Package 'robustlmm'

October 25, 2026

Type Package

Title Robust Linear Mixed Effects Models

Version 3.5.1-1

Date 2026-10-25

Author Manuel Koller [aut, cre]

Maintainer Manuel Koller <kollerma@proton.me>

Description Implements the Robust Scoring Equations estimator to fit linear mixed effects models robustly.
Robustness is achieved by modification of the scoring equations combined with the Design Adaptive Scale approach.

License GPL-2

URL https://github.com/kollerma/robustlmm

LazyLoad yes

Depends lme4 (>= 2.0-1), Matrix (>= 1.6-2), R (>= 3.6.0)

Suggests ggplot2, reshape2, microbenchmark, emmeans (>= 1.4), estimability, lqmm, MASS, RColorBrewer, skewt, fs, dplyr, ggh4x, testthat, robustvarComp, confintROB, generics, tibble, R.rsp

VignetteBuilder R.rsp

Imports lattice, nlme, methods, robustbase (>= 0.93), xtable, Rcpp (>= 0.12.2), fastGHQuad, numDeriv, parallel, rlang, utils, reformulas

Collate 'ghq.R' 'psiFunc2.R' 'redescendingPsi.R' 'sscale.R' 'checks.R' 'AllClass.R' 'control.R' 'rlmer.R' 'accessors.R' 'fromLme4.R' 'DAS-scale.R' 'fit.effects.R' 'helpers.R' 'AllGeneric.R' 'lmer.R' 'mutators.R' 'plot.R' 'generateAnovaDatasets.R' 'generateLongitudinalDatasets.R' 'plotLongitudinalBySubject.R' 'generateMixedEffectDatasets.R' 'generateSensitivityCurveDatasets.R' 'manageDatasets.R' 'ransac.R' 'influence.R' 'influence_full.R' 'df_satterthwaite.R' 'fragility.R' 'design_breakdown.R'

'bootstrapWald.R' 'simulate.R' 'anova.R' 'contrast.R'
'tidiers.R' 'fitDatasets.R' 'processFit.R' 'processFile.R'
'simulationStudies.R' 'asymptoticEfficiency.R' 'emmeans.R'
'options.R'

LinkingTo Rcpp, robustbase, Matrix

Encoding UTF-8

Language en-US

RcppModules psi_function_module

Config/build/clean-inst-doc false

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-10-25 19:50:02 UTC

Contents

robustlmm-package . . . 3
anova.rlmerMod . . . 4
asymptoticVariance . . . 7
bindDatasets . . . 8
bisquarePsi . . . 9
caseweightIF . . . 10
chgDefaults . . . 11
compare . . . 11
confint.rlmerMod . . . 13
contrast_test . . . 15
cooks.distance.rlmerMod . . . 17
createDatasetsFromList . . . 18
createRhoFunction . . . 19
design_breakdown . . . 20
extractTuningParameter . . . 22
fitDatasets_lmer . . . 23
fragility . . . 29
generateAnovaDatasets . . . 30
generateLongitudinalDatasets . . . 33
generateMixedEffectDatasets . . . 37
generateRepeatedMeasuresDatasets . . . 38
generateSensitivityCurveDatasets . . . 40
getME . . . 41
hatvalues.rlmerMod . . . 45
implicitIF_full . . . 46
influence.rlmerMod . . . 47
lapplyDatasets . . . 48
loadAndMergePartialResults . . . 49
lqqPsi . . . 49

makeBisquarePsi . . . 50
makeRobustbasePsi . . . 51
makeSScalePsi . . . 52
mergeProcessedFits . . . 53
other . . . 54
partialMoment_standardNormal . . . 55
plot-methods . . . 55
plot.rlmerMod . . . 57
plotLongitudinalBySubject . . . 58
predict.rlmerMod . . . 60
prepareMixedEffectDataset . . . 62
processDatasetsInParallel . . . 63
processFile . . . 65
processFit . . . 66
psi-functions . . . 71
psi2propII . . . 72
ransac_basin_radius . . . 72
ransac_lme4 . . . 74
refit.rlmerMod . . . 76
residuals.rlmerMod . . . 77
rlmer . . . 78
rlmerControl . . . 84
rlmerMod-class . . . 86
rlmerMod-tidiers . . . 89
rlmer_ransac . . . 91
robustlmm-options . . . 93
saveDatasets . . . 95
shortenLabelsKS2022 . . . 95
simulate.rlmerMod . . . 96
splitDatasets . . . 98
sScalePsi . . . 99
vcov.rlmerMod . . . 100
vcov_sandwich . . . 101
viewCopyOfSimulationStudy . . . 102

Index **104**

`robustlmm-package` *Robust linear mixed effects models*

Description

`robustlmm` provides functions for estimating linear mixed effects models in a robust way.

The main workhorse is the function `rlmer`; it is implemented as direct robust analogue of the popular `lmer` function of the `lme4` package. The two functions have similar abilities and limitations. A wide range of data structures can be modeled: mixed effects models with hierarchical as well as complete or partially crossed random effects structures are possible. While the `lmer` function is

optimized to handle large datasets efficiently, the computations employed in the `rlmer` function are more complex and for this reason also more expensive to compute. The two functions have the same limitations in the support of different random effect and residual error covariance structures. Both support only diagonal and unstructured random effect covariance structures.

The `robustlmm` package implements most of the analysis tool chain as is customary in R. The usual functions such as `summary`, `coef`, `resid`, etc. are provided as long as they are applicable for this type of models (see `rlmerMod-class` for a full list). The functions are designed to be as similar as possible to the ones in the `lme4` package to make switching between the two packages easy.

Details on the implementation and example analyses are provided in the package vignette available via `vignette("rlmer")` (Koller 2016).

References

Manuel Koller (2016). robustlmm: An R Package for Robust Estimation of Linear Mixed-Effects Models. Journal of Statistical Software, 75(6), 1-24. doi:10.18637/jss.v075.i06

Koller M, Stahel WA (2022). "Robust Estimation of General Linear Mixed Effects Models." In PM Yi, PK Nordhausen (eds.), Robust and Multivariate Statistical Methods, Springer Nature Switzerland AG.

Manuel Koller (2013). Robust estimation of linear mixed models. (Doctoral dissertation, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20997, 2013).

`anova.rlmerMod` *Analysis of variance for an rlmer fit.*

Description

Three modes.

Single fit `anova(fit)` a per-term robust Wald table: each term is tested with $\hat\beta_t^\top \hat V_t^{-1} \hat\beta_t$, $\hat V$ from `vcov(fit, type = vcov_type)`, against χ^2 (default covariance) or a small-sample F reference (sandwich; see `ddf`).

Nested fits differing only in the fixed effects a robust Wald test of the extra coefficients with the same covariance.

Nested fits differing in the random effects Wald asymptotics fail on the boundary of the parameter space (Self and Liang 1987), so `test = "Wald"` switches to `test = "boot"`: a parametric bootstrap from `fit0`'s estimates of the quasi-deviance difference $D = 2\sum_i[\rho_e(r_i^{(0)}/\hat\sigma_0) - \rho_e(r_i^{(1)}/\hat\sigma_0)]$ (Heritier and Ronchetti 1994; Cantoni and Ronchetti 2001; Heritier et al. 2009). `test = "score"` is an experimental robust score test for added independent variance components.

Only pairwise comparisons are implemented.

Usage

```
## S3 method for class 'rlmerMod'
anova(
  object,
  ...,
  test = c("Wald", "boot", "score"),
  null = c("parametric", "robust"),
  nuisance = c("threshold", "raw"),
  vcov_type = c("default", "sandwich"),
  ddf = c("auto", "none", "satterthwaite", "cluster"),
  nsim = 1000L,
  seed = NULL,
  verbose = FALSE,
  cluster = NULL
)
```

Arguments

object	An rlmerMod object.
...	A second rlmerMod object for pairwise comparison.
test	"Wald" (default; nested fixed-effects models), "boot" (parametric bootstrap; the default for variance-component tests) or "score" (experimental; see Details).
null	Bootstrap generator for test = "boot": "parametric" (default) generates from fit0's estimates, "robust" (experimental) from a refit without heavily down-weighted groups (see Details).
nuisance	How the test = "boot" generator treats variance components of the null model estimated close to zero: "threshold" (default) sets a component to zero when its relative variance is below $\sqrt{2 \log J}$ times its standard error at zero (J the number of groups), which keeps the bootstrap consistent when a nuisance component is zero (Guedon, Baey and Kuhn 2024); "raw" uses the estimates as they are (the behaviour before version 3.5.1). Compound-symmetric and AR(1) blocks are never thresholded. See the section "Near-zero nuisance variance components in the bootstrap test" of vignette("simulationStudies").
vcov_type	Covariance for the Wald paths, passed to vcov.
ddf	Reference distribution for the Wald paths. "auto" (default): "none" for vcov_type = "default" and "cluster" for vcov_type = "sandwich". "none": χ^2. "satterthwaite": F with robust Satterthwaite denominator degrees of freedom (default covariance and a single grouping factor only; otherwise the default reference, with a warning). "cluster": the CR2 sandwich with approximate Hotelling degrees of freedom (sandwich only; see Details).
nsim	Bootstrap replicates; default 1000 for test = "boot" and 199 for test = "score".
seed	Optional RNG seed for the bootstrap.
verbose	Bootstrap progress messages.
cluster	For vcov_type = "sandwich", the clusters, passed to vcov_sandwich; needed for models with several grouping factors, e.g. cluster = "school" for (1 | school/class).

Details

Sandwich covariance. With few clusters the G1 cluster sandwich is biased down for between-cluster terms, and χ^2 tests based on it are anti-conservative. For `vcov_type = "sandwich"` the default reference is therefore the bias-reduced CR2 sandwich (`vcov_sandwich`; Bell and McCaffrey 2002) with the approximate Hotelling F-test of Pustejovsky and Tipton (2018), $F = (\eta - k + 1)W/(\eta k)$ on $(k, \eta - k + 1)$ degrees of freedom. It is much closer to its nominal level but can stay somewhat anti-conservative for between-cluster terms; see the section "Small-sample references for the cluster sandwich" of `vignette("simulationStudies")`. CR2 needs every random effect nested within one cluster; otherwise the G1 sandwich with $F(k, J-1)$ is used, and the table heading says which. A term that a single cluster informs on its own (e.g. a factor level present in one cluster) gets no reference distribution (`NA`), with a warning.

Bootstrap test (experimental). `test = "boot"` holds its level under error contamination but is anti-conservative when whole groups are contaminated, because the bootstrap is generated from estimates those groups have biased. It warns when the null fit heavily downweights a group (smallest random-effect robustness weight below 0.5). `null = "robust"` generates from a refit without the heavily downweighted groups instead (the observed statistic still uses all the data); it falls back to the plain generator when no group or more than half of the groups are flagged, when removing them would leave the design rank-deficient, or when the refit fails, and the table heading says which applied. When both fits put the tested variance on the boundary, D sits at its atom at zero and the p-value is 1. See the ANOVA section of `vignette("robustlmm-3.5.0")`.

Score test (experimental). `test = "score"` uses the robust null fit only. Per cluster j it forms the bounded contribution $s_j = (v_j^\top \psi(\tilde{r}_j))^2 - \kappa_1 \|v_j\|^2$, with $\tilde{r}_j$ the whitened marginal residuals, v_j the whitened design column of a tested component and $\kappa_1 = E[\psi(Z)^2]$, and calibrates the statistic by a parametric bootstrap that refits only the null model. One added component is tested with the self-normalised sum of the s_j; several are tested jointly with the cone (chi-bar) statistic of the whitened score vector, with a warning when the tested directions are nearly collinear. Scope: both fits on the same data, with the same fixed effects and a single grouping factor, and an alternative that adds uncorrelated scalar components; otherwise it stops with an error. Contamination aligned with a tested direction cannot be distinguished from the alternative; the per-cluster contributions (`attr(., "boot")$s_j`) show which clusters drive a rejection. Simulation results: the ANOVA section of `vignette("robustlmm-3.5.0")` (one component) and the section "Several added variance components" of `vignette("simulationStudies")`.

Value

An `"anova"` data.frame; the bootstrap path attaches `attr(., "boot") = list(D_boot, n_fail, nsim, D_obs)`. The score path attaches `attr(., "boot") = list(S_boot, n_fail, nsim, S_obs, s_j)` with `s_j` the named per-cluster contributions to the observed statistic.

Lifecycle

The Wald tests are stable. Experimental, and subject to change: `test = "score"`, and for `test = "boot"` the options `null = "robust"` and `nuisance = "threshold"` (its default).

References

Self SG, Liang KY (1987). Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. *Journal of the American Statistical Association* 82(398),

605–610.

Guedon T, Baey C, Kuhn E (2024). *Bootstrap test procedure for variance components in nonlinear mixed effects models in the presence of nuisance parameters and a singular Fisher information matrix*. arXiv:2306.10779.

Bell RM, McCaffrey DF (2002). Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology* 28(2), 169–181.

Pustejovsky JE, Tipton E (2018). Small-sample methods for cluster-robust variance estimation and hypothesis testing in fixed effects models. *Journal of Business & Economic Statistics* 36(4), 672–683.

Heritier S, Ronchetti E (1994). *Robust bounded-influence tests in general parametric models*. JASA 89(427): 897–904.

Cantoni E, Ronchetti E (2001). *Robust inference for generalized linear models*. JASA 96(455): 1022–1030.

Heritier S, Cantoni E, Copt S, Victoria-Feser MP (2009). *Robust Methods in Biostatistics*. Wiley.

See Also

vcov, confint

`asymptoticVariance` *Compute Asymptotic Efficiencies*

Description

`asymptoticEfficiency` computes the theoretical asymptotic efficiency for an M-estimator for various types of equations.

Usage

```
asymptoticVariance(
  psi,
  equation = c("location", "scale", "eta", "tau", "mu"),
  dimension = 1
)

asymptoticEfficiency(
  psi,
  equation = c("location", "scale", "eta", "tau", "mu"),
  dimension = 1
)

findTuningParameter(
  desiredEfficiency,
  psi,
  equation = c("location", "scale", "eta", "tau", "mu"),
```

```
  dimension = 1,
  interval = c(0.15, 50),
  ...
)
```

Arguments

psi	object of class psi_func
equation	equation to base computations on. "location" and "scale" are for the univariate case. The others are for a multivariate location and scale problem. "eta" is for the shape of the covariance matrix, "tau" for the size of the covariance matrix and "mu" for the location.
dimension	dimension for the multivariate location and scale problem.
desiredEfficiency	scalar, specifying the desired asymptotic efficiency, needs to be between 0 and 1.
interval	interval in which to do the root search, passed on to uniroot.
...	passed on to uniroot.

Details

The asymptotic efficiency is defined as the ratio between the asymptotic variance of the maximum likelihood estimator and the asymptotic variance of the (M-)estimator in question.

The computations are only approximate, using numerical integration in the general case. Depending on the regularity of the psi-function, these approximations can be quite crude.

References

Maronna, R. A., Martin, R. D., Yohai, V. J., & Salibián-Barrera, M. (2019). Robust statistics: theory and methods (with R). John Wiley & Sons., equation (2.25)

Rousseeuw, P. J., Hampel, F. R., Ronchetti, E. M., & Stahel, W. A. (2011). Robust statistics: the approach based on influence functions. John Wiley & Sons., Section 5.3c, Paragraph 2 (Page 286)

---

bindDatasets *Bind Generated Datasets*

Description

This method can be used to bind multiple datasets generated using different random generators into one large dataset. The underlying dataset needs to be the same.

Usage

```
bindDatasets(..., datasetList = list(...))
```

Arguments

...	multiple datasets to be bound together
datasetList	list of datasets created with one of the generate dataset functions

Value

merged list with generators and the contents of the prepared dataset. See 'prepareMixedEffectDataset and generateAnovaDatasets for a description of the contents.

Author(s)

Manuel Koller

See Also

splitDatasets

Examples

```
datasets1 <- generateAnovaDatasets(2, 4, 4, 4)
datasets2 <- generateAnovaDatasets(2, 4, 4, 4)
datasets <- bindDatasets(datasets1, datasets2)
data <- datasets$generateData(1)
stopifnot(data$numberOfDatasets == 4,
          all.equal(datasets2$generateData(1), datasets$generateData(3),
                    check.attributes = FALSE),
      all.equal(datasets2$sphericalRandomEffects(1), datasets$sphericalRandomEffects(3)),
          all.equal(datasets2$createXMatrix(data), datasets$createXMatrix(data)),
          all.equal(datasets2$createZMatrix(data), datasets$createZMatrix(data)))

preparedDataset <- prepareMixedEffectDataset(Reaction ~ Days + (Days|Subject), sleepstudy)
datasets1 <- generateMixedEffectDatasets(2, preparedDataset)
datasets2 <- generateMixedEffectDatasets(2, preparedDataset)
datasets <- bindDatasets(datasets1, datasets2)
data <- datasets$generateData(1)
stopifnot(data$numberOfDatasets == 4,
          all.equal(datasets2$generateData(1), datasets$generateData(3),
                    check.attributes = FALSE),
      all.equal(datasets2$sphericalRandomEffects(1), datasets$sphericalRandomEffects(3)),
          all.equal(datasets2$createXMatrix(data), datasets$createXMatrix(data)),
          all.equal(datasets2$createZMatrix(data), datasets$createZMatrix(data)))
```

---

bisquarePsi *Default Tukey bisquare psi-function with* c = 4.685.

Description

Pre-built psi_func_rcpp from makeBisquarePsi with the conventional tuning constant c = 4.685 ($\approx 95\%$ efficiency at the normal model). The bisquare redescends comparatively fast; lqqPsi is the recommended redescender.

See Also

lqqPsi, makeRobustbasePsi.

---

caseweightIF — *Case-weight (Hampel) influence function of the fixed effects of a fitted* rlmerMod *object.*

Description

Returns $IF(y_i) = -\hat{J}^{-1}\psi(p\hat{a}r; y_i)$ (the Hampel object built from the score *value* ψ), reusing the Jacobian $\hat{J}$ formed by implicitIF. This is the right object for robustness quantities (gross-error sensitivity, breakdown), distinct from the local-shift sensitivity returned by implicitIF for a redescending ψ.

Usage

```
caseweightIF(fit, idx = NULL, use.expected = FALSE)
```

Arguments

fit	An rlmerMod object.
idx	Optional integer indices selecting observations to compute (default: all observations).
use.expected	Passed to implicitIF; default FALSE (empirical Jacobian).

Details

$\hat{\sigma}, \hat{\theta}$ are held fixed (partial IF).

Value

A list with IF_beta ($p \times |idx|$), IF_u ($q \times |idx|$), IF_b, and the per-observation score-contribution columns Gb_cw, Gu_cw.

See Also

vcov_sandwich, vcov

chgDefaults *Change default arguments*

Description

Change the default arguments for a psi_func_rcpp object

Usage

```
## S4 method for signature 'psi_func_rcpp'
chgDefaults(object, ...)
```

Arguments

object	instance to convert
...	arguments to change

Note

Note that names of named arguments are ignored. Only the order of the arguments considered when assigning new arguments.

Examples

```
sPsi <- chgDefaults(smoothPsi, k=2)
curve(sPsi@psi(x), 0, 3)
curve(smoothPsi@psi(x), 0, 3, col="blue", add=TRUE)
```

compare *Create comparison charts for multiple fits*

Description

Use compare to quickly compare the estimated parameters of the fits of multiple lmerMod or rlmerMod objects.

Usage

```
compare(..., digits = 3, dnames = NULL, show.rho.functions = TRUE)

## S3 method for class 'lmerMod'
getInfo(object, ...)

## S3 method for class 'rlmerMod'
getInfo(object, ...)
```

```
## S3 method for class 'comparison.table'
xtable(
  x,
  caption = NULL,
  label = NULL,
  align = NULL,
  digits = NULL,
  display = NULL,
  ...
)

## S3 method for class 'xtable.comparison.table'
print(
  x,
  add.hlines = TRUE,
  latexify.namescol = TRUE,
  include.rownames = FALSE,
  ...
)

getInfo(object, ...)
```

Arguments

`...`	objects to compare, or, for the `xtable` functions: passed to the respective `xtable` function.
`digits`	number of digits to show in output
`dnames`	names of objects given as arguments (optional)
`show.rho.functions`	whether to show rho functions in output.
`object`	object
`x`	object of class "comparison.table" or "xtable.comparison.table"
`caption`	see `xtable`.
`label`	see `xtable`.
`align`	see `xtable`.
`display`	see `xtable`.
`add.hlines`	replace empty lines in comparison table by hlines. Supersedes `hline.after` argument of `print.xtable`.
`latexify.namescol`	replace “sigma” and “x” in the first column by latex equivalents.
`include.rownames`	include row numbers (the object returned by `xtable.comparison.table` includes names in the first column)

Details

The functions `xtable.comparison.table` and `print.xtable.comparison.table` are wrapper functions for the respective `xtable` and `print.xtable` functions.

The function `getInfo` is internally used to prepare object for producing a comparison chart in `compare`.

Value

`getInfo` returns a list with estimated coefficients, estimated variance components, sigma, deviance and parameter configuration used to fit.

See Also

`xtable`

`print.xtable`

Examples

```
## Not run:
  fm1 <- lmer(Yield ~ (1|Batch), Dyestuff)
  fm2 <- rlmer(Yield ~ (1|Batch), Dyestuff)
  compare(fm1, fm2)
  require(xtable)
  xtable(compare(fm1, fm2))
  str(getInfo(fm1))

## End(Not run)
```

`confint.rlmerMod` *Confidence intervals for the fixed-effect coefficients of an rlmer fit.*

Description

Two routes.

`method = "Wald"` the default: $\hat{\beta}_k \pm q\, SE_k$ with SE_k from `vcov(object, type = vcov_type)` and q a normal or t quantile (see `df`).

`method = "boot"` **or** `"BCa"` delegated to `confintROB` (Mason, Cantoni and Ghisletta 2021, 2024), returning its fixed-effect rows; `vcov_type = "sandwich"`, `df` and `cluster` cannot be combined with it.

Usage

```
## S3 method for class 'rlmerMod'
confint(
  object,
  parm = NULL,
  level = 0.95,
  method = c("Wald", "boot", "BCa"),
  vcov_type = c("default", "sandwich"),
  df = c("auto", "none", "satterthwaite", "cluster"),
  boot.type = c("wild", "parametric"),
  nsim = 1000L,
  seed = NULL,
  cluster = NULL,
  ...
)
```

Arguments

`object`	An `rlmerMod` object.
`parm`	Either `NULL` (all fixed-effect coefficients), an integer vector of coefficient indices, or a character vector of coefficient names.
`level`	Coverage level; default 0.95.
`method`	One of `"Wald"` (default; closed form), `"boot"` (bootstrap percentile via **confintROB**), or `"BCa"` (bias-corrected bootstrap via **confintROB**).
`vcov_type`	Covariance for `method = "Wald"`: `"default"` (the model-based covariance) or `"sandwich"` (the cluster sandwich, `vcov_sandwich`).
`df`	Quantile for `method = "Wald"`. `"auto"` (default): `"none"` for `vcov_type = "default"` and `"cluster"` for `vcov_type = "sandwich"`. `"none"`: the normal quantile (with the G1 sandwich for `vcov_type = "sandwich"`). `"cluster"`: the CR2 sandwich with Bell–McCaffrey t quantiles (sandwich only). `"satterthwaite"`: robust Satterthwaite t quantiles as in `summary(object, df = "satterthwaite")` (default covariance and a single grouping factor only; otherwise the default quantile, with a warning).
`boot.type`	Bootstrap kind passed to `confintROB` when `method = "boot"` or `"BCa"`: one of `"wild"` (default, the `confintROB` recommendation) or `"parametric"`.
`nsim`	Bootstrap replicates; default 1000.
`seed`	Optional RNG seed for reproducibility of the bootstrap.
`cluster`	For `vcov_type = "sandwich"`, the clusters, passed to `vcov_sandwich`; needed for models with several grouping factors.
`...`	Additional arguments forwarded to `confintROB` (e.g. `clusterID` for the wild bootstrap).

Details

The Wald limit is adequate for about 20 or more groups; the bootstrap earns its cost mainly with fewer (Mason et al. 2024). `boot.type = "wild"` is robust to misspecification of the response

covariance, "parametric" relies on the fitted central model, and method = "BCa" corrects the percentile interval for bias and skewness. The samples of both bootstraps are also available from simulate.

Sandwich covariance. With few clusters, normal quantiles with the G1 sandwich under-cover, and for between-cluster coefficients the G1 sandwich itself is biased down. For vcov_type = "sandwich" the default (df = "auto", i.e. "cluster") is therefore the bias-reduced CR2 sandwich of vcov_sandwich with a Bell–McCaffrey t quantile per coefficient (the degrees of freedom are in attr(., "df")); see the section "Small-sample references for the cluster sandwich" of vignette("simulationStudies"). When the random effects are not nested within the clusters, the G1 sandwich with t_{J-1} quantiles is used. A coefficient that a single cluster informs on its own (e.g. a factor level present in one cluster) gets an NA interval, with a warning.

Value

A 2-column matrix with one row per selected fixed-effect coefficient, columns "<alpha/2> %" / "<1-alpha/2> %". Attributes "method", "vcov_type" (and, for the bootstrap paths, "boot.type") record the options used; the small-J sandwich reference adds "df" (per selected coefficient) and "correction" ("CR2", or "G1" when CR2 is not defined).

Lifecycle

Stable.

References

Bell RM, McCaffrey DF (2002). Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology* 28(2), 169–181.

Mason F, Cantoni E, Ghisletta P (2021). Parametric and semi-parametric bootstrap-based confidence intervals for robust linear mixed models. *Methodology* 17(4), 271–295.

Mason F, Cantoni E, Ghisletta P (2024). Linear mixed models and latent growth curve models for group comparison studies contaminated by outliers. *Psychological Methods* 30(1), 155–173.

See Also

vcov, confintROB

---

contrast_test *Wald test of linear contrasts of the fixed effects*

Description

Tests $L\beta = rhs$ for a fitted rlmer model. A single contrast gives a t test, several contrasts a joint F test. The reference distributions are those of summary, anova and the **emmeans** support.

Usage

```
contrast_test(
  object,
  L,
  rhs = 0,
  ddf = c("auto", "satterthwaite", "cluster", "none"),
  vcov_type = c("default", "sandwich"),
  cluster = NULL,
  ...
)
```

Arguments

object	an rlmerMod object.
L	the contrasts: a numeric vector (one contrast) or a matrix with one contrast per row. Either one column per fixed-effect coefficient, in the order of fixef(object), or named columns that are matched to the coefficient names; coefficients that are not named get a zero.
rhs	the hypothesised value of $L\beta$: one number or one per contrast.
ddf	the denominator degrees of freedom. "auto" (default): "satterthwaite" for vcov_type = "default" and "cluster" for vcov_type = "sandwich". "satterthwaite": the robust Satterthwaite degrees of freedom of summary(object, df = "satterthwaite") (model-based covariance only). "cluster": the bias-reduced CR2 sandwich with Bell–McCaffrey (one contrast) or approximate-Hotelling (several contrasts) degrees of freedom, as in anova(object, vcov_type = "sandwich") (sandwich only). "none": the normal and chi-square references, reported as infinite degrees of freedom.
vcov_type	"default" (the model-based covariance) or "sandwich" (the cluster sandwich, vcov_sandwich).
cluster	for vcov_type = "sandwich", the clusters, passed to vcov_sandwich; needed for models with several grouping factors.
...	not used.

Details

If the Satterthwaite degrees of freedom are not available for the fit (for example at a variance-component boundary that leaves them unidentified), the normal or chi-square reference is used, with a warning.

Value

A data frame of class "anova". For one contrast: the columns Estimate (of $L\beta - rhs$), Std. Error, df, t value and Pr(>|t|). For several contrasts: one row with NumDF, DenDF, F value and Pr(>F). The contrast matrix is in attr(., "L").

Lifecycle

Stable.

See Also

anova, confint, vcov_sandwich

Examples

```
fm <- rlmer(Reaction ~ Days + (1 | Subject), sleepstudy)
## expected reaction time on day 5
contrast_test(fm, c(1, 5))
## is it 300 ms?
contrast_test(fm, c("(Intercept)" = 1, Days = 5), rhs = 300)
## both coefficients zero, jointly
contrast_test(fm, diag(2))
```

`cooks.distance.rlmerMod` *Cook's-distance equivalent for an rlmerMod fit (per observation or per cluster).*

Description

Joint Mahalanobis influence on the fitted $(\hat{\beta}, \hat{\sigma}, \hat{\theta})$. With `groups = NULL` (default) the unit is the observation: the per-observation influence vectors are stacked into a $(p + 1 + L) \times n$ matrix W, and the result is $\sqrt{w_i^T V^{-1} w_i}$ with $V = (1/n) W W^T$. With `groups` set the unit is the cluster: the per-cluster influence functions $\Xi = -J_{par}^{-1} S$ (S the per-cluster score contributions from `.scoreByCluster`), restricted to the (β, σ, θ) rows, are scored the same way with $V = (1/J)\Xi\Xi^T$. This flags whole-cluster outliers that no single observation reveals – e.g. before relying on the bootstrap variance-component test of anova, which is anti-conservative under group contamination. If V is singular (a variance-component boundary) the Moore-Penrose pseudo-inverse is used.

Usage

```
## S3 method for class 'rlmerMod'
cooks.distance(model, groups = NULL, IF = NULL, ...)
```

Arguments

`model`	An `rlmerMod` object.
`groups`	Cluster grouping for cluster-level Cook's distance: `NULL` (default) for per-observation; `TRUE` for the top-level grouping factor; or a factor / grouping-factor name to aggregate over. Cluster-level requires a single or nested grouping structure (crossed designs error, as for the variance-parameter covariance).
`IF`	Optional pre-computed `implicitIF_full(model)`; computed on demand if `NULL`.
`...`	Currently unused.

Details

The full IF computation is the expensive part; pre-compute it once via `IF = implicitIF_full(fit)` and pass it in if you need `cooks.distance` and `influence` together.

Value

Named numeric vector: one entry per observation (`groups = NULL`) or per cluster level.

See Also

`implicitIF_full`, `influence`, `hatvalues`

`createDatasetsFromList`

Create Dataset List From List of Data Objects

Description

Convert a list of datasets to a dataset list similar to the ones created by `generateAnovaDatasets` and `generateMixedEffectDatasets`.

Usage

```
createDatasetsFromList(
  datasetList,
  formula,
  trueBeta,
  trueSigma,
  trueTheta,
  ...
)
```

Arguments

`datasetList`	list of data objects, usually of type `data.frame`.
`formula`	formula to fit the model using `lmer`.
`trueBeta`	scalar or vector with the true values of the fixed effects coefficients. Can be of length one in which case it will be replicated to the required length if needed.
`trueSigma`	scalar with the true value of the error scale.
`trueTheta`	scalar or vector with the true values for the variance component coefficients, not including sigma. Can be of length one in which case it will be replicated to the required length if needed.
`...`	all additional arguments are added to the returned list.

Details

The returned list can be passed to `processFit` and to any of the `fitDatasets` functions. Splitting and binding of datasets using `splitDatasets` and `bindDatasets` is not supported.

Value

list that can be passed to processFit and to any of the fitDatasets functions. Only generateData is implemented, all the other functions return an error if called.

See Also

generateAnovaDatasets and generateMixedEffectDatasets

Examples

```
data(sleepstudy)
sleepstudy2 <- sleepstudy
sleepstudy2[1, "Reaction"] <- sleepstudy2[1, "Reaction"] + 10
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
datasets <- createDatasetsFromList(list(sleepstudy, sleepstudy2),
                                   formula = Reaction ~ Days + (Days|Subject),
                                   trueBeta = getME(fm1, "beta"),
                                   trueSigma = sigma(fm1),
                                   trueTheta = getME(fm1, "theta"))
fitDatasets_lmer(datasets)
```

---

createRhoFunction *Create Rho-Functions With Custom Tuning Parameter*

Description

Convenience function to create rho-functions with custom tuning parameter.

Usage

```
createRhoFunction(
  tuningParameter,
  which = c("rho.e", "rho.sigma.e", "rho.b.diagonal", "rho.sigma.b.diagonal",
    "rho.b.blockDiagonal", "rho.sigma.b.blockDiagonal"),
  rho.e = smoothPsi,
  rho.sigma.e = psi2propII(rho.e),
  rho.b.diagonal = rho.e,
  rho.sigma.b.diagonal = psi2propII(rho.b.diagonal),
  rho.b.blockDiagonal = rho.e,
  rho.sigma.b.blockDiagonal = rho.b.blockDiagonal,
  ...
)
```

Arguments

tuningParameter	argument passed on to extractTuningParameter. See its documentation for details.
which	string specifying which tuning parameter should be extracted.
rho.e	PsiFunction to be used for rho.e.
rho.sigma.e	PsiFunction to be used for rho.sigma.e.
rho.b.diagonal	PsiFunction to be used for rho.b for models with diagonal random effects covariance matrix.
rho.sigma.b.diagonal	PsiFunction to be used for rho.sigma.b for models with diagonal random effects covariance matrix.
rho.b.blockDiagonal	PsiFunction to be used for rho.b for models with block-diagonal random effects covariance matrix.
rho.sigma.b.blockDiagonal	PsiFunction to be used for rho.sigma.b for models with block-diagonal random effects covariance matrix.
...	passed on to chgDefaults.

Details

'rho.b.diagonal' denotes the tuning parameter to be used for 'rho.b' for models with diagonal random effects covariance matrix. 'rho.b.blockDiagonal' is the tuning parameter to be used in the block diagonal case, respectively.

For arguments rho.sigma.e (and rho.sigma.b.diagonal), the Proposal 2 variant of the function specified for rho.e (and rho.b) is used.

Author(s)

Manuel Koller

Examples

```
createRhoFunction(c(1.345, 2.28, 1.345, 2.28, 5.14, 5.14), "rho.sigma.e")
```

---

design_breakdown *Breakdown ceiling of the fixed-effects design*

Description

Breakdown ceiling implied by the fixed-effects design.

Usage

```
design_breakdown(object, max_subsets = 2e+05)
```

Arguments

`object`	an `rlmerMod` or `merMod` fit, or a numeric model matrix.
`max_subsets`	budget for the exact enumeration of hyperplanes; above it a random search is used and `k` is a lower bound (`exact = FALSE`), so the reported ceilings are upper bounds.

Details

Computes k, the largest number of rows of the fixed-effects model matrix that lie on one hyperplane through the origin, and from it

- `max_bp`: the largest finite-sample breakdown point any regression-equivariant estimator of the fixed effects can have, $\lfloor (n - k + 1)/2 \rfloor / n$ (Mili and Coakley 1996);
- `mm_bp`: the breakdown point that an MM-type refinement is guaranteed to inherit from its start, $\min(b, 1 - b - k/n)$ at $b = 1/2$ (following Yohai 1987), and `b_opt` $= (n - k)/(2n)$, the scale constant at which that guarantee reaches `max_bp`.

In general position $k = p - 1$ and the ceiling is about 1/2. Categorical predictors raise k: with an intercept and a balanced two-level factor, $k = n/2$, the ceiling is about 1/4 and the MM guarantee at $b = 1/2$ is empty. No start, including `rlmer_ransac`, can do better than this ceiling for the fixed effects. `b_opt` is not used by `ransac_lme4`; see the section "Ranking RANSAC candidates" of `vignette("simulationStudies")`.

Value

a list with `n`, `p`, `k`, `k_share` (k/n), `max_bp`, `mm_bp`, `b_opt` and `exact`.

Lifecycle

Experimental. The interface and the results may change in a later version.

References

Mili L, Coakley CW (1996). Robust estimation in structured linear regression. *The Annals of Statistics* 24(6), 2593–2607.

Yohai VJ (1987). High breakdown-point and high efficiency robust estimates for regression. *The Annals of Statistics* 15(2), 642–656.

Examples

```
d <- data.frame(y = rnorm(40), g = gl(2, 20), x = rnorm(40))
design_breakdown(model.matrix(~ g, d))       # dummy: ceiling ~ 1/4
design_breakdown(model.matrix(~ x, d))       # general position
```

`extractTuningParameter` *Extract Tuning Parameters Used In Fitting*

Description

Methods to extract which tuning parameters have been used for fitting models. Use `extractTuningParameter` for custom configurations and `extractPredefinedTuningParameter` for predefined configurations provided in this package.

Usage

```
extractTuningParameter(
  tuningParameter,
  which = c("rho.e", "rho.sigma.e", "rho.b.diagonal", "rho.sigma.b.diagonal",
    "rho.b.blockDiagonal", "rho.sigma.b.blockDiagonal")
)

extractPredefinedTuningParameter(label, which)
```

Arguments

`tuningParameter`	vector of tuning parameters. The vector is expected to be of length 6, containing the tuning parameters for rho.e, rho.sigma.e, rho.b.diagonal, rho.sigma.b.diagonal, rho.b.blockDiagonal and rho.sigma.b.blockDiagonal. 'rho.b.diagonal' denotes the tuning parameter to be used for 'rho.b' for models with diagonal random effects covariance matrix. Names are optional.
`which`	string specifying which tuning parameter should be extracted.
`label`	label or vector of labels in results. Only predefined labels of the form 'fitDatasets_rlmer_...' are supported (for others NA is returned).

Value

scalar tuning parameter

Author(s)

Manuel Koller

Examples

```
extractPredefinedTuningParameter("fitDatasets_rlmer_DAStau", "rho.e")
```

`fitDatasets_lmer` *Fitting Functions*

Description

Methods to fit various mixed effects estimators to all generated datasets.

Usage

```
fitDatasets_lmer(datasets, control, label, postFit, datasetIndices = "all")

fitDatasets_lmer_bobyqa(datasets, postFit, datasetIndices = "all")

fitDatasets_lmer_Nelder_Mead(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer(
  datasets,
  method,
  tuningParameter,
  label,
  postFit,
  datasetIndices = "all",
  ...,
  formula,
  init
)

fitDatasets_rlmer_DAStau(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_lmerNoFit(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DASvar(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_cs(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_ar1(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_noAdj(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_k_0_5(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_k_0_5_noAdj(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_k_2(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_k_2_noAdj(datasets, postFit, datasetIndices = "all")
```

```
fitDatasets_rlmer_DAStau_k_5(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_k_5_noAdj(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_bisq(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_DAStau_sizeOBR(datasets, postFit, datasetIndices = "all")

fitDatasets_rlmer_ransac(
  datasets,
  postFit,
  datasetIndices = "all",
  K = 50L,
  sub_frac = 0.5
)

fitDatasets_rlmer_ransac_bisq(
  datasets,
  postFit,
  datasetIndices = "all",
  K = 50L,
  sub_frac = 0.5
)

fitDatasets_heavyLme(datasets, postFit, datasetIndices = "all")

fitDatasets_lqmm(datasets, postFit, datasetIndices = "all")

fitDatasets_rlme(datasets, postFit, datasetIndices = "all")

fitDatasets_varComprob(
  datasets,
  control,
  label,
  postFit,
  datasetIndices = "all"
)

fitDatasets_varComprob_compositeTau(datasets, postFit, datasetIndices = "all")

fitDatasets_varComprob_compositeTau_OGK(
  datasets,
  postFit,
  datasetIndices = "all"
)

fitDatasets_varComprob_compositeTau_2SGS(
  datasets,
```

```
  postFit,
  datasetIndices = "all"
)

fitDatasets_varComprob_compositeS(datasets, postFit, datasetIndices = "all")

fitDatasets_varComprob_compositeS_OGK(
  datasets,
  postFit,
  datasetIndices = "all"
)

fitDatasets_varComprob_compositeS_2SGS(
  datasets,
  postFit,
  datasetIndices = "all"
)

fitDatasets_varComprob_S(datasets, postFit, datasetIndices = "all")

fitDatasets_varComprob_S_OGK(datasets, postFit, datasetIndices = "all")

fitDatasets_varComprob_S_2SGS(datasets, postFit, datasetIndices = "all")
```

Arguments

datasets	Datasets list to be used to generate datasets.
control	a list (of correct class for the respective fitting function) containing control parameters to be passed through.
label	a string used to identify which fits have been created by which function.
postFit	a function, taking one argument, the resulting fit. This makes it easy to add an additional step after fitting.
datasetIndices	optional vector of dataset indices to fit, useful to try only a few datasets instead of all of them.
method	argument passed on to rlmer.
tuningParameter	argument passed on to extractTuningParameter.
...	argument passed on to createRhoFunction.
formula	optional model formula to fit; defaults to the formula stored in datasets. Used by the structured-covariance wrappers to impose a cs / ar1 covariance structure.
init	optional argument passed on to rlmer.
K	number of random subsamples used by the RANSAC initial estimator (ransac_lme4). Only used by fitDatasets_rlmer_ransac and fitDatasets_rlmer_ransac_bisq.
sub_frac	fraction of the data per RANSAC subsample. Only used by fitDatasets_rlmer_ransac and fitDatasets_rlmer_ransac_bisq.

Details

Existing fitting functions are:

`fitDatasets_lmer`: Fits datasets using `lmer` using its default options.

`fitDatasets_lmer_bobyqa`: Fits datasets using `lmer` using the bobyqa optimizer.

`fitDatasets_lmer_Nelder_Mead`: Fits datasets using `lmer` using the Nelder Mead optimizer.

`fitDatasets_rlmer`: Fits datasets using `rlmer` using a custom configuration. The argument 'tuningParameter' is passed to `extractTuningParameter`, details are documented there.

`fitDatasets_rlmer_DAStau`: Fits datasets using `rlmer` using method DAStau and `smoothPsi` for the rho functions. The tuning parameters are k = 1.345 for `rho.e`. For `rho.sigma.e`, the Proposal 2 variant is used using k = 2.28. The choices for `rho.b` and `rho.sigma.b` depend on whether the model uses a diagonal or a block diagonal matrix for Lambda. In the former case, the same psi functions and tuning parameters are use as for `rho.e` and `rho.sigma.b`. In the block diagonal case, `rho.b` and `rho.sigma.b` both use `smoothPsi` using a tuning parameter k = 5.14 (assuming blocks of dimension 2).

`fitDatasets_rlmer_DAStau_lmerNoFit`: Fits datasets using `rlmer` using the same configuration as `fitDatasets_rlmer_DAStau` except for that it is using `lmerNoFit` as initial estimator.

`fitDatasets_rlmer_DASvar`: Fits datasets using `rlmer` using method DASvar. The same rho functions and tuning parameters are used as for `fitDatasets_rlmer_DAStau`.

`fitDatasets_rlmer_cs`: Fits datasets using `rlmer` with method DASvar, forcing a compound-symmetric (`cs`) random-effects covariance structure (the single random-effects term is rewritten to `cs(...)` regardless of how the data were generated). Requires **lme4** >= 2.0-0. The same rho functions and tuning parameters are used as for `fitDatasets_rlmer_DASvar`.

`fitDatasets_rlmer_ar1`: Fits datasets using `rlmer` with method DASvar, forcing an autoregressive (`ar1`) random-effects covariance structure. Otherwise identical to `fitDatasets_rlmer_cs`. Requires **lme4** >= 2.0-0.

`fitDatasets_rlmer_DAStau_noAdj`: Fits datasets using `rlmer` using method DAStau. The same rho functions and tuning parameters are used as for `fitDatasets_rlmer_DAStau`, except for `rho.sigma.e` (and `rho.sigma.b` in the diagonal case) for which the Proposal 2 variant of `smoothPsi` using k = 1.345 is used.

`fitDatasets_rlmer_DAStau_k_0_5`: Fits datasets using `rlmer` using method DAStau. Use `smoothPsi` psi-function with tuning parameter `k = 0.5` for `rho.e` and `k = 1.47` for `rho.sigma.e`, the latter adjusted to reach the same asymptotic efficiency. In the diagonal case, the same are used for `rho.b` and `rho.sigma.b` as well. In the block-diagonal case, the tuning parameter `k = 2.17` is used for `rho.b` and `rho.sigma.b`. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

`fitDatasets_rlmer_DAStau_k_0_5_noAdj`: Fits datasets using `rlmer` using method DAStau. Use `smoothPsi` psi-function with tuning parameter `k = 0.5` for `rho.e` and `rho.sigma.e`. In the diagonal case, the same are used for `rho.b` and `rho.sigma.b` as well. In the block-diagonal case, the tuning parameter `k = 2.17` is used for `rho.b` and `rho.sigma.b`. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

`fitDatasets_rlmer_DAStau_k_2`: Fits datasets using `rlmer` using method DAStau. Use `smoothPsi` psi-function with tuning parameter `k = 2` for `rho.e` and `k = 2.9` `rho.sigma.e`, the latter adjusted to reach the same asymptotic efficiency. In the diagonal case, the same are used for `rho.b` and `rho.sigma.b` as well. In the block-diagonal case, the tuning parameter `k = 8.44` is used for `rho.b`

and rho.sigma.b. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

fitDatasets_rlmer_DAStau_k_2_noAdj: Fits datasets using rlmer using method DAStau. Use smoothPsi psi-function with tuning parameter k = 2 for rho.e and rho.sigma.e. In the diagonal case, the same are used for rho.b and rho.sigma.b as well. In the block-diagonal case, the tuning parameter k = 8.44 is used for rho.b and rho.sigma.b. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

fitDatasets_rlmer_DAStau_k_5: Fits datasets using rlmer using method DAStau. Use smoothPsi psi-function with tuning parameter k = 5 for rho.e and k = 5.03 rho.sigma.e, the latter adjusted to reach the same asymptotic efficiency. In the diagonal case, the same are used for rho.b and rho.sigma.b as well. In the block-diagonal case, the tuning parameter k = 34.21 is used for rho.b and rho.sigma.b. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

fitDatasets_rlmer_DAStau_k_5_noAdj: Fits datasets using rlmer using method DAStau. Use smoothPsi psi-function with tuning parameter k = 5 for rho.e and rho.sigma.e. In the diagonal case, the same are used for rho.b and rho.sigma.b as well. In the block-diagonal case, the tuning parameter k = 34.21 is used for rho.b and rho.sigma.b. The tuning parameter is chosen to reach about the same asymptotic efficiency for theta as for the fixed effects.

fitDatasets_rlmer_DAStau_bisq: Fits datasets using rlmer with method DAStau, replacing rho.e with the redescending bisquare psi (bisquarePsi, c = 4.685). The other rho-functions use the same smoothed Huber psi and tuning parameters as fitDatasets_rlmer_DAStau.

fitDatasets_rlmer_DAStau_sizeOBR: Fits datasets using rlmer with method DAStau and size_obr = TRUE, which replaces the finite-difference size weight in the block-diagonal V_b score equation with the Hampel-OBR form. The same rho-functions and tuning parameters are used as for fitDatasets_rlmer_DAStau. For diagonal V_b the size_obr argument is silently ignored.

fitDatasets_rlmer_ransac: Fits datasets using rlmer with method DAStau and a RANSAC-derived initial estimator (ransac_lme4). The number of random subsamples is K = 50 with subsample fraction sub_frac = 0.5. Same rho-functions and tuning parameters as fitDatasets_rlmer_DAStau.

fitDatasets_rlmer_ransac_bisq: Combines fitDatasets_rlmer_ransac (RANSAC init) with fitDatasets_rlmer_DAStau_bisq (bisquarePsi for rho.e). Designed to give redescending psi a starting value safely away from phony local minima.

fitDatasets_heavyLme: Fits datasets using heavyLme from package heavy. Additional required arguments are: lmeFormula, heavyLmeRandom and heavyLmeGroups. They are passed to the formula, random and groups arguments of heavyLme.

fitDatasets_lqmm: Fits datasets using lqmm from package lqmm. Additional required arguments are: lmeFormula, lqmmRandom, lqmmGroup and lqmmCovariance. They are passed to the formula, random, groups and covariance arguments of lqmm. lqmmCovariance is optional, if omitted pdDiag is used.

fitDatasets_rlme: Fits datasets using rlme from package rlme.

fitDatasets_varComprob: Prototype method to fit datasets using varComprob from package robustvarComp. Additional required items in datasets are: lmeFormula, groups, varcov and lower. They are passed to the fixed, groups, varcov and lower arguments of varComprob. The running of this method produces many warnings of the form "passing a char vector to .Fortran is not portable" which are suppressed.

`fitDatasets_varComprob_compositeTau`: Fits datasets with the composite Tau method using `varComprob` from package `robustvarComp`. See `fitDatasets_varComprob` for additional details.

`fitDatasets_varComprob_compositeTau_OGK`: Similar to `fitDatasets_varComprob_compositeTau` but using `covOGK` as initial covariance matrix estimator.

`fitDatasets_varComprob_compositeTau_2SGS`: Similar to `fitDatasets_varComprob_compositeTau` but using `2SGS` as initial covariance matrix estimator.

`fitDatasets_varComprob_compositeS`: Similar to `fitDatasets_varComprob_compositeTau` but using method composite S.

`fitDatasets_varComprob_compositeS_OGK`: Similar to `fitDatasets_varComprob_compositeS` but using `covOGK` as initial covariance matrix estimator.

`fitDatasets_varComprob_compositeS_2SGS`: Similar to `fitDatasets_varComprob_compositeS` but using `2SGS` as initial covariance matrix estimator.

`fitDatasets_varComprob_S`: Similar to `fitDatasets_varComprob_compositeTau` but using method S and the Rocke psi-function.

`fitDatasets_varComprob_S_OGK`: Similar to `fitDatasets_varComprob_S` but using `covOGK` as initial covariance matrix estimator.

`fitDatasets_varComprob_S_2SGS`: Similar to `fitDatasets_varComprob_S` but using `2SGS` as initial covariance matrix estimator.

Value

list of fitted models. See also `lapplyDatasets` which is called internally.

Author(s)

Manuel Koller

Examples

```
set.seed(1)
oneWay <- generateAnovaDatasets(1, 1, 10, 4,
                                lmeFormula = y ~ 1,
                                heavyLmeRandom = ~ 1,
                                heavyLmeGroups = ~ Var2,
                                lqmmRandom = ~ 1,
                                lqmmGroup = "Var2",
                                groups = cbind(rep(1:4, each = 10), rep(1:10, 4)),
                                varcov = matrix(1, 4, 4),
                                lower = 0)
fitDatasets_lmer(oneWay)
## call rlmer with custom arguments
fitDatasets_rlmer_custom <- function(datasets) {
  return(fitDatasets_rlmer(datasets,
                           method = "DASvar",
                           tuningParameter = c(1.345, 2.28, 1.345, 2.28, 5.14, 5.14),
                           label = "fitDatasets_rlmer_custom"))
}
fitDatasets_rlmer_custom(oneWay)
```

`fragility` *Fragility of variance parameters under contamination*

Description

First-order contamination fraction that would drive each variance parameter of a robust LMM to its bound.

Usage

```
fragility(object, ...)

## S3 method for class 'rlmerMod'
fragility(object, ...)
```

Arguments

`object`	an `rlmerMod` object.
`...`	ignored.

Details

For every variance parameter θ_k with a lower bound of zero (the diagonal entries of the relative Cholesky factor), `fragility` reports

$$\epsilon_k^* = \theta_k / \max_j(-\mathrm{IF}_k(j)),$$

where $\mathrm{IF}_k(j)$ is the influence of cluster j on θ_k at the fitted model. Replacing a fraction ϵ of the clusters by copies of the most influential one moves θ_k by $\epsilon\,\mathrm{IF}_k + O(\epsilon^2)$, so it reaches its bound at ϵ_k^* to first order: a zero variance for the first component of a random-effects block, a singular covariance matrix (for a 2×2 block a correlation of ± 1) for the later ones.

Only the observed clusters are searched, so ϵ^* is an upper bound on the worst case: a small value flags a fragile parameter, a large one does not certify robustness, and values above 1/2 are only indicative. `clusters`, $\epsilon^* J/(1-\epsilon^*)$, is the number of copies of the driving cluster that, added to the J observed clusters, make up the fraction ϵ^*. Crossed designs are not supported.

Value

a data frame with one row per bounded variance parameter: `parameter`, `bound`, `theta` (estimate, relative to the residual scale), `eps` (ϵ^*), `clusters` ($\epsilon^* J/(1-\epsilon^*)$) and `driver` (the most influential cluster).

Lifecycle

Experimental. The interface and the results may change in a later version.

Examples

```
fit <- rlmer(Reaction ~ Days + (Days | Subject), sleepstudy,
             method = "DASvar")
fragility(fit)
```

`generateAnovaDatasets` *Generate ANOVA type datasets*

Description

Generate balanced datasets with multiple factors. All combinations of all factor variables are generated, i.e., a fully crossed dataset will be generated. `numberOfReplicates` specifies the number of replications per unique combination.

Usage

```
generateAnovaDatasets(
  numberOfDatasetsToGenerate,
  numberOfLevelsInFixedFactor,
  numberOfSubjects,
  numberOfReplicates,
  errorGenerator = rnorm,
  randomEffectGenerator = rnorm,
  trueBeta = 1,
  trueSigma = 4,
  trueTheta = 1,
  ...,
  arrange = FALSE
)
```

Arguments

`numberOfDatasetsToGenerate`	number of datasets to generate.
`numberOfLevelsInFixedFactor`	scalar or vector with the number of levels per fixed factor or grouping variable.
`numberOfSubjects`	scalar or vector with the number of levels per variance component.
`numberOfReplicates`	number of replicates per unique combination of fixed factor and variance component.
`errorGenerator`	random number generator used for the errors.
`randomEffectGenerator`	random number generator used for the spherical random effects.
`trueBeta`	scalar or vector with the true values of the fixed effects coefficients. Can be of length one in which case it will be replicated to the required length if needed.

trueSigma	scalar with the true value of the error scale.
trueTheta	scalar of vector with the true values for the variance component coefficients, not including sigma. Can be of length one in which case it will be replicated to the required length if needed.
...	all additional arguments are added to the returned list.
arrange	If TRUE, the observations in the dataset are arranged such that the call to arrange in varComprob does not break the observation- group relationship. This requires package dplyr to be installed.

Details

numberOfLevelsInFixedFactor can either be a scalar or a vector with the number of levels for each fixed effects group. If numberOfLevelsInFixedFactor is a scalar, the value of 1 is allowed. This can be used to generate a dataset with an intercept only. If numberOfLevelsInFixedFactor is a vector with more than one entry, then all the values need to be larger than one.

numberOfSubjects can also be a scalar of a vector with the number of levels for each variance component. Each group needs to have more than one level. The vector is sorted descending before the names are assigned. This ensures that, when running lmer, the order of the random effects does not change. lmer also sorts the random effects by descending number of levels.

In order to save memory, only the generated random effects and the errors are stored. The dataset is only created on demand when the method generateData in the returned list is evaluated.

The random variables are generated in a way that one can simulate more datasets easily. When starting from the same seed, the first generated datasets will be the same as for the a previous call of generateAnovaDatasets with a smaller number of datasets to generate, see examples.

Value

list with generators and the original arguments

generateData:	function to generate data taking one argument, the dataset index.
createXMatrix:	function to generate X matrix taking one argument, the result of generateData.
createZMatrix:	function to generate Z matrix taking one argument, the result of generateData.
createLambdaMatrix:	function to generate Lambda matrix taking one argument, the result of generateData.
randomEffects:	function to return the generated random effects taking one argument, the dataset index.
sphericalRandomeffects:	function to return the generated spherical random effects taking one argument, the dataset index.
errors:	function to return the generated errors taking one argument, the dataset index.
allRandomEffects:	function without arguments that returns the matrix of all generated random effects.
allErrors:	function without arguments that returns the matrix of all generated errors.
numberOfDatasets:	numberOfDatasetsToGenerate as supplied

`numberOfLevelsInFixedFactor`:
: `numberOfLevelsInFixedFactor` as supplied

`numberOfSubjects`:
: `numberOfSubjects` sorted.

`numberOfReplicates`:
: `numberOfReplicates` as supplied

`numberOfRows`:
: number of rows in the generated dataset

`trueBeta`:
: true values used for beta

`trueSigma`:
: true value used for sigma

`trueTheta`:
: true values used for theta

`formula`:
: formula to fit the model using `lmer`

`...`:
: additional arguments passed via `...`

Author(s)

Manuel Koller

See Also

`generateMixedEffectDatasets` and `createDatasetsFromList`

Examples

```
oneWay <- generateAnovaDatasets(2, 1, 5, 4)
head(oneWay$generateData(1))
head(oneWay$generateData(2))
oneWay$formula
head(oneWay$randomEffects(1))
head(oneWay$sphericalRandomEffects(1))
head(oneWay$errors(1))

twoWayFixedRandom <- generateAnovaDatasets(2, 3, 5, 4)
head(twoWayFixedRandom$generateData(1))
twoWayFixedRandom$formula

twoWayRandom <- generateAnovaDatasets(2, 1, c(3, 5), 4)
head(twoWayRandom$generateData(1))
twoWayRandom$formula

large <- generateAnovaDatasets(2, c(10, 15), c(20, 30), 5)
head(large$generateData(1))
large$formula

## illustration how to generate more datasets
set.seed(1)
datasets1 <- generateAnovaDatasets(2, 1, 5, 4)
set.seed(1)
datasets2 <- generateAnovaDatasets(3, 1, 5, 4)
stopifnot(all.equal(datasets1$generateData(1), datasets2$generateData(1)),
          all.equal(datasets1$generateData(2), datasets2$generateData(2)))
```

`generateLongitudinalDatasets` *Generate Longitudinal Datasets*

Description

Generate balanced longitudinal datasets with random intercepts and slopes. Subjects are observed at multiple time points with optional treatment groups. Treatment and its interaction with time are coded as contrasts relative to the first level.

Usage

```
generateLongitudinalDatasets(
  numberOfDatasetsToGenerate,
  numberOfSubjects,
  numberOfTimepoints,
  numberOfTreatmentLevels = 1L,
  timeRange = c(0, 1),
  errorGenerator = rnorm,
  randomEffectGenerator = rnorm,
  trueBeta = 0,
  trueSigma = 1,
  trueTheta = c(1, 0, 1),
  contamFun = NULL,
  ...
)
```

Arguments

`numberOfDatasetsToGenerate`	number of datasets to generate.
`numberOfSubjects`	number of subjects per dataset.
`numberOfTimepoints`	number of observation time points per subject.
`numberOfTreatmentLevels`	number of treatment levels. Default: 1 (no treatment effect, intercept and time only).
`timeRange`	numeric vector of length 2, range of time values (min, max). Default: `c(0, 1)`.
`errorGenerator`	random number generator used for the errors. Called as `errorGenerator(n) * trueSigma`.
`randomEffectGenerator`	random number generator used for the spherical random effects. Called as `randomEffectGenerator(n) * trueSigma`.

`trueBeta`	scalar or vector with the true values of the fixed effects coefficients. Can be of length one in which case it will be replicated to the required length. The order is: intercept, time, treatment contrasts (if any), treatment-by-time interactions (if any).
`trueSigma`	scalar with the true value of the error scale.
`trueTheta`	numeric vector of length 3 with the true values for the Cholesky factor of the random effects covariance matrix (lme4 convention). Default: `c(1, 0, 1)` (independent random intercepts and slopes).
`contamFun`	optional contamination function. If provided, it receives the full dataset (a data frame with columns id, time, treatment, y) and an info list, and must return the (possibly modified) data frame. This allows arbitrary contamination including changing group assignments. See Details for the contents of the info list.
`...`	all additional arguments are added to the returned list.

Details

The generated data follows the model:

$$y_{ij} = \beta_0 + \beta_1 \cdot \text{time}_{ij} + \sum_{k=1}^{K-1} \beta_{1+k} \cdot \text{treatment}_{k,i} + \sum_{k=1}^{K-1} \beta_{K+k} \cdot \text{treatment}_{k,i} \cdot \text{time}_{ij} + b_{0i} + b_{1i} \cdot \text{time}_{ij} + \epsilon_{ij}$$

where K is the number of treatment groups, $b_i = (b_{0i}, b_{1i})^T \sim N(0, \sigma^2 \Lambda \Lambda^T)$ with Λ being the lower-triangular Cholesky factor reconstructed from the theta vector.

The theta parameterization follows lme4 conventions:

- For a 2x2 random-effects covariance structure (intercept and slope), theta has 3 elements: $\theta = (\lambda_{11}, \lambda_{21}, \lambda_{22})$
- The Cholesky factor is: $\Lambda = \begin{pmatrix} \lambda_{11} & 0 \\ \lambda_{21} & \lambda_{22} \end{pmatrix}$

In order to save memory, only the generated random effects and the errors are stored. The dataset is only created on demand when the method `generateData` in the returned list is evaluated.

The random variables are generated in a way that one can simulate more datasets easily. When starting from the same seed, the first generated datasets will be the same as for a previous call of `generateLongitudinalDatasets` with a smaller number of datasets to generate, see examples.

Treatment Assignment: Subjects are assigned to treatment groups in a balanced, deterministic manner. Subject i is assigned to treatment `(i - 1) mod numberOfTreatmentLevels + 1`.

Contamination Function: If `contamFun` is provided, it is called as `contamFun(data, info)` after the response `y` is computed. The `info` list contains:

- `datasetIndex`: the dataset index
- `randomEffects`: the random effects vector for this dataset
- `errors`: the error vector for this dataset
- `trueBeta`: as passed to `generateLongitudinalDatasets`
- `trueSigma`: as passed to `generateLongitudinalDatasets`

- `trueTheta`: as passed to `generateLongitudinalDatasets`

The function must return a data frame with the same structure (columns id, time, treatment, y). This allows arbitrary modifications including:

- Modifying the response `y` (e.g., adding outliers)
- Changing group assignments (e.g., moving subjects between treatments)
- Modifying time values
- Any combination of the above

Value

list with generators and the original arguments

`generateData`: function to generate data taking one argument, the dataset index.

`createXMatrix`: function to generate X matrix taking one argument, the result of `generateData`.

`createZMatrix`: function to generate Z matrix taking one argument, the result of `generateData`.

`createLambdaMatrix`: function to generate Lambda matrix taking one argument, the result of `generateData`.

`randomEffects`: function to return the generated random effects taking one argument, the dataset index.

`sphericalRandomEffects`: function to return the generated spherical random effects taking one argument, the dataset index.

`errors`: function to return the generated errors taking one argument, the dataset index.

`allRandomEffects`: function without arguments that returns the matrix of all generated random effects.

`allErrors`: function without arguments that returns the matrix of all generated errors.

`numberOfDatasets`: `numberOfDatasetsToGenerate` as supplied

`numberOfSubjects`: `numberOfSubjects` as supplied

`numberOfTimepoints`: `numberOfTimepoints` as supplied

`numberOfTreatmentLevels`: `numberOfTreatmentLevels` as supplied

`numberOfRows`: number of rows in the generated dataset

`trueBeta`: true values used for beta

`trueSigma`: true value used for sigma

`trueTheta`: true values used for theta

`formula`: formula to fit the model using `lmer`

`...`: additional arguments passed via `...`

Author(s)

Manuel Koller

See Also

generateAnovaDatasets, generateMixedEffectDatasets

Examples

```
oneGroup <- generateLongitudinalDatasets(2, 10, 5)
head(oneGroup$generateData(1))
head(oneGroup$generateData(2))
oneGroup$formula

twoGroups <- generateLongitudinalDatasets(2, 20, 5, numberOfTreatmentLevels = 2)
head(twoGroups$generateData(1))
twoGroups$formula

## illustration how to generate more datasets
set.seed(1)
datasets1 <- generateLongitudinalDatasets(2, 10, 5)
set.seed(1)
datasets2 <- generateLongitudinalDatasets(3, 10, 5)
stopifnot(all.equal(datasets1$generateData(1), datasets2$generateData(1)),
          all.equal(datasets1$generateData(2), datasets2$generateData(2)))

## contamination example: add outliers to 10% of observations
set.seed(42)
contam <- generateLongitudinalDatasets(
  numberOfDatasetsToGenerate = 5,
  numberOfSubjects = 20,
  numberOfTimepoints = 5,
  contamFun = function(data, info) {
    n <- nrow(data)
    idx <- sample(n, size = ceiling(0.1 * n))
    data$y[idx] <- data$y[idx] + 10
    data
  }
)
head(contam$generateData(1))

## contamination example: reassign some subjects to different treatment
set.seed(42)
contamGroup <- generateLongitudinalDatasets(
  numberOfDatasetsToGenerate = 5,
  numberOfSubjects = 20,
  numberOfTimepoints = 5,
  numberOfTreatmentLevels = 2,
  contamFun = function(data, info) {
    ## move first subject from T1 to T2
    data$treatment[data$id == 1] <- "T2"
    data
```

```
  }
)
head(contamGroup$generateData(1), 10)

## medsim: simulation inspired by the medication dataset from confintROB
## Two subjects from treatment are mislabeled as control, and responses
## are truncated at a measurement floor of 100.
contaminateMedsim <- function(data, info) {
  data$y <- pmax(data$y, 100)  # measurement floor
  data$treatment[data$id %in% c("2", "4")] <- "T1"
  data
}
set.seed(2000)
medsim <- generateLongitudinalDatasets(
  numberOfDatasetsToGenerate = 100,
  numberOfSubjects = 60,
  numberOfTimepoints = 7,
  numberOfTreatmentLevels = 2,
  timeRange = c(0, 18),
  trueBeta = c(240, -3.11, -2.42, 4.00),
  trueSigma = sqrt(1229.93),
  trueTheta = c(1.310266, -0.07547461, 0.2147735),
  contamFun = contaminateMedsim
)
head(medsim$generateData(1))
```

`generateMixedEffectDatasets` *Generate Mixed Effects Datasets*

Description

Generates mixed effects datasets using parametric bootstrap.

Usage

```
generateMixedEffectDatasets(
  numberOfDatasetsToGenerate,
  preparedDataset,
  errorGenerator = rnorm,
  randomEffectGenerator = rnorm
)
```

Arguments

`numberOfDatasetsToGenerate`
: number of datasets to generate.

preparedDataset
: dataset as prepared by prepareMixedEffectDataset.

errorGenerator
: random number generator used for the errors.

randomEffectGenerator
: random number generator used for the spherical random effects.

Value

list with generators and the contents of the prepared dataset. See prepareMixedEffectDataset and generateAnovaDatasets for a description of the contents.

Author(s)

Manuel Koller

See Also

generateAnovaDatasets, prepareMixedEffectDataset and createDatasetsFromList

Examples

```
preparedDataset <- prepareMixedEffectDataset(Reaction ~ Days + (Days|Subject), sleepstudy)
datasets <- generateMixedEffectDatasets(2, preparedDataset)
head(datasets$generateData(1))
head(datasets$generateData(2))
datasets$formula
head(datasets$randomEffects(1))
head(datasets$sphericalRandomEffects(1))
head(datasets$errors(1))
```

---

generateRepeatedMeasuresDatasets

Generate Repeated-Measures Datasets With Structured Covariance

---

Description

Generate balanced repeated-measures datasets with a structured random-effect covariance.

Usage

```
generateRepeatedMeasuresDatasets(
  numberOfDatasetsToGenerate,
  numberOfSubjects,
  numberOfVisits,
  numberOfReplicates = 1L,
  structure = c("unstructured", "cs", "ar1", "diag"),
  marginalSd = 1,
  correlation = 0,
```

```
  trueBeta = 0,
  trueSigma = 1,
  errorGenerator = rnorm,
  randomEffectGenerator = rnorm
)
```

Arguments

`numberOfDatasetsToGenerate`	number of datasets to generate.
`numberOfSubjects`	number of subjects (grouping levels).
`numberOfVisits`	number of levels of the within-subject factor (the dimension of the random effect).
`numberOfReplicates`	number of replicates per subject-by-visit cell.
`structure`	random-effect covariance structure, one of `"unstructured"`, `"cs"`, `"ar1"`, `"diag"`.
`marginalSd`	marginal standard deviation(s) of the random effects, recycled to length `numberOfVisits`.
`correlation`	the structure's single correlation parameter (the common correlation for `"cs"`, the lag-1 correlation for `"ar1"`); ignored for `"diag"`.
`trueBeta`	the true intercept (the only fixed effect).
`trueSigma`	the true residual standard deviation.
`errorGenerator, randomEffectGenerator`	functions used to draw the errors and spherical random effects, see `generateMixedEffectDatasets`.

Details

Each subject is observed once per level of a within-subject factor `visit` (with `numberOfVisits` levels), optionally replicated `numberOfReplicates` times, giving a `numberOfVisits`-dimensional random effect per subject through the term `(0 + visit | subject)`. The random-effect covariance follows the requested `structure`:

`"unstructured"` an arbitrary covariance (the `(0 + visit | subject)` term).

`"cs"` compound symmetry: a common `correlation` between all visits (`cs(0 + visit | subject)`).

`"ar1"` autoregressive: $\mathrm{Cor}(i, j) =$ `correlation`$^{|i-j|}$ (`ar1(0 + visit | subject)`).

`"diag"` uncorrelated visits (`diag(0 + visit | subject)`).

The data are simulated from the chosen covariance using the same machinery as `generateMixedEffectDatasets`; the returned object has the identical interface (including `generateData`, `formula` and the `fitDatasets_*` compatibility) and stores the structured covariance in `trueTheta`. Structured covariances require **lme4** >= 2.0-0.

Value

A list with the same structure as the return value of `generateMixedEffectDatasets`.

See Also

generateMixedEffectDatasets, generateLongitudinalDatasets

Examples

```
if (packageVersion("lme4") >= "2.0.0") {
  datasets <- generateRepeatedMeasuresDatasets(
      1, numberOfSubjects = 30, numberOfVisits = 3, numberOfReplicates = 4,
      structure = "cs", marginalSd = c(2, 1.5, 1.2), correlation = 0.5)
  fit <- rlmer(datasets$formula, datasets$generateData(1), method = "DASvar")
  VarCorr(fit)
}
```

---

generateSensitivityCurveDatasets

Generate Datasets To Create Sensitivity Curves

Description

This method creates a list of datasets that can be used to create sensitivity curves. The response of the dataset is modified according to the supplied arguments.

Usage

```
generateSensitivityCurveDatasets(
  data,
  observationsToChange,
  shifts,
  scales,
  center,
  formula,
  ...
)
```

Arguments

data	dataset to be modified.
observationsToChange	index or logical vector indicating which observations should be modified.
shifts	vector of shifts that should be applied one by one to each of the modified observations.
scales	vector scales that should be used to scale the observations around their original center.
center	optional scalar used to define the center from which the observations are scaled from. If missing, the mean of all the changed observations is used.
formula	formula to fit the model using lmer.
...	all additional arguments are added to the returned list.

Details

Either `shifts` or `scales` need to be provided. Both are also possible.

The argument `shifts` contains all the values that shall be added to each of the observations that should be changed. One value per generated dataset.

The argument `scales` contains all the values that shall be used to move observations away from their center. If `scales` is provided, then `observationsToChange` needs to select more than one observation.

The returned list can be passed to `processFit` and to any of the `fitDatasets` functions. Splitting and binding of datasets using `splitDatasets` and `bindDatasets` is not supported.

Value

list that can be passed to `processFit` and to any of the `fitDatasets` functions. Only `generateData` is implemented, all the other functions return an error if called.

See Also

`generateAnovaDatasets`

Examples

```
oneWay <- generateAnovaDatasets(1, 1, 10, 5)
datasets <-
    generateSensitivityCurveDatasets(oneWay$generateData(1),
                                     observationsToChange = 1:5,
                                     shifts = -10:10,
                                     formula = oneWay$formula)
datasets$generateData(1)
```

---

getME *Extract or Get Generalize Components from a Fitted Mixed Effects Model*

Description

Extract (or "get") "components" – in a generalized sense – from a fitted mixed-effects model, i.e. from an object of class `rlmerMod` or `merMod`.

Usage

```
## S3 method for class 'rlmerMod'
getME(
  object,
 name = c("X", "Z", "Zt", "Ztlist", "mmList", "y", "mu", "u", "b.s", "b", "Gp", "Tp",
   "Lambda", "Lambdat", "Tlist", "A", "U_b", "Lind", "sigma", "flist", "fixef", "beta",
   "theta", "ST", "is_REML", "n_rtrms", "n_rfacs", "N", "n", "p", "q", "p_i", "l_i",
   "q_i", "k", "m_i", "m", "cnms", "devcomp", "offset", "lower", "rho_e", "rho_b",
```

```
    "rho_sigma_e", "rho_sigma_b", "M", "w_e", "w_b", "w_b_vector", "w_sigma_e",
    "w_sigma_b", "w_sigma_b_vector", "design.weights"),
  ...
)

theta(object)
```

Arguments

object	a fitted mixed-effects model of class rlmerMod, i.e. typically the result of rlmer().
name	a character string specifying the name of the "component". Possible values are:

"X": fixed-effects model matrix

"Z": random-effects model matrix

"Zt": transpose of random-effects model matrix

"Ztlist": list of components of the transpose of the random-effects model matrix, separated by individual variance component

"mmList": list of raw model matrices associated with random effects terms

"y": response vector

"mu": conditional mean of the response

"u": conditional mode of the "spherical" random effects variable

"b.s": synonym for "u"

"b": conditional mode of the random effects variable

"Gp": groups pointer vector. A pointer to the beginning of each group of random effects corresponding to the random-effects terms.

"Tp": theta pointer vector. A pointer to the beginning of the theta sub-vectors corresponding to the random-effects terms, beginning with 0 and including a final element giving the total number of random effects

"Lambda": relative covariance factor of the random effects.

"U_b": synonym for "Lambda"

"Lambdat": transpose of the relative covariance factor of the random effects.

"Lind": index vector for inserting elements of θ into the nonzeros of Λ

"A": Scaled sparse model matrix (class dgCMatrix) for the unit, orthogonal random effects, U, equal to `getME(.,"Zt") %*% getME(.,"Lambdat")`

"sigma": residual standard error

"flist": a list of the grouping variables (factors) involved in the random effect terms

"fixef": fixed-effects parameter estimates

"beta": fixed-effects parameter estimates (identical to the result of fixef, but without names)

"theta": random-effects parameter estimates: these are parameterized as the relative Cholesky factors of each random effect term

"ST": A list of S and T factors in the TSST' Cholesky factorization of the relative variance matrices of the random effects associated with each random-effects term. The unit lower triangular matrix, T, and the diagonal matrix, S, for each term are stored as a single matrix with diagonal elements from S and off-diagonal elements from T.

"is_REML": returns TRUE for rlmerMod-objects (for compatibility with lme4)

"n_rtrms": number of random-effects terms

"n_rfacs": number of distinct random-effects grouping factors

"N": number of rows of X

"n": length of the response vector, y

"p": number of columns of the fixed effects model matrix, X

"q": number of columns of the random effects model matrix, Z

"p_i": numbers of columns of the raw model matrices, mmList

"l_i": numbers of levels of the grouping factors

"q_i": numbers of columns of the term-wise model matrices, ZtList

"k": number of random effects terms

"m_i": numbers of covariance parameters in each term

"m": total number of covariance parameters, i.e., the same as dim@nth below.

"cnms": the "component names", a 'list'.

"devcomp": a list consisting of a named numeric vector, cmp, and a named integer vector, dims, describing the fitted model. The elements of cmp are:

ldL2 always NA, for consistency with lme4 output
ldRX2 always NA, for consistency with lme4 output
wrss always NA, for consistency with lme4 output
ussq always NA, for consistency with lme4 output
pwrss always NA, for consistency with lme4 output
drsum always NA, for consistency with lme4 output
REML always NA, for consistency with lme4 output
dev always NA, for consistency with lme4 output
sigmaML always NA, for consistency with lme4 output
sigmaREML REML estimate of residual standard deviation

The elements of dims are:

N number of rows of X
n length of y
p number of columns of X
nmp n-p
nth length of theta
q number of columns of Z
nAGQ see glmer
compDev see glmerControl
useSc TRUE if model has a scale parameter

reTrms number of random effects terms

REML `0` indicates the model was fitted by maximum likelihood, any other positive integer indicates fitting by restricted maximum likelihood

GLMM `TRUE` if a GLMM

NLMM `TRUE` if an NLMM

`"offset"`: model offset

`"lower"`: lower bounds on random-effects model parameters (i.e, "theta" parameters). In order to constrain random effects covariance matrices to be semi-positive-definite, this vector is equal to 0 for elements of the `theta` vector corresponding to diagonal elements of the Cholesky factor, `-Inf` otherwise. (`getME(.,"lower")==0` can be used as a test to identify diagonal elements, as in `isSingular`.)

`"rho_e"`: rho function used for the residuals

`"rho_b"`: list of rho functions used for the random effects

`"rho_sigma_e"`: rho function used for the residuals when estimating sigma

`"rho_sigma_b"`: list of rho functions used for the random effects when estimating the covariance parameters

`"M"`: list of matrices, blocks of the Henderson's equations and the matrices used for computing the linear approximations of the estimates of beta and spherical random effects.

`"w_e"`: robustness weights associated with the observations

`"design.weights"`: Mallows design weights η_i (see argument `design.weights` of `rlmer`); a vector of ones unless design weights were specified

`"w_b"`: robustness weights associated with the spherical random effects, returned in the same format as `ranef()`

`"w_b_vector"`: robustness weights associated with the spherical random effects, returned as one long vector, in the same order as the spherical random effects (`"b_s"`)

`"w_sigma_e"`: robustness weights associated with the observations when estimating sigma

`"w_sigma_b"`: robustness weights associated with the spherical random effects when estimating the covariance parameters, returned in the same format as `ranef()`

`"w_sigma_b_vector"`: robustness weights associated with the spherical random effects when estimating the covariance parameters, returned as one long vector, in the same order as the spherical random effects (`"b_s"`)

`"ALL"`: get all of the above as a `list`.

`...` potentially further arguments; not here.

Details

The function `theta` is short for `getME(, "theta")`.

The goal is to provide "everything a user may want" from a fitted `rlmerMod` object *as far* as it is not available by methods, such as `fixef`, `ranef`, `vcov`, etc.

Value

Unspecified, as very much depending on the name.

See Also

getCall(); more standard methods for rlmerMod objects, such as ranef, fixef, vcov, etc.: see methods(class="rlmerMod")

Examples

```
## shows many methods you should consider *before* using getME():
methods(class = "rlmerMod")

## doFit = FALSE to speed up example
(fm1 <- rlmer(Reaction ~ Days + (Days|Subject), sleepstudy,
              method="DASvar", doFit=FALSE))
Z <- getME(fm1, "Z")
stopifnot(is(Z, "CsparseMatrix"),
          c(180,36) == dim(Z),
   all.equal(fixef(fm1), b1 <- getME(fm1, "beta"),
      check.attributes=FALSE, tolerance = 0))

## A way to get *all* getME()s :
## internal consistency check ensuring that all work:
parts <- getME(fm1, "ALL")
str(parts, max=2)
stopifnot(identical(Z,  parts $ Z),
          identical(b1, parts $ beta))
stopifnot(all.equal(theta(fm1), getME(fm1, "theta")))
```

---

hatvalues.rlmerMod *Robust leverage (hat values) for an rlmerMod fit.*

Description

The self-leverage A_{ii} of each observation in the robust, random-effect-whitened convolution that the estimator uses internally – the robust analogue of the linear mixed-model hat value. It reduces to the classical lmer leverage at the non-robust limit (rho = cPsi); at the robust default the effective degrees of freedom $\sum_i A_{ii}$ differ as downweighting changes each observation's pull. With groups set, the per-observation leverages are summed within each cluster, giving the cluster's leverage (its effective-df contribution).

Usage

```
## S3 method for class 'rlmerMod'
hatvalues(model, groups = NULL, ...)
```

Arguments

model	An rlmerMod object.
groups	Leverage aggregation: NULL (default) for per-observation; TRUE for the top-level grouping factor; or a factor / grouping-factor name to aggregate over.
...	Currently unused.

Value

Named numeric vector: one entry per observation (groups = NULL) or per cluster level.

See Also

cooks.distance, rlmer (the design.weights argument bounds high-leverage design points)

---

implicitIF_full *Full implicit influence function (beta, u, sigma, theta) for a fitted* rlmerMod *object.*

Description

Extends implicitIF by adding the σ-DAS and θ-DAS scoring equations to the implicit-derivative linear system. The Jacobian of the full score wrt $(\beta, u, \log\sigma, \theta)$ is computed via numDeriv::jacobian with Richardson extrapolation, reusing the converged DAS scales on the non-θ columns (they depend on θ only); the Jacobian wrt the response is closed-form. Only DASvar and DAStau methods are supported (DAStau additionally requires block sizes ≤ 2).

Usage

```
implicitIF_full(fit, eps = 1e-06, use.cache = TRUE)
```

Arguments

fit	An rlmerMod object.
eps	Numerical step for numDeriv::jacobian (default 1e-6).
use.cache	If TRUE (default), return a result cached on the fit when one is stored for the current θ, and cache the result otherwise.

Details

This function is the engine behind cooks.distance / influence; users who only want Cook's distance should call those S3 methods.

The result is cached on the fit (keyed by θ) when use.cache = TRUE, so repeated calls – and the consumers cooks.distance, the sandwich vcov, and the Satterthwaite df of summary / anova / emmeans / confint – share a single computation.

Value

A list with components `IF_beta` ($p \times n$), `IF_u` ($q \times n$), `IF_sigma` ($1 \times n$), `IF_theta` ($L \times n$), and the model-based delta-method covariance `vcov_model_delta`.

See Also

`implicitIF`, `cooks.distance`

`influence.rlmerMod` *Per-observation influence on* $(\hat{\beta}, \hat{\sigma}, \hat{\theta})$ *for an rlmerMod fit.*

Description

Thin wrapper that returns the stacked influence matrix used by `cooks.distance`. Cheaper to call this once and pass the result back to `cooks.distance` via `IF = .` than to recompute the numerical Jacobian twice.

Usage

```
## S3 method for class 'rlmerMod'
influence(model, do.coef = TRUE, ...)
```

Arguments

`model`	An `rlmerMod` object.
`do.coef`	Ignored (kept for compatibility with the `influence` generic).
`...`	Currently unused.

Value

A list with `IF_beta` ($p \times n$), `IF_sigma` ($1 \times n$), `IF_theta` ($L \times n$), and the full `implicitIF_full` object as `full`.

See Also

`implicitIF_full`, `cooks.distance`

`lapplyDatasets` *Lapply for generated datasets*

Description

Apply function for all generated datasets.

Usage

```
lapplyDatasets(datasets, FUN, ..., label, POST_FUN, datasetIndices = "all")
```

Arguments

`datasets`	Datasets list to be used to generate datasets.
`FUN`	the function to be applied to each generated dataset. The function will be called like `FUN(data, ...)`.
`...`	optional arguments to `FUN`.
`label`	optional parameter, if present, each result is added an attribute named *label* with the value of `label`.
`POST_FUN`	function to be applied to the result of `FUN`. While one could just modify `FUN` instead, this additional argument makes it a bit easier to combine different kinds of methods together.
`datasetIndices`	optional vector of dataset indices to fit, useful to try only a few datasets instead of all of them. Use `"all"` to process all datasets (default).

Value

list of results. The items in the resulting list will have two additional attributes: `datasetIndex` and `proc.time`. If `FUN` failed for an item, then the item will be the error as returned by try, i.e., it ill be of class `try-error`.

Author(s)

Manuel Koller

Examples

```
oneWay <- generateAnovaDatasets(2, 1, 5, 4)
lapplyDatasets(oneWay, function(data) sum(data$y))
lapplyDatasets(oneWay, function(data) sum(data$y), POST_FUN = function(x) x^2)
```

loadAndMergePartialResults *Load And Merge Partial Results*

Description

Convenience function that loads the results stored in each of the files and then calls `mergeProcessedFits` to merge them.

Usage

```
loadAndMergePartialResults(files)
```

Arguments

`files`	vector of filenames (including paths) of files containing the processed results

Author(s)

Manuel Koller

See Also

`processDatasetsInParallel`

lqqPsi *Default lqq (linear-quadratic-quadratic) redescending psi-function.*

Description

Pre-built `psi_func_rcpp` from `makeRobustbasePsi("lqq")` with robustbase's default lqq tuning (about 95% efficiency at the normal). The lqq psi of Koller and Stahel (2011) redescends more gradually than the bisquare and is the recommended redescender, as used by robustbase's `lmrob.control(setting = "KS2014")`. Use it as `rho.e` in `rlmer`; a redescender needs a good initial estimate, so pair it with `init = "ransac"` or `rlmer_ransac`.

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

`makeRobustbasePsi`, `bisquarePsi`.

`makeBisquarePsi` *Bisquare (Tukey biweight) psi function*

Description

Construct a redescending Tukey bisquare psi function.

Usage

```
makeBisquarePsi(c = 4.685)
```

Arguments

`c`	tuning cutoff. The default `c = 4.685` gives about 95% asymptotic efficiency for the location problem.

Details

Returns a `psi_func_rcpp` object whose psi, rho, weight and derivative slots implement Tukey's bisquare (biweight) function

$$\psi(x) = x(1 - (x/c)^2)^2 \quad \text{for } |x| \le c, \quad 0 \text{ otherwise}$$

suitable for use as `rho.e` (or `rho.b`, `rho.sigma.e`, `rho.sigma.b`) in `rlmer`.

Since robustlmm 3.5.0 the psi, rho, weight and derivative evaluations delegate to robustbase's compiled bisquare family (`Mpsi`, `Mwgt`, `Mchi`), so they match `lmrob` exactly; the returned values are identical (to numerical tolerance) to the previous hand-coded implementation. `makeBisquarePsi` is the scalar-cutoff special case of the general `makeRobustbasePsi`.

The Fisher consistency expectations (`Erho`, `Epsi2`, `EDpsi`) are computed by numerical integration against the standard normal.

The bisquare redescends comparatively fast; for a redescending fit the `"lqq"` psi (`lqqPsi`), the recommended redescender of robustbase's `lmrob.control(setting = "KS2014")`, is generally preferable (Koller and Stahel 2011).

Value

psi_func_rcpp object usable in `rho.e`, `rho.b`.

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

`makeRobustbasePsi`, `lqqPsi`.

Examples

```
pf <- bisquarePsi
pf@psi(c(-6, -3, 0, 3, 6))
```

`makeRobustbasePsi` *Redescending psi-functions from robustbase (bisquare, lqq, optimal, hampel, ggw)*

Description

Construct a redescending psi-function from one of robustbase's compiled `Mpsi` families.

Usage

```
makeRobustbasePsi(
  family = c("bisquare", "lqq", "optimal", "hampel", "ggw"),
  cc = NULL
)
```

Arguments

`family`	robustbase `Mpsi` family name, one of `"bisquare"`, `"lqq"`, `"optimal"`, `"hampel"`, `"ggw"`.
`cc`	tuning constant (a scalar for `"bisquare"` / `"optimal"`, a specification vector for `"lqq"` / `"ggw"` / `"hampel"`). `NULL` uses the family's robustbase default (about 95% efficiency at the normal).

Details

Returns a `psi_func_rcpp` object whose psi, rho, weight and derivative slots delegate to robustbase's compiled psi-function family, so the shipped functions match `lmrob` exactly. The psi, its derivative and the weight are `Mpsi(x, cc, family, 0)`, `Mpsi(x, cc, family, 1)` and `Mwgt(x, cc, family)`; the rho slot is the normalised `Mchi(x, cc, family) * MrhoInf(cc, family)`, so that `rho(Inf)` equals the supremum of rho (the `psi_func` rho convention used by `rlmer`). The Fisher consistency expectations (`Erho`, `Epsi2`, `EDpsi`) are computed by numerical integration against the standard normal.

The available families are the redescenders `"bisquare"` (Tukey biweight), `"lqq"` (linear-quadratic-quadratic), `"optimal"`, `"hampel"` and `"ggw"` (generalised Gauss-weight). When `cc` is `NULL` the family's default tuning ($\approx 95\%$ efficiency at the normal) is taken from `robustbase::.Mpsi.tuning.default(family)`; for `"lqq"`, `"ggw"` and `"hampel"` this is a short numeric *specification* vector rather than a single cut-off (see `lmrob.control`).

Of the redescenders, `bisquarePsi` redescends comparatively fast; the `"lqq"` psi of Koller and Stahel (2011), used by robustbase's `lmrob.control(setting = "KS2014")`, is the recommended redescender and is pre-built as `lqqPsi`.

Value

psi_func_rcpp object usable as `rho.e` (or `rho.b`, `rho.sigma.e`, `rho.sigma.b`) in `rlmer`.

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

`lqqPsi`, `bisquarePsi`, `Mpsi`, `lmrob.control`.

Examples

```
pf <- makeRobustbasePsi("lqq")
pf@psi(c(-6, -3, 0, 3, 6))
```

`makeSScalePsi` *S-type scale psi-function*

Description

S-type scale psi-function for high-breakdown scale estimation.

Usage

```
makeSScalePsi(c = 1.547645)
```

Arguments

`c`	tuning constant of the bisquare ρ; the default 1.547645 gives $b = 1/2$ and breakdown point 1/2.

Details

Returns a `psi_func_rcpp` object for the scale arguments `rho.sigma.e` and (for one-dimensional random-effect blocks) `rho.sigma.b` of `rlmer`. With it the scale equations become M-scale equations with a bounded ρ, as in S-estimation: $\sum_i \eta_i \tau_i^2 [\rho(t_i) - b] = 0$ with $t_i = r_i/(\tau_i \sigma)$, ρ the Tukey bisquare ρ scaled to $\sup \rho = 1$ with tuning constant `c`, and $b = E\rho(Z)$ for standard normal Z (Fisher consistency). The breakdown point of the scale equation is $\min(b, 1 - b)$: 1/2 for the default `c = 1.547645`, against 0.295 for the default `psi2propII(smoothPsi)` scale and 0.156 for `psi2propII(smoothPsi, k = 2.28)` (`setting = "RSEa"`).

It is implemented as a weight function for the existing equations, $W(t) = (\rho(t) - b)/(t^2 - \kappa_0)$ with $\rho(\sqrt{\kappa_0}) = b$, so the design-adaptive (DAS) standardisation, the consistency constant and everything built on the scale estimates (standard errors, Satterthwaite degrees of freedom) carry over unchanged.

A high-breakdown scale does not make the fit high-breakdown by itself: the fixed effects also need a redescending `rho.e` (e.g. `bisquarePsi`) and, when the start is contaminated, a high-breakdown start (`init = "robust"`); see `rlmer`. The S-scale is less efficient than the default scale estimate: its asymptotic efficiency for the scale at the normal is 0.54 with `c = 1.547645`; a larger `c` trades breakdown point for efficiency (`c = 2.937015` gives $b = 1/4$, breakdown point 1/4 and efficiency 0.87). It is meant for `rho.sigma.e` and simple variance components; `rlmer` refuses it for blocks of correlated random effects and as `rho.e` or `rho.b`. To change `c`, use `makeSScalePsi(c = ...)`, not `chgDefaults`.

Value

a `psi_func_rcpp` object; attribute `"sscale"` holds `b`, `kappa0` and `c`.

Lifecycle

Experimental. The interface and the results may change in a later version.

References

Rousseeuw PJ, Yohai VJ (1984). Robust regression by means of S-estimators. In *Robust and Nonlinear Time Series Analysis*, Lecture Notes in Statistics 26, 256–272. Springer.

See Also

`psi2propII`, `rlmer`.

Examples

```
ss <- makeSScalePsi()
attr(ss, "sscale")
## the bounded chi(t) = W(t) (t^2 - kappa0) of the scale equation
t <- c(0, 0.5, 1, 2, 5, 50)
ss@wgt(t) * (t^2 - attr(ss, "sscale")$kappa0)
```

---

mergeProcessedFits *Merge Processed Fits*

Description

Combine list of processed fits into one list in matrix form.

Usage

```
mergeProcessedFits(processedFitList)
```

Arguments

`processedFitList`
: list of processed fits as produced by `processFit`.

Value

similar list as returned by processFit just with matrix entries instead of vectors.

Examples

```
preparedDataset <-
    prepareMixedEffectDataset(Reaction ~ Days + (Days|Subject),
                              sleepstudy)
set.seed(1)
datasets <- generateMixedEffectDatasets(2, preparedDataset)

fits <- fitDatasets_lmer(datasets)
processedFits <- lapply(fits, processFit, all = TRUE)
merged <- mergeProcessedFits(processedFits)
str(merged)
```

---

other *Other methods*

Description

Other miscellaneous utilities for instances of the PsiFunction class.

Usage

```
## S4 method for signature 'Rcpp_SmoothPsi'
show(object)
## S4 method for signature 'Rcpp_HuberPsi'
show(object)
## S4 method for signature 'Rcpp_PsiFunction'
show(object)
## S4 method for signature 'Rcpp_PsiFunctionToPropIIPsiFunctionWrapper'
show(object)
```

Arguments

object	instance of class PsiFunction to be plotted

Examples

```
show(smoothPsi)
```

partialMoment_standardNormal *Compute Partial Moments*

Description

Computes a partial moment for the standard normal distribution. This is the expectation taken not from -Infinity to Infinity but just to z.

Usage

```
partialMoment_standardNormal(z, n)
```

Arguments

z	partial moment boundary, the expectation is taken from -Inf to z.
n	which moment to compute, needs to be >= 2.

References

Winkler, R. L., Roodman, G. M., & Britney, R. R. (1972). The Determination of Partial Moments. Management Science, 19(3), 290–296. https://www.jstor.org/stable/2629511, equation (2.5)

Examples

```
partialMoment_standardNormal(0, 2)
```

plot-methods *Plot an Object of the "Psi Function" Class*

Description

The plot method objects of class PsiFunction simply visualizes the $\rho()$, $\psi()$, and weight functions and their derivatives.

Usage

```
## S4 method for signature 'Rcpp_SmoothPsi'
plot(x, y,
     which = c("rho", "psi", "Dpsi", "wgt", "Dwgt"),
     main = "full",
     col = c("black", "red3", "blue3", "dark green", "light green"),
     leg.loc = "right", ...)
## S4 method for signature 'Rcpp_HuberPsi'
plot(x, y,
```

```
     which = c("rho", "psi", "Dpsi", "wgt", "Dwgt"),
     main = "full",
     col = c("black", "red3", "blue3", "dark green", "light green"),
     leg.loc = "right", ...)
## S4 method for signature 'Rcpp_PsiFunction'
plot(x, y,
     which = c("rho", "psi", "Dpsi", "wgt", "Dwgt"),
     main = "full",
     col = c("black", "red3", "blue3", "dark green", "light green"),
     leg.loc = "right", ...)
## S4 method for signature 'Rcpp_PsiFunctionToPropIIPsiFunctionWrapper'
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	instance of class PsiFunction 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

If you want to specify your own title, use main=FALSE, and a subsequent title(...) call.

See Also

psi-functions.

Examples

```
plot(huberPsiRcpp)
plot(huberPsiRcpp, which=c("psi", "Dpsi", "wgt"),
     main="short", leg = "topleft")

plot(smoothPsi)
## Plotting aspect ratio = 1:1 :
plot(smoothPsi, asp=1, main="short",
     which = c("psi", "Dpsi", "wgt", "Dwgt"))
```

`plot.rlmerMod` *Plot Method for "rlmerMod" objects.*

Description

Diagnostic plots for objects of class `rlmerMod` and `lmerMod`.

Usage

```
## S3 method for class 'rlmerMod'
plot(
  x,
  y = NULL,
  which = 1:4,
  title = c("Fitted Values vs. Residuals", "Normal Q-Q vs. Residuals",
   "Normal Q-Q vs. Random Effects", "Scatterplot of Random Effects for Group \"%s\""),
  multiply.weights = FALSE,
  add.line = c("above", "below", "none"),
  ...
)

## S3 method for class 'rlmerMod_plots'
print(x, ask = interactive() & length(x) > 1, ...)
```

Arguments

`x`	an object as created by `rlmer` or `rlmer`; or an object as created by `plot.rlmerMod`
`y`	currently ignored.
`which`	integer number between 1 and 4 to specify which plot is desired.
`title`	Titles for the different plots. The fourth item can be a format string passed to `sprintf` to add the name of the current group.
`multiply.weights`	multiply the residuals / random effects with the robustness weights when producing the Q-Q plots.
`add.line`	add reference line to plots, use `"above"` or `"below"` to show the line above or below the points. Hide the line with `"none"`.
`...`	passed on to `geom_hline` and `geom_qq_line`, to customize how the line is drawn.
`ask`	waits for user input before displaying each plot.

Details

The robustness weights for estimating the fixed and random effects are used in the plots, e.g., the ones returned by `getME(object, "w_e")` and `getME(object, "w_b")`.

Value

a list of plots of class ggplot that can be used for further modification before plotting (using print).

See Also

getME, ggplot

Examples

```
## Not run:
  rfm <- rlmer(Yield ~ (1|Batch), Dyestuff)
  plot(rfm)
  fm <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
  plot.rlmerMod(fm)

## End(Not run)
```

---

plotLongitudinalBySubject

Plot longitudinal data with robustness-weight colored lines

Description

Creates a visualization of longitudinal data with one facet per treatment group. Subject trajectories are colored by their robustness weight from a robust mixed-effects model fit, with darker lines indicating lower weights (potential outliers). Fixed-effect predictions are overlaid as reference lines.

Usage

```
plotLongitudinalBySubject(
  data,
  formula = NULL,
  idVar = "id",
  timeVar = "time",
  treatmentVar = "treatment",
  responseVar = "y",
  rlmerArgs = list(),
  lineAlpha = 0.6,
  fixedLineWidth = 1.2,
  lowColor = "black",
  highColor = "lightgray",
  fixedLineColor = "firebrick",
  fixedLinetype = "solid",
  title = NULL,
  xlab = NULL,
  ylab = NULL
)
```

Arguments

data	A data frame containing longitudinal data. Must have columns for subject ID, time, treatment group, and response variable.
formula	A formula for the mixed-effects model. Default is `y ~ treatment * time + (1 + time | id)` where `y`, `treatment`, `time`, and `id` refer to the standardized internal column names (mapped from `responseVar`, `treatmentVar`, `timeVar`, and `idVar`).
idVar	Character string naming the subject ID column in `data`. Default: `"id"`.
timeVar	Character string naming the time column in `data`. Default: `"time"`.
treatmentVar	Character string naming the treatment column in `data`. Default: `"treatment"`.
responseVar	Character string naming the response column in `data`. Default: `"y"`.
rlmerArgs	A list of additional arguments passed to `rlmer`.
lineAlpha	Numeric in [0, 1]. Transparency of subject lines. Default: 0.6.
fixedLineWidth	Numeric. Width of fixed-effect overlay lines. Default: 1.2.
lowColor	Color for low robustness weights (potential outliers). Default: `"black"`.
highColor	Color for high robustness weights (typical observations). Default: `"lightgray"`.
fixedLineColor	Color for the fixed-effect prediction lines. Default: `"firebrick"`.
fixedLinetype	Linetype for the fixed-effect prediction lines. Can be a single value (e.g., `"solid"`, `"dashed"`) applied to all lines, or `"byTreatment"` to use different linetypes for each treatment group. Default: `"solid"`.
title	Optional plot title.
xlab	Label for x-axis. If `NULL` (default), uses the value of `timeVar`.
ylab	Label for y-axis. If `NULL` (default), uses the value of `responseVar`.

Details

The function fits a robust linear mixed-effects model using `rlmer` and extracts the robustness weights for the random effects. Subjects with low weights (shown in darker colors) are those whose random effects deviate substantially from the assumed distribution.

The fixed-effect prediction lines show the population-average trajectory for each treatment group, ignoring random effects.

Value

A `ggplot` object.

See Also

`rlmer`, `generateLongitudinalDatasets`

Examples

```
## Not run:
  ## Using the medication dataset from confintROB
  library(confintROB)
  plotLongitudinalBySubject(
    medication,
    idVar = "id",
    treatmentVar = "treat",
    responseVar = "pos"
  )

  ## Using simulated data
  set.seed(123)
  simdat <- generateLongitudinalDatasets(
    numberOfDatasetsToGenerate = 1,
    numberOfSubjects = 40,
    numberOfTimepoints = 7,
    numberOfTreatmentLevels = 2,
    timeRange = c(0, 18),
    trueBeta = c(200, -2, -5, 3),
    trueSigma = 30
  )
  plotLongitudinalBySubject(simdat$generateData(1))

## End(Not run)
```

`predict.rlmerMod` *Predictions and confidence/prediction intervals for an rlmerMod fit.*

Description

By default (`interval = "none"`) returns the same numeric vector as the lme4-style point prediction (preserved byte-for-byte from the pre-existing predict method). With `interval = "confidence"` or `"prediction"`, returns a data frame with columns `fit`, `lwr`, `upr`, `se`.

Usage

```
## S3 method for class 'rlmerMod'
predict(
  object,
  newdata = NULL,
  re.form = NULL,
  ReForm,
  REForm,
  REform,
  terms = NULL,
  type = c("link", "response"),
```

```
  allow.new.levels = FALSE,
  na.action = na.pass,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

`object`	An `rlmerMod` object.
`newdata`, `re.form`, `ReForm`, `REForm`, `REform`, `terms`, `type`, `allow.new.levels`, `na.action`, `...`	See the lme4 `predict.merMod` documentation; these arguments are forwarded unchanged to the point-prediction code path.
`interval`	One of `"none"` (default, returns a numeric vector for backwards compatibility), `"confidence"` (fit-uncertainty interval for $X\hat{\beta} + Z\hat{b}$), or `"prediction"` (adds residual variance $\hat{\sigma}^2$).
`level`	Coverage level for the interval; default 0.95.

Details

The fixed-effect contribution to the SE uses the robust cluster sandwich `vcov(object, type = "sandwich")`; the random-effects contribution (when REs are part of the prediction) is computed from the partial influence function of $\hat{u}$ (the local-shift sensitivity). For `interval = "prediction"` the additional residual variance $\hat{\sigma}^2$ is added. Intervals are `fit +/- z * se` with $z = \Phi^{-1}((1 + level)/2)$; for small numbers of clusters the normal approximation may under-cover, and a bootstrap CI (via `confint(..., method = "boot")`) is preferable.

Value

A numeric vector (when `interval = "none"`) or a data frame with columns `fit`, `lwr`, `upr`, `se`.

Lifecycle

Stable.

See Also

`vcov`

prepareMixedEffectDataset

Prepare Dataset for Parametric Bootstrap

Description

This function runs lmer and extracts all information needed to generate new datasets using parametric bootstrap later.

Usage

```
prepareMixedEffectDataset(
  formula,
  data,
  REML = TRUE,
  overrideBeta,
  overrideSigma,
  overrideTheta,
  ...
)
```

Arguments

formula	passed on to lmer
data	passed on to lmer
REML	passed on to lmer
overrideBeta	use to override beta used to simulate new datasets, by default getME(fm, "beta") where fm is the fitted model returned by lmer.
overrideSigma	use to override sigma used to simulate new datasets, by default getME(fm, "sigma") where fm is the fitted model returned by lmer.
overrideTheta	use to override theta used to simulate new datasets, by default getME(fm, "theta") where fm is the fitted model returned by lmer.
...	all additional arguments are added to the returned list.

Value

List that can be passed to generateMixedEffectDatasets.

data:	the original dataset
X:	the X matrix as returned by getME
Z:	the Z matrix as returned by getME
Lambda:	the Lambda matrix as returned by getME
numberOfFixedEffects:	the number of fixed effects coefficients

numberOfRandomEffects:
: the number of random effects

numberOfRows:
: number of rows in the generated dataset

trueBeta:
: true values used for beta

trueSigma:
: true value used for sigma

trueTheta:
: true values used for theta

formula:
: formula to fit the model using lmer

...:
: additional arguments passed via ...

Author(s)

Manuel Koller

Examples

```
preparedDataset <- prepareMixedEffectDataset(Reaction ~ Days + (Days|Subject), sleepstudy)
str(preparedDataset)
```

---

processDatasetsInParallel
Process Datasets in Parallel

Description

Convenience function to run simulation study in parallel on a single machine.

Usage

```
processDatasetsInParallel(
  datasets,
  path,
  baseFilename,
  fittingFunctions,
  chunkSize,
  saveFitted = FALSE,
  checkProcessed = FALSE,
  createMinimalSaveFile = FALSE,
  ncores = 1,
  clusterType = "PSOCK",
  ...
)
```

Arguments

datasets	dataset list generated by one of the generate functions.
path	path to save the datasets to.
baseFilename	filename to use, without extension.
fittingFunctions	vector of fitDatasets functions that should be applied to each dataset.
chunkSize	number of datasets to process together in a single job.
saveFitted	logical, if true, the raw fits are also stored.
checkProcessed	logical, if true, will check whether the contents of the processed output is reproduced for the first dataset. This is useful to ensure that everything is still working as expected without having to re-run the whole simulation study.
createMinimalSaveFile	logical, if true, will create a file with the processed results of the first three datasets. This is helpful if one wants to store only the final aggregated results but still wants to make sure that the full code works as expected.
ncores	number of cores to use in processing, if set to 1, datasets are processed in the current R session. Use detectCores to find out how many cores are available on your machine.
clusterType	type of cluster to be created, passed to makeCluster.
...	passed on to processFit. Use this to control what to save.

Details

The merged results are saved in a file taking the name <path>/<baseFilename>-processed.Rdata. You can delete the intermediate result files with the numbers (the chunk index) in the name.

To run on multiple machines, use saveDatasets to save datasets into multiple files. Then call processFile on each of them on the designated machine. Finally, load and merge the results together using loadAndMergePartialResults.

Value

The list of all processed results merged together.

To help reproducibility, the output of toLatex(sessionInfo(), locale = FALSE) is stored in the sessionInfo attribute.

Author(s)

Manuel Koller

See Also

saveDatasets, processFile

`processFile` *Process File of Stored Datasets*

Description

Call this function for each file stored using `saveDatasets`. If a file hasn't been processed yet, then it is processed and a new file with the postfix "processed" is created containing the results.

Usage

```
processFile(
  file,
  fittingFunctions,
  saveFitted = FALSE,
  checkProcessed = FALSE,
  createMinimalSaveFile = FALSE,
  datasets,
  ...
)
```

Arguments

`file`	file saved by `saveDatasets`.
`fittingFunctions`	vector of `fitDatasets` functions that should be applied to each dataset.
`saveFitted`	logical, if true, the raw fits are also stored.
`checkProcessed`	logical, if true, will check whether the contents of the processed output is reproduced for the first dataset. This is useful to ensure that everything is still working as expected without having to re-run the whole simulation study.
`createMinimalSaveFile`	logical, if true, will create a file with the processed results of the first three datasets. This is helpful if one wants to store only the final aggregated results but still wants to make sure that the full code works as expected.
`datasets`	optional, datasets as stored in `file`, to avoid doing a detour of saving and loading the file.
`...`	passed on to `processFit`. Use this to control what to save.

Details

In case the raw fits may have to be inspected or `processFit` may be called with another set of arguments, then set `saveFitted` to TRUE. In that case, another file with the postfix "fitted" is created. Remove the files with postfix "processed" and run `processFile` again. The fits will not be re-done but instead loaded from the file with postfix "fitted".

Value

The list of all processed results merged together.

To help reproducibility, the output of `toLatex(sessionInfo(), locale = FALSE)` is stored in the `sessionInfo` attribute.

Author(s)

Manuel Koller

`processFit` *Process Fitted Objects*

Description

Methods to process fitted objects and convert into a data structure that is useful in post-processing.

Usage

```
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'lmerMod'
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
```

```
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'rlmerMod'
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'heavyLme'
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'lqmm'
```

```
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'rlme'
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  ...
)

## S3 method for class 'varComprob'
processFit(
  obj,
  all = FALSE,
  coefs = TRUE,
  stdErrors = all,
  tValues = all,
  sigma = TRUE,
  thetas = TRUE,
  b = all,
  meanB = all,
```

```
  meanAbsB = all,
  residuals = all,
  converged = TRUE,
  numWarnings = all,
  procTime = all,
  isInterceptCorrelationSlopeModel,
  ...
)
```

Arguments

obj	object returned by the fitting method.
all	logical, shorthand to enable all exports.
coefs	logical, if true coefficients are added to export.
stdErrors	logical, if true, standard errors are added to export.
tValues	logical, if true, t-values are added to export.
sigma	logical, if true, sigma is added to export.
thetas	logical, if true, thetas are added to export.
b	scalar logical or index vector, if true, all random effects are added to export. If an index vector is given, then only the corresponding random effects are added to the export. The same order as in lmer is used for all methods.
meanB	logical, if true, the mean of the random effects is added to the export.
meanAbsB	logical, if true, the mean of the absolute value of the random effects is added to the export.
residuals	scalar logical or index vector, similar to argument b, just returning the residuals.
converged	logical, if true, convergence code is added to export.
numWarnings	logical, if true, the number of warnings generated during the fitting process is added to export.
procTime	logical, if true, time needed to fit object is added to export.
...	optional parameters used for some implementations.
isInterceptCorrelationSlopeModel	optional logical, can be used to override the assumption that a model with three variance components can be interpreted as having intercept, correlation and slope.

Details

Warning. processFit.varComprob uses simplistic logic to convert from the parameterisation used in the robustvarComp package to theta as used in lmer and rlmer. If there are three variance components, the code assumes that they are intercept, correlation and slope. Otherwise the code assumes that the variance components are independent. Exports b and residuals are not supported.

Value

List with extracted values, most items can be suppressed to save disk space.

`label:`	Name of fitting method used to create the fit
`datasetIndex:`	Index of the dataset in the dataset list
`coefficients:`	Vector of estimated fixed-effects coefficients of the fitted model
`standardErrors:`	Vector of estimated standard errors of the fixed-effects coefficients
`tValues:`	Vector of t-Values (or z-Values depending on fitting method) of the fixed-effects coefficients
`sigma:`	Estimated residual standard error
`thetas:`	Vector of random-effects parameter estimates. As parameterized as by `lmer` and `rlmer`.
`b:`	Vector of requested predicted random-effects.
`meanB:`	Vector of means of the predicted random-effects.
`meanAbsB:`	Vector of means of the absolute values of the predicted random-effects.
`residuals:`	Vector of requested residuals.
`converged:`	Convergence status as reported by the fitting method. `0` means converged. If not available, `NA` is used. Other values are to be interpreted carefully as codes vary from method to method.
`numberOfWarnings:`	the number of warnings generated during the fitting process.
`proc.time:`	Vector of times (user, system, elapsed) as reported by `proc.time` required to fit the model.

Examples

```
  set.seed(1)
  oneWay <- generateAnovaDatasets(1, 1, 10, 4,
                                  lmeFormula = y ~ 1,
                                  heavyLmeRandom = ~ 1,
                                  heavyLmeGroups = ~ Var2,
                                  lqmmRandom = ~ 1,
                                  lqmmGroup = "Var2",
                                  groups = cbind(rep(1:4, each = 10), rep(1:10, 4)),
                                  varcov = matrix(1, 4, 4),
                                  lower = 0)
  processFit(fitDatasets_lmer(oneWay)[[1]], all = TRUE)
  processFit(fitDatasets_rlmer_DASvar(oneWay)[[1]], all = TRUE)
  ## Not run:
    processFit(fitDatasets_heavyLme(oneWay)[[1]], all = TRUE)

## End(Not run)
  if (require(lqmm)) {
    processFit(fitDatasets_lqmm(oneWay)[[1]], all = TRUE)
  }
  ## Not run:
```

```
    processFit(fitDatasets_varComprob_compositeTau(oneWay)[[1]], all = TRUE)

## End(Not run)
```

---

psi-functions *Classical, Huber and smoothed Huber psi- or rho-functions*

Description

ψ-functions are used by `rlmer` in the estimating equations and to compute robustness weights. Change tuning parameters using `chgDefaults` and convert to squared robustness weights using the `psi2propII` function.

Usage

```
## see examples
```

Details

The **"classical" ψ-function** `cPsi` can be used to get a non-robust, i.e., classical, fit. The `psi` slot equals the identity function, and the `rho` slot equals quadratic function. Accordingly, the robustness weights will always be 1 when using `cPsi`.

The **Huber ψ-function** `huberPsi` is identical to the one in the package `robustbase`. The `psi` slot equals the identity function within $\pm k$ (where k is the tuning parameter). Outside this interval it is equal to $\pm k$. The `rho` slot equals the quadratic function within $\pm k$ and a linear function outside.

The **smoothed Huber ψ-function** is very similar to the regular Huber ψ-function. Instead of a sharp bend like the Huber function, the smoothed Huber function bends smoothly. The first tuning constant, k, can be compared to the tuning constant of the original Huber function. The second tuning constant, s, determines the smoothness of the bend.

See Also

`chgDefaults` and `psi2propII` for changing tuning parameters; `psi_func-class` for a more detailed description of the slots;

Examples

```
plot(cPsi)
plot(huberPsiRcpp)
plot(smoothPsi)
curve(cPsi@psi(x), 0, 3, col="blue")
curve(smoothPsi@psi(x), 0, 3, add=TRUE)
curve(huberPsiRcpp@psi(x), 0, 3, add=TRUE, col="green")
```

psi2propII *Convert to Proposal 2 weight function*

Description

Converts the psi_func object into a function that corresponds to Proposal 2, i.e., a function of the squared weights. The other elements of the psi_func object are adapted accordingly.

Usage

```
psi2propII(object, ..., adjust = FALSE)

## S4 method for signature 'psi_func_rcpp'
psi2propII(object, ..., adjust = FALSE)
```

Arguments

object	instance of Rcpp_PsiFunction class to convert
...	optional, new default arguments passed to chgDefaults.
adjust	logical, whether tuning parameters should be adjusted automatically, such that the scale estimate has the same asymptotic efficiency as the location estimate.

Examples

```
par(mfrow=c(2,1))
plot(smoothPsi)
plot(psi2propII(smoothPsi))
```

ransac_basin_radius *Redescender half-weight (basin) radius*

Description

Half-weight (basin) radius for a fixed-effects start.

Usage

```
ransac_basin_radius(object, cc = NULL, rho = NULL)
```

Arguments

object	a fitted lmerMod or rlmerMod supplying the error scale σ and the fixed-effects design matrix.
cc	without rho: the bisquare cutoff c (default 4.685). With rho: the upper end of the search interval for $s_{1/2}$ (default: the rejection point of rho).
rho	an optional psi_func_rcpp (e.g. bisquarePsi, lqqPsi) whose Dpsi defines the expected-weight function m.

Details

Computes the half-weight radius

$$r^\star(c) = s_{1/2}(c)\,\sigma / \max_j \|x_j\|$$

of the ball around the initial fixed-effects estimate on which the population Hessian of a redescending M-estimator stays positive definite (the RANSAC-RSE basin theorem, Step 1). Write $m(s) = E\,\psi'(Z + s)$, $Z \sim N(0,1)$, for the expected redescender weight of an observation whose residual carries a deterministic mean-shift of s error-scale units; m is even in s and decreasing in $|s|$ near the origin. The half-weight shift $s_{1/2}$ is the first $s > 0$ with $m(s) = m(0)/2$. By Cauchy-Schwarz every observation's shift within the ball is at most $\|x_j\| r^\star/\sigma \le s_{1/2}$, so its expected weight is at least $m(0)/2$ and $-H(\beta) \succeq -H_0/2 \succ 0$ throughout the ball. For the bisquare at the default cutoff $c = 4.685$, $s_{1/2} = 1.410$; for `sleepstudy` this gives $r^\star \approx 4.0$ in the units of β.

Up to version 3.5.0-2 this function returned the larger support-preservation radius $c\,\sigma/(2 \max_j \|x_j\|)$ (6.6 for `sleepstudy`). That radius only kept every observation inside the redescender's support; it did not bound the Hessian away from singularity, which is what the basin guarantee needs, so it was replaced by the half-weight radius.

When `rho` is supplied, $m(s)$ is computed from its `Dpsi` slot by numerical integration, so the radius applies to any redescender (e.g. `lqqPsi`); `cc` is then only the upper end of the search interval for $s_{1/2}$ (the rejection point of `rho`, found numerically, when `cc` is `NULL`; `10` if `rho` has none). Without `rho`, the bisquare ψ' with cutoff `cc` (default `4.685`) is used. If m is not monotone on the search interval, the first crossing of $m(0)/2$ from the origin is taken.

Value

the half-weight radius $r^\star$ in the units of β (a scalar), with attributes `"max_xnorm"`, `"s_half"` (the half-weight shift $s_{1/2}$) and `"cc"`.

Lifecycle

Experimental. The interface and the results may change in a later version.

See Also

`rlmer_ransac`

Examples

```
lm0 <- lme4::lmer(Reaction ~ Days + (Days | Subject), lme4::sleepstudy)
ransac_basin_radius(lm0)                  # bisquare, c = 4.685
ransac_basin_radius(lm0, rho = lqqPsi)    # from lqq's psi'
```

`ransac_lme4` *RANSAC initial estimator for LMM*

Description

RANSAC-style random subsample initial estimator for linear mixed-effects models.

Usage

```
ransac_lme4(
  formula,
  data,
  K = 200L,
  sub_frac = 0.5,
  scale_fn = NULL,
  adaptive = TRUE,
  patience = 50L,
  K_min = 50L,
  tol = 0.001,
  stratify = TRUE,
  n_keep = 1L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`formula`	model formula in `lmer` syntax.
`data`	full data frame.
`K`	maximum number of random subsamples (default 200). With `adaptive = TRUE` the search stops early once the best score plateaus; `K` is then an upper budget rather than a fixed count.
`sub_frac`	fraction of the data per subsample (default 0.5).
`scale_fn`	function from a numeric residual vector to a scalar scale, used to rank the candidate fits. The default (`NULL`) is the bisquare M-scale of the residuals about zero (breakdown point 1/2). It is centred: a candidate whose predictions are off by a constant scores worse than the correct one, which `Qn` (the default before version 3.5.1) cannot detect; `scale_fn = robustbase::Qn` reproduces the earlier ranking. See the section "Ranking RANSAC candidates" of `vignette("simulationStudies")`.
`adaptive`	logical (default `TRUE`); stop drawing subsamples once the best score has not improved by more than `tol` (relative) for `patience` consecutive draws, after at least `K_min` draws. `FALSE` runs exactly `K` draws (the previous behaviour).
`patience`	number of consecutive non-improving draws that triggers the adaptive stop (default 50).
`K_min`	minimum draws before the adaptive stop can fire (default 50).

tol	relative-improvement threshold for the adaptive stop (default 1e-3).
stratify	logical (default TRUE); draw each subsample stratified by the random-effects grouping factor (the one with the most levels, parsed from the formula), taking ceiling(sub_frac * n_g) (at least one) rows within each level so every grouping level is represented. This keeps the per-subsample lmer fittable on designs with many small clusters, where plain random subsampling can leave a level empty. FALSE (or no grouping factor) falls back to simple random subsampling.
n_keep	number of distinct best-scoring candidate starts to return in $candidates (default 1). The multi-start consensus of rlmer_ransac (n_starts > 1) uses these to sample several basins of a redescending psi.
seed	optional RNG seed for reproducibility.
verbose	logical; print progress every 50 subsamples.

Details

For K random subsamples of the data, fit a classical lmer on each subsample, score by a robust scale of residuals computed on the *full* data, and return the lmer fit minimising that score.

The residuals are conditional: clusters in the subsample keep their predicted random effects, the others get none. A contaminated cluster inside the subsample is therefore absorbed by its own random effect, and under cluster-level contamination the start's between-cluster coefficients can be poor (see the section "Ranking RANSAC candidates" of vignette("simulationStudies")).

The breakdown point any start can give the fixed effects is limited by the design: with categorical predictors many observations lie on one hyperplane through the origin, and the ceiling drops (to about 1/4 for a balanced two-level factor with an intercept); see design_breakdown.

The motivation is that for redescending psi-functions (e.g. lqqPsi, the recommended redescender, or the faster-redescending bisquarePsi) the rlmer optimiser benefits from a starting value close to the true parameters. A bad initial estimate can produce phony local minima (e.g. random-effects correlation pinned at +/- 1; see Koller and Stahel 2022, Section 4.4). RANSAC is a classical way of generating a high-breakdown-point initial estimate by subsampling.

Value

list with fit (best lmerMod), scale (its score), subset (its row indices), scales (length-K vector of scores; NA for draws not run under an adaptive early stop), K (the requested cap), K_used (draws actually run), n_sub, n_singular (total number of collinear candidate observations skipped by the nonsingular-subsampling draw across all subsamples; see below).

Nonsingular subsampling

With categorical predictors a rank-deficient subsample often does *not* make lmer error — it silently drops the aliased (all-zero) fixed-effect columns of a dropped factor level — so a degenerate candidate would otherwise enter the scale competition with the wrong number of parameters. Rather than draw and then repair, each subsample is drawn *nonsingular by construction* using the nonsingular-subsampling algorithm of Koller and Stahel (2017, Algorithm 1; the method behind robustbase::lmrob.control(setting = "KS2014")). The draw units (clusters when a grouping factor is available, else single observations) are randomly permuted; a Gaxpy-variant LU decomposition with partial pivoting and column skipping (implemented in C++) then walks the permuted observations and greedily selects the first

p that are linearly independent, *skipping* any observation collinear with those already chosen. This yields a full-rank fixed-effects core whenever the full design has full column rank. Whole clusters carrying that core are the mandatory seed; further clusters are then added in the permuted order until the retained data are identifiable — (i) a full-rank fixed-effects design [guaranteed by the core], (ii) at least two observed levels of every random-effects grouping factor, and (iii) non-constant numeric random-slope variables — and finally up to the target subsample size. Because adding clusters never lowers the rank or removes a level this is monotone and terminates. `n_singular` counts the collinear candidate observations skipped by the LU across all draws (a measure of the collinearity encountered); it is `0` on a purely continuous, full-rank design, where the algorithm selects exactly the first p permuted observations, so the draw is a uniform random cluster subsample — statistically identical to plain random subsampling. Note that a nonsingular *start* does not guarantee the fit stays nonsingular through the `rlmer` refinement: a redescending psi can zero-weight observations back into a rank-deficient design (Koller and Stahel 2017, Remark 2), which `rlmer_ransac` checks for and warns about.

Lifecycle

Experimental. The interface and the results may change in a later version.

References

Koller, M. and Stahel, W. A. (2017) Nonsingular subsampling for regression S estimators with categorical predictors. *Computational Statistics* **32**(2), 631–646.

Examples

```
set.seed(1)
res <- ransac_lme4(Reaction ~ Days + (Days | Subject),
                   data = sleepstudy, K = 30)
res$scale
```

`refit.rlmerMod` *Refit a robust linear mixed model to a new response*

Description

Fits the model of `object` again with the response replaced by `newresp`, keeping the psi-functions, the method, the prior weights and the offset stored in `object`; the original data is not needed. The fit is started from the estimates of `object`. This is the `refit` generic of **lme4**; together with `simulate` it gives a bootstrap with robust refits.

Usage

```
## S3 method for class 'rlmerMod'
refit(object, newresp, ...)
```

Arguments

object	an rlmerMod object.
newresp	the new response: a numeric vector with one value per observation used in the fit, or a one-column data frame such as a column of simulate(object).
...	not used.

Value

An rlmerMod object. Its call is that of object.

Lifecycle

Stable.

Note

The **generics** package has a generic of the same name. If it is attached after **lme4**, call lme4::refit.

Examples

```
fm <- rlmer(Reaction ~ Days + (1 | Subject), sleepstudy)
ystar <- simulate(fm, seed = 1)
fixef(refit(fm, ystar$sim_1))
```

---

residuals.rlmerMod *Get residuals*

Description

Residuals of a fitted rlmer model: the difference of the observations and the fitted values including the random effects, or a weighted or whitened version of it.

Usage

```
## S3 method for class 'rlmerMod'
residuals(
  object,
  type = c("response", "pearson", "weighted", "normalized"),
  scaled = FALSE,
  ...
)
```

Arguments

object	rlmerMod object
type	type of residuals, see Details.
scaled	divide the "response" or "weighted" residuals by the residual standard deviation? "pearson" and "normalized" residuals are always scaled.
...	ignored

Details

`"response"` $y - \hat{\mu}$, as for `lmer`.

`"pearson"` the response residuals multiplied by the square root of the prior weights and divided by $\hat{\sigma}$. These are the "scaled residuals" of `summary`.

`"weighted"` the response residuals multiplied by the square root of the prior weights and by the robustness weights `getME(object, "w_e")`.

`"normalized"` the marginal residuals $y - X\hat{\beta}$ (minus the offset) multiplied by the inverse of the lower Cholesky factor of the fitted marginal covariance $\hat{\sigma}^2(Z\Lambda\Lambda'Z' + W^{-1})$, with W the prior weights. At the model they are uncorrelated with unit variance, so they can be used to check the fitted within-cluster covariance structure. The Cholesky factor is taken in the order of the observations in the data, so each residual is adjusted for the earlier observations of its cluster.

Lifecycle

Stable.

Examples

```
## Not run:
  fm <- rlmer(Yield ~ (1|Batch), Dyestuff)
  stopifnot(all.equal(resid(fm, type="weighted"),
                      resid(fm) * getME(fm, "w_e")))

## End(Not run)
```

`rlmer` *Robust Scoring Equations Estimator for Linear Mixed Models*

Description

Robust estimation of linear mixed effects models, for hierarchical nested and non-nested, e.g., crossed, datasets.

Usage

```
rlmer(
  formula,
  data,
  ...,
  method = c("DAStau", "DASvar"),
  setting,
  rho.e,
  rho.b,
  rho.sigma.e,
  rho.sigma.b,
```

```
  rel.tol = 1e-08,
  max.iter = 40 * (r + 1)^2,
  verbose = 0,
  doFit = TRUE,
  init,
  size_obr = FALSE,
  design.weights = NULL,
  control = rlmerControl()
)

lmerNoFit(formula, data = NULL, ..., initTheta)
```

Arguments

formula	a two-sided linear formula object describing the fixed-effects part of the model, with the response on the left of a ~ operator and the terms, separated by + operators, on the right. The vertical bar character "|" separates an expression for a model matrix and a grouping factor.
data	an optional data frame containing the variables named in formula. By default the variables are taken from the environment from which lmer is called.
...	Additional parameters passed to lmer to find the initial estimates. See lmer.
method	method to be used for estimation of theta and sigma, see Details.
setting	a string specifying suggested choices for the arguments rho.e, rho.sigma.e, rho.b and rho.sigma.b. Use "RSEn" (the default), "RSEa", or "RSEhb" for the high-breakdown configuration (see the section on breakdown in Details; it also sets init = "robust" unless init is given). "RSEn" and "RSEa" use smoothPsi for all the "rho" arguments. For rho.sigma.e, squared robustness weights are used (see psi2propII). "RSEn" uses the same tuning parameter as for rho.e, which leads to higher robustness but lower efficiency. "RSEa" adjusts the tuning parameter for higher asymptotic efficiency which results in lower robustness (k = 2.28 for default rho.e). For diagonal random effects covariance matrices, rho.sigma.b is treated exactly as rho.sigma.e. For block diagonal random effects covariance matrices (with correlation terms), regular robustness weights are used for rho.sigma.b, not squared ones, as they're not needed. But the tuning parameters are adjusted for both rho.b and rho.sigma.b according to the dimensions of the blocks (for both "RSEn" or "RSEa"). For a block of dimension 2 (e.g., correlated random intercept and slope) k = 5.14 is used.
rho.e	object of class psi_func, specifying the functions to use for the huberization of the residuals.
rho.b	object of class psi_func or list of such objects (see Details), specifying the functions to use for the huberization of the random effects.
rho.sigma.e	object of class psi_func, specifying the weight functions to use for the huberization of the residuals when estimating the variance components, use the psi2propII function to specify squared weights and custom tuning parameters.
rho.sigma.b	(optional) object of class psi_func or list of such objects, specifying the weight functions to use for the huberization of the random effects when estimating the

variance components (see Details). Use psi2propII to specify squared weights and custom tuning parameters or chgDefaults for regular weights for variance components including correlation parameters.

rel.tol — relative tolerance used as criteria in the fitting process.

max.iter — maximum number of iterations allowed.

verbose — verbosity of output. Ranges from 0 (none) to 3 (a lot of output)

doFit — logical scalar. When doFit = FALSE the model is not fit but instead a structure with the model matrices for the random-effects terms is returned (used to speed up tests). When doFit = TRUE, the default, the model is fit immediately.

init — optional lmerMod- or rlmerMod-object to use for starting values, a list with elements 'fixef', 'u', 'sigma', 'theta', the string "robust" or "ransac", or a function producing an lmerMod object.

init = "robust" is a high-breakdown start for any random-effects structure: an S-estimate (robustbase::lmrob(method = "S")) of the fixed-effects part of the model screens the observations (absolute residual at most qnorm(0.995) times its scale), and lmer on the retained observations supplies 'fixef', 'sigma' and 'theta'. It is computed for the model as specified (contrasts, subset, prior weights and offset) and is reproducible: the S-estimate's resampling uses a fixed seed and leaves the random number stream untouched. The start matters only with a redescending rho.e or rho.b, whose estimating equations have several solutions; see the section on breakdown in Details.

An lmerMod or rlmerMod passed here supplies the model, not only the starting values: the working object is built from it, so formula and data no longer have any effect. Pass such an object only when it was fitted on the same data; a warning is issued if its number of observations does not match nrow(data). To start from values obtained elsewhere while keeping the model given by formula and data, use the list form. Its components are optional — whatever is omitted keeps the value from the internal lmer initialisation, which is the right choice for 'u' when the values come from a fit on different observations.

When init = "ransac", the high-breakdown RANSAC start is obtained by calling ransac_lme4(formula, data) with its default K = 200 subsamples and sub_frac = 0.5. Its purpose is to raise the breakdown point of the starting value, which is what makes redescending psi-functions usable: their estimating equations are non-convex, so a low-breakdown start can be drawn to a bad local solution. rlmer takes 'fixef', 'sigma' and 'theta' from that fit and fits the model on the full data. The string form is random (not reproducible without an outer set.seed); for fine control of K, sub_frac, or seed, use rlmer_ransac.

size_obr — logical scalar; if TRUE (default FALSE), the size-controlling weight w_delta in the block-diagonal variance-components scoring equation is replaced by the Hampel-OBR (Stahel 1987 / Hampel et al. 1986) form $w_\tau(d) = \min(1, b_\tau/|d - s\kappa - a|)$, with b_τ taken from the tuning constant of rho.sigma.b and a determined by Fisher consistency under χ^2_s. The default finite-difference form $w_\delta(d) = (\psi(d) - \psi(d - s\kappa))/s$ is not Hampel-OBR-optimal at the central model; this option recovers asymptotic Hampel-OBR efficiency for the variance-component magnitudes (a typically modest 1-2 percentage-point gain at matched gross-error sensitivity). Has no effect for diagonal V_b (block size 1).

`design.weights` Mallows-type design weights for robustness to high-leverage design points: `NULL` (default, exact current behaviour), a numeric vector of deterministic weights $\eta_i \in (0, 1]$ of length `n`, or `"mcd"` to compute $\eta_i = \min(1, \chi^2_{p^*}(0.975)/d_i^2)^{1/2}$ from robust squared Mahalanobis distances d_i^2 of the non-constant fixed-effects covariates (`covMcd`); the `"mcd"` tuning (severity $\gamma = 1$, cutoff $c = 0.975$) is the simulation-backed default (see `vignette("rlmer")`). The weights multiply the e-side score contribution of each observation throughout the estimator – the β, u, σ and θ equations, the model `vcov`, the Satterthwaite degrees of freedom and the influence diagnostics – bounding the influence of high-leverage design points. For designs without continuous covariates `"mcd"` reduces to $\eta \equiv 1$ (the unmodified estimator). Active design weights are supported for a *single grouping factor* only; `rlmer` stops with an error if the model has more than one. Leverage robustness costs a little efficiency at the clean model (about 1% at the default tuning; see `vignette("rlmer")`).

`control` an `rlmerControl` object: `rel.tol`, `max.iter`, the settings of the Monte-Carlo DAS-tau calibration, and whether to fit again if the fit does not converge. A setting may be given either as an argument or in `control`, not in both.

`initTheta` parameter to initialize theta with (optional)

Details

Overview: This function implements the Robust Scoring Equations estimator for linear mixed effect models. It can be used much like the function `lmer` in the package `lme4`. The supported models are the same as for `lmer` (gaussian family only). The robust approach used is based on the robustification of the scoring equations and an application of the Design Adaptive Scale approach.

Example analyses and theoretical details on the method are available in the vignette (see `vignette("rlmer")`).

Models are specified using the `formula` argument, using the same syntax as for `lmer`. Additionally, one also needs to specify what robust scoring or weight functions are to be used (arguments starting with `rho.`). By default a smoothed version of the Huber function is used. Furthermore, the `method` argument can be used to speed up computations at the expense of accuracy of the results.

Computation methods: Currently, there are two different methods available for fitting models. They only differ in how the consistency factors for the Design Adaptive Scale estimates are computed. Available fitting methods for theta and sigma.e:

- `DAStau` (default): For this method, the consistency factors are computed using numerical quadrature. This is slower but yields more accurate results. This is the direct analogue to the DAS-estimate in robust linear regression.
- `DASvar`: This method computes the consistency factors using a direct approximation which is faster but less accurate. For complex models with correlated random effects with more than one correlation term, this is the only method available.

`DAStau` supports blocks of random effects of dimension at most 2; for fits containing a larger block it falls back to `DASvar` with a warning. An *experimental* option, `options(robustlmm.dastau.mc = TRUE)`, lifts this restriction via a Monte-Carlo calibration of the consistency factors; it is simulation-validated but not backed by a finite-sample theorem, see `robustlmm-options`.

Weight functions: The tuning parameters of the weight functions "rho" can be used to adjust robustness and efficiency of the resulting estimates (arguments `rho.e`, `rho.b`, `rho.sigma.e` and `rho.sigma.b`). Better robustness will lead to a decrease of the efficiency. With the default setting, `setting = "RSEn"`, the tuning parameters are set to yield estimates with approximately 95% efficiency for the fixed effects. The variance components are estimated with a lower efficiency but better robustness properties.

One has to use different weight functions and tuning parameters for simple variance components and for such including correlation parameters. By default, they are chosen appropriately to the model at hand. However, when using the `rho.sigma.e` and `rho.sigma.b` arguments, it is up to the user to specify the appropriate function. See `asymptoticEfficiency` for methods to find tuning parameters that yield a given asymptotic efficiency.

- For simple variance components and the residual error scale use the function `psi2propII` to change the tuning parameters. This is similar to Proposal 2 in the location-scale problem (i.e., using the squared robustness weights of the location estimate for the scale estimate; otherwise the scale estimate is not robust).
- For multi-dimensional blocks of random effects modeled, e.g., a model with correlated random intercept and slope, (referred to as block diagonal case below), use the `chgDefaults` function to change the tuning parameters. The parameter estimation problem is multivariate, unlike the case without correlation where the problem was univariate. For the employed estimator, this amounts to switching from simple scale estimates to estimating correlation matrices. Therefore different weight functions have to be used. Squaring of the weights (using the function `psi2propII`) is no longer necessary. To yield estimates with the same efficiency, the tuning parameters for the block diagonal are larger than for the simple case. Tables of tuning parameters are given in Table 2 and 3 of the vignette (`vignette("rlmer")`).

Breakdown point: The residual scale σ is estimated from $\sum_i \eta_i \tau_i^2 w(t_i)(t_i^2 - \kappa) = 0$, $t_i = r_i/(\tau_i\sigma)$, with the weights w of `rho.sigma.e`. For the `psi2propII` weights of the default settings $w(t)(t^2 - \kappa)$ is bounded by its limit $\chi(\infty)$, and the equation breaks down ($\hat{\sigma} \to \infty$) at the fraction $\kappa/(\kappa + \chi(\infty))$ of outliers: 0.295 for `"RSEn"` and 0.156 for `"RSEa"`. The scale is inflated well before that, and a redescending `rho.e` then no longer rejects the outliers, whatever the start. The simple variance components behave in the same way at the level of the random effects.

`setting = "RSEhb"` is a high-breakdown configuration: the S-type scale `sScalePsi` (breakdown point 1/2) for `rho.sigma.e` and for simple variance components, a bisquare `rho.b` for simple random effects, the start `init = "robust"`, and a bisquare `rho.e` with `c = 3.444` (85% efficiency). Its fixed effects break down under considerably more contamination than those of the default settings, at a cost in efficiency and in the precision of the variance components. For inference use the cluster sandwich (`vcov_type = "sandwich"` in `confint` and `anova`) and the default `method = "DAStau"`: `"DASvar"` under-estimates the residual scale in this setting (and `"DAStau"` falls back to it, with a warning, for random-effect blocks of dimension above 2). Blocks of correlated random effects keep their multivariate `rho.b` and `rho.sigma.b` and get no protection against outlying clusters. See the section "Studies for the high-breakdown configuration" of `vignette("simulationStudies")` for the breakdown, efficiency and coverage results and their limits.

Recommended tuning parameters: For a more robust estimate, use `setting = "RSEn"` (the default). For higher efficiency, use `setting = "RSEa"`. The settings described in the following paragraph are used when `setting = "RSEa"` is specified.

For the smoothed Huber function the tuning parameters to get approximately 95% efficiency are $k = 1.345$ for `rho.e` and $k = 2.28$ for `rho.sigma.e` (using the squared version). For simple variance components, the same can be used for `rho.b` and `rho.sigma.b`. For variance components including correlation parameters, use $k = 5.14$ for both `rho.b` and `rho.sigma.b`. Tables of tuning parameter are given in Table 2 and 3 of the vignette (`vignette("rlmer")`).

Specifying (multiple) weight functions: If custom weight functions are specified using the argument `rho.b` (`rho.e`) but the argument `rho.sigma.b` (`rho.sigma.e`) is missing, then the squared weights are used for simple variance components and the regular weights are used for variance components including correlation parameters. The same tuning parameters will be used when `setting = "RSEn"` is used. To get higher efficiency either use `setting = "RSEa"` (and only set arguments `rho.e` and `rho.b`). Or specify the tuning parameters by hand using the `psi2propII` and `chgDefaults` functions.

To specify separate weight functions `rho.b` and `rho.sigma.b` for different variance components, it is possible to pass a list instead of a psi_func object. The list entries correspond to the groups as shown by `VarCorr(.)` when applied to the model fitted with `lmer`. A set of correlated random effects count as just one group.

`lmerNoFit`**:** The `lmerNoFit` function can be used to get trivial starting values. This is mainly used to verify the algorithms to reproduce the fit by `lmer` when starting from trivial initial values.

Value

object of class rlmerMod.

Prior weights

`weights` is passed through to `lmer` and has the usual meaning, $\mathrm{Var}(e_i) = \sigma^2/w_i$. **Weighted `method = "DASvar"` fits made with robustlmm 3.2-0 to 3.4-5 are wrong and should be refitted**: the Design Adaptive Scale counted the prior weights twice, so $\hat{\sigma}$, the variance components and every standard error derived from them were biased by an amount that depended on the scale the weights happened to be supplied on. The default `method = "DAStau"` was affected only through its starting value, and in practice only at `rho.e = rho.b = cPsi`. Unweighted fits are unaffected in every version. See the 3.5.0 entry of `news(package = "robustlmm")` for the magnitudes and the measured exposure of each method.

Convergence

`rlmer` warns if the fit did not converge, and `print` and `summary` then show a "Convergence code" block. The code is in `object@optinfo$conv$opt`: 0 converged; 1 the iterations stopped at `max.iter`; 2 converged, but the estimating equations are not satisfied; 3 an inner iteration of the last step did not converge. The messages are in `object@optinfo$warnings`.

Lifecycle

Stable. Experimental, and subject to change: `setting = "RSEhb"`, `init = "robust"` and `init = "ransac"`; structured covariances (`cs()`, `ar1()`) in the formula; `size_obr`; and the Monte-Carlo DAS-tau calibration (`rlmerControl`).

Author(s)

Manuel Koller, with thanks to Vanda Lourenço for improvements.

See Also

lmer, vignette("rlmer")

Examples

```
## dropping of VC
system.time(print(rlmer(Yield ~ (1|Batch), Dyestuff2, method="DASvar")))

## Not run:
  ## Default method "DAStau"
  system.time(rfm.DAStau <- rlmer(Yield ~ (1|Batch), Dyestuff))
  summary(rfm.DAStau)
  ## DASvar method (faster, less accurate)
  system.time(rfm.DASvar <- rlmer(Yield ~ (1|Batch), Dyestuff,
                                  method="DASvar"))
  ## compare the two
  compare(rfm.DAStau, rfm.DASvar)

  ## Fit variance components with higher efficiency
  ## psi2propII yields squared weights to get robust estimates
  ## this is the same as using rlmer's argument `setting = "RSEa"`
  rlmer(diameter ~ 1 + (1|plate) + (1|sample), Penicillin,
        rho.sigma.e = psi2propII(smoothPsi, k = 2.28),
        rho.sigma.b = psi2propII(smoothPsi, k = 2.28))

  ## use chgDefaults for variance components including
  ## correlation terms (regular, non squared weights suffice)
  ## this is the same as using rlmer's argument `setting = "RSEa"`
  rlmer(Reaction ~ Days + (Days|Subject), sleepstudy,
        rho.sigma.e = psi2propII(smoothPsi, k = 2.28),
        rho.b = chgDefaults(smoothPsi, k = 5.14, s=10),
        rho.sigma.b = chgDefaults(smoothPsi, k = 5.14, s=10))

## End(Not run)

## Not run:
  ## start from lmer's initial estimate, not its fit
  rlmer(Yield ~ (1|Batch), Dyestuff, init = lmerNoFit)

## End(Not run)
```

---

rlmerControl *Control of the robust linear mixed model fit*

Description

Collects the tuning parameters of `rlmer`'s fitting algorithm in one object for its argument `control`.

Usage

```
rlmerControl(
  rel.tol = 1e-08,
  max.iter = NULL,
  dastau.mc = NULL,
  dasmc.nsim = NULL,
  dasmc.seed = NULL,
  retry = FALSE,
  retry.steps = c("iterations", "DASvar", "robust"),
  retry.max.iter.factor = 5,
  lmerControl = NULL
)
```

Arguments

`rel.tol`	relative tolerance used as criterion in the fitting algorithms. Default `1e-8`.
`max.iter`	maximum number of iterations. Default `NULL`: $40(r+1)^2$ for r variance parameters.
`dastau.mc`, `dasmc.nsim`, `dasmc.seed`	settings of the experimental Monte-Carlo calibration of the DAS-tau method for this fit. Default `NULL`: use the options `robustlmm.dastau.mc`, `robustlmm.dasmc.nsim` and `robustlmm.dasmc.seed`, see `robustlmm-options`.
`retry`	fit again if the fit does not converge? Default `FALSE`.
`retry.steps`	the fallbacks to try, in this order; a subset of `c("iterations", "DASvar", "robust")`.
`retry.max.iter.factor`	factor by which `max.iter` is multiplied for the retry attempts.
`lmerControl`	a control object of `lmerControl` for the classical fit that provides the starting values, or `NULL`.

Details

The arguments `rel.tol` and `max.iter` of `rlmer` keep working. A setting may be given either there or in `control`, not in both. To change a setting of `control` with `update`, update `control` itself.

Value

A list of class `"rlmerControl"`.

Retry

With `retry = TRUE`, a fit that does not converge (see the section "Convergence" of `rlmer`) is fitted again along a chain of fallbacks, and the first converged fit is returned:

1. `"iterations"`: continue from the unconverged estimate;
2. `"DASvar"`: fit with `method = "DASvar"`, and start the requested method from that fit (skipped if the method already is `"DASvar"`);
3. `"robust"`: start from the high-breakdown start, `init = "robust"`.

Every attempt may use `retry.max.iter.factor` times as many iterations as the first fit. Which attempt succeeded is recorded in `object@optinfo$retry` and shown in the convergence block of `print` and `summary`. If no attempt converges, the first fit is returned with its warnings.

Lifecycle

Stable. Experimental, and subject to change: the retry (`retry`, `retry.steps`, `retry.max.iter.factor`) and the Monte-Carlo DAS-tau settings.

See Also

`rlmer`, `robustlmm-options`

Examples

```
rlmerControl(max.iter = 200, retry = TRUE)
## Not run:
  rlmer(Reaction ~ Days + (Days | Subject), sleepstudy,
        control = rlmerControl(retry = TRUE))

## End(Not run)
```

---

rlmerMod-class *rlmerMod Class*

Description

Class "rlmerMod" of Robustly Fitted Mixed-Effect Models

Details

A robust mixed-effects model as returned by `rlmer`.

Lifecycle

Stable, including the **emmeans** support.

Objects from the Class

Objects are created by calls to `rlmer`.

Methods

Almost all methods available from objects returned from `lmer` are also available for objects returned by `rlmer`. They usage is the same.

It follows a list of some the methods that are exported by this package:

- `coef`
- `deviance` (disabled, see below)
- `extractAIC` (disabled, see below)
- `family`
- `fitted`
- `fixef`
- `formula`
- `getInfo`
- `isGLMM`
- `isLMM`
- `isNLMM`
- `isREML`
- `logLik` (disabled, see below)
- `model.frame`
- `model.matrix`
- `nobs`
- `plot`
- `predict`
- `ranef` (only partially implemented)
- `residuals`
- `sigma`
- `summary`
- `terms`
- `update`
- `VarCorr`
- `vcov`
- `weights`

Disabled methods

A log likelihood or even a pseudo log likelihood is not defined for the robust estimates returned by `rlmer`. Methods that depend on the log likelihood are therefore not available. For this reason the methods `deviance`, `extractAIC` and `logLik` stop with an error if they are called.

Coefficient-table degrees of freedom

By default (`df = "auto"`) `summary(object)` reports a Satterthwaite-type `df` and a `Pr(>|t|)` column for the fixed effects whenever computing it is cheap – either the underlying influence function is already cached on the fit (from an earlier `cooks.distance`, `vcov` sandwich, `confint` or `summary` call), or its deterministic size workload is within the cutoff `getOption("robustlmm.summary.df.max", 5000)`. On larger fits `"auto"` falls back to the historic `Estimate / Std. Error / t value` table (no p-values, as for `lmer`) and prints a one-line note on how to request the df anyway. Use `df = "satterthwaite"` to always compute it (which may be slow on large data) or `df = "none"` to never compute it. The cutoff is a dimensionless function of the problem size (number of observations, parameters and the df method), so the same fit behaves identically on every machine; raise or lower `robustlmm.summary.df.max` to show the df on larger or only on smaller fits. See `robustlmm-options` for this option. Unlike **lmerTest**, the degrees of freedom are derived from the robust, influence-function-based covariance of the variance parameters, so they stay honest under contamination. The Satterthwaite construction follows Giesbrecht and Burns (1985) and Fai and Cornelius (1996); inserting a robust sandwich covariance into the moment-matching ratio parallels the cluster-robust approach of Bell and McCaffrey (2002) and Pustejovsky and Tipton (2018). The approximation is reliable only for a moderate number of grouping levels; `summary` prints a note recommending `confint(object, method = "boot")` when the smallest grouping factor has fewer than 20 levels. The feature supports single-factor, nested (e.g. `(1 | school/class)`) and crossed (e.g. `(1 | subject) + (1 | item)`) designs with diagonal random effects, including Mallows-weighted fits, with the default `vcov`. Nested designs use a one-way cluster sandwich over the coarsest grouping factor; crossed designs use a Cameron-Gelbach-Miller multiway cluster-robust covariance, projected to the nearest positive-semidefinite matrix. Designs not covered (e.g. crossed random slopes) fall back to t-values with an explanatory note. The same Satterthwaite df is used by **emmeans** results (`emmeans`, `emtrends`, `contrast`) for this class when the default `vcov` is in effect; a user-supplied `vcov.` keeps the asymptotic (`Inf`, z-based) degrees of freedom. When a variance component is estimated at 0 (a boundary fit), the df is computed conditional on that component being held at the boundary – it equals the df of the model with that component dropped – and `summary` notes this; only a genuinely non-identifiable (singular) fit suppresses the df.

See Also

`rlmer`; corresponding class in package `lme4`: `merMod`

Examples

```
showClass("rlmerMod")

## convert an object of type 'lmerMod' to 'rlmerMod'
## to use the methods provided by robustlmm
fm <- lmer(Yield ~ (1|Batch), Dyestuff)
rfm <- as(fm, "rlmerMod")
compare(fm, rfm)
```

`rlmerMod-tidiers` *Tidying methods for robust linear mixed models*

Description

Methods for the `tidy`, `glance` and `augment` generics of the **generics** package (re-exported by **broom** and **broom.mixed**) for models fitted by `rlmer`. The column names follow **broom.mixed**. The methods are available once one of these packages is loaded. They return a tibble if the **tibble** package is installed and a data frame otherwise.

Usage

```
tidy.rlmerMod(
  x,
  effects = c("fixed", "ran_pars"),
  conf.int = FALSE,
  conf.level = 0.95,
  vcov_type = c("default", "sandwich"),
  df = c("auto", "none", "satterthwaite"),
  cluster = NULL,
  ...
)

glance.rlmerMod(x, threshold = 0.5, ...)

augment.rlmerMod(
  x,
  data = model.frame(x),
  newdata = NULL,
  se_fit = FALSE,
  influence = FALSE,
  ...
)
```

Arguments

`x`	an `rlmerMod` object.
`effects`	which rows to return: `"fixed"` (fixed-effect coefficients) and / or `"ran_pars"` (standard deviations and correlations of the random effects, and the residual standard deviation).
`conf.int`	add confidence limits for the fixed effects (`conf.low`, `conf.high`, from `confint`)?
`conf.level`	their confidence level.
`vcov_type`	covariance of the fixed effects: `"default"` (model-based) or `"sandwich"` (cluster sandwich with the bias-reduced CR2 correction, as `confint` uses).

df	for vcov_type = "default", passed to summary: the degrees of freedom and p-values are those of the coefficient table of summary(x, df = df), that is robust Satterthwaite when available and NA otherwise.
cluster	for vcov_type = "sandwich", the clusters; see vcov_sandwich.
...	augment: passed to predict. Otherwise not used.
threshold	robustness weights below this value count as downweighted in glance.
data	the data to augment; by default the model frame. The original data can be given if it has one row per observation used in the fit, or additionally the rows dropped for missing values.
newdata	new data to predict for. If given, augment returns it with .fitted (and .se.fit) only.
se_fit	add the standard errors of the fitted values (.se.fit, from predict with interval = "confidence")?
influence	add Cook's distances (.cooksd, from cooks.distance)? This needs the influence function of the fit, which is computed once and cached.

Details

augment returns the **robustness weights** of the fit next to the data: .wt_e is the weight of each observation and .wt_b the weight of the random effect of its cluster. A weight of 1 means the observation or cluster is treated as in the classical fit; a weight well below 1 marks it as outlying and downweighted. With several grouping factors there is one column per factor, .wt_b.<factor>. If a cluster has several random effects, the smallest of their weights is reported.

Value

tidy one row per parameter, with columns effect, group, term, estimate, std.error, statistic, df, p.value and, if requested, conf.low and conf.high. Random-effect parameters are named as in **broom.mixed** (sd__(Intercept), cor__(Intercept).x, sd__Observation) and have an estimate only.

glance one row: nobs, sigma, converged (NA for an object that was not fitted), the number of levels of each grouping factor (ngrps.<factor>), and the number of observations (nobs.downweighted) and of clusters per grouping factor (ngrps.downweighted.<factor>) with a robustness weight below threshold.

augment the data with the columns .fitted, .resid (response residuals), .wt_e, .wt_b (see above), .hat (hatvalues) and, if requested, .se.fit and .cooksd.

Likelihood-based statistics

glance has no logLik, AIC, BIC or deviance columns: the robust estimator solves estimating equations that are not the derivative of a likelihood, so these quantities are not defined for an rlmerMod.

Lifecycle

Stable.

Examples

```
fm <- rlmer(Reaction ~ Days + (Days | Subject), sleepstudy)
if (requireNamespace("generics", quietly = TRUE)) {
  print(generics::tidy(fm, conf.int = TRUE))
  print(generics::glance(fm))
  ## the least trusted observations and subjects
  aug <- generics::augment(fm)
  print(aug[order(aug$.wt_e)[1:5], ])
}
```

`rlmer_ransac` *rlmer with RANSAC initial estimator*

Description

Fit `rlmer` with a RANSAC-derived initial estimator.

Usage

```
rlmer_ransac(
  formula,
  data,
  K = 200L,
  sub_frac = 0.5,
  n_starts = 1L,
  phony_threshold = 0.99,
  seed = NULL,
  max_tries = 5L,
  scale_fn = NULL,
  ...
)
```

Arguments

`formula, data`	passed to `rlmer`.
`K, sub_frac, seed`	passed to `ransac_lme4`.
`n_starts`	number of distinct RANSAC starts for the multi-start consensus (default 1 = single best start, the previous behaviour).
`phony_threshold`	a fit is treated as phony (non-interior) when its maximum $\|\hat{\rho}\|$ exceeds this (default 0.99).
`max_tries`	maximum number of RANSAC re-seeds when a redescending `rho.e` zero-weights the refinement into a rank-deficient positive-weight design (Koller and Stahel 2017, Remark 2). The fit is re-drawn and re-fitted from a fresh nonsingular start until its positive-weight fixed-effects design is full rank, up to `max_tries` (default 5); if all attempts still collapse the last fit is returned with a warning. With

	a fixed `seed`, attempt t uses `seed + t - 1`, so the first attempt is reproducible and the retries are deterministic.
`scale_fn`	passed to `ransac_lme4` to rank the candidates, and used to score the fits of the multi-start consensus (default: the centred bisquare M-scale; `robustbase::Qn` gives the ranking before version 3.5.1).
`...`	other arguments to `rlmer` (e.g. `rho.e`, `rho.b`, `method`).

Details

Convenience wrapper that calls `ransac_lme4` to obtain a starting value and then passes it to `rlmer`'s `init` argument.

With `n_starts > 1` it runs a multi-start consensus: `rlmer` is fitted from each of the `n_starts` best distinct RANSAC candidate starts, and the returned fit is the one with the lowest residual scale (by default the centred bisquare M-scale about zero, see `scale_fn`) whose random-effects covariance is interior ($|\hat{\rho}| \leq$ `phony_threshold`). This samples several basins of a redescending ψ and so recovers the interior solution when the single best start happens to fall into the phony $|\hat{\rho}| \to 1$ attractor. If every start lands phony, the best-scoring fit is returned with a warning. The per-start summary is attached as `attr(fit, "consensus")`.

The nonsingular subsample guarantees a full-rank fixed-effects design at the *start* only; a redescending ψ can zero-weight observations during the refinement and collapse the positive-weight design to rank-deficiency (Koller and Stahel 2017, Remark 2). After the fit converges its e-side robustness weights are inspected and a warning is issued if the design restricted to the positively-weighted observations is rank-deficient, suggesting a positive-weight ψ (e.g. `rho.e = smoothPsi`) or a different start.

What the start can change. A start decides which solution `rlmer` reaches only where a redescending fit has several, as with the phony $|\hat{\rho}| \to 1$ attractor above. It does not raise the breakdown point of the final fit: once the scale equations of `rlmer` break down (at a fraction of 0.295 outliers for the default settings and 0.156 for `setting = "RSEa"`), a redescending `rho.e` accepts the outliers from any start. See the section "Ranking RANSAC candidates" of `vignette("simulationStudies")`. The RANSAC start is also not high-breakdown itself under contamination (every half-sample contains outliers); `rlmer(setting = "RSEhb")` combines high-breakdown scale equations, redescending ψ-functions and the start `init = "robust"`, see `rlmer`.

Value

rlmerMod object.

Lifecycle

Experimental. The interface and the results may change in a later version.

Examples

```
fit <- rlmer_ransac(Reaction ~ Days + (Days | Subject),
                    data = sleepstudy, K = 30)
```

robustlmm-options *Global options consulted by* **robustlmm**

Description

robustlmm reads a small number of global options, set with `options()` and queried with `getOption()`. With one experimental exception (the Monte-Carlo DAS-tau calibration below), none of them change the fitted estimates; they only tune optional diagnostics and the default behaviour of `summary` (see `rlmerMod-class`).

Degrees-of-freedom options

These control whether `summary(object)` computes the robust Satterthwaite degrees of freedom and `Pr(>|t|)` column by default (`df = "auto"`); see the "Coefficient-table degrees of freedom" section of `rlmerMod-class`.

`robustlmm.summary.df.max` Numeric, default `5000`. The size cutoff for computing the Satterthwaite df under `df = "auto"`. The cost of the df is a deterministic dimension-only "workload" W – one $O(n)$ score evaluation per parameter-Jacobian column: $W = (p + q + 1 + L)n$ for method `"DASvar"` and $W = 40\,L\,n$ for `"DAStau"`, where n is the number of observations, p the number of fixed effects, q the number of random effects and L the number of variance parameters. If W exceeds this cutoff and no influence function is cached on the fit, `summary` falls back to the plain `Estimate` / `Std. Error` / `t value` table and prints a note. The rule is dimensionless, so the same fit behaves identically on every machine. The default `5000` computes the df by default up to about $n = 170$ for a single random intercept fit with `"DASvar"` ($n = 125$ for `"DAStau"`). Set it higher to show the df on larger fits, or to `0` to always skip the automatic computation (you can still request it with `summary(object, df = "satterthwaite")`).

Monte-Carlo DAS-tau calibration (EXPERIMENTAL)

For a single fit these settings can also be given in `rlmerControl`, which takes precedence over the options.

These options enable and tune an *experimental* Monte-Carlo calibration of the DAS-tau fixed point in `rlmer`. Without it, `method = "DAStau"` computes the consistency factors for non-diagonal random-effect blocks by Gauss-Hermite quadrature, which is limited to blocks of dimension ≤ 2; fits containing a larger block fall back to `method = "DASvar"` with a warning. The Monte-Carlo path lifts this restriction: it computes the same self-consistent fixed point by plain Monte-Carlo integration, which works for any block dimension, including structured (`cs`/`ar1`) and unstructured blocks of dimension > 2. It is simulation-validated – in the companion ar1 simulation study it removes the small calibration residual that the `DASvar` approximation leaves in the fitted correlation – but it is not backed by a finite-sample theorem, and it has been validated on clean Gaussian data only. For blocks of dimension ≤ 2 the classical quadrature path remains the default and the Monte-Carlo path offers no improvement there.

`robustlmm.dastau.mc` Logical, default `FALSE`. Master switch. When `TRUE`, `method = "DAStau"` uses the Monte-Carlo calibration for all non-diagonal random-effect blocks of dimension > 2 (instead of falling back to `"DASvar"` for the whole fit). The Monte-Carlo sample is drawn

once per fit (common random numbers), deterministically seeded and moment-matched to exact zero mean and identity second moment, so repeated fits are identical and the caller's `.Random.seed` is left untouched. `rlmer` emits a message when the experimental path is active.

`robustlmm.dastau.mc.all` Logical, default `FALSE`. Research switch: also use the Monte-Carlo path for blocks of dimension 2, replacing the Gauss-Hermite quadrature. Intended only for comparing the two calibration paths; it offers no improvement over the quadrature.

`robustlmm.dasmc.nsim` Integer, default `1e5`. Number of Monte-Carlo draws. Larger values reduce the (deterministic, seed-dependent) residual calibration error at linear cost in time and memory.

`robustlmm.dasmc.seed` Integer, default `20260703`. Seed for the common-random-numbers draw. Fits are deterministic given this option; change it (e.g. per replicate in a simulation) to decorrelate the residual Monte-Carlo calibration error across fits. The global RNG state is saved and restored around the draw.

Messages before long computations

Some computations can take minutes on large fits. Before starting one, **robustlmm** compares a dimension-only measure of its cost with a threshold and, above it, prints a message that names the cost and how to avoid it. Nothing is skipped and no question is asked. The cost is measured in the workload units W of `robustlmm.summary.df.max` above.

`robustlmm.message.if.max` Numeric, default `50000`. Threshold on W for computing the influence function of a fit (`implicitIF_full`), which the Satterthwaite degrees of freedom, the cluster sandwich and `cooks.distance` need. It is computed once and cached.

`robustlmm.message.boot.max` Numeric, default `1e7`. Threshold on `nsim` times the number of robust refits per replicate times W, for `anova(test = "boot")` (two refits), `anova(test = "score")` (one) and `confint(method = "boot")` (counted as 0.02, because **confintROB** refits with `lmer`).

Set an option to `Inf` to silence its message, or wrap the call in `suppressMessages`.

Developer options

`robustlmm.check_rhs_optimisation` Logical, default `FALSE`. When `TRUE`, `rlmer` cross-checks the vectorised right-hand-side computation in the block-diagonal θ update against an explicit per-block loop and stops on any discrepancy. Intended for development and debugging only; it adds redundant work and is not needed in normal use.

See Also

`rlmerMod-class`, `rlmer`

Examples

```
## show the df on larger fits (raise the size cutoff)
## Not run:
options(robustlmm.summary.df.max = 20000)

## End(Not run)
```

saveDatasets *Save datasets*

Description

Saves dataset to one or more files.

Usage

```
saveDatasets(datasets, path = getwd(), file, chunkSize)
```

Arguments

datasets	dataset list generated by one of the generate functions.
path	path to save the datasets to.
file	filename to use, without extension.
chunkSize	if provided, datasets are split into chunkSize chunks and then saved.

Details

The file will be saved to path/filename.Rdata.

If chunkSize is not missing, the filename is interpreted as format specifier and passed onto sprintf. One argument is given, the index of the chunk.

Value

filename or vector of filenames.

Author(s)

Manuel Koller

shortenLabelsKS2022 *Shorten Labels*

Description

Shorten labels created by the various fitDatasets functions, for use in plotting, etc.

Usage

```
shortenLabelsKS2022(labels)
```

Arguments

labels	vector of labels as assigned by fitDatasets

Details

The labels are shortened as they are in the simulation study published in Koller and Stahel (2022).

Value

Vector of shortened labels

Author(s)

Manuel Koller

References

Koller M, Stahel WA (2022). "Robust Estimation of General Linear Mixed Effects Models." In PM Yi, PK Nordhausen (eds.), Robust and Multivariate Statistical Methods, Springer Nature Switzerland AG.

Examples

```
labels <- c("fitDatasets_lmer", "fitDatasets_rlmer_DAStau",
            "fitDatasets_rlmer_DAStau_noAdj",
            "fitDatasets_varComprob_compositeTau_OGK",
            "fitDatasets_varComprob_S_OGK",
            "fitDatasets_heavyLme",
            "fitDatasets_lqmm")
shortenLabelsKS2022(labels)
```

---

simulate.rlmerMod *Simulate responses from a fitted robust linear mixed model*

Description

Generates new responses for the design of a model fitted by `rlmer`: parametric samples from the fitted model, or wild bootstrap samples built from its residuals. These are the samples of the parametric and wild bootstraps of **confintROB**; use them to run a bootstrap of your own.

Usage

```
## S3 method for class 'rlmerMod'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  type = c("parametric", "wild"),
  use.u = FALSE,
  cluster = NULL,
  wild.weights = c("mammen", "rademacher"),
  ...
)
```

Arguments

object	an rlmerMod object.
nsim	number of responses to simulate.
seed	NULL or an integer passed to set.seed before simulating; the previous state of the random number generator is then restored on exit.
type	"parametric" (default) or "wild", see Details.
use.u	for type = "parametric": keep the fitted random effects (TRUE) or draw new ones (FALSE, default)?
cluster	for type = "wild": the clusters, as the name of a grouping factor of the model or a vector with one value per observation. See the section on clusters.
wild.weights	for type = "wild": the two-point distribution of the cluster weights, "mammen" (default; Mammen 1993) or "rademacher" (± 1 with equal probability).
...	not used.

Details

type = "parametric" draws from the central Gaussian model at the robust estimates: random effects with the fitted covariance and errors with variance $\hat{\sigma}^2/w$ for prior weights w, added to the fitted fixed part (including any offset). The samples follow the model the robust fit describes for the bulk of the data; they do *not* reproduce contamination, so they contain no outliers. With use.u = TRUE the fitted random effects are kept and only the errors are redrawn. This leaves out the between-cluster variation and is not suitable for confidence intervals of the model parameters.

type = "wild" the cluster wild bootstrap of Modugno and Giannerini (2015). The marginal residuals $y - X\hat{\beta}$ are rescaled within each cluster for the leverage of the fixed-effects design, multiplied by one random two-point weight per cluster, and added back to the fixed part. It makes no assumption on the distribution or the covariance of the response within a cluster.

Value

A data frame with nsim columns sim_1, ..., each a simulated response for the observations used in the fit. The seed is stored in attr(, "seed"), as by simulate for lmer fits.

Clusters of the wild bootstrap

All observations of a cluster get the same random weight, so the clusters have to be independent of each other. With one grouping factor these are its levels. For nested grouping factors the default is the outermost (coarsest) factor. Crossed grouping factors have no independent clusters; cluster must then be given, and the dependence across its levels is ignored.

Lifecycle

Stable.

References

Mammen E (1993). Bootstrap and wild bootstrap for high dimensional linear models. *The Annals of Statistics* 21(1), 255–285.

Mason F, Cantoni E, Ghisletta P (2021). Parametric and semi-parametric bootstrap-based confidence intervals for robust linear mixed models. *Methodology* 17(4), 271–295.

Modugno L, Giannerini S (2015). The wild bootstrap for multilevel models. *Communications in Statistics – Theory and Methods* 44(22), 4812–4825.

See Also

`confint` with `method = "boot"`, which runs the bootstrap of **confintROB**.

Examples

```
fm <- rlmer(Reaction ~ Days + (Days | Subject), sleepstudy)
ysim <- simulate(fm, nsim = 3, seed = 1)
head(ysim)

## Not run:
  ## a wild bootstrap of the slope by hand: refit each sample.
  ## confintROB refits with lmer(), which is much faster:
  ystar <- simulate(fm, nsim = 200, seed = 2, type = "wild")
  slopes <- sapply(ystar, function(y)
      fixef(lmer(y ~ Days + (Days | Subject), sleepstudy))["Days"])
  quantile(slopes, c(0.025, 0.975))
  ## the same with robust refits
  slopes <- sapply(ystar[1:20], function(y) {
      d <- sleepstudy; d$Reaction <- y
      fixef(update(fm, data = d))["Days"]
  })

## End(Not run)
```

---

splitDatasets *Split Datasets Into Chunks*

Description

Method that splits up dataset objects into smaller chunks, so that they can be processed separately.

Usage

```
splitDatasets(datasets, chunkSize = 50)
```

Arguments

`datasets`	dataset object to split into chunks
`chunkSize`	number of datasets to keep in one chunk

Value

list of dataset lists with generators and the contents of the original dataset. See prepareMixedEffectDataset and generateAnovaDatasets for a description of the contents. There is one additional entry in the list:

chunkIndex: index of the chunk

Author(s)

Manuel Koller

See Also

bindDatasets

Examples

```
oneWay <- generateAnovaDatasets(18, 1, 5, 4)
datasetList <- splitDatasets(oneWay, 5)
data <- datasetList[[4]]$generateData(1)
stopifnot(all.equal(oneWay$generateData(16), datasetList[[4]]$generateData(1),
                    check.attributes = TRUE),
          all.equal(oneWay$sphericalRandomEffects(16),
                    datasetList[[4]]$sphericalRandomEffects(1)),
          all.equal(oneWay$createXMatrix(data), datasetList[[4]]$createXMatrix(data)),
          all.equal(oneWay$createZMatrix(data), datasetList[[4]]$createZMatrix(data)))
```

---

sScalePsi *Default S-type scale psi-function*

Description

Pre-built psi_func_rcpp from makeSScalePsi with c = 1.547645 (breakdown point 1/2), for rho.sigma.e and one-dimensional rho.sigma.b.

Lifecycle

Experimental. The interface and the results may change in a later version.

See Also

makeSScalePsi.

`vcov.rlmerMod` *Variance-covariance matrix of the fixed effects of an rlmerMod fit.*

Description

By default returns the same object as lme4's `vcov.merMod` (the linearised model-based covariance). With `type = "sandwich"`, returns the robust cluster-sandwich `vcov_sandwich`.

Usage

```
## S3 method for class 'rlmerMod'
vcov(
  object,
  type = c("default", "sandwich"),
  cluster = NULL,
  correction = c("G1", "none", "CR2", "CR3"),
  ...
)
```

Arguments

`object`	An `rlmerMod` object.
`type`	`"default"` (the lme4-inherited linearised vcov; the pre-existing behaviour) or `"sandwich"` (the robust cluster-sandwich).
`cluster`	When `type = "sandwich"`, passed to `vcov_sandwich`: `NULL` (auto-detect for a single grouping factor), a character string naming a grouping factor of the model, or a length-n vector of cluster memberships.
`correction`	When `type = "sandwich"`, `"G1"` (default, applies the $J/(J-1)$ small-sample scaling), `"none"`, `"CR2"` (bias-reduced) or `"CR3"` (approximate leave-one-cluster-out jackknife); see `vcov_sandwich`.
`...`	Additional arguments passed to the default `vcov` method (only used when `type = "default"`).

Details

The cluster sandwich is exact when every grouping factor is nested within the clusters and approximate for crossed factors (with a warning). With few clusters the default G1 sandwich is biased down for between-cluster coefficients; `correction = "CR2"` is the bias-reduced version, which `anova` and `confint` use for `vcov_type = "sandwich"` (see `vcov_sandwich`). `vcov_sandwich` warns for $J < 20$ with `"G1"` or `"none"`.

Value

A $p \times p$ covariance matrix for $\hat{\beta}$.

Lifecycle

Stable.

See Also

vcov_sandwich

`vcov_sandwich` *Robust cluster-sandwich covariance of the fixed effects of a fitted* `rlmerMod` *object.*

Description

Computes the robust score sandwich $\hat{V}_{IF} = \hat{A}^{-1}\hat{B}\hat{A}^{-T}$, where $\hat{A}$ is the Schur-complement (marginal) Jacobian of the profiled β-score and $\hat{B} = \sum_j s_j s_j^T$ sums the per-cluster β-score contributions $s_j = \sum_{i \in j} \eta_i x_i \psi_e(\hat{r}_i)$ (η_i the Mallows design weights, 1 unless `design.weights` is used). Equal to the user-facing `vcov(object, type = "sandwich")`.

Usage

```
vcov_sandwich(fit, cluster = NULL, correction = c("G1", "none", "CR2", "CR3"))
```

Arguments

`fit`	`rlmerMod` object.
`cluster`	Cluster specification; see `resolveCluster`.
`correction`	One of `"G1"` (default, applies $J/(J-1)$), `"none"`, `"CR2"` or `"CR3"`; see Details.

Details

Exact when every grouping factor is nested within the clusters (a single grouping factor, or `cluster` naming the top level of a nested design); approximate for crossed factors, with a warning.

Small-sample corrections. `"G1"` (default) scales by $J/(J-1)$. This removes the small-sample bias for the intercept but not for between-cluster coefficients, for which the sandwich stays biased down when there are few clusters. `"CR2"` is the bias-reduced sandwich of Bell and McCaffrey (2002) adapted to the robust estimating equations: with $\hat{A} = \sum_j \hat{A}_j$ split into cluster contributions, each s_j is replaced by $(I - \hat{A}_j \hat{A}^{-1})^{-1/2} s_j$. For a Gaussian fit it is unbiased under the fitted model. It takes the symmetric square root after whitening by the clusters' marginal covariances, whereas the CR2 of Pustejovsky and Tipton (2018) uses their Cholesky factors; the two coincide when these covariances are multiples of the identity and otherwise differ slightly. `"CR3"` uses $(I - \hat{A}_j \hat{A}^{-1})^{-1}$: $J/(J-1)$ times the one-step leave-one-cluster-out jackknife. CR2 and CR3 need every random effect nested within one cluster. `anova` and `confint` use CR2 with Bell–McCaffrey / approximate Hotelling degrees of freedom (Pustejovsky and Tipton 2018) for `vcov_type = "sandwich"`.

$\hat{\sigma}$ and $\hat{\theta}$ are held fixed (partial sandwich).

Value

A $p \times p$ covariance matrix for $\hat{\beta}$, with dimnames from the fixed-effect coefficient names and attribute `"n.clusters"`.

Lifecycle

Stable.

References

Bell RM, McCaffrey DF (2002). Bias reduction in standard errors for linear regression with multi-stage samples. *Survey Methodology* 28(2), 169–181.

Pustejovsky JE, Tipton E (2018). Small-sample methods for cluster-robust variance estimation and hypothesis testing in fixed effects models. *Journal of Business & Economic Statistics* 36(4), 672–683.

See Also

`vcov`, `caseweightIF`

`viewCopyOfSimulationStudy` *Access Simulation Study Code*

Description

This is a convenience function to make it simple to access the simulation study script files that are shipped with robustlmm.

Usage

```
viewCopyOfSimulationStudy(
  study = c("sensitivityCurves.R", "consistencyAndEfficiencyDiagonal.R",
    "consistencyAndEfficiencyBlockDiagonal.R", "breakdown.R", "breakdownMC.R",
    "convergence.R", "robustnessDiagonal.R", "robustnessBlockDiagonal.R",
    "robustnessBlockDiagonalExtended.R"),
  destinationPath = getwd(),
  overwrite = FALSE
)
```

Arguments

`study`	Name of the script file, partial matching is supported via `match.arg`.
`destinationPath`	optional path to directory in which the copy of the script should be created. By default the current working directory is used.
`overwrite`	logical; should existing destination files be overwritten?

Details

The function creates a copy of the script file that can be safely edited without changing the original file.

Examples

```
## Not run:
  viewCopyOfSimulationStudy("sensitivityCurves")

## End(Not run)
```

Index

∗ **classes**
  rlmerMod-class, 86
∗ **methods**
  plot-methods, 55
∗ **misc**
  robustlmm-options, 93
∗ **models**
  compare, 11
  rlmer, 78
∗ **utilities**
  chgDefaults, 11
  compare, 11
  getME, 41
  other, 54
  psi2propII, 72

anova, *15*, *17*, *82*, *100*, *101*
anova.rlmerMod, 4
arrange, *31*
asymptoticEfficiency, *82*
asymptoticEfficiency *(*asymptoticVariance*)*, 7
asymptoticVariance, 7
augment.rlmerMod *(*rlmerMod-tidiers*)*, 89

bindDatasets, 8, *18*, *41*, *99*
bisquarePsi, 9, *27*, *49*, *51–53*, *72*, *75*

caseweightIF, 10, *102*
character, *56*
chgDefaults, 11, *20*, *71*, *80*, *82*, *83*
chgDefaults,psi_func_rcpp-method *(*chgDefaults*)*, 11
coef, *4*, *87*
coef.rlmerMod *(*rlmerMod-class*)*, 86
compare, 11
confint, *7*, *17*, *82*, *88*, *89*, *98*, *100*, *101*
confint.rlmerMod, 13
confintROB, *13*, *15*
contrast_test, 15
cooks.distance, *46*, *47*, *88*, *90*
cooks.distance.rlmerMod, 17
covMcd, *81*
cPsi *(*psi-functions*)*, 71
createDatasetsFromList, 18, *32*, *38*
createRhoFunction, 19, *25*

design_breakdown, 20, *75*
detectCores, *64*
deviance, *87*
deviance.rlmerMod *(*rlmerMod-class*)*, 86
dgCMatrix, *42*

extractAIC, *87*
extractAIC.rlmerMod *(*rlmerMod-class*)*, 86
extractPredefinedTuningParameter *(*extractTuningParameter*)*, 22
extractTuningParameter, *20*, 22, *25*, *26*

family, *87*
family.rlmerMod *(*rlmerMod-class*)*, 86
findTuningParameter *(*asymptoticVariance*)*, 7
fitDatasets, *18*, *19*, *41*, *64*, *65*, *95*
fitDatasets *(*fitDatasets_lmer*)*, 23
fitDatasets_heavyLme *(*fitDatasets_lmer*)*, 23
fitDatasets_lmer, 23
fitDatasets_lmer_bobyqa *(*fitDatasets_lmer*)*, 23
fitDatasets_lmer_Nelder_Mead *(*fitDatasets_lmer*)*, 23
fitDatasets_lqmm *(*fitDatasets_lmer*)*, 23
fitDatasets_rlme *(*fitDatasets_lmer*)*, 23
fitDatasets_rlmer *(*fitDatasets_lmer*)*, 23
fitDatasets_rlmer_ar1 *(*fitDatasets_lmer*)*, 23
fitDatasets_rlmer_cs *(*fitDatasets_lmer*)*, 23

fitDatasets_rlmer_DAStau
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_bisq
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_0_5
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_0_5_noAdj
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_2
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_2_noAdj
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_5
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_k_5_noAdj
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_lmerNoFit
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_noAdj
(fitDatasets_lmer), 23
fitDatasets_rlmer_DAStau_sizeOBR
(fitDatasets_lmer), 23
fitDatasets_rlmer_DASvar
(fitDatasets_lmer), 23
fitDatasets_rlmer_ransac
(fitDatasets_lmer), 23
fitDatasets_rlmer_ransac_bisq
(fitDatasets_lmer), 23
fitDatasets_varComprob
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeS
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeS_2SGS
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeS_OGK
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeTau
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeTau_2SGS
(fitDatasets_lmer), 23
fitDatasets_varComprob_compositeTau_OGK
(fitDatasets_lmer), 23
fitDatasets_varComprob_S
(fitDatasets_lmer), 23
fitDatasets_varComprob_S_2SGS
(fitDatasets_lmer), 23
fitDatasets_varComprob_S_OGK
(fitDatasets_lmer), 23
fitted, *87*
fitted.rlmerMod *(*rlmerMod-class*)*, 86
fixef, *42*, *44*, *45*, *87*
fixef.rlmerMod *(*rlmerMod-class*)*, 86
formula, *87*
formula.rlmerMod *(*rlmerMod-class*)*, 86
fragility, 29

generateAnovaDatasets, *9*, *18*, *19*, 30, *36*, *38*, *41*, *99*
generateLongitudinalDatasets, 33, *40*, *59*
generateMixedEffectDatasets, *18*, *19*, *32*, *36*, 37, *39*, *40*, *62*
generateRepeatedMeasuresDatasets, 38
generateSensitivityCurveDatasets, 40
geom_hline, *57*
geom_qq_line, *57*
getCall, *45*
getInfo, *87*
getInfo *(*compare*)*, 11
getME, 41, *58*, *62*
getOption, *93*
ggplot, *58*
glance.rlmerMod *(*rlmerMod-tidiers*)*, 89
glmer, *43*
glmerControl, *43*

hatvalues, *18*, *90*
hatvalues.rlmerMod, 45
huberPsiRcpp *(*psi-functions*)*, 71

implicitIF, *10*, *46*, *47*
implicitIF_full, *17*, *18*, 46, *47*, *94*
influence, *18*, *46*, *47*
influence.rlmerMod, 47
isGLMM, *87*
isGLMM.rlmerMod *(*rlmerMod-class*)*, 86
isLMM, *87*
isLMM.rlmerMod *(*rlmerMod-class*)*, 86
isNLMM, *87*
isNLMM.rlmerMod *(*rlmerMod-class*)*, 86
isREML, *87*
isREML.rlmerMod *(*rlmerMod-class*)*, 86

lapplyDatasets, *28*, 48
legend, *56*
list, *44*
lme4, *3*, *4*
lmer, *3*, *26*, *45*, *62*, *69*, *70*, *74*, *75*, *78*, *79*, *81*, *83*, *84*, *87*, *88*

lmerControl, *85*
lmerNoFit, *26*
lmerNoFit *(*rlmer*)*, 78
lmrob, *50, 51*, *80*
lmrob.control, *49*, *51, 52*, *75*
loadAndMergePartialResults, 49, *64*
logLik, *87*
logLik.rlmerMod *(*rlmerMod-class*)*, 86
lqmm, *27*
lqqPsi, *9, 10*, 49, *50–52*, *72, 73*, *75*

makeBisquarePsi, *9*, 50
makeCluster, *64*
makeRobustbasePsi, *10*, *49, 50*, 51
makeSScalePsi, 52, *99*
match.arg, *102*
matplot, *56*
Mchi, *50, 51*
mergeProcessedFits, *49*, 53
merMod, *41*, *88*
model.frame, *87*
model.frame.rlmerMod *(*rlmerMod-class*)*, 86
model.matrix, *87*
model.matrix.rlmerMod *(*rlmerMod-class*)*, 86
Mpsi, *50–52*
MrhoInf, *51*
Mwgt, *50, 51*

name, *45*
nobs, *87*
nobs.rlmerMod *(*rlmerMod-class*)*, 86
numDeriv::jacobian, *46*

options, *93*
other, 54

partialMoment_standardNormal, 55
plot, *55*, *87*
plot,Rcpp_HuberPsi-method *(*plot-methods*)*, 55
plot,Rcpp_PsiFunction-method *(*plot-methods*)*, 55
plot,Rcpp_PsiFunctionToPropIIPsiFunctionWrapper-method *(*plot-methods*)*, 55
plot,Rcpp_SmoothPsi-method *(*plot-methods*)*, 55
plot-methods, 55
plot.rlmerMod, 57
plotLongitudinalBySubject, 58
predict, *87*, *90*
predict.merMod, *61*
predict.rlmerMod, 60
prepareMixedEffectDataset, *9*, *38*, 62, *99*
print.rlmerMod *(*rlmerMod-class*)*, 86
print.rlmerMod_plots *(*plot.rlmerMod*)*, 57
print.summary.rlmer *(*rlmerMod-class*)*, 86
print.VarCorr.rlmerMod *(*rlmerMod-class*)*, 86
print.xtable, *13*
print.xtable.comparison.table *(*compare*)*, 11
proc.time, *70*
processDatasetsInParallel, *49*, 63
processFile, *64*, 65
processFit, *18, 19*, *41*, *53, 54*, *64, 65*, 66
psi-functions, 71
psi2propII, *53*, *71*, 72, *79, 80*, *82, 83*
psi2propII,psi_func_rcpp-method *(*psi2propII*)*, 72
psi2propII,Rcpp_SmoothPsi *(*psi2propII*)*, 72
PsiFunction, *20*
PsiFunction *(*psi-functions*)*, 71

Qn, *74*

ranef, *44, 45*, *87*
ranef.rlmerMod *(*rlmerMod-class*)*, 86
ransac_basin_radius, 72
ransac_lme4, *21*, *25*, *27*, 74, *80*, *91, 92*
refit, *76*
refit.rlmerMod, 76
resid, *4*
resid.rlmerMod *(*rlmerMod-class*)*, 86
residuals, *87*
residuals.rlmerMod, 77
resolveCluster, *101*
rlmer, *3, 4*, *15*, *25–27*, *42*, *44*, *46*, *49–53*, *59*, *69–71*, *75, 76*, 78, *80*, *85–89*, *91–94*, *96*
rlmer_ransac, *21*, *49*, *73*, *75, 76*, *80*, 91
rlmerControl, *81*, *83*, 84, *93*
rlmerMod, *41, 42*
rlmerMod-class, 86
rlmerMod-tidiers, 89
robustlmm *(*robustlmm-package*)*, 3

robustlmm-options, 93
robustlmm-package, 3

saveDatasets, *64, 65*, 95
set.seed, *97*
shortenLabelsKS2022, 95
show *(*other*)*, 54
show,Rcpp_HuberPsi-method *(*other*)*, 54
show,Rcpp_PsiFunction-method *(*other*)*, 54
show,Rcpp_PsiFunctionToPropIIPsiFunctionWrapper-method *(*other*)*, 54
show,Rcpp_SmoothPsi-method *(*other*)*, 54
show,rlmerMod-method *(*rlmerMod-class*)*, 86
show.rlmerMod *(*rlmerMod-class*)*, 86
show.summary.rlmerMod *(*rlmerMod-class*)*, 86
sigma, *87*
sigma.rlmerMod *(*rlmerMod-class*)*, 86
simulate, *15*, *76*, *97*
simulate.rlmerMod, 96
SmoothPsi *(*psi-functions*)*, 71
smoothPsi, *26, 27*, *79*
smoothPsi *(*psi-functions*)*, 71
splitDatasets, *9*, *18*, *41*, 98
sprintf, *95*
sScalePsi, *82*, 99
summary, *4*, *87*, *90*
summary.rlmerMod *(*rlmerMod-class*)*, 86
summary.summary.rlmerMod *(*rlmerMod-class*)*, 86
suppressMessages, *94*

terms, *87*
terms.rlmerMod *(*rlmerMod-class*)*, 86
theta *(*getME*)*, 41
tidy.rlmerMod *(*rlmerMod-tidiers*)*, 89
title, *56*

uniroot, *8*
update, *87*
update.rlmerMod *(*rlmerMod-class*)*, 86

VarCorr, *87*
VarCorr.rlmerMod *(*rlmerMod-class*)*, 86
VarCorr.summary.rlmerMod *(*rlmerMod-class*)*, 86
vcov, *5*, *7*, *10*, *15*, *44, 45*, *61*, *87*, *102*
vcov.rlmerMod, 100
vcov.summary.rlmerMod *(*rlmerMod-class*)*, 86
vcov_sandwich, *5, 6*, *10*, *14–17*, *90*, *100, 101*, 101
viewCopyOfSimulationStudy, 102

weights, *87*
weights.rlmerMod *(*rlmerMod-class*)*, 86

xtable, *12, 13*
xtable.comparison.table *(*compare*)*, 11