Package 'robustbase'

August 19, 2024

Version 0.99-4

VersionNote Released 0.99-3 on 2024-07-01 and 0.99-2 on 2024-01-27 to CRAN

Date 2024-08-19

Title Basic Robust Statistics

URL https://robustbase.R-forge.R-project.org/,
https://R-forge.R-project.org/R/?group_id=59,
https://R-forge.R-project.org/scm/viewvc.php/pkg/robustbase/?root=robustbase,
svn://svn.r-forge.r-project.org/svnroot/robustbase/pkg/robustbase

BugReports https://R-forge.R-project.org/tracker/?atid=302&group_id=59

Description ``Essential'' Robust Statistics.
Tools allowing to analyze data with robust methods. This includes regression methodology including model selections and multivariate statistics where we strive to cover the book ``Robust Statistics, Theory and Methods'' by 'Maronna, Martin and Yohai'; Wiley 2006.

Depends R (>= 3.5.0)

Imports stats, graphics, utils, methods, DEoptimR

Suggests grid, MASS, lattice, boot, cluster, Matrix, robust, fit.models, MPV, xtable, ggplot2, GGally, RColorBrewer, reshape2, sfsmisc, catdata, doParallel, foreach, skewt

SuggestsNote mostly only because of vignette graphics and simulation

Enhances robustX, rrcov, matrixStats, quantreg, Hmisc

EnhancesNote linked to in man/*.Rd

LazyData yes

NeedsCompilation yes

License GPL (>= 2)

Author Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>),
Peter Rousseeuw [ctb] (Qn and Sn),
Christophe Croux [ctb] (Qn and Sn),
Valentin Todorov [aut] (most robust Cov),

Andreas Ruckstuhl [aut] (nlrob, anova, glmrob),
Matias Salibian-Barrera [aut] (lmrob orig.),
Tobias Verbeke [ctb, fnd] (mc, adjbox),
Manuel Koller [aut] (mc, lmrob, psi-func.),
Eduardo L. T. Conceicao [aut] (MM-, tau-, CM-, and MTL- nlrob),
Maria Anna di Palma [ctb] (initial version of Comedian)

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Repository CRAN

Date/Publication 2024-08-19 10:30:03 UTC

Contents

`adjbox` *Plot an Adjusted Boxplot for Skew Distributions*

Description

Produces boxplots adjusted for skewed distributions as proposed in Hubert and Vandervieren (2008).

Usage

```
adjbox(x, ...)

## S3 method for class 'formula'
adjbox(formula, data = NULL, ..., subset, na.action = NULL)

## Default S3 method:
adjbox(x, ..., range = 1.5, doReflect = FALSE,
        width = NULL, varwidth = FALSE,
        notch = FALSE, outline = TRUE, names, plot = TRUE,
        border = par("fg"), col = NULL, log = "",
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
        horizontal = FALSE, add = FALSE, at = NULL)
```

Arguments

formula	a formula, such as y ~ grp, where y is a numeric vector of data values to be split into groups according to the grouping variable grp (usually a factor).
data	a data.frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used for plotting.
na.action	a function which indicates what should happen when the data contain NAs. The default is to ignore missing values in either the response or the group.
x	for specifying data from which the boxplots are to be produced. Either a numeric vector, or a single list containing such vectors. Additional unnamed arguments specify further data as separate vectors (each corresponding to a component boxplot). NAs are allowed in the data.
...	For the formula method, named arguments to be passed to the default method. For the default method, unnamed arguments are additional data vectors (unless x is a list when they are ignored), and named arguments are arguments and graphical parameters to be passed to bxp in addition to the ones given by argument pars (and override those in pars).
range	this determines how far the plot whiskers extend out from the box, and is simply passed as argument coef to adjboxStats(). If range is positive, the whiskers extend to the most extreme data point which is no more than range times the interquartile range from the box. A value of zero causes the whiskers to extend to the data extremes.
doReflect	logical indicating if the MC should also be computed on the *reflected* sample -x, and be averaged, see mc.
width	a vector giving the relative widths of the boxes making up the plot.
varwidth	if varwidth is TRUE, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.
notch	if notch is TRUE, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap this is 'strong evidence' that the two medians differ (Chambers *et al.*, 1983, p. 62). See boxplot.stats for the calculations used.
outline	if outline is not true, the outliers are not drawn (as points whereas S+ uses lines).
names	group labels which will be printed under each boxplot.
boxwex	a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower.
staplewex	staple line width expansion, proportional to box width.
outwex	outlier line width expansion, proportional to box width.
plot	if TRUE (the default) then a boxplot is produced. If not, the summaries which the boxplots are based on are returned.
border	an optional vector of colors for the outlines of the boxplots. The values in border are recycled if the length of border is less than the number of plots.
col	if col is non-null it is assumed to contain colors to be used to colour the bodies of the box plots. By default they are in the background colour.

log	character indicating if x or y or both coordinates should be plotted in log scale.
pars	a list of (potentially many) more graphical parameters, e.g., boxwex or outpch; these are passed to bxp (if plot is true); for details, see there.
horizontal	logical indicating if the boxplots should be horizontal; default FALSE means vertical boxes.
add	logical, if true *add* boxplot to current plot.
at	numeric vector giving the locations where the boxplots should be drawn, particularly when add = TRUE; defaults to 1:n where n is the number of boxes.

Details

The generic function adjbox currently has a default method (adjbox.default) and a formula interface (adjbox.formula).

If multiple groups are supplied either as multiple arguments or via a formula, parallel boxplots will be plotted, in the order of the arguments or the order of the levels of the factor (see factor).

Missing values are ignored when forming boxplots.

Extremes of the upper and whiskers of the adjusted boxplots are computed using the medcouple (mc()), a robust measure of skewness. For details, cf. TODO

Value

A list with the following components:

stats	a matrix, each column contains the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker for one group/plot. If all the inputs have the same class attribute, so will this component.
n	a vector with the number of observations in each group.
coef	a matrix where each column contains the lower and upper extremes of the notch.
out	the values of any data points which lie beyond the extremes of the whiskers.
group	a vector of the same length as out whose elements indicate to which group the outlier belongs.
names	a vector of names for the groups.

Note

The code and documentation only slightly modifies the code of boxplot.default, boxplot.formula and boxplot.stats

Author(s)

R Core Development Team, slightly adapted by Tobias Verbeke

References

Hubert, M. and Vandervieren, E. (2008). An adjusted boxplot for skewed distributions, *Computational Statistics and Data Analysis* **52**, 5186–5201. doi:10.1016/j.csda.2007.11.008

See Also

The medcouple, mc; boxplot.

Examples

```
if(require("boot")) {
 ### Hubert and Vandervieren (2008), Fig. 5.%(2006): p. 10, Fig. 4.
 data(coal, package = "boot")
 coaldiff <- diff(coal$date)
 op <- par(mfrow = c(1,2))
 boxplot(coaldiff, main = "Original Boxplot")
 adjbox(coaldiff, main  = "Adjusted Boxplot")
 par(op)
}

### Hubert and Vandervieren (2008), p. 11, Fig. 7a -- enhanced
op <- par(mfrow = c(2,2), mar = c(1,3,3,1), oma = c(0,0,3,0))
with(condroz, {
 boxplot(Ca, main = "Original Boxplot")
 adjbox (Ca, main = "Adjusted Boxplot")
 boxplot(Ca, main = "Original Boxplot [log]", log = "y")
 adjbox (Ca, main = "Adjusted Boxplot [log]", log = "y")
})
mtext("'Ca' from data(condroz)",
      outer=TRUE, font = par("font.main"), cex = 2)
par(op)
```

---

adjboxStats *Statistics for Skewness-adjusted Boxplots*

Description

Computes the "statistics" for producing boxplots adjusted for skewed distributions as proposed in Hubert and Vandervieren (2008), see adjbox.

Usage

```
adjboxStats(x, coef = 1.5, a = -4, b = 3, do.conf = TRUE, do.out = TRUE,
            ...)
```

Arguments

x	a numeric vector for which adjusted boxplot statistics are computed.
coef	number determining how far 'whiskers' extend out from the box, see boxplot.stats.
a, b	scaling factors multiplied by the medcouple mc() to determine outlyer boundaries; see the references.
do.conf, do.out	logicals; if FALSE, the conf or out component respectively will be empty in the result.
...	further optional arguments to be passed to mc(), such as doReflect.

Details

Given the quartiles Q_1, Q_3, the interquartile range $\Delta Q := Q_3 - Q_1$, and the medcouple $M :=$`mc(x)`, $c =$`coef`, the "fence" is defined, for $M \geq 0$ as

$$[Q_1 - ce^{a \cdot M}\Delta Q, Q_3 + ce^{b \cdot M}\Delta Q],$$

and for $M < 0$ as

$$[Q_1 - ce^{-b \cdot M}\Delta Q, Q_3 + ce^{-a \cdot M}\Delta Q],$$

and all observations `x` outside the fence, the "potential outliers", are returned in `out`.

Note that a typo in robustbase version up to 0.7-8, for the (rare left-skewed) case where mc(x) < 0, lead to a "fence" not wide enough in the upper part, and hence *less* outliers there.

Value

A `list` with the components

`stats`	a vector of length 5, containing the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker.
`n`	the number of observations
`conf`	the lower and upper extremes of the 'notch' (`if(do.conf)`). See `boxplot.stats`.
`fence`	length 2 vector of interval boundaries which define the non-outliers, and hence the whiskers of the plot.
`out`	the values of any data points which lie beyond the fence, and hence beyond the extremes of the whiskers.

Note

The code only slightly modifies the code of R's `boxplot.stats`.

Author(s)

R Core Development Team (`boxplot.stats`); adapted by Tobias Verbeke and Martin Maechler.

See Also

`adjbox()`, also for references, the function which mainly uses this one; further `boxplot.stats`.

Examples

```
data(condroz)
adjboxStats(ccA <- condroz[,"Ca"])
adjboxStats(ccA, doReflect = TRUE)# small difference in fence

## Test reflection invariance [was not ok, up to and including robustbase_0.7-8]
a1 <- adjboxStats( ccA, doReflect = TRUE)
a2 <- adjboxStats(-ccA, doReflect = TRUE)

nm1 <- c("stats", "conf", "fence")
```

```
stopifnot(all.equal(       a1[nm1],
                    lapply(a2[nm1], function(u) rev(-u))),
          all.equal(a1[["out"]], -a2[["out"]]))
```

---

adjOutlyingness *Compute (Skewness-adjusted) Multivariate Outlyingness*

Description

For an $n \times p$ data matrix (or data frame) x, compute the "*outlyingness*" of all n observations. Outlyingness here is a generalization of the Donoho-Stahel outlyingness measure, where skewness is taken into account via the medcouple, mc().

Usage

```
adjOutlyingness(x, ndir = 250, p.samp = p, clower = 4, cupper = 3,
                IQRtype = 7,
                alpha.cutoff = 0.75, coef = 1.5,
                qr.tol = 1e-12, keep.tol = 1e-12,
                only.outlyingness = FALSE, maxit.mult = max(100, p),
                trace.lev = 0,
                mcReflect = n <= 100, mcScale = TRUE, mcMaxit = 2*maxit.mult,
                mcEps1 = 1e-12, mcEps2 = 1e-15,
                mcTrace = max(0, trace.lev-1))
```

Arguments

x	a numeric $n \times p$ matrix or data.frame, which must be of full rank p.
ndir	positive integer specifying the number of directions that should be searched.
p.samp	the sample size to use for finding good random directions, must be at least p. The default, p had been hard coded previously.
clower, cupper	the constant to be used for the lower and upper tails, in order to transform the data towards symmetry. You can set clower = 0, cupper = 0 to get the *non*-adjusted, i.e., classical ("central" or "symmetric") outlyingness. In that case, mc() is not used.
IQRtype	a number from 1:9, denoting type of empirical quantile computation for the IQR(). The default 7 corresponds to quantile's and IQR's default. MM has evidence that type=6 would be a bit more stable for small sample size.
alpha.cutoff	number in (0,1) specifying the quantiles $(\alpha, 1-\alpha)$ which determine the "outlier" cutoff. The default, using quartiles, corresponds to the definition of the medcouple (mc), but there is no stringent reason for using the same alpha for the outlier cutoff.
coef	positive number specifying the factor with which the interquartile range (IQR) is multiplied to determine 'boxplot hinges'-like upper and lower bounds.

qr.tol	positive tolerance to be used for qr and solve.qr for determining the ndir directions, each determined by a random sample of p (out of n) observations. Note that the default 10^{-12} is rather small, and qr's default = 1e-7 may be more appropriate.
keep.tol	positive tolerance to determine which of the sample direction should be kept, namely only those for which $\|x\| \cdot \|B\|$ is larger than keep.tol.
only.outlyingness	logical indicating if the final outlier determination should be skipped. In that case, a vector is returned, see 'Value:' below.
maxit.mult	integer factor; maxit <- maxit.mult * ndir will determine the maximal number of direction searching iterations. May need to be increased for higher dimensional data, though increasing ndir may be more important.
trace.lev	an integer, if positive allows to monitor the direction search.
mcReflect	passed as doReflect to mc().
mcScale	passed as doScale to mc().
mcMaxit	passed as maxit to mc().
mcEps1	passed as eps1 to mc(); the default is slightly looser (100 larger) than the default for mc().
mcEps2	passed as eps2 to mc().
mcTrace	passed as trace.lev to mc().

Details

FIXME: Details in the comment of the Matlab code; also in the reference(s).

The method as described can be useful as preprocessing in FASTICA (http://research.ics.aalto.fi/ica/fastica/ see also the R package **fastICA**.

Value

If only.outlyingness is true, a vector adjout, otherwise, as by default, a list with components

adjout	numeric of length(n) giving the adjusted outlyingness of each observation.
cutoff	cutoff for "outlier" with respect to the adjusted outlyingnesses, and depending on alpha.cutoff.
nonOut	logical of length(n), TRUE when the corresponding observation is **non**-outlying with respect to the cutoff and the adjusted outlyingnesses.

Note

If there are too many degrees of freedom for the projections, i.e., when $n \leq 4p$, the current definition of adjusted outlyingness is ill-posed, as one of the projections may lead to a denominator (quartile difference) of zero, and hence formally an adjusted outlyingness of infinity. The current implementation avoids Inf results, but will return seemingly random adjout values of around $10^{14} - -10^{15}$ which may be completely misleading, see, e.g., the longley data example.

The result is *random* as it depends on the sample of ndir directions chosen; specifically, to get sub samples the algorithm uses sample.int(n, p.samp) which from R version 3.6.0 depends on

`RNGkind(*, sample.kind)`. Exact reproducibility of results from R versions 3.5.3 and earlier, requires setting `RNGversion("3.5.0")`. In any case, do use `set.seed()` yourself for reproducibility!

Till Aug/Oct. 2014, the default values for `clower` and `cupper` were accidentally reversed, and the signs inside `exp(.)` where swapped in the (now corrected) two expressions

```
tup <- Q3 + coef * IQR * exp(.... + clower * tmc * (tmc < 0))
tlo <- Q1 - coef * IQR * exp(.... - cupper * tmc * (tmc < 0))
```

already in the code from Antwerpen ('`mcrsoft/adjoutlingness.R`'), contrary to the published reference.

Further, the original algorithm had not been scale-equivariant in the direction construction, which has been amended in 2014-10 as well.

The results, including diagnosed outliers, therefore have changed, typically slightly, since **robustbase** version 0.92-0.

Author(s)

Guy Brys; help page and improvements by Martin Maechler

References

Brys, G., Hubert, M., and Rousseeuw, P.J. (2005) A Robustification of Independent Component Analysis; *Journal of Chemometrics*, **19**, 1–12.

Hubert, M., Van der Veeken, S. (2008) Outlier detection for skewed data; *Journal of Chemometrics* **22**, 235–246; doi:10.1002/cem.1123.

For the up-to-date reference, please consult `https://wis.kuleuven.be/statdatascience/robust`

See Also

the adjusted boxplot, `adjbox` and the medcouple, `mc`.

Examples

```
## An Example with bad condition number and "border case" outliers

dim(longley) # 16 x 7  // set seed, as result is random :
set.seed(31)
ao1 <- adjOutlyingness(longley, mcScale=FALSE)
## which are outlying ?
which(!ao1$nonOut) ## for this seed, two: "1956", "1957"; (often: none)
## For seeds 1:100, we observe (Linux 64b)
if(FALSE) {
  adjO <- sapply(1:100, function(iSeed) {
            set.seed(iSeed); adjOutlyingness(longley)$nonOut })
  table(nrow(longley) - colSums(adjO))
}
## #{outl.}:  0  1  2  3
## #{cases}: 74 17  6  3
```

```
## An Example with outliers :

dim(hbk)
set.seed(1)
ao.hbk <- adjOutlyingness(hbk)
str(ao.hbk)
hist(ao.hbk $adjout)## really two groups
table(ao.hbk$nonOut)## 14 outliers, 61 non-outliers:
## outliers are :
which(! ao.hbk$nonOut) # 1 .. 14   --- but not for all random seeds!

## here, they are(*) the same as found by (much faster) MCD:
## *) only "almost", since the 2023-05 change to covMcd()
cc <- covMcd(hbk)
table(cc = cc$mcd.wt, ao = ao.hbk$nonOut)# one differ..:
stopifnot(sum(cc$mcd.wt != ao.hbk$nonOut) <= 1)

## This is revealing: About 1--2 cases, where outliers are *not* == 1:14
## (2023: ~ 1/8 [sec] per call)
if(interactive()) {
  for(i in 1:30) {
    print(system.time(ao.hbk <- adjOutlyingness(hbk)))
    if(!identical(iout <- which(!ao.hbk$nonOut), 1:14)) {
 cat("Outliers:\n"); print(iout)
    }
  }
}

## "Central" outlyingness: *not* calling mc()  anymore, since 2014-12-11:
trace(mc)
out <- capture.output(
  oo <- adjOutlyingness(hbk, clower=0, cupper=0)
)
untrace(mc)
stopifnot(length(out) == 0)

## A rank-deficient case
T <- tcrossprod(data.matrix(toxicity))
try(adjOutlyingness(T, maxit. = 20, trace.lev = 2)) # fails and recommends:
T. <- fullRank(T)
aT <- adjOutlyingness(T.)
plot(sort(aT$adjout, decreasing=TRUE), log="y")
plot(T.[,9:10], col = (1:2)[1 + (aT$adjout > 10000)])
## .. (not conclusive; directions are random, more 'ndir' makes a difference!)
```

---

aircraft *Aircraft Data*

Description

Aircraft Data, deals with 23 single-engine aircraft built over the years 1947-1979, from Office of Naval Research. The dependent variable is cost (in units of $100,000) and the explanatory variables are aspect ratio, lift-to-drag ratio, weight of plane (in pounds) and maximal thrust.

Usage

```
data(aircraft, package="robustbase")
```

Format

A data frame with 23 observations on the following 5 variables.

`X1` Aspect Ratio

`X2` Lift-to-Drag Ratio

`X3` Weight

`X4` Thrust

`Y` Cost

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, page 154, table 22.

Examples

```
data(aircraft)
summary( lm.airc <-        lm(Y ~ ., data = aircraft))
summary(rlm.airc <- MASS::rlm(Y ~ ., data = aircraft))

aircraft.x <- data.matrix(aircraft[,1:4])
c_air <- covMcd(aircraft.x)
c_air
```

---

airmay *Air Quality Data*

Description

Air Quality Data Set for May 1973, from Chambers et al. (1983). The whole data set consists of daily readings of air quality values from May 1, 1973 to September 30, 1973, but here are included only the values for May. This data set is an example of the special treatment of the missing values.

Usage

```
data(airmay, package="robustbase")
```

Format

A data frame with 31 observations on the following 4 variables.

`X1` Solar Radiation in Longleys in the frequency band 4000-7700 from 0800 to 1200 hours at Central Park

`X2` Average windspeed (in miles per hour) between 7000 and 1000 hours at La Guardia Airport

`X3` Maximum daily temperature (in degrees Fahrenheit) at La Guardia Airport

`Y` Mean ozone concentration (in parts per billion) from 1300 to 1500 hours at Roosevelt Island

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.86, table 6.

Examples

```
data(airmay)
summary(lm.airmay <- lm(Y ~ ., data=airmay))

airmay.x <- data.matrix(airmay[,1:3])
```

---

alcohol *Alcohol Solubility in Water Data*

Description

The solubility of alcohols in water is important in understanding alcohol transport in living organisms. This dataset from (Romanelli et al., 2001) contains physicochemical characteristics of 44 aliphatic alcohols. The aim of the experiment was the prediction of the solubility on the basis of molecular descriptors.

Usage

```
data(alcohol, package="robustbase")
```

Format

A data frame with 44 observations on the following 7 numeric variables.

`SAG` solvent accessible surface-bounded molecular volume.

`V` volume

`logPC` Log(PC); PC = octanol-water partitions coefficient

`P` polarizability

`RM` molar refractivity

`Mass` the mass

`logSolubility` ln(Solubility), the response.

Source

The website accompanying the MMY-book: https://www.wiley.com/legacy/wileychi/robust_statistics/

References

Maronna, R.A., Martin, R.D. and Yohai, V.J. (2006) *Robust Statistics, Theory and Methods*, Wiley.

Examples

```
data(alcohol)
## version of data set with trivial names, as
s.alcohol <- alcohol
names(s.alcohol) <- paste("Col", 1:7, sep='')
```

---

ambientNOxCH *Daily Means of NOx (mono-nitrogen oxides) in air*

Description

This dataset contains daily means (from midnight to midnight) of NOx, i.e., mono-nitrogen oxides, in [ppb] at 13 sites in central Switzerland and Aarau for the year 2004.

Usage

```
data(ambientNOxCH, package="robustbase")
```

Format

A data frame with 366 observations on the following 14 variables.

`date` date of day, of class `"Date"`.

`ad` Site is located north of Altdorf 100 meters east of motorway A2, on an open field at the beginning of a more than 2000m deep valley (690.175, 193.55; 438; inLuft)

`ba` Site is located in the centre of the little town of Baden in a residential area. Baden has 34'000 inhabitants and is situated on the swiss plateau (666.075, 257.972; 377; inLuft).

`ef` Site is located 6 km south of altdorf and 800 m north of the village of Erstfeld. The motorway A2 passes 5 m west of the measuring site. Over 8 million vehicles have passed Erstfeld in 2004 where 13% of the counts were attributed to trucks (691.43, 187.69; 457; MFM-U).

`la` Site is located on a wooded hill in a rural area called Laegern, about 190 m above Baden, which is about 5 km away (669.8, 259; 690; NABEL).

`lu` Site is located in the center of town of Lucerne, which has 57'000 inhabitants (666.19, 211.975; 460; inLuft).

`re` Site is located 1 km west of Reiden on the Swiss plateau. The motorway A2 passes 5 m west of the measuring site (639.56, 232.11; 462; MFM-U).

`ri` Site is located at Rigi Seebodenalp, 649 m above the lake of Lucerne on an alp with half a dozen small houses (677.9, 213.5; 1030; NABEL).

`se` Site is located in Sedel next to town of Lucerne 35m above and 250m south of motorway A14 from Zug to Lucerne on a low hill with free 360° panorama (665.5, 213.41; 484; inLuft).

`si` Site is located at the border of a small industrial area in Sisseln, 300 m east of a main road (640.725, 266.25; 305; inLuft).

`st` Site is located at the south east border of Stans with 7'000 inhabitants (670.85, 201.025; 438; inLuft).

`su` Site is located in the center of Suhr (8700 inhabitants), 10 m from the main road (648.49, 246.985; 403; inLuft).

`sz` Site is located in Schwyz (14'200 inhabitants) near a shopping center (691.92, 208.03; 470; inLuft).

`zg` Site is located in the centre of Zug with 22'000 inhabitants, 24 m from the main road (681.625, 224.625; 420; inLuft).

Details

The 13 sites are part of one of the three air quality monitoring networks: inLuft (regional authorities of central Switzerland and canton Aargau)
NABEL (Swiss federal network)
MFM-U (Monitoring flankierende Massnahmen Umwelt), special Swiss federal network along transit motorways A2 and A13 from Germany to Italy through Switzerland
The information within the brackets means: Swiss coordinates km east, km north; m above sea level; network

When the measuring sites are exposed to the same atmospheric condition and when there is no singular emission event at any site, `log(mean(NOx) of a specific day at each site)` is a linear function of `log(yearly.mean(NOx) at the corresponding site)`. The offset and the slope of the straight line reflects the atmospheric conditions at this specific day. During winter time, often an inversion prevents the emissions from being diluted vertically, so that there evolve two separate atmospheric compartements: One below the inversion boundary with polluted air and one above with relatively clean air. In our example below, Rigi Seebodenalp is above the inversion boundary between December 10th and 12th.

Source

http://www.in-luft.ch/
http://www.empa.ch/plugin/template/empa/*/6794
http://www.bafu.admin.ch/umweltbeobachtung/02272/02280

See Also

another NOx dataset, `NOxEmissions`.

Examples

```
data(ambientNOxCH)
str (ambientNOxCH)
```

```
yearly <- log(colMeans(ambientNOxCH[,-1], na.rm=TRUE))
xlim <- range(yearly)
lNOx <- log(ambientNOxCH[, -1])
days <-     ambientNOxCH[, "date"]

## Subset of 9 days starting at April 4:
idays <- seq(which(ambientNOxCH$date=="2004-12-04"), length=9)
ylim <- range(lNOx[idays,],na.rm=TRUE)
op <- par(mfrow=c(3,3),mar=rep(1,4), oma = c(0,0,2,0))

for (id in idays) {
  daily <- unlist(lNOx[id,])
  plot(NA, xlim=xlim,ylim=ylim, ann=FALSE, type = "n")
  abline(0:1, col="light gray")
  abline(lmrob(daily~yearly, na.action=na.exclude),
         col="red", lwd=2)
  text(yearly, daily, names(yearly), col="blue")
  mtext(days[id], side=1, line=-1.2, cex=.75, adj=.98)
}
mtext("Daily ~ Yearly  log( NOx mean values ) at 13 Swiss locations",
      outer=TRUE)
par(op)

## do all 366 regressions:  Least Squares and Robust:
LS <- lapply(1:nrow(ambientNOxCH), function(id)
             lm(unlist(lNOx[id,]) ~ yearly,
                na.action = na.exclude))
R <- lapply(1:nrow(ambientNOxCH),
            function(id) lmrob(unlist(lNOx[id,]) ~ yearly,
                               na.action = na.exclude))
## currently 4 warnings about non-convergence;
## which ones?
days[notOk <- ! sapply(R, `[[`, "converged") ]
## "2004-01-10" "2004-05-12" "2004-05-16" "2004-11-16"

## first problematic case:
daily <- unlist(lNOx[which(notOk)[1],])
plot(daily ~ yearly,
     main = paste("lmrob() non-convergent:",days[notOk[1]]))
rr <- lmrob(daily ~ yearly, na.action = na.exclude,
            control = lmrob.control(trace=3, max.it = 100))
##-> 53 iter.

## Look at all coefficients:
R.cf <- t(sapply(R, coef))
C.cf <- t(sapply(LS, coef))
plot(C.cf, xlim=range(C.cf[,1],R.cf[,1]),
           ylim=range(C.cf[,2],R.cf[,2]))
mD1 <- rowMeans(abs(C.cf - R.cf))
lrg <- mD1 > quantile(mD1, 0.80)
arrows(C.cf[lrg,1], C.cf[lrg,2],
       R.cf[lrg,1], R.cf[lrg,2], length=.1, col="light gray")
```

```
points(R.cf, col=2)

## All robustness weights
aW <- t(sapply(R, weights, type="robustness"))
colnames(aW) <- names(yearly)
summary(aW)
sort(colSums(aW < 0.05, na.rm = TRUE)) # how often "clear outlier":
# lu st zg ba se sz su si re la ef ad ri
#  0  0  0  1  1  1  2  3  4 10 14 17 48

lattice::levelplot(aW, asp=1/2, main="Robustness weights",
                   xlab= "day", ylab= "site")
```

`Animals2` *Brain and Body Weights for 65 Species of Land Animals*

Description

A data frame with average brain and body weights for 62 species of land mammals and three others.

Note that this is simply the union of `Animals` and `mammals`.

Usage

```
Animals2
```

Format

`body` body weight in kg

`brain` brain weight in g

Note

After loading the **MASS** package, the data set is simply constructed by `Animals2 <- local({D <- rbind(Animals, mammals); unique(D[order(D$body,D$brain),])})`.

Rousseeuw and Leroy (1987)'s 'brain' data is the same as **MASS**'s `Animals` (with Rat and Brachiosaurus interchanged, see the example below).

Source

Weisberg, S. (1985) *Applied Linear Regression.* 2nd edition. Wiley, pp. 144–5.

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection.* Wiley, p. 57.

References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* Forth Edition. Springer.

Examples

```
data(Animals2)
## Sensible Plot needs doubly logarithmic scale
plot(Animals2, log = "xy")

## Regression example plot:
plotbb <- function(bbdat) {
  d.name <- deparse(substitute(bbdat))
  plot(log(brain) ~ log(body), data = bbdat, main = d.name)
  abline(       lm(log(brain) ~ log(body), data = bbdat))
  abline(MASS::rlm(log(brain) ~ log(body), data = bbdat), col = 2)
  legend("bottomright", leg = c("lm", "rlm"), col=1:2, lwd=1, inset = 1/20)
}
plotbb(bbdat = Animals2)

## The `same' plot for Rousseeuw's subset:
data(Animals, package = "MASS")
brain <- Animals[c(1:24, 26:25, 27:28),]
plotbb(bbdat = brain)

lbrain <- log(brain)
plot(mahalanobis(lbrain, colMeans(lbrain), var(lbrain)),
     main = "Classical Mahalanobis Distances")
mcd <- covMcd(lbrain)
plot(mahalanobis(lbrain,mcd$center,mcd$cov),
     main = "Robust (MCD) Mahalanobis Distances")
```

---

anova.glmrob *Analysis of Robust Quasi-Deviance for "glmrob" Objects*

Description

Compute an analysis of robust quasi-deviance table for one or more generalized linear models fitted by glmrob.

Usage

```
## S3 method for class 'glmrob'
anova(object, ..., test = c("Wald", "QD", "QDapprox"))
```

Arguments

object, ...	objects of class glmrob, typically the result of a call to glmrob.
test	a character string specifying the test statistic to be used. (Partially) matching one of "Wald", "QD" or "QDapprox". See Details.

Details

Specifying a single object gives a sequential analysis of robust quasi-deviance table for that fit. That is, the reductions in the robust residual quasi-deviance as each term of the formula is added in turn are given in as the rows of a table. *(Currently not yet implemented.)*

If more than one object is specified, the table has a row for the residual quasi-degrees of freedom (However, this information is never used in the asymptotic tests). For all but the first model, the change in degrees of freedom and robust quasi-deviance is also given. (This only makes statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

In addition, the table will contain test statistics and P values comparing the reduction in robust quasi-deviance for the model on the row to that on top of it. For all robust fitting methods, the "Wald"-type test between two models can be applied (`test = "Wald"`).

When using Mallows or Huber type robust estimators (`method="Mqle"` in `glmrob`), then there are additional test methods. One is the robust quasi-deviance test (`test = "QD"`), as described by Cantoni and Ronchetti (2001). The asymptotic distribution is approximated by a chi-square distibution. Another test (`test = "QDapprox"`) is based on a quadratic approximation of the robust quasi-deviance test statistic. Its asymptotic distribution is chi-square (see the reference).

The comparison between two or more models by `anova.glmrob` will only be valid if they are fitted to the same dataset and by the same robust fitting method using the same tuning constant c (`tcc` in `glmrob`).

Value

Basically, an object of class `anova` inheriting from class `data.frame`.

Author(s)

Andreas Ruckstuhl

References

E. Cantoni and E. Ronchetti (2001) Robust Inference for Generalized Linear Models. *JASA* **96** (455), 1022–1030.

E.Cantoni (2004) Analysis of Robust Quasi-deviances for Generalized Linear Models. *Journal of Statistical Software* **10**, https://www.jstatsoft.org/article/view/v010i04

See Also

`glmrob`, `anova`.

Examples

```
## Binomial response -----------
data(carrots)
Cfit2 <- glmrob(cbind(success, total-success) ~ logdose + block,
                family=binomial, data=carrots, method="Mqle",
                control=glmrobMqle.control(tcc=1.2))
summary(Cfit2)
```

```
Cfit4 <- glmrob(cbind(success, total-success) ~ logdose * block,
                family=binomial, data=carrots, method="Mqle",
                control=glmrobMqle.control(tcc=1.2))

anova(Cfit2, Cfit4, test="Wald")

anova(Cfit2, Cfit4, test="QD")

anova(Cfit2, Cfit4, test="QDapprox")

## Poisson response ------------
data(epilepsy)

Efit2 <- glmrob(Ysum ~ Age10 + Base4*Trt, family=poisson, data=epilepsy,
               method="Mqle", control=glmrobMqle.control(tcc=1.2,maxit=100))
summary(Efit2)

Efit3 <- glmrob(Ysum ~ Age10 + Base4 + Trt, family=poisson, data=epilepsy,
               method="Mqle", control=glmrobMqle.control(tcc=1.2,maxit=100))

anova(Efit3, Efit2, test = "Wald")

anova(Efit3, Efit2, test = "QD")

## trivial intercept-only-model:
E0 <- update(Efit3, . ~ 1)
anova(E0, Efit3, Efit2, test = "QDapprox")
```

`anova.lmrob` *Analysis of Robust Deviances ('anova') for "lmrob" Objects*

Description

Compute an analysis of robust Wald-type or deviance-type test tables for one or more linear regression models fitted by `lmrob`.

Usage

```
## S3 method for class 'lmrob'
anova(object, ..., test = c("Wald", "Deviance"),
      verbose = getOption("verbose"))
```

Arguments

`object, ...`	objects of class `"lmrob"`, typically the result of a call to `lmrob`. `...` arguments may also be symbolic descriptions of the reduced models (cf. argument `formula` in `lm`).

`test`	a character string specifying the test statistic to be used. Can be one of `"Wald"` or `"Deviance"`, with partial matching allowed, for specifying a `"Wald"`-type test or `"Deviance"`-type test.
`verbose`	logical; if true some informative messages are printed.

Details

Specifying a single object gives a sequential analysis of a robust quasi-deviance table for that fit. That is, the reductions in the robust residual deviance as each term of the formula is added in turn are given in as the rows of a table. (Currently not yet implemented.)

If more than one object is specified, the table has a row for the residual quasi-degrees of freedom (however, this information is never used in the asymptotic tests). For all but the first model, the change in degrees of freedom and robust deviance is also given. (This only makes statistical sense if the models are nested.) As opposed to the convention, the models are forced to be listed from largest to smallest due to computational reasons.

In addition, the table will contain test statistics and P values comparing the reduction in robust deviances for the model on the row to that on top of it. There are two different robust tests available: The "Wald"-type test (`test = "Wald"`) and the Deviance-type test (`test = "Deviance"`). When using formula description of the nested models in the dot arguments and `test = "Deviance"`, you may be urged to supply a `lmrob` fit for these models by an error message. This happens when the coefficients of the largest model reduced to the nested models result in invalid initial estimates for the nested models (indicated by robustness weights which are all 0).

The comparison between two or more models by `anova.lmrob` will only be valid if they are fitted to the same dataset.

Value

Basically, an object of class `anova` inheriting from class `data.frame`.

Author(s)

Andreas Ruckstuhl

See Also

`lmrob`, `anova`.

Examples

```
data(salinity)
summary(m0.sali  <- lmrob(Y ~ . , data = salinity))
anova(m0.sali, Y ~ X1 + X3)
## -> X2 is not needed
(m1.sali  <- lmrob(Y ~ X1 + X3, data = salinity))
anova(m0.sali, m1.sali) # the same as before
anova(m0.sali, m1.sali, test = "Deviance")
## whereas 'X3' is highly significant:
m2 <- update(m0.sali, ~ . -X3)
anova(m0.sali, m2)
```

```
anova(m0.sali, m2,  test = "Deviance")
## Global test [often not interesting]:
anova(m0.sali, update(m0.sali, . ~ 1), test = "Wald")
anova(m0.sali, update(m0.sali, . ~ 1), test = "Deviance")

if(require("MPV")) { ## Montgomery, Peck & Vining  datasets
  Jet <- table.b13
  Jet.rflm1 <- lmrob(y ~ ., data=Jet,
                     control = lmrob.control(max.it = 500))
  summary(Jet.rflm1)

  anova(Jet.rflm1, y ~ x1 + x5 + x6, test="Wald")

  try( anova(Jet.rflm1, y ~ x1 + x5 + x6, test="Deviance") )
  ## -> Error in anovaLm....  Please fit the nested models by lmrob

  ## {{ since  all robustness weights become 0 in the nested model ! }}

  ## Ok: Do as the error message told us:
  ##    test by comparing the two *fitted* models:

  Jet.rflm2 <- lmrob(y ~ x1 + x5 + x6, data=Jet,
                     control=lmrob.control(max.it=100))
  anova(Jet.rflm1, Jet.rflm2, test="Deviance")

} # end{"MPV" data}
```

`biomassTill` *Biomass Tillage Data*

Description

An agricultural experiment in which different tillage methods were implemented. The effects of tillage on plant (maize) biomass were subsequently determined by modeling biomass accumulation for each tillage treatment using a 3 parameter Weibull function.

A datset where the total biomass is modeled conditional on a three value factor, and hence *vector* parameters are used.

Usage

```
data("biomassTill", package="robustbase")
```

Format

A data frame with 58 observations on the following 3 variables.

`Tillage` Tillage treatments, a `factor` with levels

`CA-:` a no-tillage system with plant residues removed

`CA+`: a no-tillage system with plant residues retained

`CT`: a conventionally tilled system with residues incorporated

`DVS` the development stage of the maize crop. A DVS of 1 represents maize anthesis (flowering), and a DVS of 2 represents physiological maturity. For the data, numeric vector with 5 different values between 0.5 and 2.

`Biomass` accumulated biomass of maize plants from each tillage treatment.

`Biom.2` the same as `Biomass`, but with three values replaced by "gross errors".

Source

From Strahinja Stepanovic and John Laborde, Department of Agronomy & Horticulture, University of Nebraska-Lincoln, USA

Examples

```
data(biomassTill)
str(biomassTill)
require(lattice)
## With long tailed errors
xyplot(Biomass ~ DVS | Tillage, data = biomassTill, type=c("p","smooth"))
## With additional 2 outliers:
xyplot(Biom.2 ~ DVS | Tillage, data = biomassTill, type=c("p","smooth"))

### Fit nonlinear Regression models: ----------------------------------

## simple starting values, needed:
m00st <- list(Wm = rep(300,  3),
              a = rep( 1.5, 3),
              b = rep( 2.2, 3))

robm <- nlrob(Biomass ~ Wm[Tillage] * (-expm1(-(DVS/a[Tillage])^b[Tillage])),
              data = biomassTill, start = m00st, maxit = 200)
##                                               -----------
summary(robm) ## ... 103 IRWLS iterations
plot(sort(robm$rweights), log = "y",
     main = "ordered robustness weights (log scale)")
mtext(getCall(robm))

## the classical (only works for the mild outliers):
cl.m <- nls(Biomass ~ Wm[Tillage] * (-expm1(-(DVS/a[Tillage])^b[Tillage])),
            data = biomassTill, start = m00st)

## now for the extra-outlier data: -- fails with singular gradient !!
try(
rob2 <- nlrob(Biom.2 ~ Wm[Tillage] * (-expm1(-(DVS/a[Tillage])^b[Tillage])),
              data = biomassTill, start = m00st)
)
## use better starting values:
m1st <- setNames(as.list(as.data.frame(matrix(
                coef(robm), 3))),
                c("Wm", "a","b"))
```

```
try(# just breaks a bit later!
rob2 <- nlrob(Biom.2 ~ Wm[Tillage] * (-expm1(-(DVS/a[Tillage])^b[Tillage])),
              data = biomassTill, start = m1st, maxit= 200, trace=TRUE)
)

## Comparison  {more to come} % once we have  "MM" working...
rbind(start = unlist(m00st),
      class = coef(cl.m),
      rob   = coef(robm))
```

---

bushfire *Campbell Bushfire Data*

Description

This data set was used by Campbell (1984) to locate bushfire scars. The dataset contains satelite measurements on five frequency bands, corresponding to each of 38 pixels.

Usage

```
data(bushfire, package="robustbase")
```

Format

A data frame with 38 observations on 5 variables.

Source

Maronna, R.A. and Yohai, V.J. (1995) The Behavoiur of the Stahel-Donoho Robust Multivariate Estimator. *Journal of the American Statistical Association* **90**, 330–341.

Examples

```
data(bushfire)
plot(bushfire)
covMcd(bushfire)
```

BYlogreg *Bianco-Yohai Estimator for Robust Logistic Regression*

Description

Computation of the estimator of Bianco and Yohai (1996) in logistic regression. Now provides both the *weighted* and regular (unweighted) BY-estimator.

By default, an intercept term is included and p parameters are estimated. For more details, see the reference.

Note: This function is for "back-compatibility" with the `BYlogreg()` code web-published at KU Leuven, Belgium, and also available as file '`FunctionsRob/BYlogreg.ssc`' from https://www.wiley.com/legacy/wileychi/robust_statistics/robust.html.

However instead of using this function, the recommended interface is `glmrob(*, method = "BY")` or `... method = "WBY" ..`, see `glmrob`.

Usage

```
BYlogreg(x0, y, initwml = TRUE, addIntercept = TRUE,
         const = 0.5, kmax = 1000, maxhalf = 10, sigma.min = 1e-4,
         trace.lev = 0)
```

Arguments

`x0`	a numeric $n \times (p-1)$ matrix containing the explanatory variables.
`y`	numeric n-vector of binomial (0 - 1) responses.
`initwml`	logical for selecting one of the two possible methods for computing the initial value of the optimization process. If `initwml` is true (default), a weighted ML estimator is computed with weights derived from the MCD estimator computed on the explanatory variables. If `initwml` is false, a classical ML fit is perfomed. When the explanatory variables contain binary observations, it is recommended to set initwml to FALSE or to modify the code of the algorithm to compute the weights only on the continuous variables.
`addIntercept`	logical indicating that a column of 1 must be added the x matrix.
`const`	tuning constant used in the computation of the estimator (default=0.5).
`kmax`	maximum number of iterations before convergence (default=1000).
`maxhalf`	max number of step-halving (default=10).
`sigma.min`	smallest value of the scale parameter before implosion (and hence non-convergence) is assumed.
`trace.lev`	logical (or integer) indicating if intermediate results should be printed; defaults to `0` (the same as `FALSE`).

Value

a list with components

`convergence`	logical indicating if convergence was achieved
`objective`	the value of the objective function at the minimum
`coefficients`	vector of parameter estimates
`vcov`	variance-covariance matrix of the coefficients (if convergence is TRUE).
`sterror`	standard errors, i.e., simply `sqrt(diag(.$vcov))`, if convergence.

Author(s)

Originally, Christophe Croux and Gentiane Haesbroeck, with thanks to Kristel Joossens and Valentin Todorov for improvements.

Speedup, tweaks, more "control" arguments: Martin Maechler.

References

Croux, C., and Haesbroeck, G. (2003) Implementing the Bianco and Yohai estimator for Logistic Regression, *Computational Statistics and Data Analysis* **44**, 273–295.

Ana M. Bianco and Víctor J. Yohai (1996) Robust estimation in the logistic regression model. In Helmut Rieder, *Robust Statistics, Data Analysis, and Computer Intensive Methods*, Lecture Notes in Statistics **109**, pages 17–34.

See Also

The more typical way to compute BY-estimates (via `formula` and methods): `glmrob(*, method = "WBY")` and `.. method = "BY"`.

Examples

```
set.seed(17)
x0 <- matrix(rnorm(100,1))
y  <- rbinom(100, size=1, prob= 0.5) # ~= as.numeric(runif(100) > 0.5)
BY <- BYlogreg(x0,y)
BY <- BYlogreg(x0,y, trace.lev=TRUE)

## The "Vaso Constriction"  aka "skin" data:
data(vaso)
vX <- model.matrix( ~ log(Volume) + log(Rate), data=vaso)
vY <- vaso[,"Y"]
head(cbind(vX, vY))# 'X' does include the intercept

vWBY <- BYlogreg(x0 = vX, y = vY, addIntercept=FALSE) # as 'vX' has it already
v.BY <- BYlogreg(x0 = vX, y = vY, addIntercept=FALSE, initwml=FALSE)
## they are relatively close, well used to be closer than now,
## with the (2023-05, VT) change of covMcd() scale-correction
stopifnot( all.equal(vWBY, v.BY, tolerance = 0.008) ) # was ~ 1e-4 till 2023-05
```

carrots *Insect Damages on Carrots*

Description

The damage carrots data set from Phelps (1982) was used by McCullagh and Nelder (1989) in order to illustrate diagnostic techniques because of the presence of an outlier. In a soil experiment trial with three blocks, eight levels of insecticide were applied and the carrots were tested for insect damage.

Usage

```
data(carrots, package="robustbase")
```

Format

A data frame with 24 observations on the following 4 variables.

success integer giving the number of carrots with insect damage.

total integer giving the total number of carrots per experimental unit.

logdose a numeric vector giving log(dose) values (eight different levels only).

block factor with levels `B1` to `B3`

Source

Phelps, K. (1982). Use of the complementary log-log function to describe doseresponse relationships in insecticide evaluation field trials.
In R. Gilchrist (Ed.), *Lecture Notes in Statistics, No. 14. GLIM.82: Proceedings of the International Conference on Generalized Linear Models*; Springer-Verlag.

References

McCullagh P. and Nelder, J. A. (1989) *Generalized Linear Models.* London: Chapman and Hall.

Eva Cantoni and Elvezio Ronchetti (2001); JASA, and
Eva Cantoni (2004); JSS, see `glmrob`

Examples

```
data(carrots)
str(carrots)
plot(success/total ~ logdose, data = carrots, col = as.integer(block))
coplot(success/total ~ logdose | block, data = carrots)

## Classical glm
Cfit0 <- glm(cbind(success, total-success) ~ logdose + block,
             data=carrots, family=binomial)
summary(Cfit0)

## Robust Fit (see help(glmrob)) ....
```

chgDefaults-methods *Change Defaults (Parameters) of "Psi Function" Objects*

Description

To modify an object of class psi_func, i.e. typically change the tuning parameters, the generic function chgDefaults() is called and works via the corresponding method.

Methods

object = "psi_func" The method is used to change the default values for the tuning parameters, and returns an object of class psi_func, a copy of input object with the slot tDefs possibly changed;.

See Also

psiFunc

Examples

```
## Hampel's psi and rho:
H.38 <- chgDefaults(hampelPsi, k = c(1.5, 3.5, 8))
H.38
plot(H.38)
## for more see  ?psiFunc
```

classPC *Compute Classical Principal Components via SVD or Eigen*

Description

Compute classical principal components (PC) via SVD (svd or eigenvalue decomposition (eigen) with non-trivial rank determination.

Usage

```
classPC(x, scale = FALSE, center = TRUE, signflip = TRUE,
        via.svd = n > p, scores = FALSE)
```

Arguments

x	a numeric matrix.
scale	logical indicating if the matrix should be scaled; it is mean centered in any case (via scale(*, scale=scale)c
center	logical or numeric vector for “centering” the matrix.

signflip	logical indicating if the sign(.) of the loadings should be determined should flipped such that the absolutely largest value is always positive.
via.svd	logical indicating if the computation is via SVD or Eigen decomposition; the latter makes sense typically only for n <= p.
scores	logical indicating

Value

a list with components

rank	the (numerical) matrix rank of x; an integer number, say k, from 0:min(dim(x)). In the $n > p$ case, it is rankMM(x).
eigenvalues	the k eigenvalues, in the $n > p$ case, proportional to the variances.
loadings	the loadings, a $p \times k$ matrix.
scores	if the scores argument was true, the $n \times k$ matrix of scores, where k is the rank above.
center	a numeric p-vector of means, unless the center argument was false.
scale	if the scale argument was not false, the scale used, a p-vector.

Author(s)

Valentin Todorov; efficiency tweaks by Martin Maechler

See Also

In spirit very similar to R's standard prcomp and princomp, one of the main differences being how the *rank* is determined via a non-trivial tolerance.

Examples

```
set.seed(17)
x <- matrix(rnorm(120), 10, 12) # n < p {the unusual case}
pcx  <- classPC(x)
(k <- pcx$rank) # = 9  [after centering!]
pc2  <- classPC(x, scores=TRUE)
pcS  <- classPC(x, via.svd=TRUE)
all.equal(pcx, pcS, tol = 1e-8)
## TRUE: eigen() & svd() based PC are close here
pc0 <- classPC(x, center=FALSE, scale=TRUE)
pc0$rank # = 10  here *no* centering (as E[.] = 0)

## Loadings are orthnormal:
zapsmall( crossprod( pcx$loadings ) )

## PC Scores are roughly orthogonal:
S.S <- crossprod(pc2$scores)
print.table(signif(zapsmall(S.S), 3), zero.print=".")
stopifnot(all.equal(pcx$eigenvalues, diag(S.S)/k))
```

```
## the usual n > p case :
pc.x <- classPC(t(x))
pc.x$rank # = 10, full rank in the n > p case

cpc1 <- classPC(cbind(1:3)) # 1-D matrix
stopifnot(cpc1$rank == 1,
          all.equal(cpc1$eigenvalues, 1),
          all.equal(cpc1$loadings, 1))
```

---

cloud *Cloud point of a Liquid*

Description

This data set contains the measurements concerning the cloud point of a Liquid, from Draper and Smith (1969). The cloud point is a measure of the degree of crystallization in a stock.

Usage

```
data(cloud, package="robustbase")
```

Format

A data frame with 19 observations on the following 2 variables.

`Percentage` Percentage of I-8

`CloudPoint` Cloud point

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.96, table 10.

Examples

```
data(cloud)
summary(lm.cloud <- lm(CloudPoint ~., data=cloud))
```

coleman *Coleman Data Set*

Description

Contains information on 20 Schools from the Mid-Atlantic and New England States, drawn from a population studied by Coleman et al. (1966). Mosteller and Tukey (1977) analyze this sample consisting of measurements on six different variables, one of which will be treated as a responce.

Usage

```
data(coleman, package="robustbase")
```

Format

A data frame with 20 observations on the following 6 variables.

`salaryP` staff salaries per pupil

`fatherWc` percent of white-collar fathers

`sstatus` socioeconomic status composite deviation: means for family size, family intactness, father's education, mother's education, and home items

`teacherSc` mean teacher's verbal test score

`motherLev` mean mother's educational level, one unit is equal to two school years

`Y` verbal mean test score (y, all sixth graders)

Author(s)

Valentin Todorov

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection* Wiley, p.79, table 2.

Examples

```
data(coleman)
pairs(coleman)
summary( lm.coleman <-     lm(Y ~ . , data = coleman))
summary(lts.coleman <- ltsReg(Y ~ . , data = coleman))

coleman.x <- data.matrix(coleman[, 1:6])
(Cc <- covMcd(coleman.x))
```

colMedians *Fast Row or Column-wise Medians of a Matrix*

Description

Calculates the median for each row (column) of a matrix x. This is the same as but more efficient than apply(x, MM, median) for MM=2 or MM=1, respectively.

Usage

```
colMedians(x, na.rm = FALSE, hasNA = TRUE, keep.names=TRUE)
rowMedians(x, na.rm = FALSE, hasNA = TRUE, keep.names=TRUE)
```

Arguments

x	a numeric $n \times p$ matrix.
na.rm	if TRUE, NAs are excluded first, otherwise not.
hasNA	logical indicating if x may contain NAs. If set to FALSE, no internal NA handling is performed which typically is faster.
keep.names	logical indicating if row or column names of x should become names of the result - as is the case for apply(x, MM, median).

Details

The implementation of rowMedians() and colMedians() is optimized for both speed and memory. To avoid coercing to doubles (and hence memory allocation), there is a special implementation for integer matrices. That is, if x is an integer matrix, then rowMedians(as.double(x)) (rowMedians(as.double(x))) would require three times the memory of rowMedians(x) (colMedians(x)), but all this is avoided.

Value

a numeric vector of length n or p, respectively.

Missing values

Missing values are excluded before calculating the medians *unless* hasNA is false. Note that na.rm has no effect and is automatically false when hasNA is false, i.e., internally, before computations start, the following is executed:

```
if (!hasNA)        ## If there are no NAs, don't try to remove them
     narm <- FALSE
```

Author(s)

Henrik Bengtsson, Harris Jaffee, Martin Maechler

See Also

See `wgt.himedian()` for a weighted hi-median, and `colWeightedMedians()` etc from package **matrixStats** for *weighted* medians.
For mean estimates, see `rowMeans()` in `colSums()`.

Examples

```
set.seed(1); n <- 234; p <- 543 # n*p = 127'062
x <- matrix(rnorm(n*p), n, p)
x[sample(seq_along(x), size= n*p / 256)] <- NA
R1 <- system.time(r1 <- rowMedians(x, na.rm=TRUE))
C1 <- system.time(y1 <- colMedians(x, na.rm=TRUE))
R2 <- system.time(r2 <- apply(x, 1, median, na.rm=TRUE))
C2 <- system.time(y2 <- apply(x, 2, median, na.rm=TRUE))
R2 / R1 # speedup factor: ~= 4   {platform dependent}
C2 / C1 # speedup factor: ~= 5.8 {platform dependent}
stopifnot(all.equal(y1, y2, tol=1e-15),
          all.equal(r1, r2, tol=1e-15))

(m <- cbind(x1=3, x2=c(4:1, 3:4,4)))
stopifnot(colMedians(m) == 3,
          all.equal(colMeans(m), colMedians(m)),# <- including names !
          all.equal(rowMeans(m), rowMedians(m)))
```

---

condroz *Condroz Data*

Description

Dataset with pH-value and Calcium content in soil samples, collected in different communities of the Condroz region in Belgium. The data pertain to a subset of 428 samples with a pH-value between 7.0 and 7.5.

Usage

```
data(condroz, package="robustbase")
```

Format

A data frame with 428 observations on the following 2 variables.

`Ca` Calcium content of the soil sample

`pH` pH value of the soil sample

Details

For more information on the dataset, cf. Goegebeur et al. (2005).

Source

Hubert and Vandervieren (2006), p. 10. This dataset is also studied in Vandewalle et al. (2004).

References

See also those for adjbox.

Goegebeur, Y., Planchon, V., Beirlant, J., Oger, R. (2005). Quality Assesment of Pedochemical Data Using Extreme Value Methodology, Journal of Applied Science, 5, p. 1092-1102.

Vandewalle, B., Beirlant, J., Hubert, M. (2004). A robust estimator of the tail index based on an exponential regression model, in Hubert, M., Pison G., Struyf, A. and S. Van Aelst, ed., Theory and Applications of Recent Robust Methods, Birkhäuser, Basel, p. 367-376.

Examples

```
adjbox(condroz$Ca)
```

---

covComed *Co-Median Location and Scatter "Covariance" Estimator*

Description

Compute (versions of) the (multivariate) "Comedian" covariance, i.e., multivariate location and scatter estimator

Usage

```
covComed(X, n.iter = 2, reweight = FALSE, tolSolve = control$tolSolve,
         trace = control$trace, wgtFUN = control$wgtFUN,
         control = rrcov.control())
```

Arguments

X	data matrix of dimension, say $n \times p$.
n.iter	number of comedian() iterations. Can be as low as zero.
reweight	logical indicating if the final distances and weights should be recomputed from the final cov and center. The default is currently FALSE because that was implicit in the first version of the R code.
tolSolve	a numerical tolerance passed to solve.
trace	logical (or integer) indicating if intermediate results should be printed; defaults to FALSE; values ≥ 2 also produce print from the internal (Fortran) code.
wgtFUN	a character string or function, specifying how the weights for the reweighting step should be computed. The default, wgtFUN = "01.original" corresponds to 0-1 weights as proposed originally. Other predefined string options are available, though experimental, see the experimental .wgtFUN.covComed object.

`control` a list with estimation options - this includes those above provided in the function specification, see `rrcov.control` for the defaults. If `control` is supplied, the parameters from it will be used. If parameters are passed also in the invocation statement, they will override the corresponding elements of the control object.

Details

.. not yet ..

Value

an object of class `"covComed"` which is basically a list with components

`comp1` Description of 'comp1'

`comp2` Description of 'comp2'

... FIXME ...

Author(s)

Maria Anna di Palma (initial), Valentin Todorov and Martin Maechler

References

Falk, M. (1997) On mad and comedians. *Annals of the Institute of Statistical Mathematics* **49**, 615–644.

Falk, M. (1998). A note on the comedian for elliptical distributions. *Journal of Multivariate Analysis* **67**, 306–317.

See Also

`covMcd`, etc

Examples

```
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])
(cc1 <- covComed(hbk.x))
(ccW <- covComed(hbk.x, reweight=TRUE))
cc0  <- covComed(hbk.x, n.iter=0)
cc0W <- covComed(hbk.x, n.iter=0, reweight=TRUE)

stopifnot(all.equal(unclass(cc0), # here, the 0-1 weights don't change:
                    cc0W[names(cc0)], tol=1e-12, check.environment = FALSE),
          which(cc1$weights == 0) == 1:14,
          which(ccW$weights == 0) == 1:14,
          which(cc0$weights == 0) == 1:14)

## Martin's smooth reweighting:

## List of experimental pre-specified wgtFUN() creators:
```

```
## Cutoffs may depend on  (n, p, control$beta) :
str(.wgtFUN.covComed)
```

---

covMcd *Robust Location and Scatter Estimation via MCD*

Description

Compute the Minimum Covariance Determinant (MCD) estimator, a robust multivariate location and scale estimate with a high breakdown point, via the 'Fast MCD' or 'Deterministic MCD' ("DetMcd") algorithm.

Usage

```
covMcd(x, cor = FALSE, raw.only = FALSE,
       alpha =, nsamp =, nmini =, kmini =,
       scalefn =, maxcsteps =,
       initHsets = NULL, save.hsets = FALSE, names = TRUE,
       seed =, tolSolve =, trace =,
       use.correction =, wgtFUN =, control = rrcov.control())
```

Arguments

x	a matrix or data frame.
cor	should the returned result include a correlation matrix? Default is cor = FALSE.
raw.only	should only the "raw" estimate be returned, i.e., no (re)weighting step be performed; default is false.
alpha	numeric parameter controlling the size of the subsets over which the determinant is minimized; roughly alpha*n, (see 'Details' below) observations are used for computing the determinant. Allowed values are between 0.5 and 1 and the default is 0.5.
nsamp	number of subsets used for initial estimates or "best", "exact", or "deterministic". Default is nsamp = 500. For nsamp = "best" exhaustive enumeration is done, as long as the number of trials does not exceed 100'000 (= nLarge). For "exact", exhaustive enumeration will be attempted however many samples are needed. In this case a warning message may be displayed saying that the computation can take a very long time. For "deterministic", the *deterministic* MCD is computed; as proposed by Hubert et al. (2012) it starts from the h most central observations of *six* (deterministic) estimators.
nmini, kmini	for $n \geq 2 \times n_0$, $n_0 :=$ nmini, the algorithm splits the data into maximally kmini (by default 5) subsets, of size approximately, but at least nmini. When nmini*kmini < n, the initial search uses only a *subsample* of size nmini*kmini. The original algorithm had nmini = 300 and kmini = 5 hard coded.

scalefn	for the deterministic MCD: function to compute a robust scale estimate or character string specifying a rule determining such a function. The default, currently "hrv2012", uses the recommendation of Hubert, Rousseeuw and Verdonck (2012) who recommend Qn for $n < 1000$ and scaleTau2 for larger n. Alternatively, scalefn = "v2014", uses that rule with cutoff $n = 5000$.
maxcsteps	maximal number of concentration steps in the deterministic MCD; should not be reached.
initHsets	NULL or a Kxh integer matrix of initial subsets of observations of size h (specified by the indices in 1:n).
save.hsets	(for deterministic MCD) logical indicating if the initial subsets should be returned as initHsets.
names	logical; if true (as by default), several parts of the result have a names or dimnames respectively, derived from data matrix x.
seed	initial seed for random generator, like .Random.seed, see rrcov.control.
tolSolve	numeric tolerance to be used for inversion (solve) of the covariance matrix in mahalanobis.
trace	logical (or integer) indicating if intermediate results should be printed; defaults to FALSE; values ≥ 2 also produce print from the internal (Fortran) code.
use.correction	whether to use finite sample correction factors; defaults to TRUE.
wgtFUN	a character string or function, specifying how the weights for the reweighting step should be computed. Up to April 2013, the only option has been the original proposal in (1999), now specified by wgtFUN = "01.original" (or via control). Since **robustbase** version 0.92-3, Dec.2014, other predefined string options are available, though experimental, see the experimental .wgtFUN.covMcd object.
control	a list with estimation options - this includes those above provided in the function specification, see rrcov.control for the defaults. If control is supplied, the parameters from it will be used. If parameters are passed also in the invocation statement, they will override the corresponding elements of the control object.

Details

The minimum covariance determinant estimator of location and scatter implemented in covMcd() is similar to R function cov.mcd() in **MASS**. The MCD method looks for the $h(> n/2)$ ($h = h(\alpha, n, p) =$ h.alpha.n(alpha,n,p)) observations (out of n) whose classical covariance matrix has the lowest possible determinant.

The raw MCD estimate of location is then the average of these h points, whereas the raw MCD estimate of scatter is their covariance matrix, multiplied by a consistency factor (.MCDcons(p, h/n)) and (if use.correction is true) a finite sample correction factor (.MCDcnp2(p, n, alpha)), to make it consistent at the normal model and unbiased at small samples. Both rescaling factors (consistency and finite sample) are returned in the length-2 vector raw.cnp2.

The implementation of covMcd uses the Fast MCD algorithm of Rousseeuw and Van Driessen (1999) to approximate the minimum covariance determinant estimator.

Based on these raw MCD estimates, (unless argument raw.only is true), a reweighting step is performed, i.e., V <- cov.wt(x,w), where w are weights determined by "outlyingness" with respect to

the scaled raw MCD. Again, a consistency factor and (if `use.correction` is true) a finite sample correction factor (`.MCDcnp2.rew(p, n, alpha)`) are applied. The reweighted covariance is typically considerably more efficient than the raw one, see Pison et al. (2002).

The two rescaling factors for the reweighted estimates are returned in `cnp2`. Details for the computation of the finite sample correction factors can be found in Pison et al. (2002).

Value

An object of class `"mcd"` which is basically a `list` with components

`center`	the final estimate of location.
`cov`	the final estimate of scatter.
`cor`	the (final) estimate of the correlation matrix (only if `cor = TRUE`).
`crit`	the value of the criterion, i.e., the logarithm of the determinant. Previous to Nov.2014, it contained the determinant itself which can under- or overflow relatively easily.
`best`	the best subset found and used for computing the raw estimates, with `length(best) == quan = h.alpha.n(alpha,n,p)`.
`mah`	mahalanobis distances of the observations using the final estimate of the location and scatter.
`mcd.wt`	weights of the observations using the final estimate of the location and scatter.
`cnp2`	a vector of length two containing the consistency correction factor and the finite sample correction factor of the final estimate of the covariance matrix.
`raw.center`	the raw (not reweighted) estimate of location.
`raw.cov`	the raw (not reweighted) estimate of scatter.
`raw.mah`	mahalanobis distances of the observations based on the raw estimate of the location and scatter.
`raw.weights`	weights of the observations based on the raw estimate of the location and scatter.
`raw.cnp2`	a vector of length two containing the consistency correction factor and the finite sample correction factor of the raw estimate of the covariance matrix.
`X`	the input data as numeric matrix, without `NA`s.
`n.obs`	total number of observations.
`alpha`	the size of the subsets over which the determinant is minimized (the default is $(n + p + 1)/2$).
`quan`	the number of observations, h, on which the MCD is based. If `quan` equals `n.obs`, the MCD is the classical covariance matrix.
`method`	character string naming the method (Minimum Covariance Determinant), starting with `"Deterministic"` when `nsamp="deterministic"`.
`iBest`	(for the deterministic MCD) contains indices from 1:6 denoting which of the (six) initial subsets lead to the best set found.
`n.csteps`	(for the deterministic MCD) for each of the initial subsets, the number of C-steps executed till convergence.
`call`	the call used (see `match.call`).

Author(s)

Valentin Todorov <valentin.todorov@chello.at>, based on work written for S-plus by Peter Rousseeuw and Katrien van Driessen from University of Antwerp.

Visibility of (formerly internal) tuning parameters, notably `wgtFUN()`: Martin Maechler

References

Rousseeuw, P. J. and Leroy, A. M. (1987) *Robust Regression and Outlier Detection.* Wiley.

Rousseeuw, P. J. and van Driessen, K. (1999) A fast algorithm for the minimum covariance determinant estimator. *Technometrics* **41**, 212–223.

Pison, G., Van Aelst, S., and Willems, G. (2002) Small Sample Corrections for LTS and MCD, *Metrika* **55**, 111–123.

Hubert, M., Rousseeuw, P. J. and Verdonck, T. (2012) A deterministic algorithm for robust location and scatter. Journal of Computational and Graphical Statistics **21**, 618–637.

See Also

`cov.mcd` from package **MASS**; `covOGK` as cheaper alternative for larger dimensions.

`BACON` and `covNNC`, from package **robustX**;

Examples

```
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])
set.seed(17)
(cH <- covMcd(hbk.x))
cH0 <- covMcd(hbk.x, nsamp = "deterministic")
with(cH0, stopifnot(quan == 39,
     iBest == c(1:4,6), # 5 out of 6 gave the same
     identical(raw.weights, mcd.wt),
     identical(which(mcd.wt == 0), 1:14), all.equal(crit, -1.045500594135)))

## the following three statements are equivalent
c1 <- covMcd(hbk.x, alpha = 0.75)
c2 <- covMcd(hbk.x, control = rrcov.control(alpha = 0.75))
## direct specification overrides control one:
c3 <- covMcd(hbk.x, alpha = 0.75,
             control = rrcov.control(alpha=0.95))
c1

## Martin's smooth reweighting:

## List of experimental pre-specified wgtFUN() creators:
## Cutoffs may depend on  (n, p, control$beta) :
str(.wgtFUN.covMcd)

cMM <- covMcd(hbk.x, wgtFUN = "sm1.adaptive")

ina <- which(names(cH) == "call")
```

```
all.equal(cMM[-ina], cH[-ina]) # *some* differences, not huge (same 'best'):
stopifnot(all.equal(cMM[-ina], cH[-ina], tol = 0.2))
```

---

covOGK *Orthogonalized Gnanadesikan-Kettenring (OGK) Covariance Matrix Estimation*

Description

Computes the orthogonalized pairwise covariance matrix estimate described in in Maronna and Zamar (2002). The pairwise proposal goes back to Gnanadesikan and Kettenring (1972).

Usage

```
covOGK(X, n.iter = 2, sigmamu, rcov = covGK, weight.fn = hard.rejection,
       keep.data = FALSE, ...)

covGK (x, y, scalefn = scaleTau2, ...)
s_mad(x, mu.too = FALSE, na.rm = FALSE)
s_IQR(x, mu.too = FALSE, na.rm = FALSE)
```

Arguments

X	data in something that can be coerced into a numeric matrix.
n.iter	number of orthogonalization iterations. Usually 1 or 2; values greater than 2 are unlikely to have any significant effect on the estimate (other than increasing the computing time).
sigmamu, scalefn	a function that computes univariate robust location and scale estimates. By default it should return a single numeric value containing the robust scale (standard deviation) estimate. When mu.too is true, sigmamu() should return a numeric vector of length 2 containing robust location and scale estimates. See scaleTau2, s_Qn, s_Sn, s_mad or s_IQR for examples to be used as sigmamu argument.
rcov	function that computes a robust covariance estimate between two vectors. The default, Gnanadesikan-Kettenring's covGK, is simply $(s^2(X+Y) - s^2(X-Y))/4$ where $s()$ is the scale estimate sigmamu().
weight.fn	a function of the robust distances and the number of variables p to compute the weights used in the reweighting step.
keep.data	logical indicating if the (untransformed) data matrix X should be kept as part of the result.
...	additional arguments; for covOGK to be passed to sigmamu() and weight.fn(); for covGK passed to scalefn.
x, y	numeric vectors of the same length, the covariance of which is sought in covGK (or the scale, in s_mad or s_IQR).

mu.too	logical indicating if both location and scale should be returned or just the scale (when mu.too=FALSE as by default).
na.rm	if TRUE then NA values are stripped from x before computation takes place.

Details

Typical default values for the *function* arguments sigmamu, rcov, and weight.fn, are available as well, see the *Examples* below, **but** their names and calling sequences are still subject to discussion and may be changed in the future.

The current default, weight.fn = hard.rejection corresponds to the proposition in the litterature, but Martin Maechler strongly believes that the hard threshold currently in use is too arbitrary, and further that *soft* thresholding should be used instead, anyway.

Value

covOGK() currently returns a list with components

center	robust location: numeric vector of length p.
cov	robust covariance matrix estimate: $p \times p$ matrix.
wcenter, wcov	re-**w**eighted versions of center and cov.
weights	the robustness weights used.
distances	the mahalanobis distances computed using center and cov.

.....
but note that this might be radically changed to returning an S4 classed object!

covGK() is a trivial 1-line function returning the covariance estimate

$$\hat{c}(x, y) = \left(\hat{\sigma}(x + y)^2 - \hat{\sigma}(x - y)^2\right)/4,$$

where $\hat{\sigma}(u)$ is the scale estimate of u specified by scalefn.

s_mad(), and s_IQR() return the scale estimates mad or IQR respectively, where the s_* functions return a length-2 vector (mu, sig) when mu.too = TRUE, see also scaleTau2.

Author(s)

Kjell Konis <konis@stats.ox.ac.uk>, with modifications by Martin Maechler.

References

Maronna, R.A. and Zamar, R.H. (2002) Robust estimates of location and dispersion of high-dimensional datasets; *Technometrics* **44**(4), 307–317.

Gnanadesikan, R. and John R. Kettenring (1972) Robust estimates, residuals, and outlier detection with multiresponse data. *Biometrics* **28**, 81–124.

See Also

scaleTau2, covMcd, cov.rob.

Examples

```
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])

cO1 <- covOGK(hbk.x, sigmamu = scaleTau2)
cO2 <- covOGK(hbk.x, sigmamu = s_Qn)
cO3 <- covOGK(hbk.x, sigmamu = s_Sn)
cO4 <- covOGK(hbk.x, sigmamu = s_mad)
cO5 <- covOGK(hbk.x, sigmamu = s_IQR)


data(toxicity)
cO1tox <- covOGK(toxicity, sigmamu = scaleTau2)
cO2tox <- covOGK(toxicity, sigmamu = s_Qn)

## nice formatting of correlation matrices:
as.dist(round(cov2cor(cO1tox$cov), 2))
as.dist(round(cov2cor(cO2tox$cov), 2))

## "graphical"
symnum(cov2cor(cO1tox$cov))
symnum(cov2cor(cO2tox$cov), legend=FALSE)
```

CrohnD *Crohn's Disease Adverse Events Data*

Description

Data set issued from a study of the adverse events of a drug on 117 patients affected by Crohn's disease (a chronic inflammatory disease of the intestines).

Usage

```
data(CrohnD, package="robustbase")
```

Format

A data frame with 117 observations on the following 9 variables.

`ID` the numeric patient IDs

`nrAdvE` the number of adverse events

`BMI` Body MASS Index, i.e., $weight[kg]/(height[m])^2$.

`height` in cm

`country` a factor with levels `0` and `1`

`sex` the person's gender, a binary factor with levels `M` `F`

`age` in years, a numeric vector

`weight` in kilograms, a numeric vector

`treat` how CD was treated: a factor with levels `0`, `1` and `2`, meaning placebo, drug 1 and drug 2.

Source

form the authors of the reference, with permission by the original data collecting agency.

References

Serigne N. Lô and Elvezio Ronchetti (2006). Robust Second Order Accurate Inference for Generalized Linear Models. Technical report, University of Geneva, Switzerland.

Examples

```
data(CrohnD)
str(CrohnD)
with(CrohnD, ftable(table(sex,country, treat)))
```

---

cushny *Cushny and Peebles Prolongation of Sleep Data*

Description

The original data set was bivariate and recorded for ten subjects the prolongation of sleep caused by two different drugs. These data were used by Student as the first illustration of the paired t-test which only needs the *differences* of the two measurements. These differences are the values of cushny.

Usage

```
data(cushny, package="robustbase")
```

Format

numeric vector, sorted increasingly:
0 0.8 1 1.2 1.3 1.3 1.4 1.8 2.4 4.6

Source

Cushny, A.R. and Peebles, A.R. (1905) The action of optical isomers. II. Hyoscines. *J. Physiol.* **32**, 501–510.

These data were used by Student(1908) as the first illustration of the paired t-test, see also sleep; then cited by Fisher (1925) and thereforth copied in numerous books as an example of a normally distributed sample, see, e.g., Anderson (1958).

References

Student (1908) The probable error of a mean. *Biometrika* **6**, 1–25.

Fisher, R.A. (1925) *Statistical Methods for Research Workers*; Oliver & Boyd, Edinburgh.

Anderson, T.W. (1958) *An Introduction to Multivariate Statistical Analysis*; Wiley, N.Y.

Hampel, F., Ronchetti, E., Rousseeuw, P. and Stahel, W. (1986) *Robust Statistics: The Approach Based on Influence Functions*; Wiley, N.Y.

Examples

```
data(cushny)

plot(cushny,  rep(0, 10), pch = 3, cex = 3,
     ylab = "", yaxt = "n")
plot(jitter(cushny),  rep(0, 10), pch = 3, cex = 2,
     main = "'cushny' data (n= 10)", ylab = "", yaxt = "n")
abline(h=0, col="gray", lty=3)
myPt <- function(m, lwd = 2, ..., e = 1.5*par("cxy")[2])
  segments(m, +e, m, -e, lwd = lwd, ...)
myPt(  mean(cushny), col = "pink3")
myPt(median(cushny), col = "light blue")
legend("topright", c("mean", "median"), lwd = 2,
       col = c("pink3", "light blue"), inset = .01)

## The 'sleep' data from the standard 'datasets' package:
d.sleep <- local({ gr <- with(datasets::sleep, split(extra, group))
                   gr[[2]] - gr[[1]] })
stopifnot(all.equal(cushny,
                    sort(d.sleep), tolerance=1e-15))
```

`delivery` *Delivery Time Data*

Description

Delivery Time Data, from Montgomery and Peck (1982). The aim is to explain the time required to service a vending machine (Y) by means of the number of products stocked (X1) and the distance walked by the route driver (X2).

Usage

```
data(delivery, package="robustbase")
```

Format

A data frame with 25 observations on the following 3 variables.

`n.prod` Number of Products

`distance` Distance

`delTime` Delivery time

Source

Montgomery and Peck (1982, p.116)

References

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, page 155, table 23.

Examples

```
data(delivery)
summary(lm.deli <- lm(delTime ~ ., data = delivery))

delivery.x <- as.matrix(delivery[, 1:2])
c_deli <- covMcd(delivery.x)
c_deli
```

---

education *Education Expenditure Data*

Description

Education Expenditure Data, from Chatterjee and Price (1977, p.108). This data set, representing the education expenditure variables in the 50 US states, providing an interesting example of heteroscedacity.

Usage

```
data(education, package="robustbase")
```

Format

A data frame with 50 observations on the following 6 variables.

`State` State

`Region` Region (1=Northeastern, 2=North central, 3=Southern, 4=Western)

`X1` Number of residents per thousand residing in urban areas in 1970

`X2` Per capita personal income in 1973

`X3` Number of residents per thousand under 18 years of age in 1974

`Y` Per capita expenditure on public education in a state, projected for 1975

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.110, table 16.

Examples

```
data(education)
education.x <- data.matrix(education[, 3:5])
summary(lm.education <- lm(Y ~ Region + X1+X2+X3, data=education))

## See  example(lmrob.M.S) # for how robust regression is used
```

epilepsy *Epilepsy Attacks Data Set*

Description

Data from a clinical trial of 59 patients with epilepsy (Breslow, 1996) in order to illustrate diagnostic techniques in Poisson regression.

Usage

```
data(epilepsy, package="robustbase")
```

Format

A data frame with 59 observations on the following 11 variables.

`ID` Patient identification number

`Y1` Number of epilepsy attacks patients have during the first follow-up period

`Y2` Number of epilepsy attacks patients have during the second follow-up period

`Y3` Number of epilepsy attacks patients have during the third follow-up period

`Y4` Number of epilepsy attacks patients have during the forth follow-up period

`Base` Number of epileptic attacks recorded during 8 week period prior to randomization

`Age` Age of the patients

`Trt` a factor with levels `placebo` `progabide` indicating whether the anti-epilepsy drug Progabide has been applied or not

`Ysum` Total number of epilepsy attacks patients have during the four follow-up periods

`Age10` Age of the patients devided by 10

`Base4` Variable `Base` devided by 4

Details

Thall and Vail reported data from a clinical trial of 59 patients with epilepsy, 31 of whom were randomized to receive the anti-epilepsy drug Progabide and 28 of whom received a placebo. Baseline data consisted of the patient's age and the number of epileptic seizures recorded during 8 week period prior to randomization. The response consisted of counts of seizures occuring during the four consecutive follow-up periods of two weeks each.

Source

Thall, P.F. and Vail S.C. (1990) Some covariance models for longitudinal count data with overdispersion. *Biometrics* **46**, 657–671.

References

Diggle, P.J., Liang, K.Y., and Zeger, S.L. (1994) *Analysis of Longitudinal Data*; Clarendon Press.

Breslow N. E. (1996) Generalized linear models: Checking assumptions and strengthening conclusions. *Statistica Applicata* **8**, 23–41.

Examples

```
data(epilepsy)
str(epilepsy)
pairs(epilepsy[,c("Ysum","Base4","Trt","Age10")])

Efit1 <- glm(Ysum ~ Age10 + Base4*Trt, family=poisson, data=epilepsy)
summary(Efit1)

## Robust Fit :
Efit2 <- glmrob(Ysum ~ Age10 + Base4*Trt, family=poisson, data=epilepsy,
                method = "Mqle",
                tcc=1.2, maxit=100)
summary(Efit2)
```

---

estimethod *Extract the Estimation Method 'Estimethod' from a Fitted Model*

Description

Extract the estimation method as a `character` string from a fitted model.

Usage

```
estimethod(object, ...)
```

Arguments

`object`	a fitted model.
`...`	additional, optional arguments. (None are used in our methods)

Details

This is a (S3) generic function for which we provide methods, currently for `nlrob` only.

Value

a `character` string, the estimation method used.

See Also

`nlrob`, and `nlrob.MM`, notably for examples.

exAM *Example Data of Antille and May - for Simple Regression*

Description

This is an artificial data set, cleverly construced and used by Antille and May to demonstrate 'problems' with LMS and LTS.

Usage

```
data(exAM, package="robustbase")
```

Format

A data frame with 12 observations on 2 variables, `x` and `y`.

Details

Because the points are not in general position, both LMS and LTS typically *fail*; however, e.g., `rlm(*, method="MM")` "works".

Source

Antille, G. and El May, H. (1992) The use of slices in the LMS and the method of density slices: Foundation and comparison.
In Yadolah Dodge and Joe Whittaker, editors, *COMPSTAT: Proc. 10th Symp. Computat. Statist., Neuchatel*, **1**, 441–445; Physica-Verlag.

Examples

```
data(exAM)
plot(exAM)
summary(ls <- lm(y ~ x, data=exAM))
abline(ls)
```

foodstamp *Food Stamp Program Participation*

Description

This data consists of 150 randomly selected persons from a survey with information on over 2000 elderly US citizens, where the response, indicates participation in the U.S. Food Stamp Program.

Usage

```
data(foodstamp, package="robustbase")
```

Format

A data frame with 150 observations on the following 4 variables.

`participation` participation in U.S. Food Stamp Program; yes = 1, no = 0

`tenancy` tenancy, indicating home ownership; yes = 1, no = 0

`suppl.income` supplemental income, indicating whether some form of supplemental security income is received; yes = 1, no = 0

`income` monthly income (in US dollars)

Source

Data description and first analysis: Stefanski et al.(1986) who indicate Rizek(1978) as original source of the larger study.

Electronic version from CRAN package **catdata**.

References

Rizek, R. L. (1978) The 1977-78 Nationwide Food Consumption Survey. *Family Econ. Rev.*, Fall, 3–7.

Stefanski, L. A., Carroll, R. J. and Ruppert, D. (1986) Optimally bounded score functions for generalized linear models with applications to logistic regression. *Biometrika* **73**, 413–424.

Künsch, H. R., Stefanski, L. A., Carroll, R. J. (1989) Conditionally unbiased bounded-influence estimation in general regression models, with applications to generalized linear models. *J. American Statistical Association* **84**, 460–466.

Examples

```
data(foodstamp)

(T123 <- xtabs(~ participation+ tenancy+ suppl.income, data=foodstamp))
summary(T123) ## ==> the binary var's are clearly not independent

foodSt <- within(foodstamp, {
   logInc <- log(1 + income)
   rm(income)
})

m1 <- glm(participation ~ ., family=binomial, data=foodSt)
summary(m1)
rm1 <- glmrob(participation ~ ., family=binomial, data=foodSt)
summary(rm1)
## Now use robust weights.on.x :
rm2 <- glmrob(participation ~ ., family=binomial, data=foodSt,
              weights.on.x = "robCov")
summary(rm2)## aha, now the weights are different:
which( weights(rm2, type="robust") < 0.5)
```

`fullRank` *Remove Columns (or Rows) From a Matrix to Make It Full Rank*

Description

From the QR decomposition with pivoting, (`qr(x, tol)` if $n \geq p$), if the matrix is not of full rank, the corresponding columns ($n \geq p$) or rows ($n < p$) are omitted to form a full rank matrix.

Usage

```
fullRank(x, tol = 1e-7, qrx = qr(x, tol=tol))
```

Arguments

`x`	a numeric matrix of dimension $n \times p$, or a similar object for which `qr()` works.
`tol`	tolerance for determining rank (deficiency). Currently is simply passed to `qr`.
`qrx`	optionally may be used to pass a `qr(x, ..)`; only used when `p <= n`.

Value

a version of the matrix `x`, with less columns or rows if `x`'s rank was smaller than `min(n,p)`.

If `x` is of full rank, it is returned unchanged.

Note

This is useful for robustness algorithms that rely on X matrices of full rank, e.g., `adjOutlyingness`.

This also works for numeric data frames and whenever `qr()` works correctly.

Author(s)

Martin Maechler

See Also

`qr`; for more sophisticated rank determination, `rankMatrix` from package **Matrix**.

Examples

```
stopifnot(identical(fullRank(wood), wood))

## More sophisticated and delicate
dim(T <- tcrossprod(data.matrix(toxicity))) # 38 x 38
dim(T. <- fullRank(T)) # 38 x 10
if(requireNamespace("Matrix")) {
  rMmeths <- eval(formals(Matrix::rankMatrix)$method)
  rT. <- sapply(rMmeths, function(.m.) Matrix::rankMatrix(T., method = .m.))
  print(rT.) # "qr" (= "qrLinpack"): 13,  others rather 10
}
```

```
dim(T.2 <- fullRank(T, tol = 1e-15))# 38 x 18
dim(T.3 <- fullRank(T, tol = 1e-12))# 38 x 13
dim(T.3 <- fullRank(T, tol = 1e-10))# 38 x 13
dim(T.3 <- fullRank(T, tol = 1e-8 ))# 38 x 12
dim(T.) # default from above          38 x 10
dim(T.3 <- fullRank(T, tol = 1e-5 ))# 38 x 10 -- still

plot(svd(T, 0,0)$d, log="y", main = "singular values of T", yaxt="n")
axis(2, at=10^(-14:5), las=1)
## pretty clearly indicates that  rank 10  is "correct" here.
```

`functionX-class` *Class "functionX" of Psi-like Vectorized Functions*

Description

The class `"functionX"` of vectorized functions of one argument `x` and typically further tuning parameters.

Objects from the Class

Objects can be created by calls of the form `new("functionX", ...)`.

Slots

`.Data`**:** Directly extends class `"function"`.

Extends

Class `"function"`, from data part. Class `"OptionalFunction"`, by class `"function"`. Class `"PossibleMethod"`, by class `"function"`.

Methods

No methods defined with class "functionX" in the signature.

Author(s)

Martin Maechler

See Also

`psiFunc()`, and class descriptions of `functionXal` for *functionals* of `"functionX"`, and `psi_func` which has several `functionX` slots.

functionXal-class *Class "functionXal" of Functionals (of Psi-like functions)*

Description

The class "functionXal" is a class of functionals (typically integrals) typically of functionX functions.

Since the functionX functions typically also depend on tuning parameters, objects of this class ("functionXal") are functions of these tuning parameters.

Slots

.Data: Directly extends class "function".

Extends

Class "function", from data part. Class "OptionalFunction", by class "function". Class "PossibleMethod", by class "function".

See Also

psiFunc() and the class definitions of functionX and psi_func which has several functionXal slots.

glmrob *Robust Fitting of Generalized Linear Models*

Description

glmrob is used to fit generalized linear models by robust methods. The models are specified by giving a symbolic description of the linear predictor and a description of the error distribution. Currently, robust methods are implemented for family = binomial, = poisson, = Gamma and = gaussian.

Usage

```
glmrob(formula, family, data, weights, subset, na.action,
       start = NULL, offset, method = c("Mqle", "BY", "WBY", "MT"),
       weights.on.x = c("none", "hat", "robCov", "covMcd"), control = NULL,
       model = TRUE, x = FALSE, y = TRUE, contrasts = NULL, trace.lev = 0, ...)
```

Arguments

formula	a formula, i.e., a symbolic description of the model to be fit (cf. glm or lm).
family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
data	an optional 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 glmrob is called.
weights	an optional vector of weights to be used in the fitting process.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting in options. The "factory-fresh" default is na.omit.
start	starting values for the parameters in the linear predictor. Note that specifying start has somewhat different meaning for the different methods. Notably, for "MT", this skips the expensive computation of initial estimates via sub samples, but needs to be *robust* itself.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting.
method	a character string specifying the robust fitting method. The details of method specification are given below.
weights.on.x	a character string (can be abbreviated), a function or list (see below), or a numeric vector of length n, specifying how points (potential outliers) in x-space are downweighted. If "hat", weights on the design of the form $\sqrt{1-h_{ii}}$ are used, where h_{ii} are the diagonal elements of the hat matrix. If "robCov", weights based on the robust Mahalanobis distance of the design matrix (intercept excluded) are used where the covariance matrix and the centre is estimated by cov.rob from the package **MASS**. Similarly, if "covMcd", robust weights are computed using covMcd. The default is "none". If weights.on.x is a function, it is called with arguments (X, intercept) and must return an n-vector of non-negative weights. If it is a list, it must be of length one, and as element contain a function much like covMcd() or cov.rob() (package **MASS**), which computes multivariate location and "scatter" of a data matrix X.
control	a list of parameters for controlling the fitting process. See the documentation for glmrobMqle.control for details.
model	a logical value indicating whether *model frame* should be included as a component of the returned value.
x, y	logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.

trace.lev	logical (or integer) indicating if intermediate results should be printed; defaults to 0 (the same as FALSE).
...	arguments passed to glmrobMqle.control when control is NULL (as per default).

Details

method="model.frame" returns the model.frame(), the same as glm().
method="Mqle" fits a generalized linear model using Mallows or Huber type robust estimators, as described in Cantoni and Ronchetti (2001) and Cantoni and Ronchetti (2006). In contrast to the implementation described in Cantoni (2004), the pure influence algorithm is implemented.
method="WBY" and method="BY", available for logistic regression (family = binomial) only, call BYlogreg(*, initwml= .) for the (weighted) Bianco-Yohai estimator, where initwml is true for "WBY", and false for "BY".
method="MT", currently only implemented for family = poisson, computes an "[M]-Estimator based on [T]ransformation", by Valdora and Yohai (2013), via (hidden internal) glmrobMT(); as that uses sample(), from R version 3.6.0 it depends on RNGkind(*, sample.kind). Exact reproducibility of results from R versions 3.5.3 and earlier, requires setting RNGversion("3.5.0").

weights.on.x= "robCov" makes sense if all explanatory variables are continuous.

In the cases,where weights.on.x is "covMcd" or "robCov", or list with a "robCov" function, the mahalanobis distances D^2 are computed with respect to the covariance (location and scatter) estimate, and the weights are 1/sqrt(1+ pmax.int(0, 8*(D2 - p)/sqrt(2*p))), where D2 = D^2 and p = ncol(X).

Value

glmrob returns an object of class "glmrob" and is also inheriting from glm.
The summary method, see summary.glmrob, can be used to obtain or print a summary of the results.
The generic accessor functions coefficients, effects, fitted.values and residuals (see residuals.glmrob) can be used to extract various useful features of the value returned by glmrob().

An object of class "glmrob" is a list with at least the following components:

coefficients	a named vector of coefficients
residuals	the *working* residuals, that is the (robustly "huberized") residuals in the final iteration of the IWLS fit.
fitted.values	the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
w.r	robustness weights for each observations; i.e., residuals $\times$ w.r equals the psi-function of the Preason's residuals.
w.x	weights used to down-weight observations based on the position of the observation in the design space.
dispersion	robust estimation of dispersion paramter if appropriate
cov	the estimated asymptotic covariance matrix of the estimated coefficients.
tcc	the tuning constant c in Huber's psi-function.
family	the family object used.

linear.predictors	the linear fit on link scale.
deviance	NULL; Exists because of compatipility reasons.
iter	the number of iterations used by the influence algorithm.
converged	logical. Was the IWLS algorithm judged to have converged?
call	the matched call.
formula	the formula supplied.
terms	the terms object used.
data	the data argument.
offset	the offset vector used.
control	the value of the control argument used.
method	the name of the robust fitter function used.
contrasts	(where relevant) the contrasts used.
xlevels	(where relevant) a record of the levels of the factors used in fitting.

Author(s)

Andreas Ruckstuhl ("Mqle") and Martin Maechler

References

Eva Cantoni and Elvezio Ronchetti (2001) Robust Inference for Generalized Linear Models. *JASA* **96** (455), 1022–1030.

Eva Cantoni (2004) Analysis of Robust Quasi-deviances for Generalized Linear Models. *Journal of Statistical Software*, **10**, https://www.jstatsoft.org/article/view/v010i04 Eva Cantoni and Elvezio Ronchetti (2006) A robust approach for skewed and heavy-tailed outcomes in the analysis of health care expenditures. *Journal of Health Economics* **25**, 198–213.

S. Heritier, E. Cantoni, S. Copt, M.-P. Victoria-Feser (2009) *Robust Methods in Biostatistics*. Wiley Series in Probability and Statistics.

Marina Valdora and Víctor J. Yohai (2013) Robust estimators for Generalized Linear Models. In progress.

See Also

predict.glmrob for prediction; glmrobMqle.control

Examples

```
## Binomial response --------------
data(carrots)

Cfit1 <- glm(cbind(success, total-success) ~ logdose + block,
             data = carrots, family = binomial)
summary(Cfit1)

Rfit1 <- glmrob(cbind(success, total-success) ~ logdose + block,
```

```
                family = binomial, data = carrots, method= "Mqle",
                control= glmrobMqle.control(tcc=1.2))
summary(Rfit1)

Rfit2 <- glmrob(success/total ~ logdose + block, weights = total,
                family = binomial, data = carrots, method= "Mqle",
                control= glmrobMqle.control(tcc=1.2))
coef(Rfit2)  ## The same as Rfit1


## Binary response --------------
data(vaso)

Vfit1 <- glm(Y ~ log(Volume) + log(Rate), family=binomial, data=vaso)
coef(Vfit1)

Vfit2 <- glmrob(Y ~ log(Volume) + log(Rate), family=binomial, data=vaso,
                method="Mqle", control = glmrobMqle.control(tcc=3.5))
coef(Vfit2) # c = 3.5 ==> not much different from classical
## Note the problems with  tcc <= 3 %% FIXME algorithm ???

Vfit3 <- glmrob(Y ~ log(Volume) + log(Rate), family=binomial, data=vaso,
                method= "BY")
coef(Vfit3)## note that results differ much.
## That's not unreasonable however, see Kuensch et al.(1989), p.465

## Poisson response --------------
data(epilepsy)

Efit1 <- glm(Ysum ~ Age10 + Base4*Trt, family=poisson, data=epilepsy)
summary(Efit1)

Efit2 <- glmrob(Ysum ~ Age10 + Base4*Trt, family = poisson,
                data = epilepsy, method= "Mqle",
                control = glmrobMqle.control(tcc= 1.2))
summary(Efit2)

## 'x' weighting:
(Efit3 <- glmrob(Ysum ~ Age10 + Base4*Trt, family = poisson,
           data = epilepsy, method= "Mqle", weights.on.x = "hat",
 control = glmrobMqle.control(tcc= 1.2)))

try( # gives singular cov matrix: 'Trt' is binary factor -->
     # affine equivariance and subsampling are problematic
Efit4 <- glmrob(Ysum ~ Age10 + Base4*Trt, family = poisson,
                data = epilepsy, method= "Mqle", weights.on.x = "covMcd",
                control = glmrobMqle.control(tcc=1.2, maxit=100))
)

##--> See  example(possumDiv)  for another  Poisson-regression


### -------- Gamma family -- data from example(glm) ---
```

```
clotting <- data.frame(
           u = c(5,10,15,20,30,40,60,80,100),
        lot1 = c(118,58,42,35,27,25,21,19,18),
        lot2 = c(69,35,26,21,18,16,13,12,12))
summary(cl <- glm   (lot1 ~ log(u), data=clotting, family=Gamma))
summary(ro <- glmrob(lot1 ~ log(u), data=clotting, family=Gamma))

clotM5.high <- within(clotting, { lot1[5] <- 60 })
op <- par(mfrow=2:1, mgp = c(1.6, 0.8, 0), mar = c(3,3:1))
plot( lot1  ~ log(u), data=clotM5.high)
plot(1/lot1 ~ log(u), data=clotM5.high)
par(op)
## Obviously, there the first observation is an outlier with respect to both
## representations!

cl5.high <- glm   (lot1 ~ log(u), data=clotM5.high, family=Gamma)
ro5.high <- glmrob(lot1 ~ log(u), data=clotM5.high, family=Gamma)
with(ro5.high, cbind(w.x, w.r))## the 5th obs. is downweighted heavily!

plot(1/lot1 ~ log(u), data=clotM5.high)
abline(cl5.high, lty=2, col="red")
abline(ro5.high, lwd=2, col="blue") ## result is ok (but not "perfect")
```

`glmrob..control` *Controlling Robust GLM Fitting by Different Methods*

Description

These are auxiliary functions as user interface for `glmrob` fitting when the different methods, `"Mqle"`, `"BY"`, or `"MT"` are used. Typically only used when calling `glmrob`.

Usage

```
glmrobMqle.control(acc = 1e-04, test.acc = "coef", maxit = 50, tcc = 1.345)
glmrobBY.control  (maxit = 1000, const = 0.5, maxhalf = 10)
glmrobMT.control  (cw = 2.1, nsubm = 500, acc = 1e-06, maxit = 200)
```

Arguments

`acc`	positive convergence tolerance; the iterations converge when ???
`test.acc`	Only "coef" is currently implemented
`maxit`	integer giving the maximum number of iterations.
`tcc`	tuning constant c for Huber's psi-function

const	for "BY", the normalizing constant ..
maxhalf	for "BY"; the number of halving steps when the gradient itself no longer improves. We have seen examples when increasing maxhalf was of relevance.
cw	tuning constant c for Tukey's biweight psi-function
nsubm	the number of subsamples to take for finding an initial estimate for method = "MT".

Value

A list with the arguments as components.

Author(s)

Andreas Ruckstuhl and Martin Maechler

See Also

glmrob

Examples

```
str(glmrobMqle.control())
str(glmrobBY.control())
str(glmrobMT.control())
```

---

h.alpha.n *Compute h, the subsample size for MCD and LTS*

Description

Compute h(alpha) which is the size of the subsamples to be used for MCD and LTS. Given $\alpha = alpha$, n and p, h is an *integer*, $h \approx \alpha n$, where the exact formula also depends on p.

For $\alpha = 1/2$, h == floor(n+p+1)/2; for the general case, it's simply n2 <- (n+p+1) %/% 2; floor(2*n2 - n + 2*(n-n2)*alpha).

Usage

```
h.alpha.n(alpha, n, p)
```

Arguments

alpha	fraction, numeric (vector) in [0.5, 1], see, e.g., covMcd.
n	integer (valued vector), the sample size.
p	integer (valued vector), the dimension.

Value

numeric vector of $h(\alpha, n, p)$; when any of the arguments of length greater than one, the usual R arithmetic (recycling) rules are used.

See Also

`covMcd` and `ltsReg` which are *defined* by $h = h(\alpha, n, p)$ and hence both use `h.alpha.n`.

Examples

```
n <- c(10:20,50,100)
p <- 5
## show the simple "alpha = 1/2" case:
cbind(n=n, h= h.alpha.n(1/2, n, p), n2p = floor((n+p+1)/2))

## alpha = 3/4 is recommended by some authors :
n <- c(15, 20, 25, 30, 50, 100)
cbind(n=n, h= h.alpha.n(3/4, n, p = 6))
```

---

hbk *Hawkins, Bradu, Kass's Artificial Data*

Description

Artificial Data Set generated by Hawkins, Bradu, and Kass (1984). The data set consists of 75 observations in four dimensions (one response and three explanatory variables). It provides a good example of the masking effect. The first 14 observations are outliers, created in two groups: 1–10 and 11–14. Only observations 12, 13 and 14 appear as outliers when using classical methods, but can be easily unmasked using robust distances computed by, e.g., MCD - covMcd().

Usage

```
data(hbk, package="robustbase")
```

Format

A data frame with 75 observations on 4 variables, where the last variable is the dependent one.

X1 x[,1]

X2 x[,2]

X3 x[,3]

Y y

Note

This data set is also available in package **wle** as `artificial`.

Source

Hawkins, D.M., Bradu, D., and Kass, G.V. (1984) Location of several outliers in multiple regression data using elemental sets. *Technometrics* **26**, 197–208.

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.94.

Examples

```
data(hbk)
plot(hbk)
summary(lm.hbk <- lm(Y ~ ., data = hbk))

hbk.x <- data.matrix(hbk[, 1:3])
(cHBK <- covMcd(hbk.x))
```

---

heart *Heart Catherization Data*

Description

This data set was analyzed by Weisberg (1980) and Chambers et al. (1983). A catheter is passed into a major vein or artery at the femoral region and moved into the heart. The proper length of the introduced catheter has to be guessed by the physician. The aim of the data set is to describe the relation between the catheter length and the patient's height (X1) and weight (X2).

This data sets is used to demonstrate the effects caused by collinearity. The correlation between height and weight is so high that either variable almost completely determines the other.

Usage

```
data(heart)
```

Format

A data frame with 12 observations on the following 3 variables.

`height` Patient's height in inches

`weight` Patient's weights in pounds

`clength` Y: Catheter Length (in centimeters)

Note

There are other `heart` datasets in other R packages, notably **survival**, hence considering using `package = "robustbase"`, see examples.

Source

Weisberg (1980)

Chambers et al. (1983)

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.103, table 13.

Examples

```
data(heart, package="robustbase")
heart.x <- data.matrix(heart[, 1:2]) # the X-variables
plot(heart.x)
covMcd(heart.x)
summary( lm.heart <-     lm(clength ~ . , data = heart))
summary(lts.heart <- ltsReg(clength ~ . , data = heart))
```

`huberize` *Huberization – Bringing Outliers In*

Description

Huberization (named after Peter Huber's M-estimation algorithm for location originally) replaces outlying values in a sample `x` by their respective boundary: when $x_j < c_1$ it is replaced by c_1 and when $x_j > c_2$ it is replaced by c_2. Consequently, values inside the interval $[c_1, c_2]$ remain unchanged.

Here, $c_j = M \pm c \cdot s$ where $s := s(x)$ is the *robust* scale estimate `Qn(x)` if that is positive, and by default, M is the robust huber estimate of location μ (with tuning constant k).

In the degenerate case where `Qn(x) == 0`, trimmed means of `abs(x - M)` are tried as scale estimate s, with decreasing trimming proportions specified by the decreasing `trim` vector.

Usage

```
huberize(x, M = huberM(x, k = k)$mu, c = k,
         trim = (5:1)/16,
         k = 1.5,
         warn0 = getOption("verbose"), saveTrim = TRUE)
```

Arguments

`x`	numeric vector which is to be huberized.
`M`	a number; defaulting to `huberM(x, k)`, the robust Huber M-estimator of location.
`c`	a positive number, the tuning constant for huberization of the sample `x`.
`trim`	a *decreasing* vector of trimming proportions in $[0, 0.5]$, only used to trim the absolute deviations from `M` in case `Qn(x)` is zero.

k	used if M is not specified as huberization center M, and so, by default is taken as Huber's M-estimate huberM(x, k).
warn0	logical indicating if a warning should be signalled in case Qn(x) is zero and the trimmed means for all trimming proportions trim are zero as well.
saveTrim	a logical indicating if the last tried trim[j] value should be stored if Qn(x) was zero.

Details

- In regular cases, s = Qn(x) is positive and used to huberize values of x outside [M - c*s, M + c*s].
- In degenerate cases where Qn(x) == 0, we search for an $s > 0$ by trying the trimmed mean s := mean(abs(x-M), trim = trim[j]) with less and less trimming (as the trimming proportions trim[] must decrease). If even the last, trim[length(trim)], leads to $s = 0$, a warning is printed when warn0 is true.

Value

a numeric vector as x; in case Qn(x) was zero and saveTrim is true, also containing the (last) trim proportion used (to compute the scale s) as attribute "trim" (see attr(), attributes).

Note

For the use in mc() and similar cases where mainly numerical stabilization is necessary, a large c = 1e12 will lead to *no* huberization, i.e., all y == x for y <- huberize(x, c) for typical non-degenerate samples.

Author(s)

Martin Maechler

See Also

huberM and mc which is now stabilized by default via something like huberize(*, c=1e11).

Examples

```
## For non-degenerate data and large c, nothing is huberized,
## as there are *no* really extreme outliers :
set.seed(101)
x <- rnorm(1000)
stopifnot(all.equal(x, huberize(x, c=100)))
## OTOH, the "extremes" are shrunken towards the boundaries for smaller c:
xh <- huberize(x, c = 2)
table(x != xh)
## 45 out of a 1000:
table(xh[x != xh])# 26 on the left boundary -2.098 and 19 on the right = 2.081
## vizualization:
stripchart(x); text(0,1, "x {original}", pos=3); yh <- 0.9
stripchart(xh, at = yh, add=TRUE, col=2)
```

```
text(0, yh, "huberize(x, c=2)",   col=2, pos=1)
arrows( x[x!=xh], 1,
       xh[x!=xh], yh, length=1/8, col=adjustcolor("pink", 1/2))
```

---

huberM *Safe (generalized) Huber M-Estimator of Location*

Description

(Generalized) Huber M-estimator of location with MAD scale, being sensible also when the scale is zero where huber() returns an error.

Usage

```
huberM(x, k = 1.5, weights = NULL, tol = 1e-06,
       mu = if(is.null(weights)) median(x) else wgt.himedian(x, weights),
       s =  if(is.null(weights)) mad(x, center=mu)
    else wgt.himedian(abs(x - mu), weights),
       se = FALSE,
       warn0scale = getOption("verbose"))
```

Arguments

x	numeric vector.
k	positive factor; the algorithm winsorizes at k standard deviations.
weights	numeric vector of non-negative weights of same length as x, or NULL.
tol	convergence tolerance.
mu	initial location estimator.
s	scale estimator held constant through the iterations.
se	logical indicating if the standard error should be computed and returned (as SE component). Currently only available when weights is NULL.
warn0scale	logical; if true, and s is 0 and length(x) > 1, this will be warned about.

Details

Note that currently, when non-NULL weights are specified, the default for initial location mu and scale s is wgt.himedian, where strictly speaking a weighted "non-hi" median should be used for consistency. Since s is not updated, the results slightly differ, see the examples below.

When se = TRUE, the standard error is computed using the τ correction factor but no finite sample correction.

Value

list of location and scale parameters, and number of iterations used.

mu	location estimate
s	the s argument, typically the mad.
it	the number of "Huber iterations" used.

Author(s)

Martin Maechler, building on the MASS code mentioned.

References

Huber, P. J. (1981) *Robust Statistics.* Wiley.

See Also

`hubers` (and `huber`) in package **MASS**; `mad`.

Examples

```
huberM(c(1:9, 1000))
mad   (c(1:9, 1000))
mad   (rep(9, 100))
huberM(rep(9, 100))

## When you have "binned" aka replicated observations:
set.seed(7)
x <- c(round(rnorm(1000),1), round(rnorm(50, m=10, sd = 10)))
t.x <- table(x) # -> unique values and multiplicities
x.uniq <- as.numeric(names(t.x)) ## == sort(unique(x))
x.mult <- unname(t.x)
str(Hx  <- huberM(x.uniq, weights = x.mult), digits = 7)
str(Hx. <- huberM(x, s = Hx$s, se=TRUE), digits = 7) ## should be ~= Hx
stopifnot(all.equal(Hx[-4], Hx.[-4]))
str(Hx2 <- huberM(x, se=TRUE), digits = 7)## somewhat different, since 's' differs

## Confirm correctness of std.error :

system.time(
SS <- replicate(10000, vapply(huberM(rnorm(400), se=TRUE), as.double, 1.))
) # ~ 2.8 seconds (was 12.2 s)
rbind(mean(SS["SE",]), sd(SS["mu",]))# both ~ 0.0508
stopifnot(all.equal(mean(SS["SE",]),
                    sd ( SS["mu",]), tolerance= 0.002))
```

---

kootenay *Waterflow Measurements of Kootenay River in Libby and Newgate*

Description

The original data set is the waterflow in January of the Kootenay river, measured at two locations, namely, Libby (Montana) and Newgate (British Columbia) for 13 consecutive years, 1931–1943.

The data set is of mostly interest because it has been used as example in innumerous didactical situations about robust regression. To this end, one number (in observation 4) has been modified from the original data from originally 44.9 to 15.7 (here).

Usage

```
data(kootenay, package="robustbase")
```

Format

A data frame with 13 observations on the following 2 variables.

`Libby` a numeric vector

`Newgate` a numeric vector

Details

The original (unmodified) version of the data is easily obtainable as `kootenay0` from the examples; other modified versions of the data sets are also used in different places, see the examples below.

Source

Original Data, p.58f of Ezekiel and Fox (1959), *Methods of Correlation and Regression Analysis*. Wiley, N.Y.

References

Hampel, F., Ronchetti, E., Rousseeuw, P. and Stahel, W. (1986) *Robust Statistics: The Approach Based on Influence Functions*; Wiley, N.Y.

Rousseeuw, P. J. and Leroy, A. M. (1987) *Robust Regression & Outlier Detection*, Wiley, N. Y.

Examples

```
data(kootenay)
plot(kootenay, main = "'kootenay' data")
points(kootenay[4,], col = 2, cex =2, pch = 3)

abline(lm   (Newgate ~ Libby, data = kootenay), col = "pink")
abline(lmrob(Newgate ~ Libby, data = kootenay), col = "blue")

## The original version of Ezekiel & Fox:
kootenay0 <- kootenay
kootenay0[4, "Newgate"] <- 44.9
plot(kootenay0, main = "'kootenay0': the original data")
abline(lm   (Newgate ~ Libby, data = kootenay0), col = "pink")
abline(lmrob(Newgate ~ Libby, data = kootenay0), col = "blue")

## The version with "milder" outlier -- Hampel et al., p.310
kootenay2 <- kootenay0
kootenay2[4, "Libby"] <- 20.0 # instead of 77.6
plot(kootenay2, main = "The 'kootenay2' data",
     xlim = range(kootenay[,"Libby"]))
points(kootenay2[4,], col = 2, cex =2, pch = 3)
abline(lm   (Newgate ~ Libby, data = kootenay2), col = "pink")
abline(lmrob(Newgate ~ Libby, data = kootenay2), col = "blue")
```

lactic *Lactic Acid Concentration Measurement Data*

Description

Data on the Calibration of an Instrument that Measures Lactic Acid Concentration in Blood, from Afifi and Azen (1979) - comparing the true concentration X with the measured value Y.

Usage

```
data(lactic, package="robustbase")
```

Format

A data frame with 20 observations on the following 2 variables.

`X` True Concentration

`Y` Instrument

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.62, table 10.

Examples

```
data(lactic)
summary(lm.lactic <- lm(Y ~., data=lactic))
```

lmc *Left and Right Medcouple, Robust Measures of Tail Weight*

Description

Compute the left and right 'medcouple', *robust* estimators of tail weight, in some sense robust versions of the kurtosis, the very unrobust centralized 4th moment.

Usage

```
lmc(x, mx = median(x, na.rm=na.rm), na.rm = FALSE, doReflect = FALSE, ...)
rmc(x, mx = median(x, na.rm=na.rm), na.rm = FALSE, doReflect = FALSE, ...)
```

Arguments

x	a numeric vector

mx	number, the "center" of x wrt which the left and right parts of x are defined:
	`lmc(x, mx, *) :=  mc(x[x <= mx], *)`
	`rmc(x, mx, *) :=  mc(x[x >= mx], *)`

na.rm	logical indicating how missing values (NAs) should be dealt with.
doReflect	logical indicating if mc should also be computed on the *reflected* sample -x. Setting doReflect=TRUE makes sense for mathematical strictness reasons, as the internal MC computes the himedian() which can differ slightly from the median. Note that mc()'s own default is true iff length(x) <= 100.
...	further arguments to mc(), see its help page.

Value

each a number (unless ... contains full.result = TRUE).

References

Brys, G., Hubert, M. and Struyf, A. (2006). Robust measures of tail weight, *Computational Statistics and Data Analysis* **50(3)**, 733–759.

and those in 'References' of mc.

Examples

```
mc(1:5)  # 0 for a symmetric sample
lmc(1:5) # 0
rmc(1:5) # 0

x1 <- c(1, 2, 7, 9, 10)
mc(x1) # = -1/3
c( lmc( x1),  lmc( x1, doReflect=TRUE))#   0  -1/3
c( rmc( x1),  rmc( x1, doReflect=TRUE))# -1/3 -1/6
c(-rmc(-x1), -rmc(-x1, doReflect=TRUE)) # 2/3  1/3

data(cushny)
lmc(cushny) # 0.2
rmc(cushny) # 0.45

isSym_LRmc <- function(x, tol = 1e-14)
    all.equal(lmc(-x, doReflect=TRUE),
              rmc( x, doReflect=TRUE), tolerance = tol)

sym <- c(-20, -5, -2:2, 5, 20)
stopifnot(exprs = {
    lmc(sym) == 0.5
    rmc(sym) == 0.5
    isSym_LRmc(cushny)
    isSym_LRmc(x1)
```

```
})

## Susceptibility to large outliers:
## "Sensitivity Curve" := empirical influence function
dX10 <- function(X) c(1:5,7,10,15,25, X) # generate skewed size-10 with 'X'
x <- c(26:40, 45, 50, 60, 75, 100)
(lmc10N <- vapply(x, function(X) lmc(dX10(X)), 1))
(rmc10N <- vapply(x, function(X) rmc(dX10(X)), 1))
cols <- adjustcolor(2:3, 3/4)

plot(x, lmc10N, type="o", cex=1/2, main = "lmc & rmc( c(1:5,7,10,15,25, X) )",
     xlab=quote(X), log="x", col=cols[1])
lines(x, rmc10N, col=cols[2], lwd=3)
legend("top", paste0(c("lmc", "rmc"), "(X)"), col=cols, lty=1, lwd=c(1,3), pch = c(1, NA), bty="n")

n <- length(x)
stopifnot(exprs = {
    all.equal(current = lmc10N, target = rep(0, n))
    all.equal(current = rmc10N, target = c(3/19, 1/5, 5/21, 3/11, 7/23, rep(1/3, n-5)))
    ## and it stays stable with outlier  X --> oo :
    lmc(dX10(1e300)) == 0
    rmc(dX10(1e300)) == rmc10N[6]
})
```

`lmrob` *MM-type Estimators for Linear Regression*

Description

Computes fast MM-type estimators for linear (regression) models.

Usage

```
lmrob(formula, data, subset, weights, na.action, method = "MM",
      model = TRUE, x = !control$compute.rd, y = FALSE,
      singular.ok = TRUE, contrasts = NULL, offset = NULL,
      control = NULL, init = NULL, ...)
```

Arguments

`formula`	a symbolic description of the model to be fit. See `lm` and `formula` for more details.
`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 `lmrob` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.

weights	an optional vector of weights to be used in the fitting process (in addition to the robustness weights computed in the fitting process).
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The "factory-fresh" default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
method	string specifying the estimator-chain. MM is interpreted as SM. See *Details*, notably the currently recommended setting = "KS2014".
model, x, y	logicals. If TRUE the corresponding components of the fit (the model frame, the model matrix, the response) are returned.
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. An offset term can be included in the formula instead or as well, and if both are specified their sum is used.
control	a list specifying control parameters; use the function lmrob.control(.) and see its help page.
init	an optional argument to specify or supply the initial estimate. See *Details*.
...	additional arguments can be used to specify control parameters directly instead of (but not in addition to!) via control.

Details

Overview: This function computes an MM-type regression estimator as described in Yohai (1987) and Koller and Stahel (2011). By default it uses a bi-square redescending score function, and it returns a highly robust and highly efficient estimator (with 50% breakdown point and 95% asymptotic efficiency for normal errors). The computation is carried out by a call to lmrob.fit().

The argument setting of lmrob.control is provided to set alternative defaults as suggested in Koller and Stahel (2011) (setting="KS2011"; now do use its extension setting="KS2014"). For further details, see lmrob.control.

Initial Estimator init**:** The initial estimator may be specified using the argument init. This can either be

- a *string* used to specify built in internal estimators (currently "S" and "M-S", see *See also* below);
- a function taking arguments x, y, control, mf (where mf stands for model.frame) and returning a list containing at least the initial coefficients as component "coefficients" and the initial scale estimate as "scale".
- Or a list giving the initial coefficients and scale as components "coefficients" and "scale". See also *Examples*.

Note that when init is a function or list, the method argument must *not* contain the initial estimator, e.g., use MDM instead of SMDM.

The default, equivalent to init = "S", uses as initial estimator an S-estimator (Rousseeuw and Yohai, 1984) which is computed using the Fast-S algorithm of Salibian-Barrera and Yohai

(2006), calling `lmrob.S()`. That function, since March 2012, by default uses *nonsingular* subsampling which makes the Fast-S algorithm feasible for categorical data as well, see Koller (2012). Note that convergence problems may still show up as warnings, e.g.,

```
S refinements did not converge (to refine.tol=1e-07) in 200 (= k.max) steps
```

and often can simply be remedied by increasing (i.e. weakening) `refine.tol` or increasing the allowed number of iterations `k.max`, see `lmrob.control`.

Method `method`**:** The following chain of estimates is customizable via the `method` argument. There are currently two types of estimates available,

`"M"`**:** corresponds to the standard M-regression estimate.

`"D"`**:** stands for the Design Adaptive Scale estimate as proposed in Koller and Stahel (2011).

The `method` argument takes a string that specifies the estimates to be calculated as a chain. Setting `method='SMDM'` will result in an intial S-estimate, followed by an M-estimate, a Design Adaptive Scale estimate and a final M-step. For methods involving a `D`-step, the default value of `psi` (see `lmrob.control`) is changed to `"lqq"`.

By default, standard errors are computed using the formulas of Croux, Dhaene and Hoorelbeke (2003) (`lmrob.control` option `cov=".vcov.avar1"`). This method, however, works only for MM-estimates (i.e., `method = "MM"` or `= "SM"`). For other `method` arguments, the covariance matrix estimate used is based on the asymptotic normality of the estimated coefficients (`cov=".vcov.w"`) as described in Koller and Stahel (2011). The var-cov computation can be skipped by `cov = "none"` and (re)done later by e.g., `vcov(<obj>, cov = ".vcov.w")`.

As of robustbase version 0.91-0 (April 2014), the computation of robust standard errors for `method="SMDM"` has been changed. The old behaviour can be restored by setting the control parameter `cov.corrfact = "tauold"`.

Value

An object of class `lmrob`; a list including the following components:

`coefficients`	The estimate of the coefficient vector
`scale`	The scale as used in the M estimator.
`residuals`	Residuals associated with the estimator.
`converged`	`TRUE` if the IRWLS iterations have converged.
`iter`	number of IRWLS iterations
`rweights`	the "robustness weights" $\psi(r_i/S)/(r_i/S)$.
`fitted.values`	Fitted values associated with the estimator.
`init.S`	The `list` returned by `lmrob.S()` or `lmrob.M.S()` (for MM-estimates, i.e., `method="MM"` or `"SM"` only)
`init`	A similar list that contains the results of intermediate estimates (*not* for MM-estimates).
`rank`	the numeric rank of the fitted linear model.
`cov`	The estimated covariance matrix of the regression coefficients
`df.residual`	the residual degrees of freedom.
`weights`	the specified weights (missing if none were used).

na.action	(where relevant) information returned by model.frame on the special handling of NAs.
offset	the offset used (missing if none were used).
contrasts	(only where relevant) the contrasts used.
xlevels	(only where relevant) a record of the levels of the factors used in fitting.
call	the matched call.
terms	the terms object used.
model	if requested (the default), the model frame used.
x	if requested, the model matrix used.
y	if requested, the response used.

In addition, non-null fits will have components assign, and qr relating to the linear fit, for use by extractor functions such as summary.

Author(s)

(mainly:) Matias Salibian-Barrera and Manuel Koller

References

Croux, C., Dhaene, G. and Hoorelbeke, D. (2003) *Robust standard errors for robust estimators*, Discussion Papers Series 03.16, K.U. Leuven, CES.

Koller, M. (2012) Nonsingular subsampling for S-estimators with categorical predictors, *ArXiv e-prints* https://arxiv.org/abs/1208.5595; extended version published as Koller and Stahel (2017), see lmrob.control.

Koller, M. and Stahel, W.A. (2011) Sharpening Wald-type inference in robust regression for small samples. *Computational Statistics & Data Analysis* **55**(8), 2504–2515.

Maronna, R. A., and Yohai, V. J. (2000) Robust regression with both continuous and categorical predictors. *Journal of Statistical Planning and Inference* **89**, 197–214.

Rousseeuw, P.J. and Yohai, V.J. (1984) Robust regression by means of S-estimators, In *Robust and Nonlinear Time Series*, J. Franke, W. Härdle and R. D. Martin (eds.). Lectures Notes in Statistics 26, 256–272, Springer Verlag, New York.

Salibian-Barrera, M. and Yohai, V.J. (2006) A fast algorithm for S-regression estimates, *Journal of Computational and Graphical Statistics* **15**(2), 414–427. doi:10.1198/106186006X113629

Yohai, V.J. (1987) High breakdown-point and high efficiency estimates for regression. *The Annals of Statistics* **15**, 642–65.

Yohai, V., Stahel, W.~A. and Zamar, R. (1991) A procedure for robust estimation and inference in linear regression; in Stahel and Weisberg (eds), *Directions in Robust Statistics and Diagnostics*, Part II, Springer, New York, 365–374; doi:10.1007/9781461244448_20.

See Also

lmrob.control; for the algorithms lmrob.S, lmrob.M.S and lmrob.fit; and for methods, summary.lmrob, for the extra "statistics", notably R^2 ("R squared"); predict.lmrob, print.lmrob, plot.lmrob, and weights.lmrob.

Examples

```
data(coleman)
set.seed(0)
## Default for a very long time:
summary( m1 <- lmrob(Y ~ ., data=coleman) )

## Nowadays **strongly recommended** for routine use:
summary(m2 <- lmrob(Y ~ ., data=coleman, setting = "KS2014") )
##                                          ------------------

plot(residuals(m2) ~ weights(m2, type="robustness")) ##-> weights.lmrob()
abline(h=0, lty=3)

data(starsCYG, package = "robustbase")
## Plot simple data and fitted lines
plot(starsCYG)
  lmST <-    lm(log.light ~ log.Te, data = starsCYG)
(RlmST <- lmrob(log.light ~ log.Te, data = starsCYG))
abline(lmST, col = "red")
abline(RlmST, col = "blue")
## --> Least Sq.:/ negative slope  \ robust: slope ~= 2.2 % checked in ../tests/lmrob-data.R
summary(RlmST) # -> 4 outliers; rest perfect
vcov(RlmST)
stopifnot(all.equal(fitted(RlmST),
                    predict(RlmST, newdata = starsCYG), tol = 1e-14))
## FIXME: setting = "KS2011"  or  setting = "KS2014"  **FAIL** here

##--- 'init' argument ----------------------------------
## 1)  string
set.seed(0)
m3 <- lmrob(Y ~ ., data=coleman, init = "S")
stopifnot(all.equal(m1[-18], m3[-18]))
## 2) function
initFun <- function(x, y, control, ...) { # no 'mf' needed
    init.S <- lmrob.S(x, y, control)
    list(coefficients=init.S$coef, scale = init.S$scale)
}
set.seed(0)
m4 <- lmrob(Y ~ ., data=coleman, method = "M", init = initFun)
## list
m5 <- lmrob(Y ~ ., data=coleman, method = "M",
            init = list(coefficients = m3$init$coef, scale = m3$scale))
stopifnot(all.equal(m4[-17], m5[-17]))
```

`lmrob..D..fit` *Compute Design Adaptive Scale estimate*

Description

This function calculates a Design Adaptive Scale estimate for a given MM-estimate. This is supposed to be a part of a chain of estimates like `SMD` or `SMDM`.

Usage

```
lmrob..D..fit(obj, x=obj$x, control = obj$control,
              mf,
              method = obj$control$method)
```

Arguments

obj	lmrob-object based on which the estimate is to be calculated.
x	the design matrix; if missing, the method tries to get it from obj$x and if this fails from obj$model.
control	list of control parameters, as returned by lmrob.control.
mf	defunct.
method	optional; the method used for *obj* computation.

Details

This function is used by lmrob.fit and typically not to be used on its own. Note that lmrob.fit() specifies control potentially differently than the default, but does use the default for method.

Value

The given lmrob-object with the following elements updated:

scale	The Design Adaptive Scale estimate
converged	TRUE if the scale calculation converged, FALSE other.

Author(s)

Manuel Koller

References

Koller, M. and Stahel, W.A. (2011), Sharpening Wald-type inference in robust regression for small samples, *Computational Statistics & Data Analysis* **55**(8), 2504–2515.

See Also

lmrob.fit, lmrob

Examples

```
data(stackloss)
## Compute manual SMD-estimate:
## 1) MM-estimate
m1 <- lmrob(stack.loss ~ ., data = stackloss)
## 2) Add Design Adaptive Scale estimate
m2 <- lmrob..D..fit(m1)
print(c(m1$scale, m2$scale))
```

```
summary(m1)
summary(m2) ## the covariance matrix estimate is also updated
```

`lmrob..M..fit` *Compute M-estimators of regression*

Description

This function performs RWLS iterations to find an M-estimator of regression. When started from an S-estimated `beta.initial`, this results in an MM-estimator.

Usage

```
lmrob..M..fit(x = obj$x, y = obj$y,
              beta.initial = obj$coef, scale = obj$scale, control = obj$control,
              obj,
              mf,
              method = obj$control$method)
```

Arguments

`x`	design matrix ($n \times p$) typically including a column of 1s for the intercept.
`y`	numeric response vector (of length n).
`beta.initial`	numeric vector (of length p) of initial estimate. Usually the result of an S-regression estimator.
`scale`	robust residual scale estimate. Usually an S-scale estimator.
`control`	list of control parameters, as returned by `lmrob.control`. Currently, the components `c("max.it", "rel.tol","trace.lev", "psi", "tuning.psi", "mts", "subsampling")` are accessed.
`obj`	an optional `lmrob`-object. If specified, this is typically used to set values for the other arguments.
`mf`	defunct.
`method`	optional; the `method` used for *obj* computation.

Details

This function is used by `lmrob.fit` (and `anova(<lmrob>, type = "Deviance")`) and typically not to be used on its own.

Value

A list with the following elements:

`coef`	the M-estimator (or MM-estim.) of regression
`control`	the `control` list input used
`scale`	The residual scale estimate
`seed`	The random number generator seed
`converged`	`TRUE` if the RWLS iterations converged, `FALSE` otherwise

Author(s)

Matias Salibian-Barrera and Martin Maechler

References

Yohai, 1987

See Also

`lmrob.fit`, `lmrob`; `rlm` from package **MASS**.

Examples

```
data(stackloss)
X <- model.matrix(stack.loss ~ . , data = stackloss)
y <- stack.loss
## Compute manual MM-estimate:
## 1) initial LTS:
m0 <- ltsReg(X[,-1], y)
## 2) M-estimate started from LTS:
m1 <- lmrob..M..fit(X, y, beta.initial = coef(m0), scale = m0$scale, method = "SM",
                    control = lmrob.control(tuning.psi = 1.6, psi = 'bisquare'))
## no 'method' (nor 'obj'):
m1. <- lmrob..M..fit(X, y, beta.initial = coef(m0), scale = m0$scale,
                     control = m1$control)
stopifnot(all.equal(m1, m1., tol = 1e-15)) # identical {call *not* stored!}

cbind(m0$coef, m1$coef)
## the scale is kept fixed:
stopifnot(identical(unname(m0$scale), m1$scale))

##  robustness weights: are
r.s <- with(m1, residuals/scale) # scaled residuals
m1.wts <- Mpsi(r.s, cc = 1.6, psi="tukey") / r.s
summarizeRobWeights(m1.wts)
##--> outliers 1,3,4,13,21
which(m0$lts.wt == 0) # 1,3,4,21 but not 13

## Manually add M-step to SMD-estimate (=> equivalent to "SMDM"):
m2 <- lmrob(stack.loss ~ ., data = stackloss, method = 'SMD')
m3 <- lmrob..M..fit(obj = m2)

## Simple function that allows custom initial estimates
## (Deprecated; use init argument to lmrob() instead.) %% MM: why deprecated?
lmrob.custom <- function(x, y, beta.initial, scale, terms) {
  ## initialize object
  obj <- list(control = lmrob.control("KS2011"),
              terms = terms) ## terms is needed for summary()
  ## M-step
  obj <- lmrob..M..fit(x, y, beta.initial, scale, obj = obj)
  ## D-step
  obj <- lmrob..D..fit(obj, x)
```

```
  ## Add some missing elements
  obj$cov <- TRUE ## enables calculation of cov matrix
  obj$p <- obj$qr$rank
  obj$degree.freedom <- length(y) - obj$p
  ## M-step
  obj <- lmrob..M..fit(x, y, obj=obj)
  obj$control$method <- ".MDM"
  obj
}

m4 <- lmrob.custom(X, y, m2$init$init.S$coef,
                   m2$init$scale, m2$terms)
stopifnot(all.equal(m4$coef, m3$coef))

## Start from ltsReg:
m5 <- ltsReg(stack.loss ~ ., data = stackloss)
m6 <- lmrob.custom(m5$X, m5$Y, coef(m5), m5$scale, m5$terms)
```

`lmrob.control` *Tuning Parameters for lmrob() and Auxiliaries*

Description

Tuning parameters for `lmrob`, the MM-type regression estimator and the associated S-, M- and D-estimators. Using `setting="KS2011"` sets the defaults as suggested by Koller and Stahel (2011) and analogously for `"KS2014"`.

The `.M*.default` `function`s and `.M*.defaults` `list`s contain default tuning parameters for all the predefined ψ functions, see also `Mpsi`, etc.

Usage

```
lmrob.control(setting, seed = NULL, nResample = 500,
              tuning.chi = NULL, bb = 0.5, tuning.psi = NULL,
              max.it = 50, groups = 5, n.group = 400,
              k.fast.s = 1, best.r.s = 2,
              k.max = 200, maxit.scale = 200, k.m_s = 20,
          refine.tol = 1e-7, rel.tol = 1e-7, scale.tol = 1e-10, solve.tol = 1e-7,
              zero.tol = 1e-10,
              trace.lev = 0,
              mts = 1000, subsampling = c("nonsingular", "simple"),
              compute.rd = FALSE, method = "MM", psi = "bisquare",
              numpoints = 10, cov = NULL,
              split.type = c("f", "fi", "fii"), fast.s.large.n = 2000,
              # only for outlierStats() :
              eps.outlier = function(nobs) 0.1 / nobs,
              eps.x = function(maxx) .Machine$double.eps^(.75)*maxx,
              compute.outlier.stats = method,
              warn.limit.reject = 0.5,
```

```
              warn.limit.meanrw = 0.5, ...)

## S3 method for class 'lmrobCtrl'
update(object, ...)

.Mchi.tuning.defaults
.Mchi.tuning.default(psi)
.Mpsi.tuning.defaults
.Mpsi.tuning.default(psi)
```

Arguments

`setting`	a string specifying alternative default values. Leave empty for the defaults or use `"KS2011"` or `"KS2014"` for the defaults suggested by Koller and Stahel (2011, 2017). See *Details*.
`seed`	`NULL` or an integer vector compatible with `.Random.seed`: the seed to be used for random re-sampling used in obtaining candidates for the initial S-estimator. The current value of `.Random.seed` will be preserved if `seed` is set, i.e. non-`NULL`; otherwise, as by default, `.Random.seed` will be used and modified as usual from calls to `runif()` etc.
`nResample`	number of re-sampling candidates to be used to find the initial S-estimator. Currently defaults to 500 which works well in most situations (see references).
`tuning.chi`	tuning constant vector for the S-estimator. If `NULL`, as by default, sensible defaults are set (depending on `psi`) to yield a 50% breakdown estimator. See *Details*.
`bb`	expected value under the normal model of the "chi" (rather $\rho(rho)$) function with tuning constant equal to `tuning.chi`. This is used to compute the S-estimator.
`tuning.psi`	tuning constant vector for the redescending M-estimator. If `NULL`, as by default, this is set (depending on `psi`) to yield an estimator with asymptotic efficiency of 95% for normal errors. See *Details*.
`max.it`	integer specifying the maximum number of IRWLS iterations.
`groups`	(for the fast-S algorithm): Number of random subsets to use when the data set is large.
`n.group`	(for the fast-S algorithm): Size of each of the `groups` above. Note that this must be at least p.
`k.fast.s`	(for the fast-S algorithm): Number of local improvement steps ("*I-steps*") for each re-sampling candidate.
`k.m_s`	(for the M-S algorithm): specifies after how many unsuccessful refinement steps the algorithm stops.
`best.r.s`	(for the fast-S algorithm): Number of of best candidates to be iterated further (i.e., "***r**efined*"); is denoted t in Salibian-Barrera & Yohai(2006).
`k.max`	(for the fast-S algorithm): maximal number of refinement steps for the "fully" iterated best candidates.

`maxit.scale` integer specifying the maximum number of C level `find_scale()` iterations (in fast-S and M-S algorithms).

`refine.tol` (for the fast-S algorithm): relative convergence tolerance for the fully iterated best candidates.

`rel.tol` (for the RWLS iterations of the MM algorithm): relative convergence tolerance for the parameter vector.

`scale.tol` (for the scale estimation iterations of the S algorithm): relative convergence tolerance for the `scale` $\sigma(.)$.

`solve.tol` (for the S algorithm): relative tolerance for inversion. Hence, this corresponds to `solve.default()`'s `tol`.

`zero.tol` for checking 0-residuals in the S algorithm, non-negative number ϵ_z such that $\{i; \left|\tilde{R}_i\right| \le \epsilon_z\}$ correspond to 0-residuals, where $\tilde{R}_i$ are standardized residuals, $\tilde{R}_i = R_i/s_y$ and $s_y = \frac{1}{n}\sum_{i=1}^{n}|y_i|$.

`trace.lev` integer indicating if the progress of the MM-algorithm and the fast-S algorithms, see `lmrob.S`, should be traced (increasingly); default `trace.lev = 0` does no tracing.

`mts` maximum number of samples to try in subsampling algorithm.

`subsampling` type of subsampling to be used, a string: `"simple"` for simple subsampling (default prior to version 0.9), `"nonsingular"` for nonsingular subsampling. See also `lmrob.S`.

`compute.rd` logical indicating if robust distances (based on the MCD robust covariance estimator `covMcd`) are to be computed for the robust diagnostic plots. This may take some time to finish, particularly for large data sets, and can lead to singularity problems when there are `factor` explanatory variables (with many levels, or levels with "few" observations). Hence, is `FALSE` by default.

`method` string specifying the estimator-chain. `MM` is interpreted as `SM`. See *Details* of `lmrob` for a description of the possible values.

`psi` string specifying the type ψ-function used. See *Details* of `lmrob`. Defaults to `"bisquare"` for S and MM-estimates, otherwise `"lqq"`.

`numpoints` number of points used in Gauss quadrature.

`cov` function or string with function name to be used to calculate covariance matrix estimate. The default is `if(method %in% c('SM', 'MM')) ".vcov.avar1" else ".vcov.w"`. See *Details* of `lmrob`.

`split.type` determines how categorical and continuous variables are split. See `splitFrame`.

`fast.s.large.n` minimum number of observations required to switch from ordinary "fast S" algorithm to an efficient "large n" strategy.

`eps.outlier` limit on the robustness weight below which an observation is considered to be an outlier. Either a `numeric(1)` or a function that takes the number of observations as an argument. Used only in `summary.lmrob` and `outlierStats`.

`eps.x` limit on the absolute value of the elements of the design matrix below which an element is considered zero. Either a `numeric(1)` or a function that takes the maximum absolute value in the design matrix as an argument.

`compute.outlier.stats`
: vector of `character` strings, each valid to be used as `method` argument. Used to specify for which estimators outlier statistics (and warnings) should be produced. Set to empty (`NULL` or `character(0)`) if none are required.
Note that the default is `method` which by default is either `"MM"`, `"SM"`, or `"SMDM"`; hence using `compute.outlier.stats = "S"` provides `outlierStats()` to a `lmrob.S()` result.

`warn.limit.reject`
: limit of ratio #rejected/#obs in level above ($\geq$) which a warning is produced. Set to `NULL` to disable warning.

`warn.limit.meanrw`
: limit of the mean robustness per factor level below which ($\leq$) a warning is produced. Set to `NULL` to disable warning.

`object`
: an `"lmrobCtrl"` object, as resulting from a `lmrob.control(*)` or an `update(<lmrobCtrl>, *)` call.

`...`
: for

  `lmrob.control()`**:** further arguments to be added as `list` components to the result, e.g., those to be used in `.vcov.w()`.

  `update(object, *)`**:** (named) components from `object`, to be *modified*, **not** `setting = *`.

Details

The option `setting="KS2011"` alters the default arguments. They are changed to `method = "SMDM"`, `psi = "lqq"`, `max.it = 500`, `k.max = 2000`, `cov = ".vcov.w"`. The defaults of all the remaining arguments are not changed.

The option `setting="KS2014"` builds upon `setting="KS2011"`. More arguments are changed to `best.r.s = 20`, `k.fast.s = 2`, `nResample = 1000`. This setting should produce more stable estimates for designs with `factor`s.

By default, and in `.Mpsi.tuning.default()` and `.Mchi.tuning.default()`, `tuning.chi` and `tuning.psi` are set to yield an MM-estimate with breakdown point 0.5 and efficiency of 95% at the normal.

If numeric `tuning.chi` or `tuning.psi` are specified, say `cc`, for `psi = "ggw"` or `"lqq"`, `.psi.const(cc, psi)` is used, see its help page.

To get the defaults, e.g., `.Mpsi.tuning.default(psi)` is equivalent to but more efficient than the formerly widely used `lmrob.control(psi = psi)$tuning.psi`.

These defaults are:

```
          psi   tuning.chi                tuning.psi
    bisquare   1.54764                   4.685061
       welsh   0.5773502                 2.11
         ggw   c(-0.5, 1.5, NA, 0.5)     c(-0.5, 1.5, 0.95, NA)
         lqq   c(-0.5, 1.5, NA, 0.5)     c(-0.5, 1.5, 0.95, NA)
     optimal   0.4047                    1.060158
      hampel   c(1.5, 3.5, 8)*0.2119163  c(1.5, 3.5, 8)*0.9014
```

The values for the tuning constant for the ggw and lqq psi functions are specified differently here by a vector with four elements: minimal slope, b (controlling the bend at the maximum of the curve), efficiency, breakdown point. Use NA for an unspecified value of either efficiency or breakdown point, see examples in the tables (above and below). For these table examples, the respective "inner constants" are stored precomputed, see .psi.lqq.findc for more.

The constants for the "hampel" psi function are chosen to have a redescending slope of $-1/3$. Constants for a slope of $-1/2$ would be

```
     psi   tuning.chi             tuning.psi
"hampel"   c(2, 4, 8) * 0.1981319   c(2, 4, 8) * 0.690794
```

Alternative coefficients for an efficiency of 85% at the normal are given in the table below.

```
                psi   tuning.psi
          bisquare   3.443689
             welsh   1.456
         ggw, lqq   c(-0.5, 1.5, 0.85, NA)
           optimal   0.8684
   hampel (-1/3)   c(1.5, 3.5, 8)* 0.5704545
   hampel (-1/2)   c( 2, 4, 8) * 0.4769578
```

Value

.Mchi.tuning.default(psi) and .Mpsi.tuning.default(psi) return a short numeric vector of tuning constants which are defaults for the corresponding psi-function, see the *Details*. They are based on the named lists .Mchi.tuning.defaults and .Mpsi.tuning.defaults, respectively.

lmrob.control() returns a named list with over twenty components, corresponding to the arguments, where tuning.psi and tuning.chi are typically computed, as .Mpsi.tuning.default(psi) or .Mchi.tuning.default(psi), respectively. It is of class "lmrobCtrl" and we provide print(), update() and within methods.

update(<lmrobCtrl>, ...) does *not* allow a setting="<...>" in .....

Author(s)

Matias Salibian-Barrera, Martin Maechler and Manuel Koller

References

Koller, M. and Stahel, W.A. (2011) Sharpening Wald-type inference in robust regression for small samples. *Computational Statistics & Data Analysis* **55**(8), 2504–2515.

Koller, M. and Stahel, W.A. (2017) Nonsingular subsampling for regression S estimators with categorical predictors, *Computational Statistics* **32**(2): 631–646. doi:10.1007/s001800160679x. Referred as "KS2014" everywhere in **robustbase**; A shorter first version, Koller (2012) has been available from https://arxiv.org/abs/1208.5595.

See Also

Mpsi, etc, for the (fast!) psi function computations; lmrob, also for references and examples.

Examples

```
## Show the default settings:
str(lmrob.control())

## Artificial data for a  simple  "robust t test":
set.seed(17)
y <- y0 <- rnorm(200)
y[sample(200,20)] <- 100*rnorm(20)
gr <- as.factor(rbinom(200, 1, prob = 1/8))
lmrob(y0 ~ 0+gr)

## Use  Koller & Stahel(2011)'s recommendation but a larger  'max.it':
str(ctrl <- lmrob.control("KS2011", max.it = 1000))

str(.Mpsi.tuning.defaults)
stopifnot(identical(.Mpsi.tuning.defaults,
                    sapply(names(.Mpsi.tuning.defaults),
                           .Mpsi.tuning.default)))
## Containing (names!) all our (pre-defined) redescenders:
str(.Mchi.tuning.defaults)

## Difference between settings:
Cdef <- lmrob.control()
C11 <- lmrob.control("KS2011")
C14 <- lmrob.control("KS2014")
str(C14)
## Differences:
diffD <- names(which(!mapply(identical, Cdef,C11, ignore.environment=TRUE)))
diffC <- names(which(!mapply(identical, C11, C14, ignore.environment=TRUE)))
## KS2011 vs KS2014:  Apart from `setting` itself, they only differ in three places:
cbind(KS11 = unlist(C11[diffC[-1]]),
      KS14 = unlist(C14[diffC[-1]]))
##           KS11 KS14
## nResample  500 1000
## best.r.s     2   20
## k.fast.s     1    2
## default vs KS2011: a bit more: setting + 8
str2simpLang <-  function(x) {
    r <- if(is.null(x)) quote((NULL)) else str2lang(deparse(x))
    if(is.call(r)) format(r) else r
}
cbind(deflt= lapply(Cdef[diffD], str2simpLang),
      KS11 = lapply(C11 [diffD], str2simpLang))

## update()ing a lmrob.control() , e.g.,
C14mod <- update(C14, trace.lev = 2) # the same as
C14m.d <- C14; C14m.d$trace.lev <- 2
stopifnot(identical(C14mod, C14m.d))
## changing psi --> updates tuning.{psi,chi}:
C14mp <- update(C14, psi = "hampel", seed=101)
## updating 'method' is "smart" :
C.SMDM <- update(Cdef, method="SMDM")
```

```
all.equal(Cdef, C.SMDM) # changed also psi, tuning.{psi,chi} and cov !
chgd <- c("method", "psi", "tuning.chi",  "tuning.psi", "cov")
str(Cdef  [chgd])
str(C.SMDM[chgd])
C14m <- update(C14, method="SMM")
(ae <- all.equal(C14, C14mp))# changed tuning.psi & tuning.chi, too
stopifnot(exprs = {
    identical(C14, update(C14, method="SMDM")) # no change!
    identical(c("psi", "seed", "tuning.chi", "tuning.psi"),
              sort(gsub("[^.[:alpha:]]", "", sub(":.*", "", sub("^Component ", "", ae)))))
    identical(C14m, local({C <- C14; C$method <- "SMM"; C}))
})
##
try( update(C14, setting="KS2011") ) #--> Error: .. not allowed
```

`lmrob.fit` *MM-type estimator for regression*

Description

Compute MM-type estimators of regression: An S-estimator is used as starting value, and an M-estimator with fixed scale and redescending psi-function is used from there. Optionally a D-step (Design Adaptive Scale estimate) as well as a second M-step is calculated.

Usage

```
lmrob.fit(x, y, control, init = NULL, mf = NULL, bare.only = FALSE)
```

Arguments

`x`	design matrix ($n \times p$) typically including a column of 1s for the intercept.
`y`	numeric response vector (of length n).
`control`	a list of control parameters as returned by `lmrob.control`, used for both the initial S-estimate and the subsequent M- and D-estimates.
`init`	optional `list` of initial estimates. See *Details*.
`mf`	defunct.
`bare.only`	logical indicating if the result should be `return()`ed after the bare computation steps are done. Useful, e.g., when you only need the `coefficients`.

Details

This function is the basic fitting function for MM-type estimation, called by `lmrob` and typically not to be used on its own.

If given, `init` must be a list of initial estimates containing at least the initial coefficients and scale as `coefficients` and `scale`. Otherwise it calls `lmrob.S(..)` and uses it as initial estimator.

Value

A list with components (some missing in case `bare.only` is true)

`fitted.values`	$X\beta$, i.e., `X %*% coefficients`.
`residuals`	the raw residuals, `y - fitted.values`
`rweights`	robustness weights derived from the final M-estimator residuals (even when not converged).
`rank`	
`degree.freedom`	`n - rank`
`coefficients`	estimated regression coefficient vector
`scale`	the robustly estimated error standard deviation
`cov`	variance-covariance matrix of `coefficients`, if the RWLS iterations have converged (and `control$cov` is not `"none"`).
`control`	
`iter`	
`converged`	logical indicating if the RWLS iterations have converged.
`init.S`	the whole initial S-estimator result, including its own `converged` flag, see `lmrob.S` (only for MM-estimates).
`init`	A similar list that contains the results of intermediate estimates (not for MM-estimates).

Author(s)

Matias Salibian-Barrera, Martin Maechler and Manuel Koller

See Also

`lmrob`, `lmrob..M..fit`, `lmrob..D..fit`, `lmrob.S`

`lmrob.lar` *Least Absolute Residuals / L1 Regression*

Description

To compute least absolute residuals (LAR) or "L1" regression, `lmrob.lar` implements the routine L1 in Barrodale and Roberts (1974), which is based on the simplex method of linear programming. It is a copy of `lmRob.lar` (in early 2012) from the **robust** package.

Usage

```
lmrob.lar(x, y, control, ...)
```

Arguments

x	numeric matrix for the predictors.
y	numeric vector for the response.
control	list as returned by lmrob.control() .
...	(unused but needed when called as init(x,y,ctrl, mf) from lmrob())

Details

This method is used for computing the M-S estimate and typically not to be used on its own.

A description of the Fortran subroutines used can be found in Marazzi (1993). In the book, the main method is named RILARS.

Value

A list that includes the following components:

coef	The L1-estimate of the coefficient vector
scale	The residual scale estimate (mad)
resid	The residuals
iter	The number of iterations required by the simplex algorithm
status	Return status (0: optimal, but non unique solution, 1: optimal unique solution)
converged	Convergence status (always TRUE), needed for lmrob.fit.

Author(s)

Manuel Koller

References

Marazzi, A. (1993). *Algorithms, routines, and S functions for robust statistics*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

See Also

rq from CRAN package **quantreg**.

Examples

```
data(stackloss)
X <- model.matrix(stack.loss ~ . , data = stackloss)
y <- stack.loss
(fm.L1 <- lmrob.lar(X, y))
with(fm.L1, stopifnot(converged
  , status == 1L
  , all.equal(scale, 1.5291576438)
  , sum(abs(residuals) < 1e-15) == 4 # p=4 exactly fitted obs.
))
```

`lmrob.M.S` *M-S regression estimators*

Description

Computes an M-S-estimator for linear regression using the "M-S" algorithm.

Usage

```
lmrob.M.S(x, y, control, mf,
          split = splitFrame(mf, x, control$split.type))
```

Arguments

`x`	numeric matrix (a `model.matrix`) of the predictors.
`y`	numeric vector for the response
`control`	`list` as returned by `lmrob.control`.
`mf`	a model frame as returned by `model.frame`.
`split`	(optional) list as returned by `splitFrame`.

Details

This function is used by `lmrob` and not intended to be used on its own (because an M-S-estimator has too low efficiency 'on its own').

An M-S estimator is a combination of an S-estimator for the continuous variables and an L1-estimator (i.e. an M-estimator with $\psi(t) = sign(t)$) for the categorical variables.

The S-estimator is estimated using a subsampling algorithm. If the model includes interactions between categorical (`factor`) and continuous variables, the subsampling algorithm might fail. In this case, one can choose to assign the interaction to the categorical side of variables rather than to the continuous side. This can be accomplished via the control argument `split.type` or by specifying `split`, see `splitFrame`.

Note that the return status `converged` does not refer to the actual convergence status. The algorithm used does not guarantee convergence and thus true convergence is almost never reached. This is, however, not a problem if the estimate is only used as initial estimate part of an MM or SMDM estimate.

The algorithm sometimes produces the warning message "Skipping design matrix equilibration (dgeequ): row ?? is exactly zero.". This is just an artifact of the algorithm and can be ignored safely.

Value

A list with components

`coefficients`	numeric vector (length p) of M-S-regression coefficient estimates.
`scale`	the M-S-scale residual estimate

residuals	numeric vector (legnth n) of the residuals.
rweights	numeric vector (length n) of the robustness weights.
control	the same list as the control argument.
converged	Convergence status (always TRUE), needed for lmrob.fit.
descent.cov	logical with the true m_s_descent convergence status.

Author(s)

Manuel Koller

References

Maronna, R. A., and Yohai, V. J. (2000). Robust regression with both continuous and categorical predictors. *Journal of Statistical Planning and Inference* **89**, 197–214.

See Also

lmrob; for a description of the available split types, see splitFrame.

lmRob in package **robust** uses a version of the M-S algorithm automatically when the formula contains factors. Our version however follows Maronna and Yohai (2000) more closely.

Examples

```
data(education)
education <- within(education, Region <- factor(Region))
flm <- lm(Y ~ Region + X1 + X2 + X3, education)
x <- model.matrix(flm)
y <- education$Y # == model.response(model.frame(flm))
set.seed(17)
f.MS <- lmrob.M.S(x, y, control = lmrob.control(),
                  mf = model.frame(flm))

## The typical use of the "M-S" estimator -- as initial estimate :
fmMS <- lmrob(Y ~ Region + X1 + X2 + X3, education,
              init = "M-S")
```

---

lmrob.S *S-regression estimators*

Description

Computes an S-estimator for linear regression, using the "fast S" algorithm.

Usage

```
lmrob.S(x, y, control,
        trace.lev = control$trace.lev,
        only.scale = FALSE, mf)
```

Arguments

x	design matrix ($n \times p$)
y	numeric vector of responses (or residuals for only.scale=TRUE).
control	list as returned by lmrob.control; the following components are used for lmrob.S(): "trace.lev", "nResample", "groups", "n.group", "fast.s.large.n", "seed", "bb", "psi", "tuning.chi", "best.r.s", "k.fast.s", "k.max", "maxit.scale", "refine.tol", "solve.tol", "scale.tol", "mts", "subsampling".
trace.lev	integer indicating if the progress of the algorithm should be traced (increasingly); default trace.lev = 0 does no tracing.
only.scale	logical indicating if only the scale of y should be computed. In this case, y will typically contain *residuals*.
mf	defunct.

Details

This function is used by lmrob.fit and typically not to be used on its own (because an S-estimator has too low efficiency 'on its own').

By default, the subsampling algorithm uses a customized LU decomposition which ensures a non singular subsample (if this is at all possible). This makes the Fast-S algorithm also feasible for categorical and mixed continuous-categorical data.

One can revert to the old subsampling scheme by setting the parameter subsampling in control to "simple".

Value

By default (when only.scale is false), a list with components

coefficients	numeric vector (length p) of S-regression coefficient estimates.
scale	the S-scale residual estimate
fitted.values	numeric vector (length n) of the fitted values.
residuals	numeric vector (length n) of the residuals.
rweights	numeric vector (length n) of the robustness weights.
k.iter	(maximal) number of refinement iterations used.
converged	logical indicating if **all** refinement iterations had converged.
control	the same list as the control argument.

If only.scale is true, the computed scale (a number) is returned.

Author(s)

Matias Salibian-Barrera and Manuel Koller; Martin Maechler for minor new options and more documentation.

See Also

lmrob, also for references.

Examples

```
set.seed(33)
x1 <- sort(rnorm(30)); x2 <- sort(rnorm(30)); x3 <- sort(rnorm(30))
X. <- cbind(x1, x2, x3)
y <-  10 + X. %*% (10*(2:4)) + rnorm(30)/10
y[1] <- 500   # a moderate outlier
X.[2,1] <- 20 # an X outlier
X1  <- cbind(1, X.)

(m.lm <- lm(y ~ X.))
set.seed(12)
m.lmS <- lmrob.S(x=X1, y=y,
                 control = lmrob.control(nRes = 20), trace.lev=1)
m.lmS[c("coefficients","scale")]
all.equal(unname(m.lmS$coef), 10 * (1:4), tolerance = 0.005)
stopifnot(all.equal(unname(m.lmS$coef), 10 * (1:4), tolerance = 0.005),
          all.equal(m.lmS$scale, 1/10, tolerance = 0.09))

## only.scale = TRUE:  Compute the S scale, given residuals;
s.lmS <- lmrob.S(X1, y=residuals(m.lmS), only.scale = TRUE,
                 control = lmrob.control(trace.lev = 3))
all.equal(s.lmS, m.lmS$scale) # close: 1.89e-6 [64b Lnx]
```

---

los *Length of Stay Data*

Description

Length of stay for 201 patients that stayed at the University Hospital of Lausanne during the year 2000.

Usage

```
data(los, package="robustbase")
```

Format

Vector of integer values giving the length of stay (days):

int [1:201] 16 13 17 4 15 24 59 18 33 8 ...

Details

These data may be used to estimate and predict the total resource consumption of this group of patients.

Cf. Ruffieux, Paccaud and Marazzi (2000).

Source

The data were kindly provided by A. Marazzi.

Cf. Hubert, M. and Vandervieren, E. (2006), p. 13–15.

References

Ruffieux, C., Paccaud, F. and A. Marazzi (2000) Comparing rules for truncating hospital length of stay; *Casemix Quarterly* **2**, n. 1.

See also those for adjbox.

Examples

```
summary(los) # quite skewed, with median(.) = 8
plot(table(los))
boxplot(los, horizontal=TRUE, add=TRUE, col = "red", axes=FALSE)
##-> "outliers" instead of "just skewed"

hist(log(los))
boxplot(log(los), add=TRUE, col=2, border=2, horizontal = TRUE, at = -1)

## Hubert and Vandervieren (2006), p. 15, Fig. 11.
adjbox(los, col = "gray", staplecol="red", outcol = "red",
       main = "(Skewness-)Adjusted and original boxplot for 'los' data")
boxplot(los, add = TRUE, staplewex= 0.2, outcex= 0.5, outpch= 4,
        staplecol = "blue", outcol = "blue", staplelwd=2)
legend("topright", c("adjbox(los)", "boxplot(los)"),
       col=c("red","blue"), lwd = 1:2, bty="n")
```

---

ltsReg *Least Trimmed Squares Robust (High Breakdown) Regression*

Description

Carries out least trimmed squares (LTS) robust (high breakdown point) regression.

Usage

```
ltsReg(x, ...)

## S3 method for class 'formula'
ltsReg(formula, data, subset, weights, na.action,
       model = TRUE, x.ret = FALSE, y.ret = FALSE,
       contrasts = NULL, offset, ...)

## Default S3 method:
ltsReg(x, y, intercept = TRUE, alpha = , nsamp = , adjust = ,
       mcd = TRUE, qr.out = FALSE, yname = NULL,
      seed = , trace = , use.correction = , wgtFUN = , control = rrcov.control(),
       ...)
```

Arguments

formula	a formula of the form y ~ x1 + x2 + ...
data	data frame from which variables specified in formula are to be taken.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
weights	an optional vector of weights to be used in the fitting process. **NOT USED YET**.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The "factory-fresh" default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
model, x.ret, y.ret	logicals indicating if the model frame, the model matrix and the response are to be returned, respectively.
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. An offset term can be included in the formula instead or as well, and if both are specified their sum is used.
x	a matrix or data frame containing the explanatory variables.
y	the response: a vector of length the number of rows of x.
.	
intercept	if true, a model with constant term will be estimated; otherwise no constant term will be included. Default is intercept = TRUE
alpha	the percentage (roughly) of squared residuals whose sum will be minimized, by default 0.5. In general, alpha must between 0.5 and 1.
nsamp	number of subsets used for initial estimates or "best" or "exact". Default is nsamp = 500. For nsamp="best" exhaustive enumeration is done, as long as the number of trials does not exceed 5000. For "exact", exhaustive enumeration will be attempted however many samples are needed. In this case a warning message will be displayed saying that the computation can take a very long time.
adjust	whether to perform intercept adjustment at each step. Since this can be time consuming, the default is adjust = FALSE.
mcd	whether to compute robust distances using Fast-MCD.
qr.out	whether to return the QR decomposition (see qr); defaults to false.
yname	the name of the dependent variable. Default is yname = NULL
seed	initial seed for random generator, like .Random.seed, see rrcov.control.
trace	logical (or integer) indicating if intermediate results should be printed; defaults to FALSE; values ≥ 2 also produce print from the internal (Fortran) code.
use.correction	whether to use finite sample correction factors. Default is use.correction=TRUE

wgtFUN	a character string or function, specifying how the weights for the reweighting step should be computed. Up to April 2013, the only option has been the original proposal in (1999), now specified by wgtFUN = "01.original" (or via control).
control	a list with estimation options - same as these provided in the function specification. If the control object is supplied, the parameters from it will be used. If parameters are passed also in the invocation statement, they will override the corresponding elements of the control object.
...	arguments passed to or from other methods.

Details

The LTS regression method minimizes the sum of the h smallest squared residuals, where $h > n/2$, i.e. at least half the number of observations must be used. The default value of h (when alpha=1/2) is roughly $n/2$, more precisely, (n+p+1) %/% 2 where n is the total number of observations, but by setting alpha, the user may choose higher values up to n, where $h = h(\alpha, n, p) =$ h.alpha.n(alpha,n,p). The LTS estimate of the error scale is given by the minimum of the objective function multiplied by a consistency factor and a finite sample correction factor – see Pison et al. (2002) for details. The rescaling factors for the raw and final estimates are returned also in the vectors raw.cnp2 and cnp2 of length 2 respectively. The finite sample corrections can be suppressed by setting use.correction=FALSE. The computations are performed using the Fast LTS algorithm proposed by Rousseeuw and Van Driessen (1999).

As always, the formula interface has an implied intercept term which can be removed either by y ~ x - 1 or y ~ 0 + x. See formula for more details.

Value

The function ltsReg returns an object of class "lts". The summary method function is used to obtain (and print) a summary table of the results, and plot() can be used to plot them, see the the specific help pages.

The generic accessor functions coefficients, fitted.values and residuals extract various useful features of the value returned by ltsReg.

An object of class lts is a list containing at least the following components:

crit	the value of the objective function of the LTS regression method, i.e., the sum of the h smallest squared raw residuals.
coefficients	vector of coefficient estimates (including the intercept by default when intercept=TRUE), obtained after reweighting.
best	the best subset found and used for computing the raw estimates, with length(best) == quan = h.alpha.n(alpha,n,p).
fitted.values	vector like y containing the fitted values of the response after reweighting.
residuals	vector like y containing the residuals from the weighted least squares regression.
scale	scale estimate of the reweighted residuals.
alpha	same as the input parameter alpha.
quan	the number h of observations which have determined the least trimmed squares estimator.

intercept	same as the input parameter intercept.
cnp2	a vector of length two containing the consistency correction factor and the finite sample correction factor of the final estimate of the error scale.
raw.coefficients	vector of raw coefficient estimates (including the intercept, when intercept=TRUE).
raw.scale	scale estimate of the raw residuals.
raw.resid	vector like y containing the raw residuals from the regression.
raw.cnp2	a vector of length two containing the consistency correction factor and the finite sample correction factor of the raw estimate of the error scale.
lts.wt	vector like y containing weights that can be used in a weighted least squares. These weights are 1 for points with reasonably small residuals, and 0 for points with large residuals.
raw.weights	vector containing the raw weights based on the raw residuals and raw scale.
method	character string naming the method (Least Trimmed Squares).
X	the input data as a matrix (including intercept column if applicable).
Y	the response variable as a vector.

Author(s)

Valentin Todorov <valentin.todorov@chello.at>, based on work written for S-plus by Peter Rousseeuw and Katrien van Driessen from University of Antwerp.

References

Peter J. Rousseeuw (1984), Least Median of Squares Regression. *Journal of the American Statistical Association* **79**, 871–881.

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection.* Wiley.

P. J. Rousseeuw and K. van Driessen (1999) A fast algorithm for the minimum covariance determinant estimator. *Technometrics* **41**, 212–223.

Pison, G., Van Aelst, S., and Willems, G. (2002) Small Sample Corrections for LTS and MCD. *Metrika* **55**, 111-123.

See Also

covMcd; summary.lts for summaries, lmrob() for alternative robust estimator with HBDP.

The generic functions coef, residuals, fitted.

Examples

```
data(heart)
## Default method works with 'x'-matrix and y-var:
heart.x <- data.matrix(heart[, 1:2]) # the X-variables
heart.y <- heart[,"clength"]
ltsReg(heart.x, heart.y)

data(stackloss)
ltsReg(stack.loss ~ ., data = stackloss)
```

mc *Medcouple, a Robust Measure of Skewness*

Description

Compute the 'medcouple', a *robust* concept and estimator of skewness. The medcouple is defined as a scaled median difference of the left and right half of distribution, and hence *not* based on the third moment as the classical skewness.

Usage

```
mc(x, na.rm = FALSE, doReflect = (length(x) <= 100),
   doScale = FALSE,     # was hardwired=TRUE, then default=TRUE
   c.huberize = 1e11,   # was implicitly = Inf originally
   eps1 = 1e-14, eps2 = 1e-15,   # << new in 0.93-2 (2018-07..)
   maxit = 100, trace.lev = 0, full.result = FALSE)
```

Arguments

x	a numeric vector
na.rm	logical indicating how missing values (NAs) should be dealt with.
doReflect	logical indicating if the internal MC should also be computed on the *reflected* sample -x, with final result (mc.(x) - mc.(-x))/2. This makes sense since the internal MC, mc.() computes the himedian() which can differ slightly from the median.
doScale	logical indicating if the internal algorithm should also *scale* the data (using the most distant value from the median which is unrobust and numerically dangerous); scaling has been hardwired in the original algorithm and R's mc() till summer 2018, where it became the default. Since **robustbase** version 0.95-0, March 2022, the default is FALSE. As this may change the result, a message is printed about the new default, once per R session. You can suppress the message by specifying doScale = * explicitly, or, by setting options(mc_doScale_quiet=TRUE).
c.huberize	a positive number (default: 1e11) used to stabilize the sample via x <- huberize(x, c = c.huberize) for the mc() computations in the case of a nearly degenerate sample (many observations practically equal to the median) or very extreme outliers. In previous versions of **robustbase** no such huberization was applied which is equivalent to c.huberize = Inf.
eps1, eps2	tolerance in the algorithm; eps1 is used as a for convergence tolerance, where eps2 is only used in the internal h_kern() function to prevent underflow to zero, so could be considerably smaller. The original code implicitly *hard coded* in C eps1 := eps2 := 1e-13; only change with care!
maxit	maximal number of iterations; typically a few should be sufficient.
trace.lev	integer specifying how much diagnostic output the algorithm (in C) should produce. No output by default, most output for trace.lev = 5.
full.result	logical indicating if the full return values (from C) should be returned as a list via attr(*, "mcComp").

Value

a number between -1 and 1, which is the medcouple, $MC(x)$. For `r <- mc(x, full.result = TRUE, ....)`, then `attr(r, "mcComp")` is a list with components

`medc`	the medcouple $mc.(x)$.
`medc2`	the medcouple $mc.(-x)$ if `doReflect=TRUE`.
`eps`	tolerances used.
`iter`, `iter2`	number of iterations used.
`converged`, `converged2`	logical specifying "convergence".

Convergence Problems

For extreme cases there were convergence problems which should not happen anymore as we now use `doScale=FALSE` and huberization (when `c.huberize < Inf`).

The original algorithm and `mc(*, doScale=TRUE)` not only centers the data around the median but also scales them by the extremes which may have a negative effect e.g., when changing an extreme outlier to even more extreme, the result changes wrongly; see the 'mc10x' example.

Author(s)

Guy Brys; modifications by Tobias Verbeke and bug fixes and extensions by Manuel Koller and Martin Maechler.

The new default `doScale=FALSE`, and the new `c.huberize` were introduced as consequence of Lukas Graz' BSc thesis.

References

Guy Brys, Mia Hubert and Anja Struyf (2004) A Robust Measure of Skewness; *JCGS* **13** (4), 996–1017.

Hubert, M. and Vandervieren, E. (2008). An adjusted boxplot for skewed distributions, *Computational Statistics and Data Analysis* **52**, 5186–5201.

Lukas Graz (2021). Improvement of the Algorithms for the Medcoule and the Adjusted Outlyingness; unpublished BSc thesis, supervised by M.Maechler, ETH Zurich.

See Also

`Qn` for a robust measure of scale (aka "dispersion"), ...

Examples

```
mc(1:5) # 0 for a symmetric sample

x1 <- c(1, 2, 7, 9, 10)
mc(x1) # = -1/3

data(cushny)
mc(cushny) # 0.125
```

```
stopifnot(mc(c(-20, -5, -2:2, 5, 20)) == 0,
          mc(x1, doReflect=FALSE) ==  -mc(-x1, doReflect=FALSE),
          all.equal(mc(x1, doReflect=FALSE), -1/3, tolerance = 1e-12))

## Susceptibility of the current algorithm to large outliers :
dX10 <- function(X) c(1:5,7,10,15,25, X) # generate skewed size-10 with 'X'
x <- c(10,20,30, 100^(1:20))
## (doScale=TRUE, c.huberize=Inf)  were (implicit) defaults in earlier {robustbase}:
(mc10x <- vapply(x, function(X) mc(dX10(X), doScale=TRUE, c.huberize=Inf), 1))
## limit X -> Inf  should be 7/12 = 0.58333...  but that "breaks down a bit" :
plot(x, mc10x, type="b", main = "mc( c(1:5,7,10,15,25, X) )", xlab="X", log="x")
## The new behavior is much preferable {shows message about new 'doScale=FALSE'}:
(mc10N <- vapply(x, function(X) mc(dX10(X)), 1))
lines(x, mc10N, col=adjustcolor(2, 3/4), lwd=3)
mtext("mc(*, c.huberize=1e11)",  col=2)
stopifnot(all.equal(c(4, 6, rep(7, length(x)-2))/12, mc10N))
## Here, huberization already solves the issue:
mc10NS <- vapply(x, function(X) mc(dX10(X), doScale=TRUE), 1)
stopifnot(all.equal(mc10N, mc10NS))
```

`milk` *Daudin's Milk Composition Data*

Description

Daudin et al.(1988) give 8 readings on the composition of 86 containers of milk. They speak about 85 observations, but this can be explained with the fact that observations 63 and 64 are identical (as noted by Rocke (1996)).

The data set was used for analysing the stability of principal component analysis by the bootstrap method. In the same context, but using high breakdown point robust PCA, these data were analysed by Todorov et al. (1994). Atkinson (1994) used these data for ilustration of the forward search algorithm for identifying of multiple outliers.

Usage

```
data(milk, package="robustbase")
```

Format

A data frame with 86 observations on the following 8 variables, all but the first measure units in *grams / liter*.

`X1` density

`X2` fat content

`X3` protein content

`X4` casein content

`X5` cheese dry substance measured in the factory

`X6` cheese dry substance measured in the laboratory

`X7` milk dry substance

`X8` cheese product

Source

Daudin, J.J. Duby, C. and Trecourt, P. (1988) Stability of Principal Component Analysis Studied by the Bootstrap Method; *Statistics* **19**, 241–258.

References

Todorov, V., Neyko, N., Neytchev, P. (1994) Stability of High Breakdown Point Robust PCA, in *Short Communications, COMPSTAT'94*; Physica Verlag, Heidelberg.

Atkinson, A.C. (1994) Fast Very Robust Methods for the Detection of Multiple Outliers. *J. Amer. Statist. Assoc.* **89** 1329–1339.

Rocke, D. M. and Woodruff, D. L. (1996) Identification of Outliers in Multivariate Data; *J. Amer. Statist. Assoc.* **91** (435), 1047–1061.

Examples

```
data(milk)
(c.milk <- covMcd(milk))
summarizeRobWeights(c.milk $ mcd.wt)# 19..20 outliers
umilk <- unique(milk) # dropping obs.64 (== obs.63)
summary(cumilk <- covMcd(umilk, nsamp = "deterministic")) # 20 outliers
```

`Mpsi` *Psi / Chi / Wgt / Rho Functions for *M-Estimation*

Description

Compute Psi / Chi / Wgt / Rho functions for M-estimation, i.e., including MM, etc. For definitions and details, please use the vignette "ψ-Functions Available in Robustbase".

`MrhoInf(x)` computes $\rho(\infty)$, i.e., the normalizing or scaling constant for the transformation from $\rho(\cdot)$ to $\tilde{\rho}(\cdot)$, where the latter, aka as $\chi()$ fulfills $\tilde{\rho}(\infty) = 1$ which makes only sense for "redescending" psi functions, i.e., not for `"huber"`.

`Mwgt(x, *)` computes $\psi(x)/x$ (fast and numerically accurately).

Usage

```
Mpsi(x, cc, psi, deriv = 0)
Mchi(x, cc, psi, deriv = 0)
Mwgt(x, cc, psi)
MrhoInf(cc, psi)

.Mwgt.psi1(psi, cc = .Mpsi.tuning.default(psi))
.regularize.Mpsi(psi, redescending = TRUE)
```

Arguments

x	numeric ("abscissa" values) vector, possibly with attributes such as dim or names, etc. These are preserved for the M*() functions (but not the .M() ones).
cc	numeric tuning constant, for some psi of length > 1.
psi	a string specifying the psi / chi / rho / wgt function; either "huber", or one of the same possible specifiers as for psi in lmrob.control, i.e. currently, "bisquare", "lqq", "welsh", "optimal", "hampel", or "ggw".
deriv	an integer, specifying the *order* of derivative to consider; particularly, Mpsi(x, *, deriv = -1) is the principal function of $\psi()$, typically denoted $\rho()$ in the literature. For some psi functions, currently "huber", "bisquare", "hampel", and "lqq", deriv = 2 is implemented, for the other psi's only $d \in \{-1, 0, 1\}$
redescending	logical indicating in .regularize.Mpsi(psi,.) if the psi function is redescending.

Details

Theoretically, Mchi() would not be needed explicitly as it can be computed from Mpsi() and MrhoInf(), namely, by

```
Mchi(x, *, deriv = d)  ==  Mpsi(x, *, deriv = d-1) / MrhoInf(*)
```

for $d = 0, 1, 2$ (and '*' containing par, psi, and equality is in the sense of all.equal(x,y, tol) with a small tol.

Similarly, Mwgt would not be needed strictly, as it could be defined via Mpsi), but the explicit definition takes care of 0/0 and typically is of a more simple form.

For experts, there are slightly even faster versions, .Mpsi(), .Mwgt(), etc.

.Mwgt.psi1() mainly a utility for nlrob(), returns a function with similar semantics as psi.hampel, psi.huber, or psi.bisquare from package **MASS**. Namely, a function with arguments (x, deriv=0), which for deriv=0 computes Mwgt(x, cc, psi) and otherwise computes Mpsi(x, cc, psi, deriv=deriv).

.Mpsi(), .Mchi(), .Mwgt(), and .MrhoInf() are low-level versions of Mpsi(), Mchi(), Mwgt(), and MrhoInf(), respectively, and .psi2ipsi() provides the psi-function integer codes needed for ipsi argument of the .M*() functions.

For psi = "ggw", the $\rho()$ function has no closed form and must be computed via numerical integration, apart from 6 special cases including the defaults, see the 'Details' in help(.psi.ggw.findc).

.Mpsi.regularize() may (rarely) be used to regularize a psi function.

Value

a numeric vector of the same length as x, with corresponding function (or derivative) values.

Author(s)

Manuel Koller, notably for the original C implementation; tweaks and speedup via .Call and .M*() etc by Martin Maechler.

References

See the vignette about "ψ-Functions Available in Robustbase".

See Also

psiFunc and the psi_func class, both of which provide considerably more on the R side, but are less optimized for speed.

.Mpsi.tuning.defaults, etc, for tuning constants' defaults for`lmrob()`, and .psi.ggw.findc() utilities to construct such constants' vectors.

Examples

```
x <- seq(-5,7, by=1/8)
matplot(x, cbind(Mpsi(x, 4, "biweight"),
                 Mchi(x, 4, "biweight"),
                 Mwgt(x, 4, "biweight")), type = "l")
abline(h=0, v=0, lty=2, col=adjustcolor("gray", 0.6))

hampelPsi
(ccHa <- hampelPsi @ xtras $ tuningP $ k)
psHa <- hampelPsi@psi(x)
## using Mpsi():
Mp.Ha <- Mpsi(x, cc = ccHa, psi = "hampel")
stopifnot(all.equal(Mp.Ha, psHa, tolerance = 1e-15))

psi.huber <- .Mwgt.psi1("huber")
if(getRversion() >= "3.0.0")
stopifnot(identical(psi.huber, .Mwgt.psi1("huber", 1.345),
                    ignore.env=TRUE))
curve(psi.huber(x), -3, 5, col=2, ylim = 0:1)
curve(psi.huber(x, deriv=1), add=TRUE, col=3)

## and show that this is indeed the same as  MASS::psi.huber() :
x <- runif(256, -2,3)
stopifnot(all.equal(psi.huber(x), MASS::psi.huber(x)),
          all.equal(                psi.huber(x, deriv=1),
                    as.numeric(MASS::psi.huber(x, deriv=1))))

## and how to get  MASS::psi.hampel():
psi.hampel <- .Mwgt.psi1("Hampel", c(2,4,8))
x <- runif(256, -4, 10)
stopifnot(all.equal(psi.hampel(x), MASS::psi.hampel(x)),
          all.equal(                psi.hampel(x, deriv=1),
                    as.numeric(MASS::psi.hampel(x, deriv=1))))

## "lqq" / "LQQ" and its tuning constants:
ctl0 <- lmrob.control(psi = "lqq", tuning.psi=c(-0.5, 1.5, 0.95, NA))
ctl  <- lmrob.control(psi = "lqq", tuning.psi=c(-0.5, 1.5, 0.90, NA))
ctl0$tuning.psi  ## keeps the vector _and_ has "constants" attribute:
## [1] -0.50  1.50  0.95    NA
## attr(,"constants")
```

```
## [1] 1.4734061 0.9822707 1.5000000
ctl$tuning.psi ## ditto:
## [1] -0.5  1.5  0.9  NA \  .."constants"   1.213726 0.809151 1.500000
stopifnot(all.equal(Mpsi(0:2, cc = ctl$tuning.psi, psi = ctl$psi),
                    c(0, 0.977493, 1.1237), tol = 6e-6))
x <- seq(-4,8, by = 1/16)
## Show how you can use .Mpsi() equivalently to Mpsi()
stopifnot(all.equal( Mpsi(x, cc = ctl$tuning.psi, psi = ctl$psi),
                     .Mpsi(x, ccc = attr(ctl$tuning.psi, "constants"),
                              ipsi = .psi2ipsi("lqq"))))
stopifnot(all.equal( Mpsi(x, cc = ctl0$tuning.psi, psi = ctl0$psi, deriv=1),
                     .Mpsi(x, ccc = attr(ctl0$tuning.psi, "constants"),
                              ipsi = .psi2ipsi("lqq"),              deriv=1)))


## M*() preserving attributes :
x <- matrix(x, 32, 8, dimnames=list(paste0("r",1:32), col=letters[1:8]))
comment(x) <- "a vector which is a matrix"
px <- Mpsi(x, cc = ccHa, psi = "hampel")
stopifnot(identical(attributes(x), attributes(px)))

## The "optimal" psi exists in two versions "in the litterature": ---
## Maronna et al. 2006, 5.9.1, p.144f:
psi.M2006 <- function(x, c = 0.013)
  sign(x) * pmax(0, abs(x) - c/dnorm(abs(x)))
## and the other is the one in robustbase from 'robust': via Mpsi(.., "optimal")
## Here are both for 95% efficiency:
(c106 <- .Mpsi.tuning.default("optimal"))
c1 <- curve(Mpsi(x, cc = c106, psi="optimal"), -5, 7, n=1001)
c2 <- curve(psi.M2006(x), add=TRUE, n=1001, col=adjustcolor(2,0.4), lwd=2)
abline(0,1, v=0, h=0, lty=3)
## the two psi's are similar, but really quite different

## a zoom into Maronna et al's:
c3 <- curve(psi.M2006(x), -.5, 1, n=1001); abline(h=0,v=0, lty=3);abline(0,1, lty=2)
```

`nlrob` *Robust Fitting of Nonlinear Regression Models*

Description

`nlrob` fits a nonlinear regression model by robust methods. Per default, by an M-estimator, using iterated reweighted least squares (called "IRLS" or also "IWLS").

Usage

```
nlrob(formula, data, start, lower, upper,
      weights = NULL, na.action = na.fail,
      method = c("M", "MM", "tau", "CM", "mtl"),
      psi = .Mwgt.psi1("huber", cc=1.345), scale = NULL,
```

```
      test.vec = c("resid", "coef", "w"), maxit = 20,
      tol = 1e-06, acc, algorithm = "default", doCov = FALSE, model = FALSE,
      control = if(method == "M") nls.control() else
nlrob.control(method, optArgs = list(trace=trace), ...),
      trace = FALSE, ...)

## S3 method for class 'nlrob'
fitted(object, ...)
## S3 method for class 'nlrob'
residuals(object, type = , ...)
## S3 method for class 'nlrob'
predict(object, newdata, ...)
```

Arguments

formula	a nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta) (cf. nls). (For some checks: if $f(.)$ is linear, then we need parentheses, e.g., y ~ (a + b * x); (note that ._nlrob.w is not allowed as variable or parameter name))
data	an optional 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 nlrob is called.
start	a named numeric vector of starting parameters estimates, only for method = "M".
lower, upper	numeric vectors of lower and upper bounds; if needed, will be replicated to be as long as the longest of start, lower or upper. For (the default) method = "M", if the bounds are unspecified all parameters are assumed to be unconstrained; also, for method "M", bounds can only be used with the "port" algorithm. They are ignored, with a warning, in cases they have no effect. For all other methods, currently these bounds *must* be specified as finite values, and one of them must have names matching the parameter names in formula. For methods "CM" and "mtl", the bounds must *additionally* have an entry named "sigma" as that is determined simultaneously in the same optimization, and hence its lower bound must not be negative.
weights	an optional vector of weights to be used in the fitting process (for intrinsic weights, not the weights w used in the iterative (robust) fit). I.e., sum(w * e^2) is minimized with e = residuals, $e_i = y_i - f(xreg_i, \theta)$, where $f(x, \theta)$ is the nonlinear function, and w are the robust weights from resid * weights.
na.action	a function which indicates what should happen when the data contain NAs. The default action is for the procedure to fail. If NAs are present, use na.exclude to have residuals with length == nrow(data) == length(w), where w are the weights used in the iterative robust loop. This is better if the explanatory variables in formula are time series (and so the NA location is important). For this reason, na.omit, which leads to omission of cases with missing values on any required variable, is not suitable here since the residuals length is different from nrow(data) == length(w).
method	a character string specifying which method to use. The default is "M", for historical and back-compatibility reasons. For the other methods, primarily see

nlrob.algorithms.

"M" Computes an M-estimator, using nls(*, weights=*) iteratively (hence, IRLS) with weights equal to $\psi(r_i)/r_i$, where r_i is the i-the residual from the previous fit.

"MM" Computes an MM-estimator, starting from init, either "S" or "lts".

"tau" Computes a Tau-estimator.

"CM" Computes a "Constrained M" (=: CM) estimator.

"mtl" Compute as "Maximum Trimmed Likelihood" (=: MTL) estimator.

Note that all methods but "M" are "random", hence typically to be preceded by set.seed() in usage, see also nlrob.algorithms.

psi — a function (possibly by name) of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns $\psi(x)/x$ and for deriv=1 returns $\psi'(x)$. Note that tuning constants can *not* be passed separately, but directly via the specification of psi, typically via a simple .Mwgt.psi1() call as per default.

Note that this has been a deliberately non-backcompatible change for robustbase version 0.90-0 (summer 2013 – early 2014).

scale — when not NULL (default), a positive number specifying a scale kept *fixed* during the iterations (and returned as Scale component).

test.vec — character string specifying the convergence criterion. The relative change is tested for residuals with a value of "resid" (the default), for coefficients with "coef", and for weights with "w".

maxit — maximum number of iterations in the robust loop.

tol — non-negative convergence tolerance for the robust fit.

acc — previous name for tol, now deprecated.

algorithm — character string specifying the algorithm to use for nls, see there, only when method = "M". The default algorithm is a Gauss-Newton algorithm.

doCov — a logical specifying if nlrob() should compute the asymptotic variance-covariance matrix (see vcov) already. This used to be hard-wired to TRUE; however, the default has been set to FALSE, as vcov(obj) and summary(obj) can easily compute it when needed.

model — a logical indicating if the model.frame should be returned as well.

control — an optional list of control settings.

for method = "M"**:** settings for nls(). See nls.control for the names of the settable control values and their effect.

for all methods **but** "M"**:** a list, typically resulting from nlrob.control(method, *).

trace — logical value indicating if a "trace" of the nls iteration progress should be printed. Default is FALSE.
If TRUE, in each robust iteration, the residual sum-of-squares and the parameter values are printed at the conclusion of each nls iteration. When the "plinear" algorithm is used, the conditional estimates of the linear parameters are printed after the nonlinear parameters.

object — an R object of class "nlrob", typically resulting from nlrob(..).

`...`	for `nlrob`: only when `method` is *not* `"M"`, optional arguments for `nlrob.control`; for other functions: potentially optional arguments passed to the extractor methods.
`type`	a string specifying the *type* of residuals desired. Currently, `"response"` and `"working"` are supported.
`newdata`	a data frame (or list) with the same names as the original `data`, see e.g., `predict.nls`.

Details

For `method = "M"`, iterated reweighted least squares ("IRLS" or "IWLS") is used, calling `nls(*, weights= .)` where `weights` w_i are proportional to $\psi(r_i/\hat{\sigma})$.

All other methods minimize differently, and work **without** `nls`. See nlrob.algorithms for details.

Value

`nlrob()` returns an object of S3 class `"nlrob"`, for `method = "M"` also inheriting from class `"nls"`, (see `nls`).

It is a list with several components; they are not documented yet, as some of them will probably change. Instead, rather use "accessor" methods, where possible: There are methods (at least) for the generic accessor functions `summary()`, `coefficients()` (aka `coef()`) `fitted.values()`, `residuals()`, `sigma()` and `vcov()`, the latter for the variance-covariance matrix of the estimated parameters, as returned by `coef()`, i.e., not including the variance of the errors. For `nlrob()` results, `estimethod()` returns the "estimation method", which coincides with the `method` argument used.

`residuals(.)`, by default `type = "response"`, returns the residuals e_i, defined above as $e_i = Y_i - f_(x_i, \hat{\theta})$. These differ from the standardized or weighted residuals which, e.g., are assumed to be normally distributed, and a version of which is returned in `working.residuals` component.

Note

This function (with the only method `"M"`) used to be named `rnls` and has been in package **sfsmisc** in the past, but been dropped there.

Author(s)

`method = "M"`**:** Andreas Ruckstuhl (inspired by `rlm`() and `nls`()), in July 1994 for S-plus.
Christian Sangiorgio did the update to R and corrected some errors, from June 2002 to January 2005, and Andreas contributed slight changes and the first methods in August 2005.

`method = "MM"`**, etc:** Originally all by Eduardo L. T. Conceicao, see `nlrob.algorithms`:

Since then, the help page, testing, more cleanup, new methods: Martin Maechler.

See Also

`nls`, `rlm`.

Examples

```
DNase1 <- DNase[ DNase$Run == 1, ]

## note that selfstarting models don't work yet % <<< FIXME !!!

##--- without conditional linearity ---

## classical
fmNase1 <- nls( density ~ Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
                data = DNase1,
                start = list( Asym = 3, xmid = 0, scal = 1 ),
                trace = TRUE )
summary( fmNase1 )

## robust
RmN1  <- nlrob( density ~ Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
                data = DNase1, trace = TRUE,
                start = list( Asym = 3, xmid = 0, scal = 1 ))
summary( RmN1 )

##--- using conditional linearity ---

## classical
fm2DNase1 <- nls( density ~ 1/(1 + exp(( xmid - log(conc) )/scal ) ),
                  data = DNase1,
                  start = c( xmid = 0, scal = 1 ),
                  alg = "plinear", trace = TRUE )
summary( fm2DNase1 )

## robust
frm2DNase1 <- nlrob(density ~ 1/(1 + exp(( xmid - log(conc) )/scal ) ),
                  data = DNase1, start = c( xmid = 0, scal = 1 ),
                  alg = "plinear", trace = TRUE )
summary( frm2DNase1 )
## Confidence for linear parameter is quite smaller than "Asym" above
c1 <- coef(summary(RmN1))
c2 <- coef(summary(frm2DNase1))
rownames(c2)[rownames(c2) == ".lin"] <- "Asym"
stopifnot(all.equal(c1[,1:2], c2[rownames(c1), 1:2], tol = 0.09)) # 0.07315

### -- new examples -- "moderate outlier":
DN2 <- DNase1
DN2[10,"density"] <- 2*DN2[10,"density"]

fm3DN2 <- nls(density ~  Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
              data = DN2, trace = TRUE,
              start = list( Asym = 3, xmid = 0, scal = 1 ))

## robust
Rm3DN2 <- nlrob(density ~  Asym/(1 + exp(( xmid - log(conc) )/scal ) ),
                data = DN2, trace = TRUE,
                start = list( Asym = 3, xmid = 0, scal = 1 ))
```

```
Rm3DN2
summary(Rm3DN2) # -> robustness weight of obs. 10 ~= 0.037
confint(Rm3DN2, method = "Wald")
stopifnot(identical(Rm3DN2$dataClasses,
                    c(density = "numeric", conc = "numeric")))

## utility function sfsmisc::lseq() :
lseq <- function (from, to, length)
  2^seq(log2(from), log2(to), length.out = length)
## predict() {and plot}:
h.x <- lseq(min(DN2$conc), max(DN2$conc), length = 100)
nDat <- data.frame(conc = h.x)

h.p  <- predict(fm3DN2, newdata = nDat)# classical
h.rp <- predict(Rm3DN2, newdata = nDat)# robust

plot(density ~ conc, data=DN2, log="x",
     main = format(formula(Rm3DN2)))
lines(h.x, h.p,  col="blue")
lines(h.x, h.rp, col="magenta")
legend("topleft", c("classical nls()", "robust nlrob()"),
       lwd = 1, col= c("blue", "magenta"), inset = 0.05)

## See  ?nlrob.algorithms for examples

DNase1 <- DNase[DNase$Run == 1,]
form <- density ~ Asym/(1 + exp(( xmid -log(conc) )/scal ))
gMM  <- nlrob(form, data = DNase1, method = "MM",
              lower = c(Asym = 0, xmid = 0, scal = 0),
              upper = 3, trace = TRUE)

## "CM" (and "mtl") additionally need bounds for "sigma" :
gCM  <- nlrob(form, data = DNase1, method = "CM",
              lower = c(Asym = 0, xmid = 0, scal = 0, sigma = 0),
              upper = c(3,3,3, sigma = 0.8))
summary(gCM)# did fail; note it has  NA NA NA (std.err, t val, P val)
stopifnot(identical(Rm3DN2$dataClasses, gMM$dataClasses),
          identical(   gCM$dataClasses, gMM$dataClasses))
```

`nlrob-algorithms` *MM-, Tau-, CM-, and MTL- Estimators for Nonlinear Robust Regression*

Description

"MM": Compute an MM-estimator for nonlinear robust (constrained) regression.

"tau": Compute a Tau-estimator for nonlinear robust (constrained) regression.

"CM": Compute a "Constrained M" (=: CM) estimator for nonlinear robust (constrained) regression.

"MTL": Compute a "Maximum Trimmed Likelihood" (=: MTL) estimator for nonlinear robust (constrained) regression.

Usage

```
## You can *not* call the  nlrob(*, method = <M>) like this ==> see  help(nlrob)
## ------- ----- ----------------------------------------

nlrob.MM(formula, data, lower, upper,
 tol = 1e-06,
 psi = c("bisquare", "lqq", "optimal", "hampel"),
         init = c("S", "lts"),
 ctrl = nlrob.control("MM", psi = psi, init = init, fnscale = NULL,
       tuning.chi.scale = .psi.conv.cc(psi, .Mchi.tuning.defaults[[psi]]),
       tuning.psi.M     = .psi.conv.cc(psi, .Mpsi.tuning.defaults[[psi]]),
       optim.control = list(), optArgs = list(...)),
 ...)

nlrob.tau(formula, data, lower, upper,
  tol = 1e-06, psi = c("bisquare", "optimal"),
  ctrl = nlrob.control("tau", psi = psi, fnscale = NULL,
tuning.chi.scale = NULL, tuning.chi.tau = NULL,
optArgs = list(...)),
  ...)

nlrob.CM(formula, data, lower, upper,
 tol = 1e-06,
 psi = c("bisquare", "lqq", "welsh", "optimal", "hampel", "ggw"),
 ctrl = nlrob.control("CM", psi = psi, fnscale = NULL,
                       tuning.chi = NULL, optArgs = list(...)),
 ...)

nlrob.mtl(formula, data, lower, upper,
  tol = 1e-06,
  ctrl = nlrob.control("mtl", cutoff = 2.5, optArgs = list(...)),
  ...)
```

Arguments

formula	nonlinear regression formula, using both variable names from data and parameter names from either lower or upper.
data	data to be used, a data.frame
lower, upper	bounds aka "box constraints" for all the parameters, in the case "CM" and "mtl" these must include the error standard deviation as "sigma", see nlrob() about its names, etc. Note that one of these two must be a properly "named", e.g., names(lower) being a character vector of parameter names (used in formula above).
tol	numerical convergence tolerance.

psi, init	see nlrob.control.
ctrl	a list, typically the result of a call to nlrob.control.
tuning.psi.M	..
optim.control	..
optArgs	a list of optional arguments for optimization, e.g., trace = TRUE, passed to to the optimizer, which currently must be JDEoptim(.).
...	alternative way to pass the optArgs above.

Details

Copyright 2013, Eduardo L. T. Conceicao. Available under the GPL (>= 2)

Currently, all four methods use JDEoptim() from **DEoptimR**, which subsamples using sample(). From R version 3.6.0, sample depends on RNGkind(*, sample.kind), such that exact reproducibility of results from R versions 3.5.3 and earlier requires setting RNGversion("3.5.0"). In any case, do use set.seed() additionally for reproducibility!

Value

an R object of class "nlrob.<meth>", basically a list with components

Author(s)

Eduardo L. T. Conceicao; compatibility (to nlrob) tweaks and generalizations, inference, by Martin Maechler.

Source

For "MTL": Maronna, Ricardo A., Martin, R. Douglas, and Yohai, Victor J. (2006). *Robust Statistics: Theory and Methods* Wiley, Chichester, p. 133.

References

"MM": Yohai, V.J. (1987) High breakdown-point and high efficiency robust estimates for regression. *The Annals of Statistics* **15**, 642–656.

"tau": Yohai, V.J., and Zamar, R.H. (1988). High breakdown-point estimates of regression by means of the minimization of an efficient scale. *Journal of the American Statistical Association* **83**, 406–413.

"CM": Mendes, B.V.M., and Tyler, D.E. (1996) Constrained M-estimation for regression.

In: *Robust Statistics, Data Analysis and Computer Intensive Methods*, Lecture Notes in Statistics 109, Springer, New York, 299–320.

"MTL": Hadi, Ali S., and Luceno, Alberto (1997). Maximum trimmed likelihood estimators: a unified approach, examples, and algorithms. Computational Statistics & Data Analysis **25**, 251–272.

Gervini, Daniel, and Yohai, Victor J. (2002). A class of robust and fully efficient regression estimators. The Annals of Statistics **30**, 583–616.

Examples

```
DNase1 <- DNase[DNase$Run == 1,]
form <- density ~ Asym/(1 + exp(( xmid -log(conc) )/scal ))
pnms <- c("Asym", "xmid", "scal")
set.seed(47) # as these by default use randomized optimization:

fMM <- robustbase:::nlrob.MM(form, data = DNase1,
           lower = setNames(c(0,0,0), pnms), upper = 3,
           ## call to nlrob.control to pass 'optim.control':
           ctrl = nlrob.control("MM", optim.control = list(trace = 1),
                                optArgs = list(trace = TRUE)))

## The same via nlrob() {recommended; same random seed to necessarily give the same}:
set.seed(47)
gMM  <- nlrob(form, data = DNase1, method = "MM",
              lower = setNames(c(0,0,0), pnms), upper = 3, trace = TRUE)
gMM
summary(gMM)
## and they are the same {apart from 'call' and 'ctrl' and new stuff in gMM}:
ni <- names(fMM); ni <- ni[is.na(match(ni, c("call","ctrl")))]
stopifnot(all.equal(fMM[ni], gMM[ni]))
```

`nlrob.control` *Control Nonlinear Robust Regression Algorithms*

Description

Allow the user to specify details for the different nonlinear robust regression algorithms in `nlrob`.

Usage

```
nlrob.control(method,
              psi = c("bisquare", "lqq", "welsh", "optimal", "hampel", "ggw"),
              init = c("S", "lts"),
              optimizer = "JDEoptim", optArgs  = list(),
              ...)
```

Arguments

`method`	`character` string specifying the method
`psi`	string specifying the psi-function which defines the estimator.
`init`	for some methods, currently, `"MM"` only, a string specifying the initial estimator.
`optimizer`	currently only `"JDEoptim"` from package **DEoptimR**.
`optArgs`	a `list` of optional arguments to the optimizer. Currently, that is `JDEoptim` from package **DEoptimR**.

`...` optional arguments depending on `method`, such as `fnscale`, `tuning.chi` or both `tuning.chi.tau` and `tuning.chi.scale`; for `method = "MM"` also `optim.control` to be passed to the `optim(.., hessian=TRUE)` call. Internally, `nlrob.control()` will choose (or check) defaults for the psi/rho/chi related tuning parameters, also depending on the `method` chosen; see e.g., the 'Examples'.

Value

a `list` with several named components. The contents depend quite a bit on the `method`.

See Also

`nlrob`; for some details, `nlrob.algorithms`.

Examples

```
## Show how the different 'method's  have different smart defaults :
str(nlrob.control("MM"))
str(nlrob.control("MM", psi = "hampel"))# -> other tuning.psi.M and tuning.chi.scale defaults
str(nlrob.control("MM", psi = "lqq", tol = 1e-10))# other tuning.psi.M & tuning.chi.scale defaults
str(nlrob.control("tau"))
str(nlrob.control("tau",psi= "lqq"))
str(nlrob.control("CM")) # tuning.chi undefined, unneeded
str(nlrob.control("CM", psi= "optimal"))
str(nlrob.control("mtl"))
```

NOxEmissions *NOx Air Pollution Data*

Description

A typical medium sized environmental data set with hourly measurements of NOx pollution content in the ambient air.

Usage

```
data(NOxEmissions, package="robustbase")
```

Format

A data frame with 8088 observations on the following 4 variables.

`julday` day number, a factor with levels `373` ... `730`, typically with 24 hourly measurements.

`LNOx` log of hourly mean of NOx concentration in ambient air [ppb] next to a highly frequented motorway.

`LNOxEm` log of hourly sum of NOx emission of cars on this motorway in arbitrary units.

`sqrtWS` Square root of wind speed [m/s].

Details

The original data set had more observations, but with missing values. Here, all cases with missing values were omitted (na.omit(.)), and then only those were retained that belonged to days with at least 20 (fully) observed hourly measurements.

Source

René Locher (at ZHAW, Switzerland).

See Also

another NOx dataset, ambientNOxCH.

Examples

```
data(NOxEmissions)
plot(LNOx ~ LNOxEm, data = NOxEmissions, cex = 0.25, col = "gray30")

## Not run: ## these take too much time --
 ## p = 340  ==> already Least Squares is not fast
 (lmNOx <- lm(LNOx ~ . ,data = NOxEmissions))
 plot(lmNOx) #->  indication of 1 outlier

 M.NOx <- MASS::rlm(LNOx ~ . , data = NOxEmissions)
 ## M-estimation works
 ## whereas  MM-estimation fails:
 try(MM.NOx <- MASS::rlm(LNOx ~ . , data = NOxEmissions, method = "MM"))
 ## namely because S-estimation fails:
 try(lts.NOx <- ltsReg(LNOx ~ . , data = NOxEmissions))
 try(lmR.NOx <- lmrob (LNOx ~ . , data = NOxEmissions))

## End(Not run)
```

---

outlierStats *Robust Regression Outlier Statistics*

Description

Simple statistics about observations with robustness weight of almost zero for models that include factor terms. The number of rejected observations and the mean robustness weights are computed for each level of each factor included in the model.

Usage

```
outlierStats(object, x = object$x, control = object$control
           , epsw = control$eps.outlier
           , epsx = control$eps.x
           , warn.limit.reject = control$warn.limit.reject
           , warn.limit.meanrw = control$warn.limit.meanrw
           , shout = NA)
```

Arguments

object	object of class "lmrob", typically the result of a call to lmrob.
x	design matrix
control	list as returned by lmrob.control().
epsw	limit on the robustness weight below which an observation is considered to be an outlier. Either a numeric(1) or a function that takes the number of observations as an argument.
epsx	limit on the absolute value of the elements of the design matrix below which an element is considered zero. Either a numeric(1) or a function that takes the maximum absolute value in the design matrix as an argument.
warn.limit.reject	limit of ratio #rejected/#obs in level above ($\geq$) which a warning is produced. Set to NULL to disable warning.
warn.limit.meanrw	limit of the mean robustness per factor level below which ($\leq$) a warning is produced. Set to NULL to disable warning.
shout	a logical (scalar) indicating if large "Ratio" or small "Mean.RobWeight" should lead to corresponding warning()s; cutoffs are determined by warn.limit.reject and warn.limit.meanrw, above. By default, NA; setting it to FALSE or TRUE disables or unconditionally enables "shouting".

Details

For models that include factors, the fast S-algorithm used by lmrob can produce "bad" fits for some of the factor levels, especially if there are many levels with only a few observations. Such a "bad" fit is characterized as a fit where most of the observations in a level of a factor are rejected, i.e., are assigned robustness weights of zero or nearly zero. We call such a fit a "local exact fit".

If a local exact fit is detected, then we recommend to increase some of the control parameters of the "fast S"-algorithm. As a first aid solution in such cases, one can use setting="KS2014", see also lmrob.control.

This function is called internally by lmrob to issue a warning if a local exact fit is detected. The output is available as ostats in objects of class "lmrob" (only if the statistic is computed).

Value

A data frame for each column with any zero elements as well as an overall statistic. The data frame consist of the names of the coefficients in question, the number of non-zero observations in that level (N.nonzero), the number of rejected observations (N.rejected), the ratio of rejected observations to the number of observations in that level (Ratio) and the mean robustness weight of all the observations in the corresponding level (Mean.RobWeight).

Author(s)

Manuel Koller

References

Koller, M. and Stahel, W.A. (2017) Nonsingular subsampling for regression S estimators with categorical predictors, *Computational Statistics* **32**(2): 631–646. doi:10.1007/s001800160679x

See Also

`lmrob.control` for the default values of the control parameters; `summarizeRobWeights`.

Examples

```
## artificial data example
data <- expand.grid(grp1 = letters[1:5], grp2 = letters[1:5], rep=1:3)
set.seed(101)
data$y <- c(rt(nrow(data), 1))
## compute outlier statistics for all the estimators
control <- lmrob.control(method = "SMDM",
                         compute.outlier.stats = c("S", "MM", "SMD", "SMDM"))
## warning is only issued for some seeds
set.seed(2)
fit1 <- lmrob(y ~ grp1*grp2, data, control = control)
## do as suggested:
fit2 <- lmrob(y ~ grp1*grp2, data, setting = "KS2014")

## the plot function should work for such models as well
plot(fit1)

## Not run:
  ## access statistics:
  fit1$ostats ## SMDM
  fit1$init$ostats ## SMD
  fit1$init$init$ostats ## SM
  fit1$init$init$init.S$ostats ## S

## End(Not run)
```

`pension` *Pension Funds Data*

Description

The total 1981 premium income of pension funds of Dutch firms, for 18 Professional Branches, from de Wit (1982).

Usage

```
data(pension, package="robustbase")
```

Format

A data frame with 18 observations on the following 2 variables.

`Income` Premium Income (in millions of guilders)

`Reserves` Premium Reserves (in millions of guilders)

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.76, table 13.

Examples

```
data(pension)
plot(pension)

summary(lm.p  <-    lm(Reserves ~., data=pension))
summary(lmR.p <- lmrob(Reserves ~., data=pension))
summary(lts.p <- ltsReg(Reserves ~., data=pension))
abline( lm.p)
abline(lmR.p, col=2)
abline(lts.p, col=2, lty=2)

## MM: "the" solution is much simpler:
plot(pension, log = "xy")
lm.lp  <-    lm(log(Reserves) ~ log(Income), data=pension)
lmR.lp <- lmrob(log(Reserves) ~ log(Income), data=pension)
plot(log(Reserves) ~ log(Income), data=pension)
## no difference between LS and robust:
abline( lm.lp)
abline(lmR.lp, col=2)
```

---

phosphor *Phosphorus Content Data*

Description

This dataset investigates the effect from inorganic and organic Phosphorus in the soil upon the phosphorus content of the corn grown in this soil, from Prescott (1975).

Usage

```
data(phosphor, package="robustbase")
```

Format

A data frame with 18 observations on the following 3 variables.

`inorg` Inorganic soil Phosphorus

`organic` Organic soil Phosphorus

`plant` Plant Phosphorus content

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection.* Wiley, p.156, table 24.

Examples

```
data(phosphor)
plot(phosphor)
summary(lm.phosphor <- lm(plant ~ ., data = phosphor))
summary(lts.phosphor <- ltsReg(plant ~ ., data = phosphor))

phosphor.x <- data.matrix(phosphor[, 1:2])
cPh <- covMcd(phosphor.x)
plot(cPh, "dd")
```

---

pilot *Pilot-Plant Data*

Description

Pilot-Plant data from Daniel and Wood (1971). The response variable corresponds to the acid content determined by titration and the explanatory variable is the organic acid content determined by extraction and weighing. This data set was analyzed also by Yale and Forsythe (1976).

Usage

```
data(pilot, package="robustbase")
```

Format

A data frame with 20 observations on the following 2 variables.

`X` Organic acid content - extraction

`Y` Acid content - titration

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, page 21, table 1.

Examples

```
data(pilot)
summary(lm.pilot <- lm(Y ~.,data=pilot))
```

plot-methods *Plot an Object of the "Psi Function" Class*

Description

The plot method objects of class psi_func simply visualizes the $\rho()$, $\psi()$, and weight functions and their derivatives.

Usage

```
## S4 method for signature 'psi_func'
plot(x, y,
     which = c("rho", "psi", "Dpsi", "wgt", "Dwgt"),
     main = "full",
     col = c("black", "red3", "blue3", "dark green", "light green"),
     leg.loc = "right", ...)
```

Arguments

x	object of class psi_func to be plotted
y	(optional) vector of abscissa values (to plot object at).
which	character vector of slots to be included in plot; by default, all of the slots are included
main	string or logical indicating the kind of plot title; either "full", "short" or FALSE which chooses a full, a short or no main title at all.
col	colors to be used for the different slots
leg.loc	legend placement, see also x argument of legend
...	passed to matplot

Note

An earlier version had argument shortMain which is deprecated now. Use main = "short" instead of shortMain = TRUE.

If you want to specify your own title, use main=FALSE, and a subsequent title(...) call.

See Also

psiFunc() and the class psi_func.

Examples

```
plot(huberPsi)
plot(huberPsi, which=c("psi", "Dpsi", "wgt"),
     main="short", leg = "topleft")

plot(hampelPsi)
```

```
## Plotting aspect ratio = 1:1 :
plot(hampelPsi, asp=1, main="short",
     which = c("psi", "Dpsi", "wgt", "Dwgt"))
```

---

plot.lmrob *Plot Method for "lmrob" Objects*

Description

Diagnostic plots for elements of class lmrob

Usage

```
## S3 method for class 'lmrob'
plot(x, which = 1:5,
     caption = c("Standardized residuals vs. Robust Distances",
   "Normal Q-Q vs. Residuals", "Response vs. Fitted Values",
   "Residuals vs. Fitted Values" , "Sqrt of abs(Residuals) vs. Fitted Values"),
   panel = if(add.smooth) panel.smooth else points,
   sub.caption = deparse(x$call), main = "",
   compute.MD = TRUE,
   ask = prod(par("mfcol")) < length(which) && dev.interactive(),
   id.n = 3, labels.id = names(residuals(x)), cex.id = 0.75,
   label.pos = c(4,2), qqline = TRUE, add.smooth = getOption("add.smooth"),
   ..., p=0.025)
```

Arguments

x	an object as created by lmrob
which	integer number between 1 and 5 to specify which plot is desired
caption	Caption for the different plots
panel	panel function. The useful alternative to points, panel.smooth can be chosen by add.smooth = TRUE.
main	main title
sub.caption	sub titles
compute.MD	logical indicating if the robust Mahalanobis distances should be recomputed, using covMcd() when needed, i.e., if which contains 1.
ask	waits for user input before displaying each plot
id.n	number of points to be labelled in each plot, starting with the most extreme.
labels.id	vector of labels, from which the labels for extreme points will be chosen. NULL uses observation numbers.
cex.id	magnification of point labels.
label.pos	positioning of labels, for the left half and right half of the graph respectively.
qqline	logical indicating if a qqline() should be added to the normal Q-Q plot.

add.smooth	logical indicating if a smoother should be added to most plots; see also panel above.
...	optional arguments for par, title, etc.
p	threshold for distance-distance plot

Details

if compute.MD = TRUE and the robust Mahalanobis distances need to be computed, they are stored ("cached") with the object x when this function has been called from top-level.

References

Robust diagnostic plots as in Rousseeuw and van Zomeren (1990), see 'References' in ltsPlot.

See Also

lmrob, also for examples, plot.lm.

Examples

```
data(starsCYG)
## Plot simple data and fitted lines
plot(starsCYG)
 lmST <-    lm(log.light ~ log.Te, data = starsCYG)
RlmST <- lmrob(log.light ~ log.Te, data = starsCYG)
RlmST
abline(lmST, col = "red")
abline(RlmST, col = "blue")

op <- par(mfrow = c(2,2), mgp = c(1.5, 0.6, 0), mar= .1+c(3,3,3,1))
plot(RlmST, which = c(1:2, 4:5))
par(op)
```

---

plot.lts *Robust LTS Regression Diagnostic Plots*

Description

Four plots (selectable by which) are currently provided:

1. a plot of the standardized residuals versus their index,
2. a plot of the standardized residuals versus fitted values,
3. a Normal Q-Q plot of the standardized residuals, and
4. a regression diagnostic plot (standardized residuals versus robust distances of the predictor variables).

Usage

```
## S3 method for class 'lts'
plot(x, which = c("all","rqq","rindex","rfit","rdiag"),
     classic=FALSE, ask = (which[1] == "all" && dev.interactive()),
     id.n, ...)
```

Arguments

x	a lts object, typically result of ltsReg.
which	string indicating which plot to show. See the *Details* section for a description of the options. Defaults to "all".
.	
classic	whether to plot the classical distances too. Default is FALSE.
.	
ask	logical indicating if the user should be *ask*ed before each plot, see par(ask=.). Defaults to which == "all" && dev.interactive().
id.n	number of observations to be identified by a label starting with the most extreme. Default is the number of identified outliers (can be different for the different plots - see Details).
...	other parameters to be passed through to plotting functions.

Details

This function produces several plots based on the robust and classical regression estimates. Which of them to select is specified by the attribute which. The possible options are:

rqq: Normal Q-Q plot of the standardized residuals;

rindex: plot of the standardized residuals versus their index;

rfit: plot of the standardized residuals versus fitted values;

rdiag: regression diagnostic plot.

The normal quantile plot produces a normal Q-Q plot of the standardized residuals. A line is drawn which passes through the first and third quantile. The id.n residuals with largest distances from this line are identified by labels (the observation number). The default for id.n is the number of regression outliers (lts.wt==0).

In the Index plot and in the Fitted values plot the standardized residuals are displayed against the observation number or the fitted value respectively. A horizontal dashed line is drawn at 0 and two solid horizontal lines are located at +2.5 and -2.5. The id.n residuals with largest absolute values are identified by labels (the observation number). The default for id.n is the number regression outliers (lts.wt==0).

The regression diagnostic plot, introduced by Rousseeuw and van Zomeren (1990), displays the standardized residuals versus robust distances. Following Rousseeuw and van Zomeren (1990), the horizontal dashed lines are located at +2.5 and -2.5 and the vertical line is located at the upper 0.975 percent point of the chi-squared distribution with p degrees of freedom. The id.n residuals with

largest absolute values and/or largest robust Mahalanobis distances are identified by labels (the observation number). The default for id.n is the number of all outliers: regression outliers (lts.wt==0) + leverage (bad and good) points (RD > 0.975 percent point of the chi-squared distribution with p degrees of freedom).

References

P. J. Rousseeuw and van Zomeren, B. C. (1990). Unmasking Multivariate Outliers and Leverage Points. *Journal of the American Statistical Association* **85**, 633–639.

P. J. Rousseeuw and K. van Driessen (1999) A fast algorithm for the minimum covariance determinant estimator. *Technometrics* **41**, 212–223.

See Also

covPlot

Examples

```
data(hbk)
lts <- ltsReg(Y ~ ., data = hbk)
lts
plot(lts, which = "rqq")
```

`plot.mcd` *Robust Distance Plots*

Description

Shows the Mahalanobis distances based on robust and classical estimates of the location and the covariance matrix in different plots. The following plots are available:

- index plot of the robust and mahalanobis distances
- distance-distance plot
- Chisquare QQ-plot of the robust and mahalanobis distances
- plot of the tolerance ellipses (robust and classic)
- Scree plot - Eigenvalues comparison plot

Usage

```
## S3 method for class 'mcd'
plot(x,
     which = c("all", "dd", "distance", "qqchi2",
               "tolEllipsePlot", "screeplot"),
     classic = FALSE, ask = (which[1] == "all" && dev.interactive()),
     cutoff, id.n, labels.id = rownames(x$X), cex.id = 0.75,
     label.pos = c(4,2), tol = 1e-7, ...)
```

```
covPlot(x,
     which = c("all", "dd", "distance", "qqchi2",
               "tolEllipsePlot", "screeplot"),
     classic = FALSE, ask = (which[1] == "all" && dev.interactive()),
     m.cov = covMcd(x),
     cutoff = NULL, id.n, labels.id = rownames(x), cex.id = 0.75,
     label.pos = c(4,2), tol = 1e-07, ...)
```

Arguments

x	For the plot() method, a mcd object, typically result of covMcd. For covPlot(), the numeric data matrix such as the X component as returned from covMcd.
which	string indicating which plot to show. See the *Details* section for a description of the options. Defaults to "all".
.	
classic	whether to plot the classical distances too. Defaults to FALSE.
.	
ask	logical indicating if the user should be *ask*ed before each plot, see par(ask=.). Defaults to which == "all" && dev.interactive().
cutoff	the cutoff value for the distances.
id.n	number of observations to be identified by a label. If not supplied, the number of observations with distance larger than cutoff is used.
labels.id	vector of labels, from which the labels for extreme points will be chosen. NULL uses observation numbers.
cex.id	magnification of point labels.
label.pos	positioning of labels, for the left half and right half of the graph respectively (used as text(.., pos=*)).
tol	tolerance to be used for computing the inverse, see solve. Defaults to tol = 1e-7.
m.cov	an object similar to those of class "mcd"; however only its components center and cov will be used. If missing, the MCD will be computed (via covMcd()).
...	other parameters to be passed through to plotting functions.

Details

These functions produce several plots based on the robust and classical location and covariance matrix. Which of them to select is specified by the attribute which. The plot method for "mcd" objects is calling covPlot() directly, whereas covPlot() should also be useful for plotting other (robust) covariance estimates. The possible options are:

distance index plot of the robust distances

dd distance-distance plot

qqchi2 a qq-plot of the robust distances versus the quantiles of the chi-squared distribution

`tolEllipsePlot` a tolerance ellipse plot, via `tolEllipsePlot()`

`screeplot` an eigenvalues comparison plot - screeplot

The Distance-Distance Plot, introduced by Rousseeuw and van Zomeren (1990), displays the robust distances versus the classical Mahalanobis distances. The dashed line is the set of points where the robust distance is equal to the classical distance. The horizontal and vertical lines are drawn at values equal to the cutoff which defaults to square root of the 97.5% quantile of a chi-squared distribution with p degrees of freedom. Points beyond these lines can be considered outliers.

References

P. J. Rousseeuw and van Zomeren, B. C. (1990). Unmasking Multivariate Outliers and Leverage Points. *Journal of the American Statistical Association* **85**, 633–639.

P. J. Rousseeuw and K. van Driessen (1999) A fast algorithm for the minimum covariance determinant estimator. *Technometrics* **41**, 212–223.

See Also

`tolEllipsePlot`

Examples

```
data(Animals, package ="MASS")
brain <- Animals[c(1:24, 26:25, 27:28),]
mcd <- covMcd(log(brain))

plot(mcd, which = "distance", classic = TRUE)# 2 plots
plot(mcd, which = "dd")
plot(mcd, which = "tolEllipsePlot", classic = TRUE)
op <- par(mfrow = c(2,3))
plot(mcd) ## -> which = "all" (5 plots)
par(op)

## same plots for another robust Cov estimate:
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])
cOGK <- covOGK(hbk.x, n.iter = 2, sigmamu = scaleTau2,
               weight.fn = hard.rejection)
covPlot(hbk.x, m.cov = cOGK, classic = TRUE)
```

`possumDiv` *Possum Diversity Data*

Description

Possum diversity data: As issued from a study of the diversity of possum (arboreal marsupials) in the Montane ash forest (Australia), this dataset was collected in view of the management of hardwood forest to take conservation and recreation values, as well as wood production, into account.

The study is fully described in the two references. The number of different species of arboreal marsupials (possum) was observed on 151 different 3ha sites with uniform vegetation. For each site the nine variable measures (see below) were recorded. The problem is to model the relationship between `diversity` and these other variables.

Usage

```
data(possumDiv, package="robustbase")
```

Format

Two different representations of the same data are available:

`possumDiv` is a data frame of 151 observations of 9 variables, where the last two are factors, `eucalyptus` with 3 levels and `aspect` with 4 levels.

`possum.mat` is a numeric (integer) matrix of 151 rows (observations) and 14 columns (variables) where the last seven ones are 0-1 dummy variables, three (`E.*`) are coding for the kind of `eucalyptus` and the last four are 0-1 coding for the `aspect` factor.

The variables have the following meaning:

Diversity main variable of interest is the number of different species of arboreal marsupial (possum) observed, with values in 0:5.

Shrubs the number of shrubs.

Stumps the number of cut stumps from past logging operations.

Stags the number of stags (hollow-bearing trees).

Bark bark index (integer) vector reflecting the quantity of decorticating bark.

Habitat an integer score indicating the suitability of nesting and foraging habitat for Leadbeater's possum.

BAcacia a numeric vector giving the basal area of acacia species.

eucalyptus a 3-level `factor` specifying the species of eucalypt with the greatest stand basal area. This has the same information as the following three variables

E.regnans 0-1 indicator for Eucalyptus regnans

E.delegatensis 0-1 indicator for Eucalyptus deleg.

E.nitens 0-1 indicator for Eucalyptus nitens

aspect a 4-level `factor` specifying the aspect of the site. It is the same information as the following four variables.

NW-NE 0-1 indicator

NW-SE 0-1 indicator

SE-SW 0-1 indicator

SW-NW 0-1 indicator

Source

Eva Cantoni (2004) Analysis of Robust Quasi-deviances for Generalized Linear Models. *Journal of Statistical Software* **10**, 04, https://www.jstatsoft.org/article/view/v010i04

References

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Nix, H. A. and Smith, A. P. (1991) The conservation of arboreal marsupials in the montane ash forests of the central highlands of victoria, south-east australia: III. The habitat requirements of leadbeater's possum *gymnobelideus leadbeateri* and models of the diversity and abundance of arboreal marsupials. *Biological Conservation* **56**, 295–315.

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Smith, A. P. and Nix, H. A. (1990) The conservation of arboreal marsupials in the montane ash forests of the victoria, south-east australia, I. Factors influencing the occupancy of trees with hollows, *Biological Conservation* **54**, 111–131.

See also the references in glmrob.

Examples

```
data(possumDiv)
head(possum.mat)

str(possumDiv)
## summarize all variables as multilevel factors:
summary(as.data.frame(lapply(possumDiv, function(v)
                             if(is.integer(v)) factor(v) else v)))

## Following Cantoni & Ronchetti (2001), JASA, p.1026 f.:% cf. ../tests/poisson-ex.R
pdFit <- glmrob(Diversity ~ . , data = possumDiv,
                family=poisson, tcc = 1.6, weights.on.x = "hat", acc = 1e-15)
summary(pdFit)
summary(pdF2 <- update(pdFit, ~ . -Shrubs))
summary(pdF3 <- update(pdF2,  ~ . -eucalyptus))
summary(pdF4 <- update(pdF3,  ~ . -Stumps))
summary(pdF5 <- update(pdF4,  ~ . -BAcacia))
summary(pdF6 <- update(pdF5,  ~ . -aspect))# too much ..
anova(pdFit, pdF3, pdF4, pdF5, pdF6, test = "QD") # indeed,
## indeed, the last simplification is too much
possumD.2 <- within(possumDiv, levels(aspect)[1:3] <- rep("other", 3))
## and use this binary 'aspect' instead of the 4-level one:
summary(pdF5.1 <- update(pdF5, data = possumD.2))

if(FALSE) # not ok, as formually not nested.
anova(pdF5, pdF5.1)

summarizeRobWeights(weights(pdF5.1, type="rob"), eps = 0.73)
##->  "outliers"  (1, 59, 110)
wrob <- setNames(weights(pdF5.1, type="rob"), rownames(possumDiv))
head(sort(wrob))
```

predict.glmrob *Predict Method for Robust GLM ("glmrob") Fits*

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted *robust* generalized linear model (GLM) object.

Usage

```
## S3 method for class 'glmrob'
predict(object, newdata = NULL,
       type = c("link", "response", "terms"), se.fit = FALSE,
       dispersion = NULL, terms = NULL, na.action = na.pass, ...)
```

Arguments

object	a fitted object of class inheriting from "glmrob".
newdata	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
type	the type of prediction required. The default is on the scale of the linear predictors; the alternative "response" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and type = "response" gives the predicted probabilities. The "terms" option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale. The value of this argument can be abbreviated.
se.fit	logical switch indicating if standard errors are required.
dispersion	the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by summary applied to the object is used.
terms	with type="terms" by default all terms are returned. A character vector specifies which terms are to be returned
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
...	optional further arguments, currently simply passed to predict.lmrob().

Value

If se = FALSE, a vector or matrix of predictions. If se = TRUE, a list with components

fit	Predictions
se.fit	Estimated standard errors
residual.scale	A scalar giving the square root of the dispersion used in computing the standard errors.

Author(s)

Andreas Ruckstuhl

See Also

`glmrob()` to fit these robust GLM models, `residuals.glmrob()` and other methods; `predict.lm()`, the method used for a non-robust fit.

Examples

```
data(carrots)
## simplistic testing & training:
i.tr <- sample(24, 20)
fm1 <- glmrob(cbind(success, total-success) ~ logdose + block,
              family = binomial, data = carrots, subset = i.tr)
fm1
predict(fm1, carrots[-i.tr, ]) # --> numeric vector
predict(fm1, carrots[-i.tr, ],
        type="response", se = TRUE)# -> a list

data(vaso)
Vfit <- glmrob(Y ~ log(Volume) + log(Rate), family=binomial, data=vaso)
newd <- expand.grid(Volume = (V. <- seq(.5, 4, by = 0.5)),
                    Rate   = (R. <- seq(.25,4, by = 0.25)))
p <- predict(Vfit, newd)
filled.contour(V., R., matrix(p, length(V.), length(R.)),
      main = "predict(glmrob(., data=vaso))", xlab="Volume", ylab="Rate")
```

---

predict.lmrob *Predict method for Robust Linear Model ("lmrob") Fits*

Description

Predicted values based on robust linear model object.

Usage

```
## S3 method for class 'lmrob'
predict(object, newdata, se.fit = FALSE,
       scale = NULL, df = NULL,
       interval = c("none", "confidence", "prediction"), level = 0.95,
       type = c("response", "terms"), terms = NULL,
       na.action = na.pass, pred.var = res.var/weights, weights = 1, ...)
```

Arguments

object	object of class inheriting from "lmrob"
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
se.fit	a switch indicating if standard errors are required.
scale	scale parameter for std.err. calculation
df	degrees of freedom for scale
interval	type of interval calculation.
level	tolerance/confidence level
type	Type of prediction (response or model term).
terms	if type="terms", which terms (default is all terms)
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
pred.var	the variance(s) for future observations to be assumed for prediction intervals. See 'Details'.
weights	variance weights for prediction. This can be a numeric vector or a one-sided model formula. In the latter case, it is interpreted as an expression evaluated in newdata
...	further arguments passed to or from other methods.

Details

Note that this lmrob method for predict is closely modeled after the method for lm(), predict.lm, maybe see there for caveats with missing value treatment. The prediction intervals are for a single observation at each case in newdata (or by default, the data used for the fit) with error variance(s) pred.var. This can be a multiple of res.var, the estimated value of σ^2: the default is to assume that future observations have the same error variance as those used for fitting. If weights is supplied, the inverse of this is used as a scale factor. For a weighted fit, if the prediction is for the original data frame, weights defaults to the weights used for the model fit, with a warning since it might not be the intended result. If the fit was weighted and newdata is given, the default is to assume constant prediction variance, with a warning.

Value

predict.lmrob produces a vector of predictions or a matrix of predictions and bounds with column names fit, lwr, and upr if interval is set. If se.fit is TRUE, a list with the following components is returned:

fit	vector or matrix as above
se.fit	standard error of predicted means
residual.scale	residual standard deviations
df	degrees of freedom for residual

Author(s)

Andreas Ruckstuhl

See Also

`lmrob` and the (non-robust) traditional `predict.lm` method.

Examples

```
## Predictions --- artificial example -- closely following  example(predict.lm)

set.seed(5)
n <- length(x <- sort(c(round(rnorm(25), 1), 20)))
y <- x + rnorm(n)
iO <- c(sample(n-1, 3), n)
y[iO] <- y[iO] + 10*rcauchy(iO)

p.ex <- function(...) {
  plot(y ~ x, ...); abline(0,1, col="sky blue")
  points(y ~ x, subset=iO, col="red", pch=2)
  abline(lm   (y ~ x), col = "gray40")
  abline(lmrob(y ~ x), col = "forest green")
  legend("topleft", c("true", "Least Squares", "robust"),
         col = c("sky blue", "gray40", "forest green"), lwd=1.5, bty="n")
}
p.ex()

fm <- lmrob(y ~ x)
predict(fm)
new <- data.frame(x = seq(-3, 10, 0.25))
str(predict(fm, new, se.fit = TRUE))
pred.w.plim <- predict(fm, new, interval = "prediction")
pred.w.clim <- predict(fm, new, interval = "confidence")
pmat <- cbind(pred.w.clim, pred.w.plim[,-1])

matlines(new$x, pmat, lty = c(1,2,2,3,3))# add to first plot
## show zoom-in region :
rect(xleft = -3, ybottom = -20, xright = 10, ytop = 40,
     lty = 3, border="orange4")

## now zoom in :
p.ex(xlim = c(-3,10), ylim = c(-20, 40))
matlines(new$x, pmat, lty = c(1,2,2,3,3))
box(lty = 3, col="orange4", lwd=3)
legend("bottom", c("fit", "lwr CI", "upr CI", "lwr Pred.I", "upr Pred.I"),
       col = 1:5, lty=c(1,2,2,3,3), bty="n")

## Prediction intervals, special cases
##  The first three of these throw warnings
w <- 1 + x^2
fit <- lmrob(y ~ x)
wfit <- lmrob(y ~ x, weights = w)
```

```
predict(fit,       interval = "prediction")
predict(wfit,      interval = "prediction")
predict(wfit, new, interval = "prediction")
predict(wfit, new, interval = "prediction", weights = (new$x)^2) -> p.w2
p.w2
stopifnot(identical(p.w2, ## the same as using formula:
     predict(wfit, new, interval = "prediction", weights = ~x^2)))
```

`print.lmrob` *Print Method for Objects of Class "lmrob"*

Description

Print method for elements of class `"lmrob"`.

Usage

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

Arguments

`x`	an R object of class `lmrob`, typically created by `lmrob`.
`digits`	number of digits for printing, see `digits` in `options`.
`...`	potentially more arguments passed to methods.

See Also

`lmrob`, `summary.lmrob`, `print` and `summary`.

Examples

```
data(coleman)
( m1 <- lmrob(Y ~ ., data=coleman) ) # -> print.lmrob() method
```

`psi.findc` *Find Tuning Constant(s) for "lqq" and "ggw" Psi Functions*

Description

Find psi function tuning constant sets for `"LQQ"` and `"GGW"` psi (ψ) functions by specifying largest descent (minimal slope), efficiency and or breakdown point.

`.psi.const()` is called from `lmrob.control()` to set the tuning constants for psi and chi for `"LQQ"` and `"GGW"` psi. Unless the specified tuning constants are from fixed small set where the computations are stored precomputed, `.psi.const()` calls the corresponding `.psi.<psi>.findc()`.

Usage

```
.psi.ggw.findc(ms, b, eff = NA, bp = NA,
               subdivisions = 100L,
               rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol,
               tol = .Machine$double.eps^0.25, ms.tol = tol/64, maxiter = 1000)

.psi.lqq.findc(ms, b.c, eff = NA, bp = NA,
               interval = c(0.1, 4), subdivisions = 100L,
               rel.tol = .Machine$double.eps^0.25, abs.tol = rel.tol,
               tol = .Machine$double.eps^0.25, maxiter = 1000)
.psi.const(cc, psi)
```

Arguments

ms	number, the minimal slope, typically negative.
b, b.c	number, specifying b or b/c for "ggw" or "lqq" respectively.
eff	a number (or NA), the desired *efficiency*, in $[0, 1]$ of the estimator. If NA, bp must be specified as valid number.
bp	a number (or NA), the desired ***b**reakdown **p**oint* of the estimator, in $[0, 1]$.
interval	for finding c via uniroot().
subdivisions	passed to integrate().
rel.tol, abs.tol	relative and absolute tolerance for integrate().
tol	relative tolerance for uniroot().
ms.tol	relative tolerance for the internal .psi.ggw.finda(), eventually passed to optimize inside (internal) .psi.ggw.mxs().
maxiter	maximal number of iterations for uniroot().
cc	(for .psi.const():) numeric vector of length 4, containing all constants c(ms, b*, eff, bp), where b* = b for "ggw" and b* = b.c for "lqq", and one of (eff, bp) is NA.
psi	a string, either "ggw" or "lqq".

Details

For some important special cases, the result of .psi.*.findc() are stored precomputed for efficiency reasons. These cases are (the defaults for tuning.chi and tuning.psi respectively in lmrob.control()s result,

tuning.chi	tuning.psi
c(-0.5, 1.5, NA, 0.5)	c(-0.5, 1.5, 0.95, NA)

and for "ggw" additionally, these four cases:

tuning.chi	tuning.psi

	`c(-0.5, 1.5, 0.85, NA)`
`c(-0.5, 1, NA, 0.5)`	`c(-0.5, 1, 0.95, NA)`
	`c(-0.5, 1, 0.85, NA)`

Note that for `"ggw"`, exactly these $2 + 4 = 6$ cases also allow fast ρ and χ (aka $\tilde{\rho}(\cdot)$, see `Mchi`), function evaluations. For all other tuning constant settings, rho() evaluations are based on numerical integration via R's own `Rdqags()` C function (part of R's official API).

Value

a `numeric` vector of constants, for `"lqq"` or `"ggw"` psi functions, respectively:

`"lqq"`: $(b, c, s) = (b/c * c, c, s = 1 - min_s lope)$,

`"ggw"`: $(0, a, b, c, \rho(\infty))$.

`.psi.const(cc, psi)` returns the argument `cc` with the above constant vectors as attribute `"constants"`, in the case of `psi = "lqq"` in all cases (since **robustbase** version >= 0.93), for `psi = "ggw"` only in the non-standard cases.

Author(s)

Manuel Koller (original) and Martin Maechler (arguments, export, docs).

References

See the vignette about "ψ-Functions Available in Robustbase".

See Also

`Mpsi()` etc for the psi function definitions; `.Mpsi.tuning.defaults`, etc, for tuning constants' defaults for `lmrob()`.

Examples

```
(c.ge95 <- .psi.ggw.findc(ms = -0.5, b = 1.5, eff = 0.95))
(c.ge90 <- .psi.ggw.findc(ms = -0.5, b = 1.5, eff = 0.90))
(c.gb50 <- .psi.ggw.findc(ms = -0.5, b = 1.5,  bp = 0.50))
stopifnot(all.equal(c.ge95, c(0, 1.386362,   1.5, 1.0628199,  4.7773893), tol = 1e-5),
          all.equal(c.ge90, c(0, 1.0282811,  1.5, 0.87086259, 3.2075233), tol = 1e-5),
          all.equal(c.gb50, c(0, 0.20367394, 1.5, 0.29591308, 0.37033962),tol = 1e-5))

(cl.e.95 <- .psi.lqq.findc(ms = -0.5, b.c = 1.5, eff = .95))
(cl.b.50 <- .psi.lqq.findc(ms = -0.5, b.c = 1.5,  bp = .50))
stopifnot(all.equal(cl.e.95, c(1.4734061,  0.98227073, 1.5), tol = 1e-5),
          all.equal(cl.b.50, c(0.40154568, 0.26769712, 1.5), tol = 1e-5))
```

psiFunc *Constructor for Objects "Psi Function" Class*

Description

psiFunc(..) is a convenience interface to new("psi_func",..), i.e. for constructing objects of class "psi_func".

Usage

```
psiFunc(rho, psi, wgt, Dpsi,Dwgt, Erho = NULL, Epsi2 = NULL, EDpsi = NULL, name, ...)

huberPsi
hampelPsi
```

Arguments

rho, psi, wgt, Dpsi, Dwgt	each a function of x and tuning parameters typically. Specification of Dwgt is optional.
Erho, Epsi2, EDpsi	see psi_func, and note that these may change in the future.
name	Name of ψ-function used for printing.
...	potential further arguments for specifying tuning parameter names and defaults.

Author(s)

Martin Maechler

See Also

The description of class psi_func.

Examples

```
plot(huberPsi) # => shows "all" {as an object with a smart plot() method}

## classical (Gaussian / "least-squares") psi {trivial}:
F1 <- function(x, .) rep.int(1, length(x))
FF <- function(.) rep.int(1, length(.))
cPsi <- psiFunc(rho = function(x,.) x^2 / 2, psi = function(x, .) x,
                wgt = F1, Dpsi = F1,
                Erho = function(.) rep.int(1/2, length(.)),
                Epsi2 = FF, EDpsi = FF, name = "classic", . = Inf)
show(cPsi)
plot(cPsi)
## is the same as the limit of Huber's:
plot(chgDefaults(huberPsi, k = Inf))
```

```
## Hampel's psi and rho:
H.38 <- chgDefaults(hampelPsi, k = c(1.5, 3.5, 8))
k. <- H.38@xtras$tuningP$k ; k.. <- as.vector(outer(c(-1,1), k.))
c.t <- adjustcolor("skyblue3", .8)
.ax.k <- function(side) { abline(h=0, v=0, lty=2)
  axis(side, at = k.., labels=formatC(k..), pos=0, col=c.t, col.axis=c.t) }
op <- par(mfrow=c(2,1), mgp = c(1.5, .6, 0), mar = .6+c(2,2,1,.5))
curve(H.38@psi(x), -10, 10, col=2, lwd=2, n=512)
lines(k.., H.38@psi(k..), type = "h", lty=3, col=c.t); .ax.k(1)
curve(H.38@rho(x), -10, 10, col=2, lwd=2, n=512); abline(h=0, v=0, lty=2)
lines(k.., H.38@rho(k..), type = "h", lty=3, col=c.t); .ax.k(1)
title(expression("Hampel's " ~~~ psi(x) ~~ "and" ~~ rho(x) ~~~ " functions"))
par(op)

## Not the same, but similar, directly using the plot() method:
plot(H.38)
```

---

psi_func-class *Class of "Psi Functions" for M-Estimation*

Description

The class "psi_func" is used to store ψ (psi) functions for M-estimation. In particular, an object of the class contains $\rho(x)$ (rho), its derivative $\psi(x)$ (psi), the weight function $\psi(x)/x$, and first derivative of ψ, Dpsi = $\psi'(x)$.

Objects from the Class

Objects can be created by calls of the form new("psi_func", ...), but preferably by psiFunc(...).

Slots

rho: the $\rho()$ function, an object of class "functionX". This is used to formulate the objective function; $\rho()$ can be regarded as generalized negative log-likelihood.

psi: $\psi()$ is the derivative of ρ, $\psi(x) = \frac{d}{dx}\rho(x)$; also of class "functionX".

wgt: The weight function $\psi(x)/x$, of class "functionX".

Dpsi: the derivative of ψ, $Dpsi(x) = psi'(x)$; of class "functionX".

Dwgt: the derivative of the weight function, of class "functionX", is generated automatically if psiFunc constructor is used.

tDefs: *named* numeric vector of **t**uning parameter **Def**ault values.

Erho: A function of class "functionXal" for computing $E[\rho(X)]$ when X is standard normal $\mathcal{N}(0,1)$.

Epsi2: A function of class "functionXal" for computing $E[\psi^2(X)]$ when X is standard normal.

EDpsi: A function of class "functionXal" for computing $E[\psi'(X)]$ when X is standard normal.

name: Name of ψ-function used for printing.

xtras: Potentially further information.

Methods

Currently, only chgDefaults(), plot() and show().

Author(s)

Martin Maechler

See Also

psiFunc.

Examples

```
str(huberPsi, give.attr = FALSE)

plot(hampelPsi)# calling the plot method (nicely showing "all" !)
```

---

pulpfiber *Pulp Fiber and Paper Data*

Description

Measurements of aspects pulp fibers and the paper produced from them. Four properties of each are measured in sixty-two samples.

Usage

```
data(pulpfiber, package="robustbase")
```

Format

A data frame with 62 observations on the following 8 variables.

- X1 numeric vector of arithmetic fiber length
- X2 numeric vector of long fiber fraction
- X3 numeric vector of fine fiber fraction
- X4 numeric vector of zero span tensile
- Y1 numeric vector of breaking length
- Y2 numeric vector of elastic modulus
- Y3 numeric vector of stress at failure
- Y4 numeric vector of burst strength

Details

Cited from the reference article: *The dataset contains measurements of properties of pulp fibers and the paper made from them. The aim is to investigate relations between pulp fiber properties and the resulting paper properties. The dataset contains $n = 62$ measurements of the following four pulp fiber characteristics: arithmetic fiber length, long fiber fraction, fine fiber fraction, and zero span tensile. The four paper properties that have been measured are breaking length, elastic modulus, stress at failure, and burst strength.*

The goal is to predict the $q = 4$ paper properties from the $p = 4$ fiber characteristics.

Author(s)

port to R and this help page: Martin Maechler

Source

Rousseeuw, P. J., Van Aelst, S., Van Driessen, K., and Agulló, J. (2004) Robust multivariate regression; *Technometrics* **46**, 293–305.

Till 2016 available from `http://users.ugent.be/~svaelst/data/pulpfiber.txt`

References

Lee, J. (1992) *Relationships Between Properties of Pulp-Fibre and Paper*, unpublished doctoral thesis, U. Toronto, Faculty of Forestry.

Examples

```
data(pulpfiber)
str(pulpfiber)

pairs(pulpfiber, gap=.1)
## 2 blocks of 4 ..
c1 <- cov(pulpfiber)
cR <- covMcd(pulpfiber)
## how different are they: The robust estimate has more clear high correlations:
symnum(cov2cor(c1))
symnum(cov2cor(cR$cov))
```

`Qn` *Robust Location-Free Scale Estimate More Efficient than MAD*

Description

Compute the robust scale estimator Q_n, an efficient alternative to the MAD.

By default, $Q_n(x_1, \ldots, x_n)$ is the k-th order statistic (a quantile) of the `choose(n, 2)` absolute differences $|x_i - x_j|$, (for $1 \le i < j \le n$), where by default (originally only possible value) $k = choose(n\%/\%2 + 1, 2)$ which is about the first quartile (25% quantile) of these pairwise differences. See the references for more.

Usage

```
Qn(x, constant = NULL, finite.corr = is.null(constant) && missing(k),
   na.rm = FALSE, k = choose(n %/% 2 + 1, 2), warn.finite.corr = TRUE)

s_Qn(x, mu.too = FALSE, ...)
```

Arguments

x	numeric vector of observations.
constant	number by which the result is multiplied; the default achieves consistency for normally distributed data. Note that until Nov. 2010, "thanks" to a typo in the very first papers, a slightly wrong default constant, 2.2219, was used instead of the correct one which is equal to `1 / (sqrt(2) * qnorm(5/8))` (as mentioned already on p.1277, after (3.7) in Rousseeuw and Croux (1993)). If you need the old slightly off version for historical reproducibility, you can use `Qn.old()`. Note that the relative difference is only about 1 in 1000, and that the correction should not affect the finite sample corrections for $n \leq 9$.
finite.corr	logical indicating if the finite sample bias correction factor should be applied. Defaults to `TRUE` unless `constant` is specified. Note the for non-default `k`, the consistency `constant` already depends on `n` leading to *some* finite sample correction, but no simulation-based small-`n` correction factors are available.
na.rm	logical specifying if missing values (`NA`) should be removed from `x` before further computation. If false as by default, and if there are `NA`s, i.e., `if(anyNA(x))`, the result will be `NA`.
k	integer, typically half of n, specifying the "quantile", i.e., rather the order statistic that `Qn()` should return; for the Qn() proper, this has been hard wired to `choose(n%/%2 +1, 2)`, i.e., $\lfloor \frac{n}{2} \rfloor + 1$. Choosing a large `k` is less robust but allows to get non-zero results in case the default `Qn()` is zero.
warn.finite.corr	logical indicating if a `warning` should be signalled when `k` is non-default, in which case specific small-n correction is not yet provided.
mu.too	logical indicating if the `median(x)` should also be returned for `s_Qn()`.
...	potentially further arguments for `s_Qn()` passed to `Qn()`.

Details

As the (default, consistency) constant needed to be corrected, the finite sample correction has been based on a much more extensive simulation, and on a 3rd or 4th degree polynomial model in $1/n$ for odd or even n, respectively.

Value

`Qn()` returns a number, the Q_n robust scale estimator, scaled to be consistent for σ^2 and i.i.d. Gaussian observations, optionally bias corrected for finite samples.

`s_Qn(x, mu.too=TRUE)` returns a length-2 vector with location (μ) and scale; this is typically only useful for `covOGK(*, sigmamu = s_Qn)`.

Author(s)

Original Fortran code: Christophe Croux and Peter Rousseeuw <rousse@wins.uia.ac.be>.
Port to C and R: Martin Maechler, <maechler@R-project.org>

References

Rousseeuw, P.J. and Croux, C. (1993) Alternatives to the Median Absolute Deviation, *Journal of the American Statistical Association* **88**, 1273–1283. doi:10.2307/2291267

Christophe Croux and Peter J. Rousseeuw (1992) A class of high-breakdown scale estimators based on subranges , *Communications in Statistics - Theory and Methods* **21**, 1935–1951; doi:10.1080/03610929208830889

Christophe Croux and Peter J. Rousseeuw (1992) Time-Efficient Algorithms for Two Highly Robust Estimators of Scale, *Computational Statistics, Vol. 1*, ed. Dodge and Whittaker, Physica-Verlag Heidelberg, 411–428; available via Springer Link.

About the typo in the `constant`:
Christophe Croux (2010) Private e-mail, Fri Jul 16, w/ Subject *Re: Slight inaccuracy of Qn implementation* .....

See Also

`mad` for the 'most robust' but much less efficient scale estimator; `Sn` for a similar faster but less efficient alternative. Finally, `scaleTau2` which some consider "uniformly" better than Qn or competitors.

Examples

```
set.seed(153)
x <- sort(c(rnorm(80), rt(20, df = 1)))
s_Qn(x, mu.too = TRUE)
Qn(x, finite.corr = FALSE)

## A simple pure-R version of Qn() -- slow and memory-rich for large n: O(n^2)
Qn0R <- function(x, k = choose(n %/% 2 + 1, 2)) {
    n <- length(x <- sort(x))
    if(n == 0) return(NA) else if(n == 1) return(0.)
    stopifnot(is.numeric(k), k == as.integer(k), 1 <= k, k <= n*(n-1)/2)
    m <- outer(x,x,"-")# abs not needed as x[] is sorted
    sort(m[lower.tri(m)], partial = k)[k]
}
(Qx1 <- Qn(x, constant=1)) # 0.5498463
## the C-algorithm "rounds" to 'float' single precision ..
stopifnot(all.equal(Qx1, Qn0R(x), tol = 1e-6))


(qn <- Qn(c(1:4, 10, Inf, NA), na.rm=TRUE))
stopifnot(is.finite(qn), all.equal(4.075672524, qn, tol=1e-10))

## -- compute for different 'k' :

n <- length(x) # = 100 here
```

```
(k0 <- choose(floor(n/2) + 1, 2)) # 51*50/2 == 1275
stopifnot(identical(Qx1, Qn(x, constant=1, k=k0)))
nn2 <- n*(n-1)/2
all.k <- 1:nn2
system.time(Qss <- sapply(all.k, function(k) Qn(x, 1, k=k)))
system.time(Qs  <- Qn  (x, 1, k = all.k))
system.time(Qs0 <- Qn0R(x,    k = all.k) )
stopifnot(exprs = {
   Qs[1]   == min(diff(x))
   Qs[nn2] == diff(range(x))
   all.equal(Qs,  Qss, tol = 1e-15) # even exactly
   all.equal(Qs0, Qs, tol = 1e-7) # see 2.68e-8, as Qn() C-code rounds to (float)
})

plot(2:nn2, Qs[-1], type="b", log="y", main = "Qn(*, k),  k = 2..n(n-1)/2")
```

---

r6pack *Robust Distance based observation orderings based on robust "Six pack"*

Description

Compute six initial robust estimators of multivariate location and "scatter" (scale); then, for each, compute the distances d_{ij} and take the h ($h > n/2$) observations with smallest distances. Then compute the statistical distances based on these h observations.

Return the indices of the observations sorted in increasing order.

Usage

```
r6pack(x, h, full.h, scaled = TRUE, scalefn = rrcov.control()$scalefn)
```

Arguments

x	n x p data matrix
h	integer, typically around (and slightly larger than) $n/2$.
full.h	logical specifying if the full (length n) observation ordering should be returned; otherwise only the first h are. For `.detmcd()`, `full.h=FALSE` is typical.
scaled	logical indicating if the data x is already scaled; if false, we apply `x <- doScale(x, median, scalefn)`.
scalefn	a `function(u)` to compute a robust univariate scale of u.

Details

The six initial estimators are

1. Hyperbolic tangent of standardized data
2. Spearmann correlation matrix

3. Tukey normal scores
4. Spatial sign covariance matrix
5. BACON
6. Raw OGK estimate for scatter

Value

a $h' \times 6$ matrix of observation indices, i.e., with values from $1, \ldots, n$. If `full.h` is true, $h' = n$, otherwise $h' = h$.

Author(s)

Valentin Todorov, based on the original Matlab code by Tim Verdonck and Mia Hubert. Martin Maechler for tweaks (performance etc), and `full.h`.

References

Hubert, M., Rousseeuw, P. J. and Verdonck, T. (2012) A deterministic algorithm for robust location and scatter. Journal of Computational and Graphical Statistics **21**, 618–637.

See Also

`covMcd(*, nsamp = "deterministic")`; `CovSest(*, nsamp = "sdet")` from package **rrcov**.

Examples

```
data(pulpfiber)
dim(m.pulp <- data.matrix(pulpfiber)) #  62 x 8
dim(fr6  <- r6pack(m.pulp, h = 40, full.h= FALSE)) #  h x 6  = 40 x 6
dim(fr6F <- r6pack(m.pulp, h = 40, full.h= TRUE )) #  n x 6  = 62 x 6
stopifnot(identical(fr6, fr6F[1:40,]))
```

---

radarImage *Satellite Radar Image Data from near Munich*

Description

The data were supplied by A. Frery. They are a part of a synthetic aperture satellite radar image corresponding to a suburb of Munich. Provided are coordinates and values corresponding to three frequency bands for each of 1573 pixels.

Usage

```
data(radarImage, package="robustbase")
```

Format

A data frame with 1573 observations on the following 5 variables.

`X.coord` a numeric vector

`Y.coord` a numeric vector

`Band.1` a numeric vector

`Band.2` a numeric vector

`Band.3` a numeric vector

Source

The website accompanying the MMY-book: https://www.wiley.com/legacy/wileychi/robust_statistics/

Examples

```
data(radarImage)
plot(Y.coord ~ X.coord, data = radarImage)

## The 8 "clear" outliers (see also below)
ii8 <- c(1548:1549, 1553:1554, 1565:1566, 1570:1571)
outF <- 1+(seq_len(nrow(radarImage)) %in% ii8)
pairs(radarImage[, 3:5], main = "radarImage (n = 1573)",
      col = outF, pch=outF)

## Finding outliers ----------------------------------------

set.seed(1)
system.time(cc.ri <- covMcd(radarImage))# ~ 0.1 sec
## check for covMcd() consistency:
ii0 <- as.integer(
 c(262, 450:451, 480:481, 509, 535, 542, 597, 643, 669, 697, 803:804, 832:834,
   862:864, 892, 989, 1123, 1145, 1223:1224, 1232:1233, 1249:1250, 1267, 1303,
   1347, 1357, 1375, 1411, 1419:1420, 1443, 1453, 1504, 1510:1512,
   1518:1521, 1525:1526, 1543:1544, 1546:1555, 1557:1558, 1561:1562, 1564:1566,
   1569:1571, 1573))
length(ii0) # 73 -- other seeds sometimes give 72, rarely 71 "outliers"
table(is0 <- cc.ri$mcd.wt == 0) # 2023-05: 118
stopifnot(exprs = {
    ## identical(ii0, which(is0)) -- TRUE before 2023-05  covMcd() change
    ii8 %in% which(is0) # ii8 is subset of is0
    identical(ii8, which(cc.ri$mah > 200))
    length(intersect(cc.ri$best, ii0)) == 0
})

cc <- c(adjustcolor("black", 0.4), adjustcolor("tomato", 0.8))
pairs(radarImage, main = "radarImage (n = 1573) + Outliers", gap=0,
      col = cc[1+is0], pch = c(1,8)[1+is0], cex = 0.8)
```

rankMM *Simple Matrix Rank*

Description

Compute the rank of a matrix A in simple way, based on the SVD, svd(), and "the same as Matlab".

Usage

```
rankMM(A, tol = NULL, sv = svd(A, 0, 0)$d)
```

Arguments

A	a numerical matrix, maybe non-square. When sv is specified, only dim(A) is made use of.
tol	numerical tolerance (compared to singular values). By default, when NULL, the tolerance is determined from the maximal value of sv and the computer epsilon.
sv	vector of *non-increasing* singular values of A, (to be passed if already known).

Value

an integer from the set 0:min(dim(A)).

Author(s)

Martin Maechler, Date: 7 Apr 2007

See Also

There are more sophisticated proposals for computing the rank of a matrix; for a couple of those, see rankMatrix in the **Matrix** package.

Examples

```
rankMM # - note the simple function definition

hilbert <- function(n) { i <- seq_len(n); 1/outer(i - 1L, i, "+") }
hilbert(4)
H12 <- hilbert(12)
rankMM(H12)        # 11 - numerically more realistic
rankMM(H12, tol=0) # -> 12
## explanation :
round(log10(svd(H12, 0,0)$d), 1)
```

`residuals.glmrob` *Residuals of Robust Generalized Linear Model Fits*

Description

Compute residuals of a fitted `glmrob` model, i.e., robust generalized linear model fit.

Usage

```
## S3 method for class 'glmrob'
residuals(object,
          type = c("deviance", "pearson", "working",
                   "response", "partial"),
          ...)
```

Arguments

`object`	an object of class `glmrob`, typically the result of a call to `glmrob`.
`type`	the type of residuals which should be returned. The alternatives are: `"deviance"` (default), `"pearson"`, `"working"`, `"response"`, and `"partial"`.
`...`	further arguments passed to or from other methods.

Details

The references in `glm` define the types of residuals: Davison & Snell is a good reference for the usages of each.

The partial residuals are a matrix of working residuals, with each column formed by omitting a term from the model.

The `residuals` (S3) method (see `methods`) for `glmrob` models has been modeled to follow closely the method for classical (non-robust) `glm` fitted models. Possibly, see its documentation, i.e., residuals.glm, for further details.

References

See those for the classical GLM's, `glm`.

See Also

`glmrob` for computing `object`, `anova.glmrob`; the corresponding *generic* functions, `summary.glmrob`, `coef`, `fitted`, `residuals`.

Examples

```
### -------- Gamma family -- data from example(glm) ---
clotting <- data.frame(
            u = c(5,10,15,20,30,40,60,80,100),
         lot1 = c(118,58,42,35,27,25,21,19,18),
         lot2 = c(69,35,26,21,18,16,13,12,12))
summary(cl <- glm   (lot1 ~ log(u), data=clotting, family=Gamma))
summary(ro <- glmrob(lot1 ~ log(u), data=clotting, family=Gamma))
clotM5.high <- within(clotting, { lot1[5] <- 60 })
cl5.high <- glm   (lot1 ~ log(u), data=clotM5.high, family=Gamma)
ro5.high <- glmrob(lot1 ~ log(u), data=clotM5.high, family=Gamma)
rr <- range(residuals(ro), residuals(cl), residuals(ro5.high))
plot(residuals(ro5.high) ~ residuals(cl5.high), xlim = rr, ylim = rr, asp = 1)
abline(0,1, col=2, lty=3)
points(residuals(ro) ~ residuals(cl), col = "gray", pch=3)

## Show all kinds of residuals:
r.types <- c("deviance", "pearson", "working", "response")
sapply(r.types, residuals, object = ro5.high)
```

---

rrcov.control *Control Settings for covMcd and ltsReg*

Description

Auxiliary function for passing the estimation options as parameters to the estimation functions.

Usage

```
rrcov.control(alpha = 1/2, method = c("covMcd", "covComed", "ltsReg"),
              nsamp = 500, nmini = 300, kmini = 5,
              seed = NULL, tolSolve = 1e-14,
              scalefn = "hrv2012", maxcsteps = 200,
              trace = FALSE,
              wgtFUN = "01.original", beta,
              use.correction = identical(wgtFUN, "01.original"),
              adjust = FALSE)
```

Arguments

alpha	This parameter controls the size of the subsets over which the determinant is minimized, i.e., alpha*n observations are used for computing the determinant. Allowed values are between 0.5 and 1 and the default is 0.5.
method	a string specifying the "main" function for which rrcov.control() is used. This currently only makes a difference to determine the default for beta.

nsamp	number of subsets used for initial estimates or "best" or "exact". Default is nsamp = 500. If nsamp="best" exhaustive enumeration is done, as far as the number of trials do not exceed 5000. If nsamp="exact" exhaustive enumeration will be attempted however many samples are needed. In this case a warning message will be displayed saying that the computation can take a very long time.
nmini, kmini	for covMcd: For large n, the algorithm splits the data into maximally $kmini$ subsets of targetted size nmini. See covMcd for more details.
seed	initial seed for R's random number generator; see .Random.seed and the description of the seed argument in lmrob.control.
tolSolve	numeric tolerance to be used for inversion (solve) of the covariance matrix in mahalanobis.
scalefn	(for deterministic covMcd():) a character string or function for computing a robust scale estimate. The current default "hrv2012" uses the recommendation of Hubert et al (2012); see covMcd for more.
maxcsteps	integer specifying the maximal number of concentration steps for the deterministic MCD.
trace	logical or integer indicating whether to print intermediate results. Default is trace = FALSE.
wgtFUN	a character string or function, specifying how the weights for the reweighting step should be computed, see ltsReg, covMcd or covComed, respectively. The default is specified by "01.original", as the resulting weights are 0 or 1. Alternative string specifications need to match names(.wgtFUN.covComed) - which currently is experimental.
beta	a quantile, experimentally used for some of the prespecified wgtFUNs, see e.g., .wgtFUN.covMcd and .wgtFUN.covComed.
use.correction	whether to use finite sample correction factors. Defaults to TRUE.
adjust	(for ltsReg():) whether to perform intercept adjustment at each step. Because this can be quite time consuming, the default is adjust = FALSE.

Value

A list with components, as the parameters passed by the invocation

Author(s)

Valentin Todorov

See Also

For details, see the documentation about ltsReg and covMcd, respectively.

Examples

```
data(Animals, package = "MASS")
brain <- Animals[c(1:24, 26:25, 27:28),]
```

```
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])

ctrl <- rrcov.control(alpha=0.75, trace=TRUE)
covMcd(hbk.x,      control = ctrl)
covMcd(log(brain), control = ctrl)
```

`salinity` *Salinity Data*

Description

This is a data set consisting of measurements of water salinity (i.e., its salt concentration) and river discharge taken in North Carolina's Pamlico Sound, recording some bi-weekly averages in March, April, and May from 1972 to 1977. This dataset was listed by Ruppert and Carroll (1980). In Carrol and Ruppert (1985) the physical background of the data is described. They indicated that observations 5 and 16 correspond to periods of very heavy discharge and showed that the discrepant observation 5 was masked by observations 3 and 16, i.e., only after deletion of these observations it was possible to identify the influential observation 5.

This data set is a prime example of the *masking effect*.

Usage

```
data(salinity, package="robustbase")
```

Format

A data frame with 28 observations on the following 4 variables (in parentheses are the names used in the 1980 reference).

`X1`: Lagged Salinity ('SALLAG')

`X2`: Trend ('TREND')

`X3`: Discharge ('H2OFLOW')

`Y`: Salinity ('SALINITY')

Note

The **boot** package contains another version of this salinity data set, also attributed to Ruppert and Carroll (1980), but with two clear transcription errors, see the examples.

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.82, table 5.

Ruppert, D. and Carroll, R.J. (1980) Trimmed least squares estimation in the linear model. *JASA* **75**, 828–838; table 3, p.835.

Carroll, R.J. and Ruppert, D. (1985) Transformations in regression: A robust analysis. *Technometrics* **27**, 1–12

Examples

```
data(salinity)
summary(lm.sali  <-        lm(Y ~ . , data = salinity))
summary(rlm.sali <- MASS::rlm(Y ~ . , data = salinity))
summary(lts.sali <-    ltsReg(Y ~ . , data = salinity))

salinity.x <- data.matrix(salinity[, 1:3])
c_sal <- covMcd(salinity.x)
plot(c_sal, "tolEllipsePlot")

## Connection with boot package's version :
if(requireNamespace("boot")) { ## 'always'
 print( head(boot.sal <- boot::salinity        ) )
 print( head(robb.sal <- salinity [, c(4, 1:3)]) ) # difference: has one digit more
 ## Otherwise the same ?
 dimnames(robb.sal) <- dimnames(boot.sal)
 ## apart from the 4th column, they are "identical":
 stopifnot( all.equal(boot.sal[, -4], robb.sal[, -4], tol = 1e-15) )

 ## But the discharge ('X3', 'dis' or 'H2OFLOW')  __differs__ in two places:
 plot(cbind(robustbase = robb.sal[,4], boot = boot.sal[,4]))
 abline(0,1, lwd=3, col=adjustcolor("red", 1/4))
 D.sal <- robb.sal[,4] - boot.sal[,4]
 stem(robb.sal[,4] - boot.sal[,4])
 which(abs(D.sal) > 0.01) ## 2 8
 ## *two* typos (=> difference ~= 1) in the version of 'boot': obs. 2 & 8 !!!
 cbind(robb = robb.sal[,4], boot = boot.sal[,4], D.sal)
}# boot
```

---

scaleTau2 *Robust Tau-Estimate of Scale*

Description

Computes the robust τ-estimate of univariate scale, as proposed by Maronna and Zamar (2002); improved by a consistency factor,

Usage

```
scaleTau2(x, c1 = 4.5, c2 = 3.0, na.rm = FALSE, consistency = TRUE,
          mu0 = median(x),
          sigma0 = median(x.), mu.too = FALSE, iter = 1, tol.iter = 1e-7)
```

Arguments

x	numeric vector
c1, c2	non-negative numbers, specifying cutoff values for the biweighting of the mean and the rho function respectively.

na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
consistency	logical indicating if the consistency correction factor (for the scale) should be applied.
mu0	the initial location estimate μ_0, defaulting to the median.
sigma0	the initial scale estimate s_0, defaulting to the MAD; may be set to a positive value when the MAD is zero.
mu.too	logical indicating if both location and scale should be returned or just the scale (when mu.too=FALSE as by default).
iter	positive integer or logical indicating if and how many iterations should be done. The default, iter = 1 computes the "traditional" tau-estimate of scale.
tol.iter	if iter is true, or iter > 1, stop the iterations when $\lvert s_n - s_o \rvert \le \epsilon s_n$, where $\epsilon :=$tol.iter, and s_o and s_n are the previous and current estimates of σ.

Details

First, s_0 := MAD, i.e. the equivalent of mad(x, constant=1) is computed. Robustness weights $w_i := w_{c1}((x_i - med(X))/s_0)$ are computed, where $w_c(u) = max(0, (1-(u/c)^2)^2)$. The robust location measure is defined as $\mu(X) := (\sum_i w_i x_i)/(\sum_i w_i)$, and the robust $\tau(tau)$-estimate is $s(X)^2 := s_0^2 * (1/n) \sum_i \rho_{c2}((x_i - \mu(X))/s_0)$, where $\rho_c(u) = min(c^2, u^2)$.
When iter=TRUE or iter > 1, the above estimate is *iterated* in a fixpoint iteration, setting s_0 to the current estimate $s(X)$ and iterating until the number of iterations is larger than iter or the fixpoint is found in the sense that \
scaleTau2(*, consistency=FALSE) returns $s(X)$, whereas this value is divided by its asymptotic limit when consistency = TRUE as by default.

Note that for n = length(x) == 2, all equivariant scale estimates are proportional, and specifically, scaleTau2(x, consistency=FALSE) == mad(x, constant=1). See also the reference.

Value

numeric vector of length one (if mu.too is FALSE as by default) or two (when mu.too = TRUE) with robust scale or (location,scale) estimators $\hat{\sigma}(x)$ or $(\hat{\mu}(x), \hat{\sigma}(x))$.

Author(s)

Original by Kjell Konis with substantial modifications by Martin Maechler.

References

Maronna, R.A. and Zamar, R.H. (2002) Robust estimates of location and dispersion of high-dimensional datasets; *Technometrics* **44**(4), 307–317.

Yohai, V.J., and Zamar, R.H. (1988). High breakdown-point estimates of regression by means of the minimization of an efficient scale. *Journal of the American Statistical Association* **83**, 406–413.

See Also

Sn, Qn, mad; further covOGK for which scaleTau2 was designed.

Examples

```
x <- c(1:7, 1000)
sd(x) # non-robust std.deviation
scaleTau2(x)
scaleTau2(x, mu.too = TRUE)
(sI  <- scaleTau2(c(x,Inf), mu.too = TRUE))
(sIN <- scaleTau2(c(x,Inf,NA), mu.too = TRUE, na.rm=TRUE))
stopifnot({
  identical(sI, sIN)
  all.equal(scaleTau2(c(x, 999), mu.too = TRUE), sIN,
            tol = 1e-15)
})

if(doExtras <- robustbase:::doExtras()) {
 set.seed(11)
 ## show how much faster this is, compared to Qn
 x <- sample(c(rnorm(1e6), rt(5e5, df=3)))
 (system.time(Qx <- Qn(x)))         ## 2.04 [2017-09, lynne]
 (system.time(S2x <- scaleTau2(x))) ## 0.25    (ditto)
 cbind(Qn = Qx, sTau2 = S2x)
}##       Qn    sTau2
##  1.072556 1.071258
```

`SiegelsEx` *Siegel's Exact Fit Example Data*

Description

A small counterexample data set devised by Andrew Siegel. Six (out of nine) data points lie on the line $y = 0$ such that some robust regression estimators exhibit the "*exact fit*" property.

Usage

```
data(SiegelsEx, package="robustbase")
```

Format

A data frame with 9 observations on the following 2 variables.

`x` a numeric vector

`y` a numeric vector

Source

Emerson and Hoaglin (1983, p.139)

References

Peter J. Rousseeuw and Annick M. Leroy (1987) *Robust Regression and Outlier Detection* Wiley, p.60–61

Examples

```
data(SiegelsEx)
plot(SiegelsEx, main = "Siegel's example for 'exact fit'")
abline(          lm(y ~ x, data = SiegelsEx))
abline(MASS::lqs(y ~ x, data = SiegelsEx, method = "lms"), col = 2)
legend("topright", leg = c("lm", "LMS"), col=1:2, lwd=1, inset = 1/20)
```

---

sigma *Extract 'Sigma' - Standard Deviation of Errors for Robust Models*

Description

Extract the estimated standard deviation of the errors, the "residual standard deviation" (misnomed also "residual standard error") from a fitted model.

Usage

```
## S3 method for class 'lmrob'
sigma(object, ...)
```

Arguments

`object`	a fitted model.
`...`	additional, optional arguments. (None are used in our methods)

Details

For R <= `3.2.x`, we provide an (S3) generic function (as e.g., package **lme4**) and methods for `lmrob`, `nlrob`, and `nls`.

From R >= `3.3.0`, we provide methods for our `lmrob` and `nlrob` models.

Value

the residual standard error as a scalar

Examples

```
      m.cl <-   lm (Y ~ ., data=coleman)
if(getRversion() >= "3.3.0")  sigma(m.cl)  else  summary(m.cl)$sigma
sigma( m1  <- lmrob(Y ~ ., data=coleman) )
sigma( m2  <- lmrob(Y ~ ., data=coleman, setting = "KS2014") )
```

`smoothWgt` *Smooth Weighting Function - Generalized Biweight*

Description

"The Biweight on a Stick" — Compute a smooth (when $h > 0$) weight function typically for computing weights from large (robust) "distances" using a piecewise polynomial function which in fact is a 2-parameter generalization of Tukey's 1-parameter "biweight".

Usage

```
smoothWgt(x, c, h)
```

Arguments

`x`	numeric vector of abscissa values
`c`	"cutoff", a typically positive number.
`h`	"bandwidth", a positive number.

Details

Let $w(x; c, h) :=$`smoothWgt(x, c, h)`. Then,

$$w(x; c, h) := 0 \quad \text{if } |x| \geq c + h/2,$$

$$w(x; c, h) := 1 \quad \text{if } |x| \leq c - h/2,$$

$$w(x; c, h) := \big((1 - |x| - (c - h/2))^2\big)^2 \text{ if } c - h/2 < |x| < c + h/2,$$

`smoothWgt()` is *scale invariant* in the sense that

$$w(\sigma x; \sigma c, \sigma h) = w(x; c, h),$$

when $\sigma > 0$.

Value

a numeric vector of the same length as `x` with weights between zero and one. Currently all `attributes` including `dim` and `names` are dropped.

Author(s)

Martin Maechler

See Also

`Mwgt(.., psi = "bisquare")` of which `smoothWgt()` is a generalization, and `Mwgt(.., psi = "optimal")` which looks similar for larger `c` with its constant one part around zero, but also has only one parameter.

Examples

```
## a somewhat typical picture:
curve(smoothWgt(x, c=3, h=1), -5,7, n = 1000)

csW <- curve(smoothWgt(x, c=1/2, h=1), -2,2) # cutoff 1/2, bandwidth 1
## Show that the above is the same as
## Tukey's "biweight" or "bi-square" weight function:
bw <- function(x) pmax(0, (1 - x^2))^2
cbw <- curve(bw,                    col=adjustcolor(2, 1/2), lwd=2, add=TRUE)
cMw <- curve(Mwgt(x,c=1,"biweight"), col=adjustcolor(3, 1/2), lwd=2, add=TRUE)
stopifnot(## proving they are all the same:
   all.equal(csW, cbw, tol=1e-15),
   all.equal(csW, cMw, tol=1e-15))
```

Sn *Robust Location-Free Scale Estimate More Efficient than MAD*

Description

Compute the robust scale estimator S_n, an efficient alternative to the MAD.

Usage

```
Sn(x, constant = 1.1926, finite.corr = missing(constant), na.rm = FALSE)

s_Sn(x, mu.too = FALSE, ...)
```

Arguments

x	numeric vector of observations.
constant	number by which the result is multiplied; the default achieves consisteny for normally distributed data.
finite.corr	logical indicating if the finite sample bias correction factor should be applied. Default to TRUE unless constant is specified.
na.rm	logical specifying if missing values (NA) should be removed from x before further computation. If false as by default, and if there are NAs, i.e., if(anyNA(x)), the result will be NA.
mu.too	logical indicating if the median(x) should also be returned for s_Sn().
...	potentially further arguments for s_Sn() passed to Sn().

Details

..... FIXME .....

Value

`Sn()` returns a number, the S_n robust scale estimator, scaled to be consistent for σ^2 and i.i.d. Gaussian observations, optionally bias corrected for finite samples.

`s_Sn(x, mu.too=TRUE)` returns a length-2 vector with location (μ) and scale; this is typically only useful for `covOGK(*, sigmamu = s_Sn)`.

Author(s)

Original Fortran code: Christophe Croux and Peter Rousseeuw <rousse@wins.uia.ac.be>.
Port to C and R: Martin Maechler, <maechler@R-project.org>

References

Rousseeuw, P.J. and Croux, C. (1993) Alternatives to the Median Absolute Deviation, *Journal of the American Statistical Association* **88**, 1273–1283.

See Also

`mad` for the 'most robust' but much less efficient scale estimator; `Qn` for a similar more efficient but slower alternative; `scaleTau2`.

Examples

```
x <- c(1:10, 100+1:9)# 9 outliers out of 19
Sn(x)
Sn(x, c=1)# 9
Sn(x[1:18], c=1)# 9
set.seed(153)
x <- sort(c(rnorm(80), rt(20, df = 1)))
s_Sn(x, mu.too=TRUE)

(s <- Sn(c(1:4, 10, Inf, NA), na.rm=TRUE))
stopifnot(is.finite(s), all.equal(3.5527554, s, tol=1e-10))
```

---

splitFrame *Split Continuous and Categorical Predictors*

Description

Splits the design matrix into categorical and continuous predictors. Categorical variables are variables that are `factor`s, `ordered` factors, *or* `character`.

Usage

```
splitFrame(mf, x = model.matrix(mt, mf),
     type = c("f","fi", "fii"))
```

Arguments

mf	model frame (as returned by model.frame).
x	(optional) design matrix, defaulting to the derived model.matrix.
type	a character string specifying the split type (see details).

Details

Which split type is used can be controlled with the setting split.type in lmrob.control.

There are three split types. The only differences between the types are how interactions between categorical and continuous variables are handled. The extra types of splitting can be used to avoid *Too many singular resamples* errors.

Type "f", the default, assigns only the intercept, categorical and interactions of categorical variables to x1. Interactions of categorical and continuous variables are assigned to x2.

Type "fi" assigns also interactions between categorical and continuous variables to x1.

Type "fii" assigns not only interactions between categorical and continuous variables to x1, but also the (corresponding) continuous variables themselves.

Value

A list that includes the following components:

x1	design matrix containing only categorical variables
x1.idx	logical vectors of the variables considered categorical in the original design matrix
x2	design matrix containing the continuous variables

Author(s)

Manuel Koller

References

Maronna, R. A., and Yohai, V. J. (2000). Robust regression with both continuous and categorical predictors. *Journal of Statistical Planning and Inference* **89**, 197–214.

See Also

lmrob.M.S

Examples

```
data(education)
education <- within(education, Region <- factor(Region))
educaCh   <- within(education, Region <- as.character(Region))

## no interactions -- same split for all types:
fm1 <- lm(Y ~ Region + X1 + X2 + X3, education)
fmC <- lm(Y ~ Region + X1 + X2 + X3, educaCh  )
```

```
splt <- splitFrame(fm1$model) ; str(splt)
splC <- splitFrame(fmC$model)
stopifnot(identical(splt, splC))

## with interactions:
fm2 <- lm(Y ~ Region:X1:X2 + X1*X2, education)
s1 <- splitFrame(fm2$model, type="f"  )
s2 <- splitFrame(fm2$model, type="fi" )
s3 <- splitFrame(fm2$model, type="fii")
cbind(s1$x1.idx,
      s2$x1.idx,
      s3$x1.idx)
rbind(p.x1 = c(ncol(s1$x1), ncol(s2$x1), ncol(s3$x1)),
      p.x2 = c(ncol(s1$x2), ncol(s2$x2), ncol(s3$x2)))
```

`starsCYG` *Hertzsprung-Russell Diagram Data of Star Cluster CYG OB1*

Description

Data for the Hertzsprung-Russell Diagram of the Star Cluster CYG OB1, which contains 47 stars in the direction of Cygnus, from C.Doom. The first variable is the logarithm of the effective temperature at the surface of the star (Te) and the second one is the logarithm of its light intencity (L/L_0).

In the Hertzsprung-Russell diagram, which is the scatterplot of these data points, where the log temperature is plotted from left to right, two groups of points are seen:
the majority which tend to follow a steep band and four stars in the upper corner. In the astronomy the 43 stars are said to lie on the main sequence and the four remaining stars are called "giants" (the points 11, 20, 30, 34).

Usage

```
data(starsCYG, package="robustbase")
```

Format

A data frame with 47 observations on the following 2 variables

`log.Te` Logarithm of the effective temperature at the surface of the star (Te).

`log.light` Logarithm of its light intencity (L/L_0)

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, p.27, table 3.

Examples

```
data(starsCYG)
plot(starsCYG)
cst <- covMcd(starsCYG)
lm.stars <- lm(log.light ~ log.Te, data = starsCYG)
summary(lm.stars)
plot(lm.stars)
lts.stars <- ltsReg(log.light ~ log.Te, data = starsCYG)
plot(lts.stars)
```

---

steamUse *Steam Usage Data (Excerpt)*

Description

The monthly use of steam (`Steam`) in a factory may be modeled and described as function of the operating days per month (`Operating.Days`) and mean outside temperature per month (`Temperature`).

Usage

```
data("steamUse", package="robustbase")
```

Format

A data frame with 25 observations on the following 9 variables.

`Steam`: regression response Y, the poinds of steam used monthly.

`fattyAcid`: pounds of Real Fatty Acid in storage per month.

`glycerine`: pounds of crude glycerine made.

`wind`: average wind velocity in miles per hour (a numeric vector).

`days`: an integer vector with number of days of that month, i.e., in $28..31$.

`op.days`: the number of operating days for the given month (integer).

`freeze.d`: the number of days below 32 degrees Fahrenheit ($= 0$°C (C=Celsius) $=$ freezing temperature of water).

`temperature`: a numeric vector of average outside temperature in Fahrenheit (F).

`startups`: the number of startups (of production in that month).

Details

Nor further information is given in Draper and Smith, about the place and exacts years of the measurements, though some educated guesses should be possible, see the examples.

Source

Data from Draper and Smith, 1st ed, 1966; appendix A.

A version of this has been used in teaching at SfS ETH Zurich, since at least 1996, https://stat.ethz.ch/Teaching/Datasets/NDK/dsteam.dat

The package **aprean3** contains all data sets from the 3rd edition of Draper and Smith (1998), and this data set with variable names `x1 .. x10` (`x9` being `wind^2`, hence extraneous).

References

Draper and Smith (1981) Applied Regression Analysis (2nd ed., p. 615 ff)

Examples

```
## Not run:
if(require("aprean3")) { # show how  'steamUse'  is related to 'dsa01a'
  stm <- dsa01a
  names(stm) <- c("Steam", "fattyAcid", "glycerine", "wind",
  "days", "op.days", "freeze.d",
  "temperature", "wind.2", "startups")
  ## prove that wind.2 is  wind^2, "traditionally" rounded to 1 digit:
  stopifnot(all.equal(floor(0.5 + 10*stm[,"wind"]^2)/10,
                      stm[,"wind.2"], tol = 1e-14))
  ## hence drop it
  steamUse <- stm[, names(stm) != "wind.2"]
}

## End(Not run)
data(steamUse)
str(steamUse)
## Looking at this,
cbind(M=rep_len(month.abb, 25), steamUse[,5:8, drop=FALSE])
## one will conjecture that these were 25 months, Jan--Jan in a row,
## starting in a leap year (perhaps 1960 ?).

plot(steamUse)

summary(fm1 <- lmrob(Steam ~ temperature + op.days, data=steamUse))
## diagnoses 2 outliers: month of July, maybe company-wide summer vacations
## KS2014 alone seems not robust enough:
summary(fm.14 <- lmrob(Steam ~ temperature + op.days, data=steamUse,
         setting="KS2014"))
pairs(Steam ~ temperature+op.days, steamUse)
```

---

summarizeRobWeights *Print a Nice "summary" of Robustness Weights*

Description

Print a nice "summary" about a numeric vector of robustness weights. Observations with weights around zero are marked as outliers.

Usage

```
summarizeRobWeights(w, digits = getOption("digits"),
                    header = "Robustness weights:",
              eps = 0.1 / length(w), eps1 = 1e-3, ...)
```

Arguments

w	numeric vector of robustness weigths.
digits	digits to be used for printing.
header	string to be printed as header line.
eps	numeric tolerance ϵ: values of w with $\|w_i\| < \epsilon/n$ are said to be outliers.
eps1	numeric tolerance: values of w with $\|1 - w_i\| < eps1$ are said to have weight '~= 1'.
...	potential further arguments, passed to print().

Value

none; the function is used for its side effect of printing.

Author(s)

Martin Maechler

See Also

The summary methods for lmrob and glmrob make use of summarizeRobWeights().

Our methods for weights(), weights.lmrob(*, type="robustness") and weights.glmrob(*, type="robustness").

Examples

```
w <- c(1,1,1,1,0,1,1,1,1,0,1,1,.9999,.99999, .5,.6,1e-12)
summarizeRobWeights(w) # two outside ~= {0,1}
summarizeRobWeights(w, eps1 = 5e-5)# now three outside {0,1}

## See the summary(<lmrob>) outputs
```

---

summary.glmrob *Summarizing Robust Fits of Generalized Linear Models*

Description

The summary method for class "glmrob" summarizes robust fits of (currently only discrete) generalized linear models.

Usage

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

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

Arguments

object	an object of class "glmrob", usually, a result of a call to glmrob.
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.
...	further arguments passed to or from other methods.
x	an object of class "summary.glrob".
digits	the number of digits to use for printing.
signif.stars	logical indicating if the P-values should be visualized by so called "significance stars".

Details

summary.glmrob returns an object of class "summary.glmrob".

Its print() method tries to be smart about formatting the coefficients, standard errors, etc, and gives "significance stars" if signif.stars is TRUE (as per default when options where not changed).

Value

The function summary.glmrob computes and returns a list of summary statistics of the robustly fitted linear model given in object. The following elements are in the list:

...	FIXME

Author(s)

Andreas Ruckstuhl

See Also

glmrob; the generic summary and also summary.glm.

Examples

```
data(epilepsy)
Rmod <- glmrob(Ysum ~ Age10 + Base4*Trt, family = poisson,
               data = epilepsy, method= "Mqle")
ss <- summary(Rmod)
ss ## calls print.summary.glmrob()
str(ss) ## internal STRucture of summary object
```

`summary.lmrob` *Summary Method for "lmrob" Objects*

Description

Summary method for R object of class `"lmrob"` and `print` method for the summary object.

Further, methods `fitted()`, `residuals()` work (via the default methods), and `predict()` (see `predict.lmrob`, `vcov()`, `weights()` (see `weights.lmrob`), `model.matrix()`, `confint()`, `dummy.coef()`, `hatvalues()`, etc., have explicitly defined `lmrob` methods. `.lmrob.hat()` is the lower level "work horse" of the `hatvalues()` method.

Usage

```
## S3 method for class 'lmrob'
summary(object, correlation = FALSE,
        symbolic.cor = FALSE, ...)
## S3 method for class 'summary.lmrob'
print(x, digits = max(3, getOption("digits") - 3),
      symbolic.cor= x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"),
      showAlgo = TRUE, ...)

## S3 method for class 'lmrob'
vcov(object, cov = object$control$cov, complete = TRUE, ...)
## S3 method for class 'lmrob'
model.matrix(object, ...)
```

Arguments

`object`	an R object of class `lmrob`, typically created by `lmrob`.
`correlation`	logical variable indicating whether to compute the correlation matrix of the estimated coefficients.
`symbolic.cor`	logical indicating whether to use symbols to display the above correlation matrix.
`x`	an R object of class `summary.lmrob`, typically resulting from `summary(lmrob(..),..)`.
`digits`	number of digits for printing, see `digits` in `options`.

`signif.stars` logical variable indicating whether to use stars to display different levels of significance in the individual t-tests.

`showAlgo` optional `logical` indicating if the algorithmic parameters (as mostly inside the `control` part) should be shown.

`cov` covariance estimation function to use, a `function` or character string naming the function; **robustbase** currently provides `".vcov.w"` and `".vcov.avar1"`, see *Details* of `lmrob`. Particularly useful when `object` is the result of `lmrob(.., cov = "none")`, where

```
object$cov <- vcov(object, cov = ".vcov.w")
```

allows to *update* the fitted object.

`complete` (mainly for R `>= 3.5.0`:) `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 `NA`s correspondingly. When `complete = TRUE`, `vcov()` is compatible with `coef()` also in this singular case.

`...` potentially more arguments passed to methods.

Value

`summary(object)` returns an object of S3 class `"summary.lmrob"`, basically a `list` with components "call", "terms", "residuals", "scale", "rweights", "converged", "iter", "control" all copied from `object`, and further components, partly for compatibility with `summary.lm`,

`coefficients` a `matrix` with columns `"Estimate"`, `"Std. Error"`, `"t value"`, and `"Pr(>|t|)"`, where "Estimate" is identical to `coef(object)`. Note that `coef(<summary.obj>)` is slightly preferred to access this matrix.

`df` degrees of freedom, in an `lm` compatible way.

`sigma` identical to `sigma(object)`.

`aliased` ..

`cov` derived from `object$cov`.

`r.squared` robust "R squared" or R^2, a coefficient of determination: This is the consistency corrected robust coefficient of determination by Renaud and Victoria-Feser (2010).

`adj.r.squared` an adjusted R squared, see `r.squared`.

References

Renaud, O. and Victoria-Feser, M.-P. (2010). A robust coefficient of determination for regression, *Journal of Statistical Planning and Inference* **140**, 1852-1862.

See Also

`lmrob`, `predict.lmrob`, `weights.lmrob`, `summary.lm`, `print`, `summary`.

Examples

```
mod1 <- lmrob(stack.loss ~ ., data = stackloss)
sa <- summary(mod1)  # calls summary.lmrob(....)
sa                   # dispatches to call print.summary.lmrob(....)

## correlation between estimated coefficients:
cov2cor(vcov(mod1))

cbind(fit = fitted(mod1), resid = residuals(mod1),
      wgts= weights(mod1, type="robustness"),
      predict(mod1, interval="prediction"))

data(heart)
sm2 <- summary( m2 <- lmrob(clength ~ ., data = heart) )
sm2
```

---

summary.lts *Summary Method for LTS objects*

Description

summary method for class "lts".

Usage

```
## S3 method for class 'lts'
summary(object, correlation = FALSE, ...)

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

Arguments

object	an object of class "lts", usually, a result of a call to ltsReg.
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
x	an object of class "summary.lts", usually, a result of a call to summary.lts.
digits	the number of significant digits to use when printing.
signif.stars	logical indicating if "significance stars" should be printer, see printCoefmat.
...	further arguments passed to or from other methods.

Details

These functions compute and print summary statistics for weighted least square estimates with weights based on LTS estimates. Therefore the statistics are similar to those for LS but all terms are multiplied by the corresponding weight.

Correlations are printed to two decimal places: to see the actual correlations print `summary(object)$correlation` directly.

Value

The function `summary.lts` computes and returns a list of summary statistics of the fitted linear model given in `object`, using the components of this object (list elements).

`residuals` the residuals - a vector like the response `y` containing the residuals from the weighted least squares regression.

`coefficients` a $p \times 4$ matrix with columns for the estimated coefficient, its standard error, t-statistic and corresponding (two-sided) p-value.

`sigma` the estimated scale of the reweighted residuals

$$\hat{\sigma}^2 = \frac{1}{n-p}\sum_i R_i^2,$$

where R_i is the i-th residual, `residuals[i]`.

`df` degrees of freedom, a 3-vector $(p, n-p, p*)$, the last being the number of non-aliased coefficients.

`fstatistic` (for models including non-intercept terms) a 3-vector with the value of the F-statistic with its numerator and denominator degrees of freedom.

`r.squared` R^2, the "fraction of variance explained by the model",

$$R^2 = 1 - \frac{\sum_i R_i^2}{\sum_i (y_i - y^*)^2},$$

where y^* is the mean of y_i if there is an intercept and zero otherwise.

`adj.r.squared` the above R^2 statistic "*adjusted*", penalizing for higher p.

`cov.unscaled` a $p \times p$ matrix of (unscaled) covariances of the $\hat{\beta}_j$, $j = 1, \ldots, p$.

`correlation` the correlation matrix corresponding to the above `cov.unscaled`, if `correlation = TRUE` is specified.

See Also

`ltsReg`; the generic `summary`.

Examples

```
data(Animals2)
ltsA <- ltsReg(log(brain) ~ log(body), data = Animals2)
(slts <- summary(ltsA))
## non-default options for printing the summary:
print(slts, digits = 5, signif.stars = FALSE)
```

summary.mcd *Summary Method for MCD objects*

Description

summary method for class "mcd".

Usage

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

Arguments

object, x	an object of class "mcd" (or "summary.mcd"); usually, a result of a call to covMcd.
digits	the number of significant digits to use when printing.
print.gap	number of horizontal spaces between numbers; see also print.default.
...	further arguments passed to or from other methods.

Details

summary.mcd(), the S3 method, simply returns an (S3) object of class "summary.mcd" for which there's a print method:

print.summary.mcd prints summary statistics for the weighted covariance matrix and location estimates with weights based on MCD estimates. While the function print.mcd prints only the robust estimates of the location and the covariance matrix, print.summary.mcd will print also the correlation matrix (if requested in the call to covMcd with cor=TRUE), the eigenvalues of the covariance or the correlation matrix and the robust ("Mahalanobis") distances.

Value

summary.mcd returns an summary.mcd object, whereas the print methods returns its first argument via invisible, as all print methods do.

See Also

covMcd, summary

Examples

```
data(Animals, package = "MASS")
brain <- Animals[c(1:24, 26:25, 27:28),]
lbrain <- log(brain)
summary(cLB <- covMcd(lbrain))
```

summary.nlrob *Summarizing Robust Fits of Nonlinear Regression Models*

Description

summary method for objects of class "nlrob", i.e., nlrob() results. Currently it only works for nlrob(*, method="M").

Usage

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

Arguments

object	an object of class "nlrob", usually, a result of a call to nlrob.
correlation	logical variable indicating whether to compute the correlation matrix of the estimated coefficients.
symbolic.cor	logical indicating whether to use symbols to display the above correlation matrix.
...	further arguments passed to or from other methods.

Value

The function summary.nlrob computes and returns an object of class "summary.nlrob" of summary statistics of the robustly fitted linear model given in object. There is a print method, print.summary.lmrob(), which nicely formats the output.

The result keeps a large part of object's components such as residuals, cov or w, and additionally contains

coefficients	the matrix of coefficients, standard errors and p-values.
correlation	if the correlation argument was true, the correlation matrix of the parameters.

Author(s)

Andreas Ruckstuhl

See Also

nlrob(), also for examples.

telef *Number of International Calls from Belgium*

Description

Number of international calls from Belgium, taken from the Belgian Statistical Survey, published by the Ministry of Economy.

Usage

```
data(telef, package="robustbase")
```

Format

A data frame with 24 observations on the following 2 variables.

`Calls` Number of Calls (in tens of millions)

`Year` Year (1950 - 1973)

Source

P. J. Rousseeuw and A. M. Leroy (1987) *Robust Regression and Outlier Detection*; Wiley, page 26, table 2.

Examples

```
data(telef)
summary(lm.telef <- lm(Year~., data=telef))
```

tolEllipsePlot *Tolerance Ellipse Plot*

Description

Plots the 0.975 tolerance ellipse of the bivariate data set `x`. The ellipse is defined by those data points whose distance is equal to the squareroot of the 0.975 chisquare quantile with 2 degrees of freedom.

Usage

```
tolEllipsePlot(x, m.cov = covMcd(x), cutoff = NULL, id.n = NULL,
               classic = FALSE, tol = 1e-07,
               xlab = "", ylab = "",
               main = "Tolerance ellipse (97.5%)",
               txt.leg = c("robust", "classical"),
               col.leg = c("red", "blue"),
               lty.leg = c("solid","dashed"))
```

Arguments

x	a two dimensional matrix or data frame.
m.cov	an object similar to those of class "mcd"; however only its components center and cov will be used. If missing, the MCD will be computed (via covMcd()).
cutoff	numeric distance needed to flag data points outside the ellipse.
id.n	number of observations to be identified by a label. If not supplied, the number of observations with distance larger than cutoff is used.
classic	whether to plot the classical distances as well, FALSE by default.
tol	tolerance to be used for computing the inverse, see solve. Defaults to 1e-7.
xlab, ylab, main	passed to plot.default.
txt.leg, col.leg, lty.leg	character vectors of length 2 for the legend, only used if classic = TRUE.

Author(s)

Peter Filzmoser, Valentin Todorov and Martin Maechler

See Also

covPlot which calls tolEllipsePlot() when desired. ellipsoidhull and predict.ellipsoid from package **cluster**.

Examples

```
data(hbk)
hbk.x <- data.matrix(hbk[, 1:3])
mcd <- covMcd(hbk.x)       # compute mcd in advance
## must be a 2-dimensional data set: take the first two columns :
tolEllipsePlot(hbk.x[,1:2])

## an "impressive" example:
data(telef)
tolEllipsePlot(telef, classic=TRUE)
```

---

toxicity *Toxicity of Carboxylic Acids Data*

Description

The aim of the experiment was to predict the toxicity of carboxylic acids on the basis of several molecular descriptors.

Usage

```
data(toxicity, package="robustbase")
```

Format

A data frame with 38 observations on the following 10 variables which are attributes for carboxylic acids:

`toxicity` aquatic toxicity, defined as $\log(IGC_{50}^{-1})$; typically the "response".

`logKow` $logKow$, the partition coefficient

`pKa` pKa: the dissociation constant

`ELUMO` **E**nergy of the **l**owest **u**noccupied **m**olecular **o**rbital

`Ecarb` Electrotopological state of the **carb**oxylic group

`Emet` Electrotopological state of the **met**hyl group

`RM` Molar refractivity

`IR` Refraction index

`Ts` Surface tension

`P` Polarizability

Source

The website accompanying the MMY-book: https://www.wiley.com/legacy/wileychi/robust_statistics/

References

Maguna, F.P., Núñez, M.B., Okulik, N.B. and Castro, E.A. (2003) Improved QSAR analysis of the toxicity of aliphatic carboxylic acids; *Russian Journal of General Chemistry* **73**, 1792–1798.

Examples

```
data(toxicity)
summary(toxicity)
plot(toxicity)
plot(toxicity ~ pKa, data = toxicity)

## robustly scale the data (to scale 1) using Qn
(scQ.tox <- sapply(toxicity, Qn))
scTox <- scale(toxicity, center = FALSE, scale = scQ.tox)
csT <- covOGK(scTox, n.iter = 2,
              sigmamu = s_Qn, weight.fn = hard.rejection)
as.dist(round(cov2cor(csT$cov), 2))
```

tukeyPsi1 *Tukey's Bi-square Score (Psi) and "Chi" (Rho) Functions and Derivatives*

Description

These are **deprecated**, replaced by `Mchi(*, psi="tukey")`, `Mpsi(*, psi="tukey")`

`tukeyPsi1()` computes Tukey's bi-square score (psi) function, its first derivative or it's integral/"principal function". This is scaled such that $\psi'(0) = 1$, i.e., $\psi(x) \approx x$ around 0.

`tukeyChi()` computes Tukey's bi-square loss function, `chi(x)` and its first two derivatives. Note that in the general context of M-estimators, these loss functions are called $\rho(rho)$-functions.

Usage

```
tukeyPsi1(x, cc, deriv = 0)
tukeyChi (x, cc, deriv = 0)
```

Arguments

`x`	numeric vector.
`cc`	tuning constant
`deriv`	integer in $\{-1, 0, 1, 2\}$ specifying the order of the derivative; the default, `deriv = 0` computes the psi-, or chi- ("rho"-)function.

Value

a numeric vector of the same length as `x`.

Note

`tukeyPsi1(x, d)` and `tukeyChi(x, d+1)` are just re-scaled versions of each other (for `d in -1:1`), i.e.,

$$\chi^{(\nu)}(x, c) = (6/c^2)\psi^{(\nu-1)}(x, c),$$

for $\nu = 0, 1, 2$.

We use the name 'tukeyPsi**1**', because `tukeyPsi` is reserved for a future "Psi Function" class object, see `psiFunc`.

Author(s)

Matias Salibian-Barrera, Martin Maechler and Andreas Ruckstuhl

See Also

`lmrob` and `Mpsi`; further `anova.lmrob` which needs the `deriv = -1`.

Examples

```
op <- par(mfrow = c(3,1), oma = c(0,0, 2, 0),
          mgp = c(1.5, 0.6, 0), mar= .1+c(3,4,3,2))
x <- seq(-2.5, 2.5, length = 201)
cc <- 1.55 # as set by default in lmrob.control()
plot. <- function(...) { plot(...); abline(h=0,v=0, col="gray", lty=3)}
plot.(x, tukeyChi(x, cc), type = "l", col = 2)
plot.(x, tukeyChi(x, cc, deriv = 1), type = "l", col = 2)
plot.(x, tukeyChi(x, cc, deriv = 2), type = "l", col = 2)
mtext(sprintf("tukeyChi(x, c = %g, deriv),  deriv = 0,1,2", cc),
      outer = TRUE, font = par("font.main"), cex = par("cex.main"))
par(op)

op <- par(mfrow = c(3,1), oma = c(0,0, 2, 0),
          mgp = c(1.5, 0.6, 0), mar= .1+c(3,4,1,1))
x <- seq(-5, 5, length = 201)
cc <- 4.69 # as set by default in lmrob.control()
plot. <- function(...) { plot(..., asp = 1); abline(h=0,v=0, col="gray", lty=3)}
plot.(x, tukeyPsi1(x, cc), type = "l", col = 2)
abline(0:1, lty = 3, col = "light blue")
plot.(x, tukeyPsi1(x, cc, deriv = -1), type = "l", col = 2)
plot.(x, tukeyPsi1(x, cc, deriv =  1), type = "l", col = 2); abline(h=1,lty=3)
mtext(sprintf("tukeyPsi1(x, c = %g, deriv),  deriv = 0, -1, 1", cc),
      outer = TRUE, font = par("font.main"), cex = par("cex.main"))
par(op)
```

---

vaso *Vaso Constriction Skin Data Set*

Description

Finney's data on vaso constriction in the skin of the digits.

Usage

```
data(vaso, package="robustbase")
```

Format

A data frame with 39 observations on the following 3 variables.

`Volume` Inhaled volume of air

`Rate` Rate of inhalation

`Y` vector of 0 or 1 values.

Details

The data taken from Finney (1947) were obtained in a carefully controlled study in human physiology where a reflex "vaso constriction" may occur in the skin of the digits after taking a single deep breath. The response y is the occurence (y = 1) or non-occurence (y = 0) of vaso constriction in the skin of the digits of a subject after he or she inhaled a certain volume of air at a certain rate. The responses of three subjects are available. The first contributed 9 responses, the second contributed 8 responses, and the third contributed 22 responses.

Although the data represent repeated measurements, an analysis that assumes independent observations may be applied, as claimed by Pregibon (1981).

Source

Finney, D.J. (1947) The estimation from individual records of the relationship between dose and quantal response. *Biometrika* **34**, 320–334

References

Atkinson, A.C. and Riani, M. (2000) *Robust Diagnostic Regression Analysis*, First Edition. New York: Springer, Table A.23.

Fahrmeir, L. and Tutz, G. (2001) *Multivariate Statistical Modelling Based on Generalized Linear Models*, Springer, Table 4.2.

Kuensch, H.R., Stefanski, A. and Carrol, R.J. (1989) Conditionally unbiased bounded influence estimation in general regression models, with applications to generalized linear models, *JASA* **84**, 460–466.

Pregibon, D. (1981) Logistic regression diagnostics, *Annals of Statistics* **9**, 705–724.

Examples

```
data(vaso)
str(vaso)
pairs(vaso)

glmV <- glm(Y ~ log(Volume) + log(Rate), family=binomial, data=vaso)
summary(glmV)
## -->  example(glmrob)  showing classical & robust GLM
```

`wagnerGrowth` *Wagner's Hannover Employment Growth Data*

Description

Wagner (1994) investigates the rate of employment growth (y) as function of percentage of people engaged in **p**roducation **a**ctivities (`PA`) and **h**igher **s**ervices (`HS`) and of the **g**rowth of these percentages (`GPA`, `GHS`) during three time periods in 21 geographical regions of the greater Hannover area.

Usage

```
data(wagnerGrowth, package="robustbase")
```

Format

A data frame with $21 \times 3 = 63$ observations (one per `Region` x `Period`) on the following 7 variables.

`Region` a `factor` with 21 levels, denoting the corresponding region in Hannover (conceptually a "block factor").

`PA` numeric: percent of people involved in production activities.

`GPA` **g**rowth of `PA`.

`HS` a numeric vector

`GHS` a numeric vector

`y` a numeric vector

`Period` a `factor` with levels `1:3`, denoting the time period, 1 = 1979-1982, 2 = 1983-1988, 3 = 1989-1992.

Source

Hubert, M. and Rousseeuw, P. J. (1997). Robust regression with both continuous and binary regressors, *Journal of Statistical Planning and Inference* **57**, 153–163.

References

Wagner J. (1994). Regionale Beschäftigungsdynamik und höherwertige Produktionsdienste: Ergebnisse für den Grossraum Hannover (1979-1992). *Raumforschung und Raumordnung* **52**, 146–150.

Examples

```
data(wagnerGrowth)
## maybe
str(wagnerGrowth)


require(lattice)
(xyplot(y ~ Period | Region, data = wagnerGrowth,
         main = "wagnerGrowth: 21 regions @ Hannover"))

(dotplot(y ~ reorder(Region,y,median), data = wagnerGrowth,
         main = "wagnerGrowth",
         xlab = "Region [ordered by  median(y | Region) ]"))
```

`weights.lmrob` *Extract Robustness and Model Weights*

Description

`weights()` extracts robustness weights or fitting (or prior) weights from a `lmrob` or `glmrob` object.

Usage

```
## S3 method for class 'lmrob'
weights(object, type = c("prior", "robustness"), ...)
```

Arguments

`object`	an object of class `"lmrob"` or `"glmrob"`, typically the result of a call to `lmrob`, or `glmrob`, respectively.
`type`	the type of weights to be returned. Either `"prior"` (default), or `"robustness"`.
`...`	not used currently.

Details

The "prior weights" correspond to the weights specified using the "weights" argument when calling `lmrob`. The "robustness weights" are the weights assigned by the M-estimator of regression, $\psi(r_i/S)/(r_i/S)$. The robust coefficient estimate then numericarlly corresponds to a weighted least squares fit using the product of both types of weights as weights.

Value

Weights extracted from the object `object`.

Author(s)

Manuel Koller and Martin Maechler.

See Also

`lmrob`, `glmrob` and `weights`

wgt.himedian *Weighted Hi-Median*

Description

Compute the weighted Hi-Median of x.

Usage

```
wgt.himedian(x, weights = rep(1, n))
```

Arguments

x	numeric vector
weights	numeric vector of weights; of the same length as x.

Note

this is rather a by-product of the code used in Sn and Qn. We currently plan to replace it with more general weighted quantiles.

See Also

median; also wtd.quantile from package **Hmisc**.

Examples

```
x <- c(1:6, 20)
median(x) ## 4
stopifnot(all.equal(4, wgt.himedian(x)),
          all.equal(6, wgt.himedian(x, c(rep(1,6), 5))))
```

wood *Modified Data on Wood Specific Gravity*

Description

The original data are from Draper and Smith (1966) and were used to determine the influence of anatomical factors on wood specific gravity, with five explanatory variables and an intercept. These data were contaminated by replacing a few observations with outliers.

Usage

```
data(wood, package="robustbase")
```

Format

A data frame with 20 observations on the following 6 variables.

x1, x2, x3, x4, x5 explanatory "anatomical" wood variables.

y wood specific gravity, the target variable.

Source

Draper and Smith (1966, p.227)

Peter J. Rousseeuw and Annick M. Leroy (1987) *Robust Regression and Outlier Detection* Wiley, p.243, table 8.

Examples

```
data(wood)
plot(wood)
summary( lm.wood <-        lm(y ~ ., data = wood))
summary(rlm.wood <- MASS::rlm(y ~ ., data = wood))
summary(lts.wood <-    ltsReg(y ~ ., data = wood))

wood.x <- as.matrix(wood)[,1:5]
c_wood <- covMcd(wood.x)
c_wood
```

---

xtrData *Extreme Data examples*

Description

`x30o50`, called ''XX" in the thesis, has been a running case for which `mc()` had failed to converge. A numeric vector of 50 values, 30 of which are very close to zero, specifically, their absolute values are less than `1.5e-15`.

The remaining 20 values (11 negative, 9 positive) have absolute values between 0.0022 and 1.66.

Usage

```
data(x30o50, package="robustbase")
```

Format

A summary is

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.66006  0.00000  0.00000 -0.04155  0.00000  1.29768
```

notably the 1st to 3rd quartiles are all very close to zero.

Details

a good robust method will treat the 60% "almost zero" values as "good" data and all other as outliers.

This is somewhat counter intuitive to typical human perception where the 30 almost-zero numbers would be considered as inliers and the remaining 20 as "good" data.

The original `mc()` algorithm and also the amendments up to 2022 (**robustbase** versions before 0.95) would fail to converge unless (in newer versions) `eps1` was increased, e.g., only by a factor of 10, to `eps1 = 1e-13`.

References

Lukas Graz (2021); unpublished BSc thesis, see mc.

Examples

```
data(x30o50)
## have 4  duplicated values :
table(dX <- duplicated(x30o50))
     x30o50[dX] # 0 2.77e-17 4.16e-17 2.08e-16
sort(x30o50[dX]) * 2^56 #  0  2  3 15
## and they are  c(0,2,3,15)*2^-56

table(sml <- abs(x30o50) < 1e-11)# 20 30
summary(x30o50[ sml]) # -1.082e-15 ... 1.499e-15 ; mean = 9.2e-19 ~~ 0
summary(x30o50[!sml])
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.6601 -0.4689 -0.0550 -0.1039  0.3986  1.2977

op <- par(mfrow=c(3,1), mgp=c(1.5, .6, 0), mar = .3+c(2,3:1))
(Fn. <- ecdf(x30o50)) # <- only 46 knots (as have 4 duplications)
plot(Fn.) ## and zoom in (*drastically*) to around x=0 :
for(f in c(1e-13, 1.5e-15)) {
  plot(Fn., xval=f*seq(-1,1, length.out = 1001), ylim=c(0,1), main="[zoomed in]")
  if(f == 1e-13) rect(-1e-15,0, +1e-15, 1, col="thistle", border=1)
  plot(Fn., add=TRUE)
}
par(op)

mcOld <- function(x, ..., doScale=TRUE)  mc(x, doScale=doScale, c.huberize=Inf, ...)
try( mcOld(x30o50) ) # Error: .. not 'converged' in 100 iteration
mcOld(x30o50, eps1 = 1e-12) # -0.152
(mcX <- mc(x30o50)) # -7.10849e-13
stopifnot(exprs = {
    all.equal(-7.10848988e-13, mcX, tol = 1e-9)
    all.equal(mcX, mc(1e30*x30o50), tol = 4e-4) # not so close
})
table(sml <- abs(x30o50) < 1e-8)# 20 30
range(x30o50[sml])
x0o50 <- x30o50; x0o50[sml] <- 0
(mcX0 <- mc(x0o50))
stopifnot(exprs = {
```

```
    all.equal(-0.378445401788, mcX0, tol=1e-12)
    all.equal(-0.099275805349, mc(x30o50[!sml]) -> mcL, tol=2e-11)
    all.equal(mcL, mcOld(x30o50[!sml]))
})
## -- some instability also wrt c.huberize:
mcHubc <- function(dat, ...)
    function(cc) vapply(cc, function(c) mc(dat, c.huberize = c, ...), -1.)
mcH50 <- mcHubc(x30o50)
head(cHs <- c(sort(outer(c(1, 2, 5), 10^(2:15))), Inf), 9)
mcXc <- mcH50(cHs)
plot(  mcXc  ~ cHs, type="b", log="x" , xlab=quote(c[huberize]))
plot((-mcXc) ~ cHs, type="b", log="xy", xlab=quote(c[huberize]))
## but for "regular" outlier skew data, there's no such dependency:
mcXcu <- mcHubc(cushny)(cHs)
stopifnot( abs(mcXcu - mcXcu[1]) < 1e-15)
```

Index