Package ‘csranks’

July 22, 2025

Type Package

Title Statistical Tools for Ranks

Version 1.2.3

Description Account for uncertainty when working with ranks.
Estimate standard errors consistently in linear regression with ranked variables.
Construct confidence sets of various kinds for positions of populations in a ranking
based on values of a certain feature and their estimation errors.
Theory based on Mogstad, Romano, Shaikh, and Wil-
helm (2023)<doi:10.1093/restud/rdad006> and
Chetverikov and Wilhelm (2023) <doi:10.48550/arXiv.2310.15512>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL `https://github.com/danielwilhelm/R-CS-ranks`,
`https://danielwilhelm.github.io/R-CS-ranks/`

BugReports `https://github.com/danielwilhelm/R-CS-ranks/issues`

RoxygenNote 7.3.1

Depends R (>= 3.5.0)

Imports stats, ggplot2, scales, MASS, cli, lifecycle

Suggests spelling, testthat (>= 2.1.0), grid, knitr, rmarkdown

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Daniel Wilhelm [aut, cre],
Pawel Morgen [aut]

Maintainer Daniel Wilhelm <`d.wilhelm@lmu.de`>

Repository CRAN

Date/Publication 2024-09-12 21:50:05 UTC

Contents

`csranks` *Confidence sets for ranks*

Description

Marginal and simultaneous confidence sets for ranks.

Usage

```
csranks(
  x,
  Sigma,
  coverage = 0.95,
  cstype = "two-sided",
  stepdown = TRUE,
  R = 1000,
  simul = TRUE,
  indices = NA,
  na.rm = FALSE,
  seed = NA
)
```

Arguments

`x`	vector of estimates containing estimated features by which the populations are to be ranked.
`Sigma`	estimated covariance matrix of `x`.
`coverage`	nominal coverage of the confidence set. Default is 0.95.
`cstype`	type of confidence set (`two-sided`, `upper`, `lower`). Default is `two-sided`.
`stepdown`	logical; if `TRUE` (default), stepwise procedure is used, otherwise single step procedure is used. See Details section for more.
`R`	number of bootstrap replications. Default is 1000.

`simul`	logical; if `TRUE` (default), then simultaneous confidence sets are computed, which jointly cover all populations indicated by `indices`. Otherwise, for each population indicated in `indices` a marginal confidence set is computed.
`indices`	vector of indices of `x` for whose ranks the confidence sets are computed. `indices=NA` (default) means computation of confidence sets for all populations.
`na.rm`	logical; if `TRUE`, then `NA`'s are removed from `x` and `Sigma` (if any).
`seed`	seed for bootstrap random variable draws. If set to `NA` (default), then seed is not set.

Value

A `csranks` object, which is a list with three items:

`L` Lower bounds of the confidence sets for ranks indicated in `indices`

`rank` Estimated ranks from `irank` with default parameters

`U` Upper bounds of the confidence sets.

Details

Suppose $j = 1, \ldots, p$ populations (e.g., schools, hospitals, political parties, countries) are to be ranked according to some measure $\theta = (\theta_1, \ldots, \theta_p)$. We do not observe the true values $\theta_1, \ldots, \theta_p$. Instead, for each population, we have data from which we have estimated these measures, $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$. The values $\hat{\theta}_1, \ldots, \hat{\theta}_p$ are estimates of the true values $\theta_1, \ldots, \theta_p$ and thus contain statistical uncertainty. In consequence, a ranking of the populations by the values $\hat{\theta}_1, \ldots, \hat{\theta}_p$ contains statistical uncertainty and is not necessarily equal to the true ranking of $\theta_1, \ldots, \theta_p$.

The function computes confidence sets for the rank of one, several or all of the populations (`indices` indicates which of the $1, \ldots, p$ populations are of interest). `x` is a vector containing the estimates $\hat{\theta}_1, \ldots, \hat{\theta}_p$ and `Sigma` is an estimate of the covariance matrix of `x`. The method assumes that the estimates are asymptotically normal and the sample sizes of the datasets are large enough so that $\hat{\theta} - \theta$ is approximately distributed as $N(0, \Sigma)$. The argument `Sigma` should contain an estimate of the covariance matrix Σ. For instance, if for each population j

$$\sqrt{n_j}(\hat{\theta}_j - \theta_j) \rightarrow_d N(0, \sigma_j^2)$$

and the datasets for each population are drawn independently of each other, then `Sigma` is a diagonal matrix

$$diag(\hat{\sigma}_1^2/n_1, \ldots, \hat{\sigma}_p^2/n_p)$$

containing estimates of the asymptotic variances divided by the sample size. More generally, the estimates in `x` may be dependent, but then `Sigma` must be an estimate of its covariance matrix including off-diagonal terms.

Marginal confidence sets (`simul=FALSE`) are such that the confidence set for a population j contains the true rank of that population j with probability approximately equal to the nominal coverage level. Simultaneous confidence sets (`simul=TRUE`) on the other hand are such that the confidence sets for populations indicated in `indices` cover the true ranks of all of these populations simultaneously with probability approximately equal to the nominal coverage level. For instance, in the PISA example below, a marginal confidence set of a country j covers the true rank of country j with probability approximately equal to 0.95. A simultaneous confidence set for all countries covers the true ranks of all countries simultaneously with probability approximately equal to 0.95.

The function implements the procedures developed and described in more detail in Mogstad, Romano, Shaikh, and Wilhelm (2023). The procedure is based on on testing a large family of hypotheses for pairwise comparisons. Stepwise methods can be used to improve the power of the procedure by, potentially, rejecting more hypotheses without violating the desired coverage property of the resulting confidence set. These are employed when `stepdown=TRUE`. From a practical point of view, `stepdown=TRUE` is computationally more demanding, but often results in tighter confidence sets.

The procedure uses a parametric bootstrap procedure based on the above approximate multivariate normal distribution.

References

Mogstad, Romano, Shaikh, and Wilhelm (2023), "Inference for Ranks with Applications to Mobility across Neighborhoods and Academic Achievements across Countries", forthcoming at Review of Economic Studies cemmap working paper doi:10.1093/restud/rdad006

Examples

```
# simple simulated example:
n <- 100
p <- 10
X <- matrix(rep(1:p,n)/p, ncol=p, byrow=TRUE) + matrix(rnorm(n*p), 100, 10)
thetahat <- colMeans(X)
Sigmahat <- cov(X) / n
csranks(thetahat, Sigmahat)

# PISA example:
data(pisa2018)
math_score <- pisa2018$math_score
math_se <- pisa2018$math_se
math_cov_mat <- diag(math_se^2)

# marginal confidence set for each country:
csranks(math_score, math_cov_mat, simul=FALSE)

# simultaneous confidence set for all countries:
csranks(math_score, math_cov_mat, simul=TRUE)
```

`csranks_multinom` *Confidence sets for ranks based on multinomial data*

Description

Marginal and simultaneous confidence sets for ranks of categories, where categories are ranked by the probabilities of being chosen.

Usage

```
csranks_multinom(
  x,
  coverage = 0.95,
  cstype = "two-sided",
  simul = TRUE,
  multcorr = "Holm",
  indices = NA,
  na.rm = FALSE
)
```

Arguments

`x`	vector of counts indicating how often each category was chosen.
`coverage`	nominal coverage of the confidence set. Default is 0.95.
`cstype`	type of confidence set (`two-sided`, `upper`, `lower`). Default is `two-sided`.
`simul`	logical; if `TRUE` (default), then simultaneous confidence sets are computed, which jointly cover all populations indicated by `indices`. Otherwise, for each population indicated in `indices` a marginal confidence set is computed.
`multcorr`	multiplicity correction to be used: `Holm` (default) or `Bonferroni`. See Details section for more.
`indices`	vector of indices of `x` for whose ranks the confidence sets are computed. `indices=NA` (default) means computation of confidence sets for all populations.
`na.rm`	logical; if `TRUE`, then `NA`'s are removed from `x` and `Sigma` (if any).

Value

A `csranks` object, which is a list with three items:

`L` Lower bounds of the confidence sets for ranks indicated in `indices`

`rank` Estimated ranks from `irank` with default parameters

`U` Upper bounds of the confidence sets.

Details

This function computes confidence sets for ranks similarly as `csranks`, but it is tailored to the special case of multinomial data. Suppose there are p populations (for the case of multinomial data, we will refer to them as "categories") such as political parties, for example, that one wants to rank by the probabilities of them being chosen. For political parties, this would correspond to the share of votes each party obtains. Here, the underlying data are multinomial: each observation corresponds to a choice among the p categories. The vector `x` contains the counts of how often each category was chosen in the data.

In this setting, `link{csranks}` could be applied to compute confidence sets for the ranks of each category, but instead this function implements a different method proposed by Bazylik, Mogstad, Romano, Shaikh, and Wilhelm (2023), which exploits the multinomial structure of the problem and yields confidence sets for the ranks that are valid in finite samples (whereas `csranks` produces confidence sets that are valid only asymptotically).

The procedure involves testing multiple hypotheses. The `\code{multcorr}` indicates a method for multiplicity correction. See the paper for details.

References

Bazylik, Mogstad, Romano, Shaikh, and Wilhelm. "Finite-and large-sample inference for ranks using multinomial data with an application to ranking political parties". cemmap working paper

Examples

```
x <- c(rmultinom(1, 1000, 1:10))
csranks_multinom(x)
```

`cstaubest` *Confidence sets for the tau-best*

Description

Computation of confidence sets for the identities of populations among the tau best.

Usage

```
cstaubest(
  x,
  Sigma,
  tau = 2,
  coverage = 0.95,
  stepdown = TRUE,
  R = 1000,
  na.rm = FALSE,
  seed = NA
)

cstauworst(
  x,
  Sigma,
  tau = 2,
  coverage = 0.95,
  stepdown = TRUE,
  R = 1000,
  na.rm = FALSE,
  seed = NA
)
```

Arguments

`x`	vector of estimates containing estimated features by which the populations are to be ranked.
`Sigma`	estimated covariance matrix of `x`.
`tau`	the confidence set contains indicators for the elements in `x` whose rank is less than or equal to `tau`.
`coverage`	nominal coverage of the confidence set. Default is 0.95.
`stepdown`	logical; if `TRUE` (default), stepwise procedure is used, otherwise single step procedure is used. See Details section for more.
`R`	number of bootstrap replications. Default is 1000.
`na.rm`	logical; if `TRUE`, then `NA`'s are removed from `x` and `Sigma` (if any).
`seed`	seed for bootstrap random variable draws. If set to `NA` (default), then seed is not set.

Value

logical vector indicating which of the elements of `x` are in the confidence set for the tau-best.

Functions

- `cstauworst()`: Confidence sets for the tau-worst
  Equivalent to calling `cstaubest` with `-x`.

Details

The function computes a confidence set containing indicators for the elements in `x` whose rank is less than or equal to `tau` with probability approximately equal to the nominal coverage (`coverage`).

The function implements the projection confidence set for the tau-best developed and described in more detail in Mogstad, Romano, Shaikh, and Wilhelm (2023).

References

Mogstad, Romano, Shaikh, and Wilhelm (2023), "Inference for Ranks with Applications to Mobility across Neighborhoods and Academic Achievements across Countries", forthcoming at Review of Economic Studies cemmap working paper, doi:10.1093/restud/rdad006

Examples

```
# simple simulated example:
n <- 100
p <- 10
X <- matrix(rep(1:p,n)/p, ncol=p, byrow=TRUE) + matrix(rnorm(n*p), 100, 10)
thetahat <- colMeans(X)
Sigmahat <- cov(X) / n

# confidence set for the populations that may be among the top-3
# (with probability approximately 0.95):
```

```
cstaubest(thetahat, Sigmahat, tau=3)

# confidence set for the populations that may be among the bottom-3
# (with probability approximately 0.95):
cstauworst(thetahat, Sigmahat, tau=3)
```

irank *Compute ranks*

Description

Compute integer of fractional ranks with flexible handling of ties.

Usage

```
irank(x, omega = 0, increasing = FALSE, na.rm = FALSE)

frank(x, omega = 0, increasing = FALSE, na.rm = FALSE)
```

Arguments

x	vector of values to be ranked
omega	numeric value in [0,1], defining how ties in x (if any) are handled; default is 0. See Details.
increasing	logical; if FALSE (default), then large elements in x receive a small rank. Otherwise, large elements in x receive a large rank.
na.rm	logical; if TRUE, then NA's are removed from x. Default: FALSE.

Details

irank implements all possible definitions of ranks of the values in x. Different definitions of the ranks are chosen through combinations of the two arguments omega and increasing. Suppose x is of length p. If increasing=TRUE, then the largest value in x receives the rank p and the smallest the rank 1. If increasing=FALSE, then the largest value in x receives the rank 1 and the smallest the rank p.

The value of omega indicates how ties are handled. If there are no ties in x, then the value of omega does not affect the ranks and the only choice to be made is whether the ranks should be increasing or decreasing with the values in x. When there are ties in x, however, then there are infinitely many possible ranks that can be assigned to a tied value.

When increasing=TRUE, then omega=0 leads to the smallest possible and omega=1 to the largest possible rank of a tied value. Values of omega between 0 and 1 lead to values of the rank between the largest and smallest.

frank takes the ranking returned by irank and divides the result by length(x). The result is a ranking with ranks in the interval [0,1]. An important special case occurs for increasing=TRUE and omega=1: in this case, the rank of the value x[j] is equal to the empirical cdf of x evaluated at x[j].

Value

Numeric vector of the same length as `x` containing the integer (for `irank`) or fractional (for `frank`) ranks.

Examples

```
# simple example without ties:
x <- c(3,8,-4,10,2)
irank(x, increasing=TRUE)
irank(x, increasing=FALSE)

# since there are no ties, the value of omega has no impact:
irank(x, increasing=TRUE, omega=0)
irank(x, increasing=TRUE, omega=0.5)
irank(x, increasing=TRUE, omega=1)

# simple example with ties:
x <- c(3,4,7,7,10,11,15,15,15,15)
irank(x, increasing=TRUE, omega=0) # smallest possible ranks
irank(x, increasing=TRUE, omega=0.5) # mid-ranks
irank(x, increasing=TRUE, omega=1) # largest possible ranks

# simple example of fractional ranks without ties:
x <- c(3,8,-4,10,2)
frank(x, increasing=TRUE)
frank(x, increasing=FALSE)
```

`irank_against` *Compute integer ranks in another reference vector*

Description

The method `irank` compares ranks using the same vector as reference. `irank_against` returns integer ranks, that values from `x` would assume if (individually) inserted into `v`. `frank_against` acts analogously, returning fractional ranks.

Usage

```
irank_against(x, v, omega = 0, increasing = FALSE, na.rm = FALSE)

frank_against(x, v, omega = 0, increasing = FALSE, na.rm = FALSE)
```

Arguments

`x`	numeric query vector.
`v`	numeric reference vector.
`omega`	numeric value in [0,1], defining how ties in `x` (if any) are handled; default is `0`. See Details.

`increasing`	logical; if `FALSE` (default), then large elements in `x` receive a small rank. Otherwise, large elements in `x` receive a large rank.
`na.rm`	logical; if `TRUE`, then `NA`'s are removed from `x`. Default: `FALSE`.

Details

It's useful to think about `frank_against(x,v)` as a generalization of Empirical Cumulative Distribution Function, created for `v` and evaluated for points in `x`. `frank_agaist(x,v,increasing=TRUE,omega=1)` is identical to `ecdf(v)(x)`.

`increasing` switches the inequality sign in ECDF definition from $F_V(t) = \hat{P}(V <= t)$ to $\hat{P}(V >= t)$.

`omega=0` introduces the strict inequality ($\hat{P}(V < t)$ instead of $\hat{P}(V <= t)$). Any `omega` in between is a weighted average of the cases `omega=1` and `omega=0`.

Finally, `irank_against` is equal to `frank_against` multiplied by the `length(v)`.

This particular choice of default parameters was made for compatibility with default parameters of `irank` and `frank`. `irank(x)` is always equal to `irank_against(x,x)` and `frank(x)` is always equal to `frank_against(x,x)`.

Value

Numeric vector of the same length as `x` containing the integer (for `irank_against`) or fractional (for `frank_against`) ranks.

See Also

`irank()`, `ecdf()`

Examples

```
irank_against(1:10, c(4,4,4,3,1,10,7,7))
```

`lmranks` *Regressions Involving Ranks*

Description

Estimation and inference for regressions involving ranks, i.e. regressions in which the dependent and/or the independent variable has been transformed into ranks before running the regression.

Usage

```
lmranks(
  formula,
  data,
  subset,
  weights,
  na.action = stats::na.fail,
  method = "qr",
  model = TRUE,
  x = FALSE,
  qr = TRUE,
  y = FALSE,
  singular.ok = TRUE,
  contrasts = NULL,
  offset = offset,
  omega = 1,
  ...
)

## S3 method for class 'lmranks'
plot(x, which = 1, ...)

## S3 method for class 'lmranks'
proj(object, onedf = FALSE, ...)

## S3 method for class 'lmranks'
predict(object, newdata, ...)

## S3 method for class 'lmranks'
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'lmranks'
vcov(object, complete = TRUE, ...)
```

Arguments

`formula`	An object of class "`formula`": a symbolic description of the model to be fitted. Exactly like the formula for linear model except that variables to be ranked can be indicated by `r()`. See Details and Examples below.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	currently not supported.
`weights`	currently not supported.
`na.action`	currently not supported. User is expected to handle NA values prior to the use of this function.
`method`	the method to be used; for fitting, currently only `method = "qr"` is supported; `method = "model.frame"` returns the model frame (the same as with `model = TRUE`, see below).
`model, y, qr`	logicals. If TRUE the corresponding components of the fit (the model frame, the response, the QR decomposition) are returned.
`x`	• For `lmranks`: Logical. Should model matrix be returned?

	• For `plot` method: An `lmranks` object.
`singular.ok`	logical. If `FALSE` (the default in S but not in R) a singular fit is an error.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector or matrix of extents matching those of the response. One or more `offset` terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See `model.offset`.
`omega`	real number in the interval [0,1] defining how ties are handled (if there are any). The value of `omega` is passed to `frank` for computation of ranks. The default is 1 so that the rank of a realized value is defined as the empirical cdf evaluated at that realized value. See Details below.
`...`	For `lm()`: additional arguments to be passed to the low level regression fitting functions (see below).
`which`	As in `plot.lm`. Currently only no.1 is available.
`object`	A `lmranks` object.
`onedf`	A logical flag. If `TRUE`, a projection is returned for all the columns of the model matrix. If `FALSE`, the single-column projections are collapsed by terms of the model (as represented in the analysis of variance table).
`newdata`	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
`correlation`	logical; if `TRUE`, the correlation matrix of the estimated parameters is returned and printed.
`symbolic.cor`	logical. If `TRUE`, print the correlations in a symbolic form (see `symnum`) rather than as numbers.
`complete`	logical indicating if the full variance-covariance matrix should be returned also in case of an over-determined system where some coefficients are undefined and `coef(.)` contains NAs correspondingly. When `complete = TRUE`, `vcov()` is compatible with `coef()` also in this singular case.

Details

This function performs estimation and inference for regressions involving ranks. Suppose there is a dependent variable Y_i and independent variables X_i and W_i, where X_i is a scalar and W_i a vector (possibly including a constant). Instead of running a linear regression of Y_i on X_i and W_i, we want to first transform Y_i and/or X_i into ranks. Denote by R_i^Y the rank of Y_i and R_i^X the rank of X_i. Then, a **rank-rank regression**,

$$R_i^Y = \rho R_i^X + W_i'\beta + \varepsilon_i,$$

is run using the formula `r(Y)~r(X)+W`. Similarly, a **regression of the raw dependent variable on the ranked regressor**,

$$Y_i = \rho R_i^X + W_i'\beta + \varepsilon_i,$$

can be implemented by the formula `Y~r(X)+W`, and a **regression of the ranked dependent variable on the raw regressors**,

$$R_i^Y = W_i'\beta + \varepsilon_i,$$

can be implemented by the formula `r(Y)~W`.

The function works, in many ways, just like `lm` for linear regressions. Apart from some smaller details, there are two important differences: first, in `lmranks`, the mark `r()` can be used in formulas to indicate variables to be ranked before running the regression and, second, subsequent use of `summary` produces a summary table with the correct standard errors, t-values and p-values (while those of the `lm` are not correct for regressions involving ranks). See Chetverikov and Wilhelm (2023) for more details.

Many other aspects of the function are similar to `lm`. For instance, `.` in a formula means 'all columns not otherwise in the formula' just as in `lm`. An intercept is included by default. In a model specified as `r(Y)~r(X)+.`, both `r(X)` and `X` will be included in the model - as it would have been in `lm` and, say, `log()` instead of `r()`. One can exclude `X` with a `-`, i.e. `r(Y)~r(X)+.-X`. See `formula` for more about model specification.

The `r()` is a private alias for `frank`. The `increasing` argument, provided at individual regressor level, specifies whether the ranks should increase or decrease as regressor values increase. The `omega` argument of `frank`, provided at `lmranks` function level, specifies how ties in variables are to be handled and can be supplied as argument in `lmranks`. For more details, see `frank`. By default `increasing` is set to `TRUE` and `omega` is set equal to `1`, which means `r()` computes ranks by transforming a variable through its empirical cdf.

Many functions defined for `lm` also work correctly with `lmranks`. These include `coef`, `model.frame`, `model.matrix`, `resid`, `update` and others. On the other hand, some would return incorrect results if they treated `lmranks` output in the same way as `lm`'s. The central contribution of this package are `vcov`, `summary` and `confint` implementations using the correct asymptotic theory for regressions involving ranks.

See the `lm` documentation for more.

Value

An object of class `lmranks`, inheriting (as much as possible) from class `lm`.

Additionally, it has an `omega` entry, corresponding to the `omega` argument, a `ranked_response` logical entry, and a `rank_terms_indices` - an integer vector with indices of entries of `terms.labels` attribute of `terms(formula)`, which correspond to ranked regressors.

Methods (by generic)

- `plot(lmranks)`: Plot diagnostics for an `lmranks` object

  Displays plots useful for assessing quality of model fit. Currently, only one plot is available, which plots fitted values against residuals (for homoscedacity check).
- `proj(lmranks)`: Projections of the data onto terms of rank-rank regression model
- `predict(lmranks)`: Predict method for Linear Model for Ranks Fits
- `summary(lmranks)`: Summarizing fits of rank-rank regressions
- `vcov(lmranks)`: Calculate Variance-Covariance Matrix for a Fitted `lmranks` object

  Returns the variance-covariance matrix of the regression coefficients (main parameters) of a fitted `lmranks` object. Its result is theoretically valid and asymptotically consistent, in contrast to naively running `vcov(lm(...))`.

Rank-rank regressions with clusters

Sometimes, the data is divided into clusters (groups) and one is interested in running rank-rank regressions separately within each cluster, where the ranks are not computed within each cluster, but using all observations pooled across all clusters. Specifically, let $G_i = 1, \ldots, n_G$ denote a variable that indicates the cluster to which the i-th observation belongs. Then, the regression model of interest is

$$R_i^Y = \sum_{g=1}^{n_G} 1\{G_i = g\}(\rho_g R_i^X + W_i' \beta_g) + \varepsilon_i,$$

where ρ_g and β_g are now cluster-specific coefficients, but the ranks R_i^Y and R_i^X are computed as ranks among all observations Y_i and X_i, respectively. That means the rank of an observation is not computed among the other observations in the same cluster, but rather among all available observations across all clusters.

This type of regression is implemented in the `lmranks` function using interaction notation: `r(Y)~(r(X)+W):G`. Here, the variable G **must** be a `factor`.

Since the theory for clustered regression mixing grouped and ungrouped (in)dependent variables is not yet developed, such a model will raise an error. Also, by default the function includes a cluster-specific intercept, i.e. `r(Y)~(r(X)+W):G` is internally interpreted as `r(Y)~(r(X)+W):G+G-1`.

`contrasts` of `G` must be of `contr.treatment` kind, which is the default.

Warning

As a consequence of the order, in which `model.frame` applies operations, `subset` and `na.action` would be applied after evaluation of `r()`. That would drop some rank values from the final model frame and returned coefficients and standard errors could no longer be correct. The user must handle NA values and filter the data on their own prior to usage in `lmranks`.

Wrapping `r()` with other functions (like `log(r(x))`) will not recognize correctly the mark (because it will not be caught in `terms(formula, specials = "r")`). The ranks will be calculated correctly, but their transformation will be treated later in `lm` as a regular regressor. This means that the corresponding regression coefficient will be calculated correctly, but the standard errors, statistics etc. will not.

`r`, `.r_predict` and `.r_cache` are special expressions, used internally to interpret `r` mark correctly. Do not use them in `formula`.

A number of methods defined for `lm` do not yield theoretically correct results when applied to `lmranks` objects; errors or warnings are raised in those instances. Also, the `df.residual` component is set to NA, since the notion of effects of freedom for the rank models is not theoretically established (at time of 1.2 release).

References

Chetverikov and Wilhelm (2023), "Inference for Rank-Rank Regressions". arXiv preprint arXiv:2310.15512

See Also

`lm` for details about other arguments; `frank`.

Generic functions `coef`, `effects`, `residuals`, `fitted`, `model.frame`, `model.matrix`, `update` .

Examples

```
# rank-rank regression:
X <- rnorm(500)
Y <- X + rnorm(500)
rrfit <- lmranks(r(Y) ~ r(X))
summary(rrfit)

# naive version of the rank-rank regression:
RY <- frank(Y, increasing=TRUE, omega=1)
RX <- frank(X, increasing=TRUE, omega=1)
fit <- lm(RY ~ RX)
summary(fit)
# the coefficient estimates are the same as in the lmranks function, but
# the standard errors, t-values, p-values are incorrect

# support of `data` argument:
data(mtcars)
lmranks(r(mpg) ~ r(hp) + ., data = mtcars)
# Same as above, but use the `hp` variable only through its rank
lmranks(r(mpg) ~ r(hp) + . - hp, data = mtcars)

# rank-rank regression with clusters:
G <- factor(rep(LETTERS[1:4], each=nrow(mtcars) / 4))
lmr <- lmranks(r(mpg) ~ r(hp):G, data = mtcars)
summary(lmr)
model.matrix(lmr)
# Include all columns of mtcars as usual covariates:
lmranks(r(mpg) ~ (r(hp) + .):G, data = mtcars)
```

`parent_child_income` *Income of parents and children*

Description

An artificial dataset containing income of children and their parents together with some information about them.

Usage

```
parent_child_income
```

Format

A data frame with 3894 rows and 4 variables:

c_faminc Family income of a child
p_faminc Family income of parent
gender Gender
race Race: hisp (Hispanic), black or neither

`pisa` *Cross-country comparison of students' achievement*

Description

[Deprecated]

New code should use `data(pisa2018)` instead.

Dataset containing average scores on math, reading, and science together with standard errors for all OECD countries. These are from the 2018 Program for International Student Assessment (PISA) study by the Organization for Economic Cooperation and Development (OECD). The average scores are over all 15-year-old students in the study.

Usage

```
pisa
```

Format

A data frame with 37 rows and 7 variables:

jurisdiction country, from which data was collected

math_score average score in math

math_se standard error for the average score in math

reading_score average score in reading

reading_se standard error for the average score in reading

science_score average score in science

science_se standard error for the average score in science

Source

https://www.oecd.org/en/about/programmes/pisa/pisa-data.html

`pisa2018` *Cross-country comparison of students' achievement*

Description

Datasets containing average scores on math, reading, and science together with standard errors for all OECD countries. These are from the 2018 and 2022 editions of Program for International Student Assessment (PISA) study by the Organization for Economic Cooperation and Development (OECD). The average scores are over all 15-year-old students in the study.

Usage

```
pisa2018

pisa2022
```

Format

jurisdiction country, from which data was collected

math_score average score in math

math_se standard error for the average score in math

reading_score average score in reading

reading_se standard error for the average score in reading

science_score average score in science

science_se standard error for the average score in science

An object of class `data.frame` with 38 rows and 7 columns.

Source

https://www.oecd.org/en/about/programmes/pisa/pisa-data.html

`plot.csranks` *Plot ranking with confidence sets*

Description

Display ranks together with their confidence set bounds.

Usage

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

plotranking(
  ranks,
  L,
  U,
  popnames = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  colorbins = 1,
  horizontal = TRUE
)
```

Arguments

x	An csranks object, likely produced by csranks.
...	Other arguments, passed to plotranking.
ranks	vector of ranks
L	vector of lower bounds of confidence sets for the ranks
U	vector of lower bounds of confidence sets for the ranks
popnames	vector containing names of the populations whose ranks are in ranks. If popnames=NULL (default), then populations are automatically numbered.
title	character string containing the main title of the graph. title=NULL (default) means no title.
subtitle	character string containing the subtitle of the graph. subtitle=NULL (default) means no subtitle.
caption	character string containing the caption of the graph. caption=NULL (default) means no caption.
colorbins	integer indicating the number of quantile bins into which populations are grouped and color-coded. Value has to lie between 1 (default) and the number of populations.
horizontal	logical. Should be the bars displayed horizontally, or vertically?

Value

A ggplot plot displaying confidence sets.

Functions

- plot(csranks): Plot csranks output

Examples

```
x <- seq(1, 3, length = 10)
V <- diag(rep(0.04, 10))
CS <- csranks(x, V)
grid::current.viewport()
plot(CS)
# Equivalent:
plotranking(CS$rank, CS$L, CS$U)

# plotranking returns a ggplot object. It can be customized further:
library(ggplot2)
pl <- plot(CS)
pl + xlab("position in ranking") + ylab("population label") + theme_gray()

# horizontal = FALSE uses ggplot2::coord_flip underneath. The x and y axes swap places.
pl <- plot(CS, horizontal = FALSE)
pl + xlab("position in ranking") + # Note, that xlab refers to vertical axis now
  ylab("population label") + theme_gray()
```

Index