Package: STB (via r-universe)

September 10, 2024

Version 0.6.5

Date 2021-09-14

Title Simultaneous Tolerance Bounds

Author Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Maintainer Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Depends R (>= 3.2.2)

Imports stats, graphics, grDevices, Matrix, methods, VCA (>= 1.3.1), parallel

Description Provides an implementation of simultaneous tolerance bounds (STB), useful for checking whether a numeric vector fits to a hypothetical null-distribution or not. Furthermore, there are functions for computing STB (bands, intervals) for random variates of linear mixed models fitted with package 'VCA'. All kinds of, possibly transformed (studentized, standardized, Pearson-type transformed) random variates (residuals, random effects), can be assessed employing STB-methodology.

License GPL (>= 3)

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-09-15 09:40:02 UTC

Contents

STB-package *(S)imultaneous (T)olerance (B)ounds*

Description

Compute simultaneous tolerance bounds for arbitrary null-distributions or random variates of linear mixed models (LMM). A common problem is to check whether a numeric values follow a distinct distribution. Formal significance tests lack power when its most needed, when sample size is small and have sort of too much power for large sample sizes, i.e. rejecting e.g. normality even if departure from normality is negligible for drawing inference.

Graphical methods are usually preferred over formal testing but lack some objectivity, i.e. the same plot is differently interpreted by two different persons. Simultaneous tolerance bounds (intervals or bands) add some objectivity to plots like the QQ-plot, a plot frequently used to check for specific distributions. See the first reference for details about simultaneous tolerance bounds (there, used for checking normality in the general linear model). Various other null-distributions can be checked, since resampling can be applied to all sorts of distributions (see argument 'rand.func' of `stb`).

Besides checking distributional assumptions of numeric vectors, graphical methods are available for linear mixed models fitted using R-package `VCA`. Various types of random variates exist in this framework, i.e. random effects and at least two types of residuals. All types of random variates of LMM need to be checked for their particular distributional assumptions. Departure from these may indicate that a transformation of the response variable is required or that extreme values (possible outliers) have a negative effect on the distribution of random variates. See the 2nd reference for a discussion of this topic and for practical examples.

Generic function `stb` applies to either numeric vectors or to objects of class `VCA`. The former generates a quantile-quantile (QQ) plot following a specified null-distribution with simultaneous tolerance band (see `stb.default`). The latter checks residuals or random effects, possibly transformed (e.g. studentized) to be normally distributed, incorporating the covariance-structure of the fitted model (see `stb.VCA`). Plotting methods exists for the resulting objects (see `plot.STB` and `plot.stbVCA` for details). There are two graphical methods available for random variates of LMM, the usual QQ-plot with simultaneous tolerance band (argument `type` set to 1 or 3) and a plot of the random variates with a simultaneous tolerance interval (STI) (argument `type` set to 2 or 3), where the STI is derived from the extreme values (first and last order statistics).

Details

Package:	STB
Type:	Package
Version:	0.6.5
Date:	2021-09-14
License:	GPL (>= 3)
LazyLoad:	yes

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A., Jensen, U., Piepho, H.P. (2011), Checking assumptions of normality and homoscedasticity in the general linear model. Communications in Statistics - Simulation and Computation; S. 141-154

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

`fastSTB` *Simultaneous Tolerance Bands Using a Fast C-Implementation.*

Description

This function represents the interface to a C-implementation of the bisection algorithm for computing simultaneous tolerance bounds as described in Schuetzenmeister et al. 2012.

Usage

```
fastSTB(
  mat,
  alpha = 0.05,
  tol = 1e-04,
  max.iter = 100L,
  Ncpu = 2,
  timer = FALSE
)
```

Arguments

`mat`	(numeric) matrix with rows representing simulations of 'ncol' values, rows are supposed to be sorted
`alpha`	(numeric) 100(1-alpha)% simultaneous tolerance level (coverage)
`tol`	(numeric) convergence tolerance level for the bisection algorithm
`max.iter`	(integer) number of bisection-steps for the algorithm
`Ncpu`	(integer) specifying the number of processors (CPUs, cores, ...) to be used
`timer`	(logical) TRUE = the time spent for computing the STB will be printed

Details

Quantiles will be computed according to SAS PCTLDEF5 definition of quantiles, which is identical to 'type=2' in function quantile. This is also the default-option throughout this package. Function SASquantile is a R-implementation identical to the C-implementation used here.

Value

(list) with elements:

mat	(numeric) matrix with sorted random vector stored in rows
nCol	(integer) number of columns of 'mat'
nRow	(integer) number of rows of 'mat'
alpha	(numeric) values used for the 100(1-alpha)% STB
tol	(numeric) the tolerance value used as stopping criterion for the bisection algorithm
max.iter	(integer) the max. number of iteration of the bisection algorithm
Q	(numeric) matrix with 2 rows, corresponding to the bounds of the STB (1st row = lower bounds, 2nd row = upper bounds)
coverage	(numeric) value corresponding to the actual coverage of the STB

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A., Jensen, U., Piepho, H.P. (2011), Checking assumptions of normality and homoscedasticity in the general linear model. Communications in Statistics - Simulation and Computation; S. 141-154

See Also

getSTB stb

Examples

```
## Not run:
### example for 10000 x 30 matrix
 set.seed(333)
 mat <- t(apply(matrix(rnorm(10000*30), ncol=30), 1, sort))
 stb.obj1 <- fastSTB(mat, timer=TRUE)
 stb.obj1$coverage
 stb.obj2 <- fastSTB(mat, timer=TRUE, Ncpu=4)
 stb.obj3 <- fastSTB(mat, timer=TRUE, Ncpu=6)
 stb.obj4 <- fastSTB(mat, timer=TRUE, Ncpu=8)

## End(Not run)
```

getSTB *Load/unload C-lib. Simultaneous Tolerance Bands Using R-Implementation.*

Description

Compute simultaneous tolerance bounds (STB) from a matrix.

Usage

```
getSTB(
  mat,
  alpha = 0.05,
  tol = 1e-04,
  max.iter = 100L,
  q.type = 2L,
  logfile = NULL,
  output = TRUE,
  timer = FALSE,
  Ncpu = 2
)
```

Arguments

mat	(numeric) matrix with nrow=N_simulation, ncol=N_points, where each row is sorted in increasing order
alpha	(numeric) 100(1-alpha)% simultaneous tolerance-level will be computed
tol	(numeric) value controlling the termination of the bisection algorithm, i.e. the condition coverage-(1-alpha)<=tol has to be TRUE.
max.iter	(integer) maximum number of iterations of the bisection algorithm. If this number is reached the algorithm terminates and returns the best approximation of lower and upper bounds reached up to this iteration.
q.type	(integer) the quantile-type used if algo="R", see ? quantile for details.
logfile	(character) string specifying the name of the (optional) log-file, storing the iteration history
output	(logical) TRUE a txtProgressBar will be printed as well as additional status information
timer	(logical) TRUE = the time spent for computing the STB will be printed
Ncpu	(integer) specifying the number cores/CPUs to be used in computing the coverage on C-level, for N>1 multi-processing is applied

Details

Function computes 100(1-alpha)% simultaneous tolerance bounds (intervals, bands) from a matrix, where rows correspond to order statistics (sorted) of random samples of a, e.g. normal distribution. It starts by computing joint coverage of the naive unadjusted (point-wise) alpha-level intervals, computed as percentiles across each order statistic (columns). Alpha is decreased using a bisection search until the joint coverage becomes at least 100(1-alpha)% depending on arguments `max.iter` and `tol`. If the number of simulated random samples goes to infinity, the STB becomes exact. Note that checking whether a numeric vector is normally distributed is equal to checking whether the residuals of the simplest linear model y = mu + e (y~1) are normally distributed [e ~ N(0, sigma^2)].

Value

(list) with elements:

`Q`	(numeric) matrix with two rows, where the 1st row = lower bounds of the STB, 2nd row = upper bounds of the STB
`cov`	(numeric) value indicating the actual coverage of the STB

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A., Jensen, U., Piepho, H.P. (2011), Checking assumptions of normality and homoscedasticity in the general linear model. Communications in Statistics - Simulation and Computation; S. 141-154

See Also

fastSTB stb

Examples

```
## Not run:
set.seed(333)
mat <- t(apply(matrix(rnorm(10000*30), ncol=30), 1, sort))
stb.obj <- getSTB(mat, timer=TRUE)
stb.obj$cov

## End(Not run)
```

plot.STB *Plot Objects of Class 'STB'.*

Description

Standard plotting method for objects of class 'STB'.

Usage

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

Arguments

x	(object) of class 'STB' as generated by function getSTB
...	arguments passed to other methods

Details

This function plots objects of class 'STB' as generated by function stb. Objects of S3-class 'STB' are list-type objects storing all the information needed to plot QQ-plots with simultaneous tolerance bounds.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

stb

Examples

```
## Not run:
### generate an 'STB' object without plotting
obj <- stb(rnorm(30), plot=FALSE)
plot(obj)

### manipulate the 'STB' object for plotting
obj$legend=TRUE
plot(obj)

### add a previously generated STB-ocject to an existing plot
plot(sort(rnorm(30)), sort(rnorm(30)))
obj$add <- TRUE
plot(obj)

## End(Not run)
```

`plot.stbVCA` *Plot Objects of Class 'stbVCA'.*

Description

Standard plotting method for objects of Class 'stbVCA'.

Usage

```
## S3 method for class 'stbVCA'
plot(x, orient = 1, pick = FALSE, type = NULL, ...)
```

Arguments

`x`	(object) of class 'STB' as generated by function `getSTB`
`orient`	(integer) in 'type=1' plots, 1 = expected vs. observed values, 2 = observed vs. expected values
`pick`	(logical, integer) TRUE = triggers function `identify` for identification of points using the mouse. Stop interactive identification by pressing 'Esc' or "Right-Mousclick –> Stop". If 'TRUE' and 'type=3', identified points will be added to the residual plot with STI automatically. If 'pick=1', only in the QQ-plot identification will be toggled, and if 'pick=2' only in the residual plot identification will be possible.
`type`	(integer) 1 = plot simultaneous tolerance band (STB), 2 = plot simultaneous tolerance interval (STI), 3 = plot STB and STI
`...`	additional arguments changing the visual appearance of the plot, e.g. 'pch', 'pch.out', 'col', 'col.out', 'stb.col', 'stb.main', "'sti.main', 'legend', ... in fact all elements of 'x'

Details

This function plots objects of class 'stbVCA' as generated by function `stb.VCA`. Objects of S3-class 'stbVCA' are list-type objects storing all the information needed to plot QQ-plots with simultaneous tolerance bounds. Additionally to the information contained in ordinary 'STB' objects, a copy of the 'VCA' object is stored as well as the type of random variate and the mode, i.e. the type of transformation applied.

One can specify additional parameters for changing the appearance of the plot(s). Any of the following parameters can be set to a value different from the default:

`legend`	...	(logical) TRUE = will add legend to plot(s) (is default)
`pch`	...	plotting symbol for non-outlying points (default=16)
`col`	...	point color for non-outlying points (default="black")
`col.out`	...	point color for outlying points (default="red")
`pch.out`	...	plotting symbold for outlying points (default=16)

stb.col	...	color of the STB in the QQ-plot (default="#0000FF40")
stb.lpos	...	position placement as done in function 'legend', or, character string "title" indicating that legend information should be displayed as (sub-main) title (default="title")
stb.main	...	character string used as main title in the QQ-plot with STB
sti.lty	...	line type for STI-bounds in the residual plot (default=2)
sti.lwd	...	line width for STI-bounds (default=1)
sti.col	...	line color for STI bounds (default="red")
sti.ylab	...	character string specifying the Y-axis label in the resiudal plot with STI
sti.lpos	...	position placement as done in function 'legend' (default="topright")
sti.main	...	character string used as main title in the residual plot with STI

Value

(stbVCA) object is invisibly returned, any additional compution results are added

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

stb.VCA

Examples

```
## Not run:
library(VCA)
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
fit

# use studentized conditional residuals
stb.obj1 <- stb.VCA(fit, term="cond", mode="student", N=1000)

# plot it again
plot(stb.obj1)

# use random effects "day" and apply standardization
stb.obj2 <- stb.VCA(fit, term="day", mode="stand", N=1000)

# plot it again
plot(stb.obj2)

# initially, request QQ-plot with STB
stb.obj3 <- stb.VCA(fit, term="day", mode="stand", N=1000, type=1)

# now request plotting of the residual plot as well
# catch computation result which are invisibly returned
stb.obj4 <- plot(stb.obj3, type=3)
```

```
# individualize the appearance of the plot
plot(stb.obj4, sti.lpos="top", col="darkblue", out.pch=17, out.col="green")

## End(Not run)
```

---

rankSTB *Rank-Based Algorithm for Computing 100(1-alpha)% Simulataneous Tolerance Bounds.*

Description

Implementation of a rank-based algorithm for constructing 100(1-alpha)% STBs as outlined in the reference.

Usage

```
rankSTB(mat, alpha = 0.05)
```

Arguments

mat	(numeric) matrix of dimension (N, n), where i-th row corrsponds to ordered values of the i-th simulation
alpha	(numeric) value defining the desired coverage as 100(1-alpha)%

Details

This function is a performance optimized version of the original rank-based algorithm avoiding the time-consuming iteration. In principle it sorts out simulation results which have at least one extreme order statistic untill exactly 100(1-alpha)% of all simulation results remain. From these, bounds of the STB are constructed determining extreme-values per order-statistic (column).

This implementation also corrects step 4) of the published algorithm, which has to find those indices of elements being equal to "min(c)" OR being equal to "N-min(c)+1". This reflects the construction of vector "c", where max. rank-values are transformed to min. rank-values. In step 6) the "N_k-1-(1-alpha)*N largest" elements of "d_l^theta" have to be selected, which needs also correction.

Parallel processing did not minimize the computation time in contrast to the algorithms for computing the quantile-based algorithm. Thus, parallel processing is not supported for this algorithm.

Value

(list) with two elements:

Q	(matrix) 1st row stores lower bounds, 2nd row upper bounds
cov	(numeric) value corresponding the coverage

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

Examples

```
## Not run:
# for following problem size the rank-based algo
# outperforms the quantile based one, although,
# ran serially
mat <- matrix(rnorm(10000*100), ncol=100)
mat <- t(apply(mat, 1, sort))
system.time(stb.rank <- rankSTB(mat))
system.time(stb.q.R  <- getSTB(mat))
system.time(stb.q.C  <- fastSTB(mat))
x <- apply(mat, 2, mean)
plot(x,x, ylim=c(-5,5))
lines(x, stb.q.R$Q[1,], col="blue", lwd=2)
lines(x, stb.q.R$Q[2,], col="blue", lwd=2)
lines(x, stb.q.C$Q[1,], col="red",  lwd=2)
lines(x, stb.q.C$Q[2,], col="red",  lwd=2)
lines(x, stb.rank$Q[1,],  col="cyan", lwd=2)
lines(x, stb.rank$Q[2,],  col="cyan", lwd=2)
legend("top", legend=c("R-quantile", "C-quantile", "rank-based"),
       fill=c("blue", "red", "cyan"))

# varying Ncpu for the C-implementation of the quantile-based algo
system.time(stb.q.C  <- fastSTB(mat, Ncpu=4))
system.time(stb.q.C  <- fastSTB(mat, Ncpu=6))
system.time(stb.q.C  <- fastSTB(mat, Ncpu=8))
system.time(stb.q.C  <- fastSTB(mat, Ncpu=10))
system.time(stb.q.C  <- fastSTB(mat, Ncpu=12))

## End(Not run)
```

`SASquantile` *Implements SAS-quantile Calculation (PCTLDEF=5) as Described by SAS-Help.*

Description

This implementation seems to be identical with 'type=2' in function `quantile` but less efficiently implemented, i.e. for large vectors x it is much slower than the built-in quantile-function.

Usage

```
SASquantile(x, prob, tol = 1e-12, type = c("R", "C"))
```

Arguments

x	(numeric) vector
prob	(numeric) value 0 < prob < 1
tol	(numeric) value used for checking numerical equivalence
type	(character) "R" = uses the R-implementation, "C" = calls the C-implementation n

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
SASquantile(1:100, .75)

### compare to R-default
quantile(1:100, .75)

### or to what R calls SAS-definition
quantile(1:100, .75, type=3)

# should work for any vector (no seed)
v <- rnorm(50000,20,7)
Q.R2   <- quantile(v, probs=c(.01, .025, .05, .1, .25, .5, .75, .9, .95, .975, .99), type=2)
Q.SAS.R <- SASquantile(v, prob=c(.01, .025, .05, .1, .25, .5, .75, .9, .95, .975, .99), type="R")
Q.SAS.C <- SASquantile(v, prob=c(.01, .025, .05, .1, .25, .5, .75, .9, .95, .975, .99), type="C")

Q.R2
Q.SAS.R
Q.SAS.C

## End(Not run)
```

---

stb *Generic Method for computing Simultaneous Tolerance Bounds (bands and intervals) for fitted models or numeric vectors.*

Description

If the generic method is applied to an object of class VCA function stb.VCA is called and all its arguments may be specified. Otherwise, function stb.default is called and all its arguments may be specified. The latter was developed for numeric vectors (see ?stb.default). Therefore this method will most likely produce strange results on other types of objects than VCA and numeric vectors.

Usage

```
stb(obj, ...)
```

Arguments

obj	(object) passed on
...	additional parameters

`stb.default` *Simultaneous Tolerance Bands (STB).*

Description

Compute And/Or Plot Simultaneous Tolerance Bands for numeric vectors.

Usage

```
## Default S3 method:
stb(
  obj,
  N = 10000L,
  alpha = 0.05,
  rand.func = rnorm,
  tol = 1e-04,
  max.iter = 100L,
  algo = c("rank", "C", "R"),
  Ncpu = 1,
  q.type = 2L,
  stb.col = "#0000FF40",
  col.points = "black",
  col.out = "red",
  col.pwb = "#0000FF40",
  main = NULL,
  add.pwb = FALSE,
  quiet = FALSE,
  add = FALSE,
  plot = TRUE,
  legend = FALSE,
  timer = FALSE,
  pch = 16,
  pch.out = 16,
  seed = NULL,
  ...
)
```

Arguments

obj	(numeric) vector, which is supposed to be N(my, sigma^2)-distributed
N	(integer) value specifying the number of random samples to be used for constructing the STB

alpha	(numeric) value specifying the simultaneous tolerance level, i.e. 100(1-alpha)% of all 'N' random samples have to be completely enclosed by the bounds of the STB
rand.func	(function) a function which generates random samples, e.g. rand.func=rnorm which corresponds to random sampling from the standard normal distribution. Another example is defining func=function(n)rchisq(n=n, df=3, ncp=2) and using rand.func=func. See examples for further examples.
tol	(numeric) value specifying the max. acceptable deviation from 'alpha' used in the bisection algorithm
max.iter	(integer) value specifying the max. number of iteration for finding the bounds of the STB
algo	(character) (string) specifying the method to be used for constructing a 100(1-alpha)% STB, choose "rank" for the rank-based, "C" for a C-implementation of the quantile-based, and "R" for an R-implentation of the quantile-based algorithm (see details). "C" uses SAS PCTLDEF5 definition of quantiles, whereas "R" can use any of the built-in R types of quantiles (see quantile.
Ncpu	(integer) specifying the number cores/CPUs to be used, for N>1 multi-processing is applied
q.type	(integer) the quantile-type used if algo="R", see ? quantile for details.
stb.col	(character) string, a valid specification of a color to be used for the STB
col.points	(character) color for the points in the QQ-plot
col.out	(character) color for points outsied of the 100(1-alpha)% STB
col.pwb	(character) color for the point-wise STB (not adjusted for multiplicity), defaults to "#0000FF40" which is "blue" with 80% transparency
main	(characer) string for a main title appearing over the plot
add.pwb	(logical) should the point-wise tolerance band be plotted for comparison?
quiet	(logical) TRUE = no additional output ist printed (progress bar etc.)
add	(logical) TRUE = the 100(1-alpha)% STB is added to an existing plot
plot	(logical) TRUE = either a QQ-plot with STB (add=FALSE) or a STB added to an existing plot (add=TRUE) is plotted. FALSE = only computations are carried out without plotting, an object of class 'STB' is returned which can be stored an plotted later on, e.g. to avoid computing an STB every time a Sweave/mWeave report is updated
legend	(logical) TRUE a legend is plotted "topleft"
timer	(logical) TRUE = the time spent for computing the STB will be printed
pch	(integer) plotting symbols for the QQ-plot
pch.out	(integer) plotting symbols for outlying points
seed	(numeric) value interpreted as integer, setting the random number generator (RNG) to a defined state
...	further graphical parameters passed on

Details

Function takes a numeric vector 'vec' and computes the 100(1-alpha)%-simultaneous tolerance band (STB) for the (DEFAULT)Null-hypothesis H0: vec~N(my, sigma^2) distributed, which is equal to checking whether the residuals of the simplest linear model y = mu + e (y~1) are normally distributed, i.e. 'e ~ N(0, sigma^2)'. By specification of argument `rand.func` other null-distributions can be specified. One has to specify a function with a single argument 'n', which returns a random sample with 'n' observations, randomly drawn from the desired null-distribution (see description argument `rand.func` below). Note that all random samples as well as vector `vec` will be centered to mean=0 and scaled to sd=1.

One can choose between three methods for constructing the 100(1-alpha)% STB. There are two implementations of the quantile-based algorithm ("C", "R" see 1st reference) and one of the rank-based algorithm (see 2nd reference). Methods "C" and "R" can be run in parallel. The rank-based algorithm does not benefit form parallel processing, at least not in the current implementation. It is still the default and recommended for small to medium sized vectors and 10000 <= N <= 20000 simulations, which should be sufficiently accurate reflect the null-distribution. The "C" and "R" options refer to implementations of the quantile-based algorithm. The "C" implementation benefits most from using multiple cores, i.e. the larger 'Ncpu' the better, and should be used for large problems, i.e. rather large number of elements and large number of simulations.

The table below gives an impression how these algorithms perform. Runtimes were measured under Windows 7 on a Intel Xeon E5-2687W 3.1 GHz workstation with 16 logical cores and 16 GB RAM. The runtime of the C-implementation of the quantile-based algorithm is denoted as "t_qC12" applied parallely with 12 cores. Each setting was repeated 10 times and the overall run time was then divided by 10 providing sufficiently robust simulation results. Note, that for smaller problem sizes a large proportion of the overall runtime is due to simulating, i.e. drawing from the null-distribution.

_____N_obs	_____N_sim	____t_rank	____t_qC12
25	5000	0.4s	0.5s
25	10000	0.8s	1.3s
50	10000	1.0s	3.2s
100	10000	1.7s	2.9s
100	20000	3.0s	4.8s
225	20000	5.1s	8.3s
300	30000	9.6s	17.2s
300	50000	16.1s	24.9s
1000	50000	47.8s	123.5s

Value

invisibly returns a list-type object of class `STB`, which comprises all arguments accepted by this function.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A., Jensen, U., Piepho, H.P. (2011), Checking assumptions of normality and homoscedasticity in the general linear model. Communications in Statistics - Simulation and Computation; S. 141-154

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

See Also

plot.STB

Examples

```
### log-normal vector to be checked for normality
## Not run:
set.seed(111)
stb(exp(rnorm(30)), col.out="red", legend=TRUE)

### uniformly distributed sample checked for Chi-Squared Distribution with DF=1, degrees of freedom
set.seed(707)
stb(runif(25, -5, 5), rand.func=function(n){rchisq(n=n, df=1)},
    col.out="red", legend=TRUE, main="Chi-Squared with DF=1")

### check whether an Chi-Squared (DF=1) random sample better fits
stb(rchisq(25, df=1), rand.func=function(n){rchisq(n=n, df=1)},
    col.out="red", legend=TRUE, main="Chi-Squared with DF=1")

### add STB to an existing plot
plot(sort(rnorm(30)), sort(rnorm(30)))
stb(rnorm(30), add=TRUE)

### compute STB for later use and prevent plotting
STB <- stb(rnorm(30), plot=FALSE)

## End(Not run)
```

`stb.VCA` *Simultaneous Tolerance Bounds on Residuals and Random Effects for 'VCA' Objects.*

Description

Simulate N-times data incorporating the estimated variance-covariance matrix of observations y and construct a 100(1-alpha)% simultaneous tolerance band.

Usage

```
## S3 method for class 'VCA'
stb(
  obj,
  term = NULL,
  mode = c("raw", "student", "standard", "pearson"),
  N = 5000,
  alpha = 0.05,
  algo = c("rank", "R", "C"),
  q.type = 2L,
  plot = TRUE,
  legend = TRUE,
  orient = 1,
  main1 = NULL,
  main2 = NULL,
  seed = NULL,
  type = 1,
  pb = TRUE,
  parallel = TRUE,
  Ncpu = 2,
  ...
)
```

Arguments

obj	(VCA) object
term	(character, integer) specifying a type of residuals if one of c("conditional", "marginal"), or, the name of a random term (one of obj$re.assign$terms). If 'term' is a integer, it is interpreted as the i-th random term in 'obj$re.assign$terms'.
mode	(character) string specifying a possible transformation of random effects or residuals (see residuals.VCA and ranef.VCAfor details)
N	(integer) specifying the number of simulated datasets y_sim
alpha	(numeric) value 0 < alpha < 1 specifying the min. 100(1-alpha)% coverage of the simultaneous tolerance band (STB)
algo	(character) (string) specifying the method to be used for constructing a 100(1-alpha)% STB, choose "rank" for the rank-based, "C" for a C-implementation of the quantile-based, and "R" for an R-implentation of the quantile-based algorithm (see details).
q.type	(integer) value specifying the quantile type to be used as offered in quantile in case 'algo="R"'. Whenever 'algo="C"', quantiles are computed according to SAS PCTLDEF5, which is identical to type 2. The rank-based algorithm does not employ quantiles.
plot	(logical) TRUE = create 'stbVCA' object and plot it, FALSE = only create the 'stbVCA' object
legend	(logical) TRUE = add legend to the plot(s)

orient	(integer) in QQ-plot, specifying whether to plot expected values vs. observed values (1) or observed vs. expected (2)
main1	(character) string specifying an user-defined main-title of the 'type=1' plot (STB)
main2	(character) string specifying an user-defined main-title of the 'type=2' plot (STI)
seed	(integer) value used as seed for the RNG
type	(integer) 1 = QQ-plot with simultaneous tolerance band (STB), 2 = residual plot with simultaneous tolerance interval (STI), 3 = both plot at once
pb	(logical) TRUE = a text-based progress bar will display the simulation progress
parallel	(logical) TRUE = parallel processing will be attempted on 'Ncpu' cores of the local machine. FALSE = no parallel processing applied, only in this case, 'pb' will have an effect, since this is only available for non-parallel processing.
Ncpu	(integer) specifying the number of CPUs on which the parallelization will be carried out. In case that 'Ncup' is larger than the number of existing CPUs, the max. number of CPUs will be used instead. Note, that setting 'Ncpu' to the max. number available may not result in the min. time spent on computing.
...	additional arguments passed to other methods

Details

A Linear Mixed Models, noted in standard matrix notation, can be written as $y = Xb + Zg + e$, where y is the column vector of observations, X and Z are design matrices assigning fixed (b), respectively, random (g) effects to observations, and e is the column vector of residual errors.

Here, simulation is performed incorporating the variance-covariance matrix $V = ZGZ^T + R$ of observations y. There are two types of random variates in a mixed model, random effects g and residual errors e. These follow a multivariate normal distribution with expectation zero and covariance matrices G and R. See the 1st reference for a detailed description of the properties. Following Schuetzenmeister and Piepho (2012), a Cholesky decomposition $V = CC'$ is applied to V, yielding the upper triangular matrix C, which can be used to simulate a new set of observations $y_{sim} = Cz$, where z is a vector of independent standard normal deviates of the same size as y. Actually, $y_sim = C'z$ is used here, because the Cholesky decomposition in R is defined as $V = C'C$. For each simulated dataset, random variates of interest ('term') are extracted, possibly transformed ('mode') and stored in ordered form (order statistics) in a Nxn matrix, N being the number of simulated datasets and n being the number of random variates of interest. For each column of this matrix tolerance intervals are computed iteratively untill the joint coverage is as close to but >= 100(1-alpha)/ iterations is reached. This quantile-based algorithm is exact for $N - - > Infinity$.

SAS-quantile definition PCTLDEF=5 is used in the fast C-implementation of the STB-algorithm (SASquantile), i.e. in case algo="C". One can compute and plot two types of plots (see argument 'type'). Simultaneous tolerance bands (STB) are helpful in assessing the general distribution of a random variate, i.e. checking for departure from the normality assumption. Outlying observations may also be detected using STB. Simultaneous tolerance intervals (STI) are taylored for identification of extreme values (possible outliers). STI are a simplification of STB, where simultaneous coverage is only required for extreme values of each simulation, i.e. an STB is constructed from the min and max values from all N simulations. This results in lower and upper bounds, which can be used in residuals plots for assessing outliers.

One can choose between 3 methods for constructing the 100(1-alpha)% STB. The fastest one is the rank-based algorithm ("rank"), which should only be applied for reasonably large number of simulations (rule of thumb: N>5000). For fewer simulations, the quantile-based algorithm is recommended. It exists in two flavours, a native R-implementation ("R") and a pure C-implementation ("C"). Both can be applied using parallel-processing (see arguments 'parallel' and 'Ncpu'). Only the R-implementation allows to specify a specific quantile-definition other than `type=2` of function `quantile`.

Value

(stbVCA) object, a list with all information needed to create the QQ-plot with ~100(1-alpha)% STB.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

Schuetzenmeister, A., Jensen, U., Piepho, H.P., 2012. Checking the assumptions of normality and homoscedasticity in the general linear model using diagnostic plots. Communications in Statistics-Simulation and Computation 41, 141-154.

See Also

`getSTB`, `fastSTB`, `rankSTB`

`fastSTB`

Examples

```
## Not run:
library(VCA)
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
fit

# use studentized conditional residuals
stb.obj1 <- stb(fit, term="cond", mode="student", N=1000)

# plot it again
plot(stb.obj1)

# now request also plotting the corresponding residual plot
# capture additional computation results which are invisibly
# returned
stb.obj1 <- plot(stb.obj1, type=3)

# use other type of legend in QQ-plot
```

```
plot(stb.obj1, stb.lpos="topleft")

# use random effects "day" and apply standardization
stb.obj2 <- stb(fit, term="day", mode="stand", N=1000)
# plot it again
plot(stb.obj2)

# more complex example
data(Orthodont)
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))
fit.Ortho <- anovaMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)

# studentized conditional residuals
stb.cr.stud <- stb(fit.Ortho, term="cond", mode="stud", N=1000)

# same model fitted via REML (same covariance structure of random effects by
# constraining it to be diagonal)
fit.Ortho.reml1 <- remlMM(distance~Sex*age2+(Subject)*age2, Ortho, cov=FALSE)

# allow block-diagonal covariance structure of random effects due to non-zero
# correlation between intercept and slope of random regression part,
# not 'cov=TRUE' is the default
fit.Ortho.reml2 <- remlMM(distance~Sex*age2+(Subject)*age2, Ortho)
fit.Ortho.reml1
fit.Ortho.reml2

# covariance matrices of random effects 'G' differ
getMat(fit.Ortho.reml1, "G")[1:10, 1:10]
getMat(fit.Ortho.reml2, "G")[1:10, 1:10]

# therefore, (conditional) residuals differ
resid(fit.Ortho.reml1)
resid(fit.Ortho.reml2)

# therefore, STB differ

# studentized conditional residuals
system.time({
stb.cr.stud.reml1 <- stb(fit.Ortho.reml1, term="cond", mode="stud",
                         N=5000, Ncpu=2, seed=11) })
system.time({
stb.cr.stud.reml2 <- stb(fit.Ortho.reml2, term="cond", mode="stud",
                         N=5000, Ncpu=4, seed=11) })

# same seed-value should yield identical results
system.time({
stb.cr.stud.reml3 <- stb(fit.Ortho.reml2, term="cond", mode="stud",
                         N=5000, Ncpu=4, seed=11) })

par(mfrow=c(1,2))
plot(stb.cr.stud.reml2)
```

```
plot(stb.cr.stud.reml3)

# both type of plots side-by-side
plot(stb.cr.stud.reml2, type=3)

# and enabling identification of points
# identified elements in the 1st plot will
# be automatically added to the 2nd one
plot(stb.cr.stud.reml2, type=3, pick=TRUE)

# raw "day" random effects
stb.re.subj <- stb(fit.Ortho, term="Subject", N=1000)

# identify points using the mouse
stb.re.subj <- plot(stb.re.subj, pick=TRUE, type=3)

# now click on points

## End(Not run)
```

Index