ci(b)      # indirect call

# confidence intervals for regression parameteres
data(state)
reg  <-  lm(Area ~ Population, data=as.data.frame(state.x77))
ci(reg)
```

---

coefFrame — *Return model parameters in a data frame*

Description

Fits a model to each subgroup defined by by, then returns a data frame with one row for each fit and one column for each parameter.

Usage

```
coefFrame(
  mod,
  data,
  by = NULL,
  fit.on = TRUE,
  fitfun,
  keep.unused.levels = TRUE,
  byvar.sep = "\001",
  ...
)
```

Arguments

mod	a model formula, to be passed to by `fitfun`.
data	a data frame, row subsets of which will be used as the `data` argument to `fitfun`.
by	names of columns in `x` that will be used to define the subgroups.
fit.on	a logical vector indicating which rows of `x` are to be used to fit the model (like the `subset` argument in a lot of other functions). Can be given in terms of variables in `x`
fitfun	a model fitting function (e.g. lm, nls). More specifically, a function that expects at least a formula object (as the first argument) and a data.frame object (passed as an argument named `data`) and returns a model object for which a `coef` method has been defined (e.g. coef.lm, coef.nls) to extract fit values of model parameters.
keep.unused.levels	Include rows in output for all unique values of `by`, even those which were excluded by `fit.on`. The default value `TRUE` should be left alone if you are going to go on to pass the result to `backFit`.

`byvar.sep`	passed to frameApply, used to form the subsets of the data.
`...`	other arguments to pass to `fitfun`.

Value

a data frame with a row for each unique row of `x[by]`, and column for each model paramter, as well as columns specified in `by`.

Author(s)

Jim Rogers <`james.a.rogers@pfizer.com`>

Examples

```
# load example data
library(gtools)
data(ELISA)

# Coefficients for four parameter logistic fits:
coefFrame(log(Signal) ~ SSfpl(log(Concentration), A, B, xmid, scal),
           data = ELISA, fitfun = nls,
           by = c("PlateDay", "Read"),
           fit.on = Description == "Standard" & Concentration != 0)

# Coefficients for linear fits:
coefFrame(log(Signal) ~ log(Concentration),
           data = ELISA, fitfun = lm,
           by = c("PlateDay", "Read"),
           fit.on = Description == "Standard" & Concentration != 0 )

# Example passing arguments to fitfun, and example of
# error handling during model fitting:
ELISA$Signal[1] <- NA
coefFrame(log(Signal) ~ log(Concentration),
           data = ELISA, fitfun = lm, na.action = na.fail,
           by = c("PlateDay", "Read"),
           fit.on = Description == "Standard" & Concentration != 0 )
```

`CrossTable` *Cross Tabulation with Tests for Factor Independence*

Description

An implementation of a cross-tabulation function with output similar to S-Plus crosstabs() and SAS Proc Freq (or SPSS format) with Chi-square, Fisher and McNemar tests of the independence of all table factors.

Usage

```
CrossTable(
  x,
  y,
  digits = 3,
  max.width = 5,
  expected = FALSE,
  prop.r = TRUE,
  prop.c = TRUE,
  prop.t = TRUE,
  prop.chisq = TRUE,
  chisq = FALSE,
  fisher = FALSE,
  mcnemar = FALSE,
  resid = FALSE,
  sresid = FALSE,
  asresid = FALSE,
  missing.include = FALSE,
  format = c("SAS", "SPSS"),
  dnn = NULL,
  ...
)
```

Arguments

`x`	A vector or a matrix. If y is specified, x must be a vector
`y`	A vector in a matrix or a dataframe
`digits`	Number of digits after the decimal point for cell proportions
`max.width`	In the case of a 1 x n table, the default will be to print the output horizontally. If the number of columns exceeds max.width, the table will be wrapped for each successive increment of max.width columns. If you want a single column vertical table, set max.width to 1
`expected`	If `TRUE`, chisq will be set to `TRUE` and expected cell counts from the χ^2 will be included
`prop.r`	If `TRUE`, row proportions will be included
`prop.c`	If `TRUE`, column proportions will be included
`prop.t`	If `TRUE`, table proportions will be included
`prop.chisq`	If `TRUE`, chi-square contribution of each cell will be included
`chisq`	If `TRUE`, the results of a chi-square test will be included
`fisher`	If `TRUE`, the results of a Fisher Exact test will be included
`mcnemar`	If `TRUE`, the results of a McNemar test will be included
`resid`	If `TRUE`, residual (Pearson) will be included
`sresid`	If `TRUE`, standardized residual will be included
`asresid`	If `TRUE`, adjusted standardized residual will be included

`missing.include`	If `TRUE`, then remove any unused factor levels
`format`	Either SAS (default) or SPSS, depending on the type of output desired.
`dnn`	the names to be given to the dimensions in the result (the dimnames names).
`...`	optional arguments

Details

A summary table will be generated with cell row, column and table proportions and marginal totals and proportions. Expected cell counts can be printed if desired (if 'chisq = TRUE'). In the case of a 2 x 2 table, both corrected and uncorrected values will be included for appropriate tests. In the case of tabulating a single vector, cell counts and table proportions will be printed.

Note: If 'x' is a vector and 'y' is not specified, no statistical tests will be performed, even if any are set to `TRUE`.

Value

A list with multiple components including key table data and statistical test results, where performed.

t: An n by m matrix containing table cell counts

prop.col: An n by m matrix containing cell column proportions

prop.row: An n by m matrix containing cell row proportions

prop.tbl: An n by m matrix containing cell table proportions

chisq: Results from the Chi-Square test. A list with class 'htest'. See ?chisq.test for details

chisq.corr: Results from the corrected Chi-Square test. A list with class 'htest'. See ?chisq.test for details. ONLY included in the case of a 2 x 2 table.

fisher.ts: Results from the two-sided Fisher Exact test. A list with class 'htest'. See ?fisher.test for details. ONLY included if 'fisher' = TRUE.

fisher.lt: Results from the Fisher Exact test with HA = "less". A list with class 'htest'. See ?fisher.test for details. ONLY included if 'fisher' = TRUE and in the case of a 2 x 2 table.

fisher.gt: Results from the Fisher Exact test with HA = "greater". A list with class 'htest'. See ?fisher.test for details. ONLY included if 'fisher' = TRUE and in the case of a 2 x 2 table.

mcnemar: Results from the McNemar test. A list with class 'htest'. See ?mcnemar.test for details. ONLY included if 'mcnemar' = TRUE.

mcnemar.corr: Results from the corrected McNemar test. A list with class 'htest'. See ?mcnemar.test for details. ONLY included if 'mcnemar' = TRUE and in the case of a 2 x 2 table.

resid/sresid/asresid: Pearson Residuals (from chi-square tests).

Author(s)

Marc Schwartz <`marc_schwartz@comcast.net`>. Original version posted to r-devel on Jul 27, 2002. SPSS format modifications added by Nitin Jain based upon code provided by Dirk Enzmann <`dirk.enzmann@jura.uni-hamburg.de`>

See Also

`stats::xtabs()`, `base::table()`, `base::prop.table()`

Examples

```
# Simple cross tabulation of education versus prior induced abortions
# using infertility data
data(infert, package = "datasets")
CrossTable(infert$education, infert$induced, expected = TRUE)
CrossTable(infert$education, infert$induced, expected = TRUE, format="SAS")
CrossTable(infert$education, infert$induced, expected = TRUE, format="SPSS")
CrossTable(warpbreaks$wool, warpbreaks$tension, dnn = c("Wool", "Tension"))
```

`estimable` *Contrasts and estimable linear functions of model coefficients*

Description

Compute and test contrasts and other estimable linear functions of model coefficients for for lm, glm, lme, mer, and geese objects

Usage

```
estimable(obj, cm, beta0, conf.int = NULL, show.beta0, ...)

## Default S3 method:
estimable(obj, cm, beta0, conf.int = NULL, show.beta0, joint.test = FALSE, ...)
```

Arguments

`obj`	Regression (lm, glm, lme, mer, mlm) object.
`cm`	Vector, List, or Matrix specifying estimable linear functions or contrasts. See below for details.
`beta0`	Vector of null hypothesis values
`conf.int`	Confidence level. If provided, confidence intervals will be computed.
`show.beta0`	Logical value. If TRUE a column for beta0 will be included in the output table. Defaults to TRUE when beta0 is specified, FALSE otherwise.
`...`	ignored
`joint.test`	Logical value. If TRUE a 'joint' Wald test for the hypothesis $L\beta = \beta_0$ is performed. Otherwise 'row-wise' tests are performed, i.e. $(L\beta)[i] = \beta_0[i]$.

Details

`estimable` computes an estimate, test statitic, significance test, and (optional) confidence interval for each linear functions of the model coefficients specified by `cm`.

The estimable function(s) may be specified via a vector, list, or matrix. If `cm` is a vector, it should contained named elements each of which gives the coefficient to be applied to the corresponding parameter. These coefficients will be used to construct the contrast matrix, with unspecified model parameters assigned zero coefficients. If `cm` is a list, it should contain one or more coefficient vectors, which will be used to construct rows of the contrast matrix. If `cm` is a matrix, column names must match (a subset of) the model parameters, and each row should contain the corresponding coefficient to be applied. Model parameters which are not present in the set of column names of `cm` will be set to zero.

The estimates and their variances are obtained by applying the contrast matrix (generated from) `cm` to the model estimates variance-covariance matrix. Degrees of freedom are obtained from the appropriate model terms.

The user is responsible for ensuring that the specified linear functions are meaningful.

For computing contrasts among levels of a single factor, `fit.contrast` may be more convenient. For computing contrasts between two specific combinations of model parameters, the `contrast` function in Frank Harrell's 'rms' library (formerly 'Design') may be more convenient.

%The `.wald` function is called internally by `estimable` and %is not intended for direct use.

Value

Returns a matrix with one row per linear function. Columns contain the beta0 value (optional, see `show.beta0` above), estimated coefficients, standard errors, t values, degrees of freedom, two-sided p-values, and the lower and upper endpoints of the 1-alpha confidence intervals.

Note

The estimated fixed effect parameters of `lme` objects may have different degrees of freedom. If a specified contrast includes nonzero coefficients for parameters with differing degrees of freedom, the smallest number of degrees of freedom is used and a warning message is issued.

Author(s)

BXC (Bendix Carstensen) <`b@bxc.dk`>, Gregory R. Warnes <`greg@warnes.net`>, Soren Hojsgaard <`sorenh@agrsci.dk`>, and Randall C Johnson <`rjohnson@ncifcrf.gov`>

See Also

`fit.contrast()`, `stats::lm()`, `nlme::lme()`, `stats::contrasts()`, `rms::contrast()`

Examples

```
# setup example data
y <- rnorm(100)
x <-  cut(rnorm(100, mean=y, sd=0.25),c(-4,-1.5,0,1.5,4))
levels(x) <- c("A","B","C","D")
```

```
x2 <- rnorm(100, mean=y, sd=0.5)

# simple contrast and confidence interval
reg <- lm(y ~ x)
estimable(reg, c(    0,   1,    0,   -1) )  # full coefficient vector
estimable(reg, c("xB"=1,"xD"=-1) )          # just the nonzero terms


# Fit a spline with a single knot at 0.5 and plot the *pointwise*
# confidence intervals
library(gplots)
pm <- pmax(x2-0.5, 0) # knot at 0.5
reg2 <- lm(y ~ x + x2 + pm )

range <- seq(-2, 2, , 50)
tmp <- estimable(reg2,
                 cm=cbind(
                          '(Intercept)'=1,
                          'xC'=1,
                          'x2'=range,
                          'pm'=pmax(range-0.5, 0)
                           ),
                 conf.int=0.95)
plotCI(x=range, y=tmp[, 1], li=tmp[, 6], ui=tmp[, 7])

# Fit both linear and quasi-Poisson models to iris data, then compute
# joint confidence intervals on contrasts for the Species and
# Sepal.Width by Species interaction terms.
data(iris)
lm1  <- lm (Sepal.Length ~ Sepal.Width + Species + Sepal.Width:Species, data=iris)
glm1 <- glm(Sepal.Length ~ Sepal.Width + Species + Sepal.Width:Species, data=iris,
            family=quasipoisson("identity"))

cm <- rbind(
            'Setosa vs. Versicolor'   = c(0, 0, 1, 0, 1, 0),
            'Setosa vs. Virginica'    = c(0, 0, 0, 1, 0, 1),
            'Versicolor vs. Virginica'= c(0, 0, 1,-1, 1,-1)
            )
estimable(lm1, cm)
estimable(glm1, cm)
```

`est_p_ci` *Display estimate, confidence interval and p-value for one model term*

Description

Display estimate, confidence interval and p-value for one model term

Usage

```
est_p_ci(model, term, mult = 1, digits = 2, ...)
```

Arguments

`model`	model object
`term`	model term
`mult`	scale (multiply) the parameter by this factor
`digits`	number of significant digits to display
`...`	optional arguments

Examples

```
set.seed(42)

# fit an example model with 3 groups
y <- rnorm(100)
x <-  cut(rnorm(100, mean=y, sd=0.25),c(-4,-1.5,0,1.5,4))
reg <- lm(y ~ x)
reg

# show model estimate, p-value, and confidence interval
# for the first group
est_p_ci(reg, 2)

# estimate some group contrasts
cmat <- rbind( "1 vs 4"    =c(-1, 0, 0, 1),
               "1+2 vs 3+4"=c(-1/2,-1/2, 1/2, 1/2),
               "1 vs 2+3+4"=c(-3/3, 1/3, 1/3, 1/3))
cont <- fit.contrast(reg, x, cmat, conf.int = 0.95)
cont

# show the contrast estimate, p-value, and confidence interval
# for the first contrast
est_p_ci(cont, 2:3)
```

`fast.prcomp` *Efficient computation of principal components and singular value decompositions.*

Description

The standard `stats::prcomp()` and `svd()` function are very inefficient for wide matrixes. `fast.prcomp` and `fast.svd` are modified versions which are efficient even for matrixes that are very wide.

Usage

```
fast.prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL)
```

Arguments

`x`	data matrix
`retx`	a logical value indicating whether the rotated variables should be returned.
`center`	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of `x` can be supplied. The value is passed to `scale`.
`scale.`	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is `FALSE` for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of `x` can be supplied. The value is passed to `scale`.
`tol`	a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to `tol` times the standard deviation of the first component.) With the default null setting, no components are omitted (unless `rank.` is specified less than `min(dim(x))`.). Other settings for tol could be `tol = 0` or `tol = sqrt(.Machine$double.eps)`, which would omit essentially constant components.

Details

The current implementation of the function `svd()` in S-Plus and R is much slower when operating on a matrix with a large number of columns than on the transpose of this matrix, which has a large number of rows. As a consequence, `stats::prcomp()`, which uses `svd()`, is also very slow when applied to matrixes with a large number of rows.

The simple solution is to use `La.svd()` instead of `svd()`. A suitable patch to `stats::prcomp()` has been submitted. In the mean time, the function `fast.prcomp` has been provided as a short-term work-around.

list("fast.prcomp") is a modified versiom of `stats::prcomp()` that calls `La.svd()` instead of `svd()`

list("fast.svd") is simply a wrapper around `La.svd()`.

Value

See the documetation for `stats::prcomp()` or `svd()` .

Author(s)

Modifications by Gregory R. Warnes <`greg@warnes.net`>

See Also

`stats::prcomp()`, `base::svd()`, `base::La.svd()`

Examples

```
# create test matrix
set.seed(4943546)
nr <- 50
nc <- 2000
x  <- matrix( rnorm( nr*nc), nrow=nr, ncol=nc )
tx <- t(x)

# SVD directly on matrix is SLOW:
system.time( val.x <- svd(x)$u )

# SVD on t(matrix) is FAST:
system.time( val.tx <- svd(tx)$v )

# and the results are equivalent:
max( abs(val.x) - abs(val.tx) )

# Time gap dissapears using fast.svd:
system.time( val.x <- fast.svd(x)$u )
system.time( val.tx <- fast.svd(tx)$v )
max( abs(val.x) - abs(val.tx) )


library(stats)

# prcomp directly on matrix is SLOW:
system.time( pr.x <- prcomp(x) )

# prcomp.fast is much faster
system.time( fast.pr.x <- fast.prcomp(x) )

# and the results are equivalent
max( pr.x$sdev - fast.pr.x$sdev )
max( abs(pr.x$rotation[,1:49]) - abs(fast.pr.x$rotation[,1:49]) )
max( abs(pr.x$x) - abs(fast.pr.x$x)  )

# (except for the last and least significant component):
max( abs(pr.x$rotation[,50]) - abs(fast.pr.x$rotation[,50]) )
```

`fit.contrast` *Compute and test arbitrary contrasts for regression objects*

Description

Compute and test arbitrary contrasts for regression objects.

Usage

```
fit.contrast(model, varname, coeff, showall, conf.int, df, ...)
```

Arguments

`model`	regression (lm,glm,aov,lme) object for which the contrast(s) will be computed.
`varname`	variable name
`coeff`	vector or matrix specifying contrasts (one per row).
`showall`	return all regression coefficients. If `TRUE`, all model cofficients will be returned. If `FALSE` (the default), only the coefficients corresponding to the specified contrast will be returned.
`conf.int`	numeric value on (0,1) or NULL. If a numeric value is specified, confidence intervals with nominal coverage probability `conf.int` will be computed. If `NULL`, confidence intervals will not be computed.
`df`	boolean indicating whether to return a column containing the degrees of freedom.
`...`	optional arguments provided by methods.

Details

Computes the specified contrast(s) by re-fitting the model with the appropriate arguments. A contrast of the form `c(1,0,0,-1)` would compare the mean of the first group with the mean of the fourth group.

Value

Returns a matrix containing estimated coefficients, standard errors, t values, two-sided p-values. If `df` is TRUE, an additional column containing the degrees of freedom is included. If `conf.int` is specified lower and upper confidence limits are also returned.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

References

Venables & Ripley, Section 6.2

See Also

- `stats::lm()`, `stats::contrasts()`, `stats::contr.treatment()`, `stats::contr.poly()`,
- Computation and testing of General Linear Hypothesis: `glh.test()`,
- Computation and testing of estimable functions of model coefficients: `estimable()`, `make.contrasts()`

Examples

```
set.seed(42)

y <- rnorm(100)
x <-  cut(rnorm(100, mean=y, sd=0.25),c(-4,-1.5,0,1.5,4))
reg <- lm(y ~ x)
summary(reg)

# look at the group means
gm <- sapply(split(y,x),mean)
gm


# mean of 1st group vs mean of 4th group
fit.contrast(reg, x, c(    1,    0,    0,   -1) )
# estimate should be equal to:
gm[1] - gm[4]

# mean of 1st and 2nd groups vs mean of 3rd and 4th groups
fit.contrast(reg, x, c( -1/2, -1/2,  1/2,  1/2) )
# estimate should be equal to:
sum(-1/2*gm[1], -1/2*gm[2], 1/2*gm[3], 1/2*gm[4])

# mean of 1st group vs mean of 2nd, 3rd and 4th groups
fit.contrast(reg, x, c( -3/3,  1/3,  1/3,  1/3) )
# estimate should be equal to:
sum(-3/3*gm[1], 1/3*gm[2], 1/3*gm[3], 1/3*gm[4])

# all at once
cmat <- rbind( "1 vs 4"    =c(-1, 0, 0, 1),
               "1+2 vs 3+4"=c(-1/2,-1/2, 1/2, 1/2),
               "1 vs 2+3+4"=c(-3/3, 1/3, 1/3, 1/3))
fit.contrast(reg,x,cmat)

#
x2 <- rnorm(100,mean=y,sd=0.5)
reg2 <- lm(y ~ x + x2 )
fit.contrast(reg2,x,c(-1,0,0,1))

#
# Example for Analysis of Variance
#

set.seed(03215)
Genotype <- sample(c("WT","KO"), 1000, replace=TRUE)
Time <- factor(sample(1:3, 1000, replace=TRUE))
y <- rnorm(1000)
data <- data.frame(y, Genotype, Time)


# Compute Contrasts & obtain 95% confidence intervals
```

```
model <- aov( y ~ Genotype + Time + Genotype:Time, data=data )

fit.contrast( model, "Genotype", rbind("KO vs WT"=c(-1,1) ), conf=0.95 )

fit.contrast( model, "Time",
            rbind("1 vs 2"=c(-1,1,0),
                  "2 vs 3"=c(0,-1,1)
                  ),
            conf=0.95 )


cm.G <- rbind("KO vs WT"=c(-1,1) )
cm.T <- rbind("1 vs 2"=c(-1,1,0),
              "2 vs 3"=c(0,-1,1) )

# Compute contrasts and show SSQ decompositions

model <- aov( y ~ Genotype + Time + Genotype:Time, data=data,
              contrasts=list(Genotype=make.contrasts(cm.G),
                             Time=make.contrasts(cm.T) )
            )

summary(model, split=list( Genotype=list( "KO vs WT"=1 ),
                           Time = list( "1 vs 2" = 1,
                                        "2 vs 3" = 2 ) ) )


# example for lme
library(nlme)
data(Orthodont)
fm1 <- lme(distance ~ Sex, data = Orthodont,random=~1|Subject)

# Contrast for sex.  This example is equivalent to standard treatment
# contrast.
#
fit.contrast(fm1, "Sex", c(-1,1), conf.int=0.95 )
#
# and identical results can be obtained using lme built-in 'intervals'
#
intervals(fm1)

# Cut age into quantile groups & compute some contrasts
Orthodont$AgeGroup <- gtools::quantcut(Orthodont$age)
fm2 <- lme(distance ~ Sex + AgeGroup, data = Orthodont,random=~1|Subject)
#
fit.contrast(fm2, "AgeGroup", rbind("Linear"=c(-2,-1,1,2),
                                    "U-Shaped"=c(-1,1,1,-1),
                                    "Change-Point at 11"=c(-1,-1,1,1)),
                              conf.int=0.95)
```

glh.test *Test a General Linear Hypothesis for a Regression Model*

Description

Test, print, or summarize a general linear hypothesis for a regression model

Usage

```
glh.test(reg, cm, d = rep(0, nrow(cm)))
```

Arguments

`reg`	Regression model
`cm`	contrast matrix `C` . Each row specifies a linear combination of the coefficients
`d`	vector `d` specifying the null hypothesis values for each linear combination

Details

Test the general linear hypothesis $C\hat{\beta} = d$ for the regression model `reg`.

The test statistic is obtained from the formula:

$$f = \frac{(C\hat{\beta} - d)'(C(X'X)^{-1}C')(C\hat{\beta} - d)/r}{SSE/(n-p)}$$

where

- `r` is the number of contrasts contained in `C`, and
- `n-p` is the model degrees of freedom.

Under the null hypothesis, `f` will follow a F-distribution with `r` and `n-p` degrees of freedom

Value

Object of class `c("glh.test","htest")` with elements:

`call`	Function call that created the object
`statistic`	F statistic
`parameter`	vector containing the numerator (r) and denominator (n-p) degrees of freedom
`p.value`	p-value
`estimate`	computed estimate for each row of `cm`
`null.value`	d
`method`	description of the method
`data.name`	name of the model given for `reg`
`matrix`	matrix specifying the general linear hypotheis (`cm`)

Note

When using treatment contrasts (the default) the first level of the factors are subsumed into the intercept term. The estimated model coefficients are then contrasts versus the first level. This should be taken into account when forming contrast matrixes, particularly when computing contrasts that include 'baseline' level of factors.

See the comparison with `fit.contrast` in the examples below.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

References

R.H. Myers, Classical and Modern Regression with Applications, 2nd Ed, 1990, p. 105

See Also

`fit.contrast()`, `estimable()`, `stats::contrasts()`

Examples

```
# fit a simple model
y <- rnorm(100)
x <-  cut(rnorm(100, mean=y, sd=0.25),c(-4,-1.5,0,1.5,4))
reg <- lm(y ~ x)
summary(reg)

# test both group 1 = group 2  and group 3 = group 4
# *Note the 0 in the column for the intercept term*

C <- rbind( c(0,-1,0,0), c(0,0,-1,1) )
ret <- glh.test(reg, C)
ret  # same as 'print(ret) '
summary(ret)

# To compute a contrast between the first and second level of the factor
# 'x' using 'fit.contrast' gives:

fit.contrast( reg, x,c(1,-1,0,0) )

# To test this same contrast using 'glh.test', use a contrast matrix
# with a zero coefficient for the intercept term.  See the Note section,
# above, for an explanation.

C <- rbind( c(0,-1,0,0) )
glh.test( reg, C )
```

`make.contrasts` *Construct a User-Specified Contrast Matrix*

Description

This function converts human-readable contrasts into the form that R requires for computation.

Usage

```
make.contrasts(contr, how.many = ncol(contr))
```

Arguments

`contr`	vector or matrix specifying contrasts (one per row).
`how.many`	dimensions of the desired contrast matrix. This must equal the number of levels of the target factor variable.

Details

Specifying a contrast row of the form `c(1,0,0,-1)` creates a contrast that will compare the mean of the first group with the mean of the fourth group.

Value

`make.contrasts` returns a matrix with dimensions (`how.many`, `how.many`) containing the specified contrasts augmented (if necessary) with orthogonal "filler" contrasts.

This matrix can then be used as the argument to `contrasts()` or to the `contrasts` argument of model functions (eg, `lm()`).

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

- `stats::lm()`, `stats::contrasts()`, `stats::contr.treatment()`, `stats::contr.poly()`,
- Computation and testing of General Linear Hypothesis: `glh.test()`,
- Computation and testing of estimable functions of model coefficients: `estimable()`,
- Estimate and Test Contrasts for a previously fit linear model: `fit.contrast()`

Examples

```
set.seed(4684)
y <- rnorm(100)
x.true <- rnorm(100, mean=y, sd=0.25)
x <-  factor(cut(x.true,c(-4,-1.5,0,1.5,4)))
reg <- lm(y ~ x)
summary(reg)

# Mirror default treatment contrasts
test <- make.contrasts(rbind( c(-1,1,0,0), c(-1,0,1,0), c(-1,0,0,1) ))
lm( y ~ x, contrasts=list(x = test ))

# Specify some more complicated contrasts
#   - mean of 1st group vs mean of 4th group
#   - mean of 1st and 2nd groups vs mean of 3rd and 4th groups
#   - mean of 1st group vs mean of 2nd, 3rd and 4th groups
cmat <- rbind( "1 vs 4"    =c(-1, 0, 0, 1),
               "1+2 vs 3+4"=c(-1/2,-1/2, 1/2, 1/2),
               "1 vs 2+3+4"=c(-3/3, 1/3, 1/3, 1/3))

summary(lm( y ~ x, contrasts=list(x=make.contrasts(cmat) )))
# or
contrasts(x) <- make.contrasts(cmat)
summary(lm( y ~ x ) )

# or use contrasts.lm
reg <- lm(y ~ x)
fit.contrast( reg, "x", cmat )

# compare with values computed directly using group means
gm <- sapply(split(y,x),mean)
gm %*% t(cmat)


#
# Example for Analysis of Variance
#

set.seed(03215)
Genotype <- sample(c("WT","KO"), 1000, replace=TRUE)
Time <- factor(sample(1:3, 1000, replace=TRUE))
data <- data.frame(y, Genotype, Time)
y <- rnorm(1000)

data <- data.frame(y, Genotype, as.factor(Time))

# Compute Contrasts & obtain 95% confidence intervals

model <- aov( y ~ Genotype + Time + Genotype:Time, data=data )

fit.contrast( model, "Genotype", rbind("KO vs WT"=c(-1,1) ), conf=0.95 )
fit.contrast( model, "Time",
            rbind("1 vs 2"=c(-1,1,0),
                  "2 vs 3"=c(0,-1,1)
                  ),
            conf=0.95 )


cm.G <- rbind("KO vs WT"=c(-1,1) )
cm.T <- rbind("1 vs 2"=c(-1,1,0),
              "2 vs 3"=c(0,-1,1) )

# Compute contrasts and show SSQ decompositions

model <- model <- aov( y ~ Genotype + Time + Genotype:Time, data=data,
                      contrasts=list(Genotype=make.contrasts(cm.G),
                                     Time=make.contrasts(cm.T) )
                      )

summary(model, split=list( Genotype=list( "KO vs WT"=1 ),
                           Time = list( "1 vs 2" = 1,
                                        "2 vs 3" = 2 ) ) )
```

Index