Package ‘robustlmm’

July 30, 2026

Type Package

Title Robust Linear Mixed Effects Models

Version 3.5.0-2

Date 2026-07-30

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 (>= 3.5.0)

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

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' 'AllClass.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' 'bootstrapWald.R' 'anova.R' 'fitDatasets.R' 'processFit.R' 'processFile.R' 'simulationStudies.R' 'asymptoticEfficiency.R' 'emmeans.R' 'options.R'

LinkingTo Rcpp, robustbase, Matrix

Encoding UTF-8

RcppModules psi_function_module

Config/build/clean-inst-doc false

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-07-30 15:00:02 UTC

Contents

robustlmm-package . . . 3
anova.rlmerMod . . . 4
asymptoticVariance . . . 7
bindDatasets . . . 8
bisquarePsi . . . 9
caseweightIF . . . 10
chgDefaults . . . 10
compare . . . 11
confint.rlmerMod . . . 13
cooks.distance.rlmerMod . . . 15
createDatasetsFromList . . . 16
createRhoFunction . . . 17
extractTuningParameter . . . 18
fitDatasets_lmer . . . 19
generateAnovaDatasets . . . 25
generateLongitudinalDatasets . . . 28
generateMixedEffectDatasets . . . 32
generateRepeatedMeasuresDatasets . . . 34
generateSensitivityCurveDatasets . . . 35
getME . . . 37
hatvalues.rlmerMod . . . 41
implicitIF_full . . . 42
influence.rlmerMod . . . 43
lapplyDatasets . . . 43
loadAndMergePartialResults . . . 44
lqqPsi . . . 45
makeBisquarePsi . . . 45
makeRobustbasePsi . . . 46
mergeProcessedFits . . . 48
other . . . 48
partialMoment_standardNormal . . . 49
plot-methods . . . 50
plot.rlmerMod . . . 51
plotLongitudinalBySubject . . . 52

predict.rlmerMod 54
prepareMixedEffectDataset 56
processDatasetsInParallel 57
processFile 59
processFit 60
psi-functions 65
psi2propII 66
ransac_basin_radius 66
ransac_lme4 67
residuals.rlmerMod 69
rlmer 70
rlmerMod-class 75
rlmer_ransac 77
robustlmm-options 79
saveDatasets 80
shortenLabelsKS2022 81
splitDatasets 82
vcov.rlmerMod 83
vcov_sandwich 84
viewCopyOfSimulationStudy 85

Index **86**

`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)` returns a per-term robust Wald chi-square table from `vcov(fit, type = vcov_type)`. Each term is tested marginally as $T_t = \hat{\beta}_t^\top \hat{V}_t^{-1} \hat{\beta}_t \sim \chi^2_{k_t}$ under $H_0 : \beta_t = 0$.

Nested fits differing only in fixed effects (default `test = "Wald"`) robust Wald restriction test on the extra coefficients. The same V used by the single-fit table is used here; `vcov_type = "sandwich"` carries through.

Nested fits differing in random-effects structure the Wald and score asymptotics are invalid (boundary problem; see Self-Liang 1987 and Koller 2026 paper 2). `test = "Wald"` warns and switches to the parametric bootstrap automatically; `test = "boot"` is the default valid path. The bootstrap simulates `nsim` datasets from the fitted central LMM at `fit0`'s estimates, refits both rlmer models per replicate, and uses the quasi-deviance difference $D = 2\sum[\rho_e(r_i^{(0)}/\hat{\sigma}_0) - \rho_e(r_i^{(1)}/\hat{\sigma}_1)]$ as the discrepancy statistic (Heritier and Ronchetti 1994; Cantoni and Ronchetti 2001; Heritier, Cantoni, Copt and Victoria-Feser 2009 sec. 5). For the special case of a *single added variance component* the experimental `test = "score"` offers a contamination-robust alternative (see Details).

Usage

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

Arguments

`object`	An `rlmerMod` object.
`...`	A second `rlmerMod` object for pairwise comparison.
`test`	One of `"Wald"` (default; closed form for nested fixed-effects-only tests), `"boot"` (parametric bootstrap quasi-deviance; the default valid path for variance-component tests), or `"score"` (experimental one-sided robust score test for a single added variance component, calibrated by a score-only parametric bootstrap; see Details for its scope and validation record).
`null`	Bootstrap null generation for `test = "boot"`. `"parametric"` (default) generates from `fit0`'s robust estimates (the exact current behaviour). `"robust"` is an experimental contamination-robust null that generates from a cleaned null fit (see Details); it only affects the variance-component bootstrap path and is ignored for the Wald paths.
`vcov_type`	Forwarded to `vcov` for the Wald paths; ignored for the bootstrap.
`ddf`	Denominator degrees of freedom for the Wald paths. `"none"` (default) reports the chi-square table as before; `"satterthwaite"` reports an F-test with a Satterthwaite denominator df (the multivariate `lmerTest::contestMD` generalisation, built on the robust IF-based covariance of the variance parameters). It requires `vcov_type = "default"` and a single grouping factor; otherwise it warns and falls back to the chi-square table.
`nsim`	Bootstrap replicates when `test = "boot"` or `test = "score"`; default 1000 for `"boot"` and 199 for `"score"` (whose null-only refits are ~4x cheaper).
`seed`	Optional RNG seed for reproducibility of the bootstrap.
`verbose`	Bootstrap progress messages.

Details

Only pairwise comparison is implemented; chains `anova(fit0,fit1, fit2, ...)` are not yet supported.

Small-J caveat for the sandwich. `vcov_type = "sandwich"` is markedly anti-conservative at small J: in a simulation study the pairwise Wald Type-I rate reached 0.15-0.20 (vs. nominal 0.05) at $J = 8$ and was still elevated at $J = 18$, returning to nominal only by $J \gtrsim 50$. `vcov_sandwich` emits a warning for $J < 20$; prefer `vcov_type = "default"` for hypothesis tests at small J.

Subject-contamination caveat for the bootstrap (experimental). The parametric-bootstrap quasi-deviance path (`test = "boot"`) is experimental — it is currently the only exposed variance-component path. It calibrates correctly under clean Gaussian and heavy-tailed errors but is markedly anti-conservative when a subset of subjects is contaminated. In a simulation study with 10% of subjects shifted by 5 standard errors (`shift_subj`), Type-I climbed to 0.13-0.24 (2.5-5x nominal) across $J \in \{8, 18, 50\}$. The mechanism: the bootstrap simulates from the fitted central LMM at `fit0`'s estimates, so any bias `fit0`'s $\hat{\sigma}_0, \hat{\theta}_0$ picked up from the contaminated subjects propagates into a too-narrow bootstrap null. As an automatic guard, `anova(test = "boot")` now inspects the null fit's own random-effects robustness weights and **warns** when a group is heavily downweighted (smallest weight below 0.5) — a direct signal that the fit treated that group as an outlier, so the bootstrap null may be poisoned. (The robust fit absorbs a contaminated group into a downweighted random effect, which is why `cooks.distance` — an *influence* measure — does *not* reliably flag it here: the

downweighting that removes the bias also removes the influence.) When the warning fires, treat the p-value with caution.

Experimental robust null (`null = "robust"`). As a mitigation for the subject-contamination anti-conservativeness above, the bootstrap path accepts `null = "robust"`: instead of generating the parametric bootstrap from `fit0`'s raw estimates, it generates from a *contamination-cleaned* null fit. Clusters the robust fit heavily downweights (smallest per-cluster random-effect weight below a tuned threshold, the same signal the automatic guard uses) are flagged, `fit0` is refitted with those clusters removed, and the bootstrap is generated from the resulting de-biased $(\hat{\sigma}, \hat{\theta}, \hat{\beta})$ (with the generator's $U_b = \Lambda(\theta)$ rebuilt at the full cleaned $\hat{\theta}$ vector). The observed discrepancy D is still taken from the original untrimmed fits, so the test uses all the data; only the bootstrap null generation is cleaned. If no cluster is flagged, more than half the clusters are flagged, trimming would leave the design rank-deficient (a dropped contrast level, a collapsed grouping factor, or a constant random-slope covariate — checked by the same nonsingular-subsampling test used by the RANSAC initial estimator), or the trimmed refit fails, the call falls back to the plain parametric null (so on clean data `null = "robust"` closely matches `null = "parametric"`); the table heading states which case applied. In simulation this reduces the contamination-induced Type-I inflation at larger J while preserving power, at the cost of mild conservatism on clean data. It is **experimental**, validated by simulation rather than a finite-sample theorem; the table heading notes when the robust null was applied.

Experimental robust score test (`test = "score"`). For the common special case where the alternative adds exactly *one independent scalar variance component* relative to the null — e.g. `(1|g)` vs `(1|g) + (0 + x|g)`, or diagonal structures adding one component — `test = "score"` runs a one-sided robust score test computed from the robust *null* fit only and calibrated by a score-only parametric bootstrap that refits just the null model per replicate (roughly 4x cheaper than `test = "boot"`'s double refits; hence the smaller default `nsim = 199`). Per cluster j, the whitened marginal residuals $\tilde{r}_j = V_j^{-1/2}(y_j - X_j\hat{\beta}_0)$ and the whitened tested direction $v_j = V_j^{-1/2} z_j$ (with V_j the null fit's marginal covariance and z_j the added term's design column) give the bounded contribution $s_j = (v_j^\top \psi(\tilde{r}_j))^2 - \kappa_1 \|v_j\|^2$ (ψ the fit's `rho.e` psi-function, $\kappa_1 = E[\psi(Z)^2]$); the statistic is the self-normalised cluster sum $S = \sum_j s_j / \sqrt{\sum_j (s_j - \bar{s})^2}$, and the one-sided p-value is $(1 + \#\{S^* \geq S\})/(n_{\text{eff}} + 1)$. Because each s_j is psi-bounded, whole-cluster contamination shifts the statistic and its bootstrap reference law together instead of inflating the test: in simulation (Gaussian balanced designs, one scalar tested component; not a theorem) contaminated-null Type-I was 0.035 vs 0.115 for `test = "boot"` with 10% of clusters shifted at $J = 50$, clean-null Type-I 0.045, and power 0.920 vs 0.900 — with no null cleaning needed (raw vs cleaned generating parameters changed at most 1-2 decisions in 600+ paired replicates, so `null = "robust"` is ignored and no downweighted-group warning is issued for this path). Across an adversarial sweep the pattern held: $J = 18$ clean/contaminated 0.055/0.015, uncentered-x designs clean/contaminated 0.045/0.075, 5% single-observation outliers 0.060. *Scope*: both fits on the same data with identical fixed effects and a single shared grouping factor, and the alternative adds exactly one uncorrelated scalar component (`length(theta)` differs by 1); anything else — multi-component or correlated-slope alternatives, crossed or nested factors — stops with an error, use `test = "boot"` there. *Caveat*: contamination aligned with the tested direction is indistinguishable from the alternative for any test with power; inspect the attached per-cluster contributions (`attr(., "boot")$s_j`; the largest values identify the clusters driving the statistic) and cluster diagnostics (`cooks.distance(fit, groups = )`, `hatvalues`) when a rejection is suspect. **Experimental**: simulation-validated, not proven; the deviance bootstrap `test = "boot"` remains the default variance-component path.

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.

References

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 genrators 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 and the only path implemented in this package. Per-coefficient closed-form CI $\hat{\beta}_k \pm z_{1-\alpha/2} \cdot SE_k$ with V taken from `vcov(object, type = vcov_type)` – so the `"sandwich"` option carries through to the interval.

`method = "boot"` **or** `"BCa"` a thin dispatch to `confintROB` (Mason, Cantoni & Ghisletta 2021, 2024) with `boot.type` forwarded. The wrapper subsets the returned matrix to the fixed-effect rows so the shape matches the `"Wald"` path; variance-component CIs from `confintROB` are dropped here. `vcov_type` is not honoured on these paths (`confintROB` uses its own internal covariance).

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("none", "satterthwaite"),
  boot.type = c("wild", "parametric"),
  nsim = 1000L,
  seed = 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 to use for V when method = "Wald": "default" (the linearised lme4 vcov; pre-existing behaviour) or "sandwich" (the robust cluster-sandwich vcov_sandwich; exact for a single nested grouping factor, approximate for crossed designs). Ignored when method = "boot" or "BCa".
df	Critical-value degrees of freedom for method = "Wald". "none" (default) uses the normal quantile $z_{1-\alpha/2}$ as before; "satterthwaite" uses a per-coefficient Satterthwaite t-quantile (the robust IF-based df), matching summary(object, df = "satterthwaite"). It requires vcov_type = "default" and a single grouping factor; otherwise it warns and falls back to the normal quantile.
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.
...	Additional arguments forwarded to confintROB (e.g. clusterID for the wild bootstrap).

Details

Guidance (Koller 2014; Mason et al. 2024). The chi-sq-p Wald limit is adequate for $J \gtrsim 20$ groups; the bootstrap earns its (substantial) cost mainly at smaller J. boot.type = "wild" (the default, following **confintROB**) is robust to misspecification of the response covariance, while "parametric" is exact under the fitted central LMM. method = "BCa" adds the bias-correction-and-acceleration adjustment to the bootstrap percentile (preferred when the bootstrap distribution is skewed).

Small-J caveat for the sandwich. vcov_type = "sandwich" under-covers at $J < 20$ (in a simulation study: coverage ~0.89 at $J = 8$ vs. nominal 0.95); the sandwich path emits a warning. For Wald CIs at small J prefer vcov_type = "default" or use method = "boot" / "BCa".

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.

References

Mason F, Cantoni E, Ghisletta P (2021). *Parametric and bootstrap-based inference for linear mixed-effects models in the presence of outliers*. Methodology 17(4): 271–293.

Mason F, Cantoni E, Ghisletta P (2024). *Bootstrap confidence intervals for fixed effects in mixed-effects models with outliers*. Psychological Methods.

See Also

vcov, confintROB

---

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 $V = (1/n)WW^T$. 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 specifiying 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")
```

`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 specifiying 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)
```

---

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 decending 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: $\mathrm{Cor}(i, j) = \texttt{correlation}^{|i-j|}$ (`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))
```

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`.

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 reproduciblility, 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 reproduciblility, 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 contant, 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 basin (support-preservation) radius*

Description

Support-preservation (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`	the rejection point c of the redescender. If `NULL` and `rho` is supplied, c is the rejection point of `rho` found numerically; if both are `NULL`, the bisquare default `4.685`.
`rho`	an optional redescending `psi_func_rcpp` (e.g. `bisquarePsi`, `lqqPsi`) whose rejection point supplies c when `cc` is `NULL`.

Details

Computes the radius $r^\star(c) = c\,\sigma/(2\max_j \|x_j\|)$ of the ball around the initial fixed-effects estimate within which every observation keeps its redescender-support membership, so the population Hessian stays positive definite (Koller and Stahel; the RANSAC-RSE basin theorem). Here c is the *rejection point* of the redescending ψ — the smallest $x > 0$ with $\psi(x) = 0$. For the bisquare this is the tuning cutoff (default `4.685`); the geometry generalises to any finite-rejection-point redescender (e.g. `lqqPsi`) by finding its rejection point numerically. A redescending ψ is safe to engage from a start only if the eventual estimate stays within this radius of the (high-breakdown) start — otherwise the iteration may leave the basin and converge to a phony solution.

Value

the basin radius $r^\star(c)$ in the units of β (a scalar), with attribute `"max_xnorm"`.

`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 = robustbase::Qn,
  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. Default Qn.
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 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.

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
```

`residuals.rlmerMod` *Get residuals*

Description

The per-observation residuals are returned, i.e., the difference of the observation and the fitted value including random effects. With type one can specify whether the weights should be used or not.

Usage

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

Arguments

`object`	rlmerMod object
`type`	type of residuals
`scaled`	scale residuals by residual standard deviation (=scale parameter)?
`...`	ignored

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
)

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) or `"RSEa"`. Both 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 `"ransac"`, or a function producing an lmerMod object.

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")`).

`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")`).

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.

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)
```

`rlmerMod-class` *rlmerMod Class*

Description

Class "rlmerMod" of Robustly Fitted Mixed-Effect Models

Details

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

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)
```

`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,
  ...
)
```

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.

`...` 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 lowest-residual-scale one 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.

Value

rlmerMod object.

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)

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.

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)
```

---

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)))
```

`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"),
  ...
)
```

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) or `"none"`.
`...`	Additional arguments passed to the default `vcov` method (only used when `type = "default"`).

Details

The cluster sandwich is exact for a single (nested) grouping factor; for crossed factors it is approximate (a warning is issued, see vcov_sandwich).

Small-J caveat for the sandwich. The G1 correction is necessary but not sufficient at very small J: in a simulation study sandwich CI coverage drops to ~0.89 at $J = 8$ (vs. nominal 0.95). vcov_sandwich emits a warning for $J < 20$; for inference at small J prefer type = "default" or pair the sandwich CI with a bootstrap calibration (e.g. confintROB).

Value

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

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 $s_j = \sum_{i \in j} x_i \psi_e(\hat{r}_i)$. Equal to the user-facing vcov(object, type = "sandwich").

Usage

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

Arguments

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

Details

Exact for a single (nested) grouping factor; approximate for crossed factors (a warning is issued via resolveCluster). With few clusters, set correction = "G1" (default) for the $J/(J-1)$ small-sample scaling.

Small-J caveat. The G1 correction is necessary but not sufficient at very small J: in a simulation study CI coverage drops to ~0.89 at $J = 8$ (vs. nominal 0.95), and Wald-style hypothesis tests using the sandwich are anti-conservative (Type-I ~3-4x nominal). The function emits a warning for $J < 20$. For inference at small J prefer vcov_type = "default" or pair the sandwich CI with a bootstrap calibration (e.g. confintROB); the sandwich is most useful at $J \gtrsim 50$.

$\hat{\sigma}, \hat{\theta}$ are held fixed (partial sandwich); the returned variance is the leading-order fixed-effects covariance.

Value

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

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, 75
∗ **methods**
plot-methods, 50
∗ **misc**
robustlmm-options, 79
∗ **models**
compare, 11
rlmer, 70
∗ **utilities**
chgDefaults, 10
compare, 11
getME, 37
other, 48
psi2propII, 66

anova, *15*
anova.rlmerMod, 4
arrange, *26*
asymptoticEfficiency, *73*
asymptoticEfficiency (asymptoticVariance), 7
asymptoticVariance, 7

bindDatasets, 8, *16*, *36*, *82*
bisquarePsi, 9, *23, 24*, *45*, *47*, *66*, *68*

caseweightIF, 10, *85*
character, *50*
chgDefaults, 10, *18*, *65*, *71*, *73, 74*
chgDefaults,psi_func_rcpp-method (chgDefaults), 10
coef, *3*, *75*
coef.rlmerMod (rlmerMod-class), 75
compare, 11
confint, *7*, *76, 77*
confint.rlmerMod, 13
confintROB, *13*, *15*
cooks.distance, *5*, *41–43*, *76*
cooks.distance.rlmerMod, 15
covMcd, *72*
cPsi (psi-functions), 65
createDatasetsFromList, 16, *27*, *33*
createRhoFunction, 17, *22*

detectCores, *58*
deviance, *75*
deviance.rlmerMod (rlmerMod-class), 75
dgCMatrix, *38*

extractAIC, *75*
extractAIC.rlmerMod (rlmerMod-class), 75
extractPredefinedTuningParameter (extractTuningParameter), 18
extractTuningParameter, *17*, 18, *22*

family, *76*
family.rlmerMod (rlmerMod-class), 75
findTuningParameter (asymptoticVariance), 7
fitDatasets, *16*, *36*, *58, 59*, *81*
fitDatasets (fitDatasets_lmer), 19
fitDatasets_heavyLme (fitDatasets_lmer), 19
fitDatasets_lmer, 19
fitDatasets_lmer_bobyqa (fitDatasets_lmer), 19
fitDatasets_lmer_Nelder_Mead (fitDatasets_lmer), 19
fitDatasets_lqmm (fitDatasets_lmer), 19
fitDatasets_rlme (fitDatasets_lmer), 19
fitDatasets_rlmer (fitDatasets_lmer), 19
fitDatasets_rlmer_ar1 (fitDatasets_lmer), 19
fitDatasets_rlmer_cs (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_bisq (fitDatasets_lmer), 19

fitDatasets_rlmer_DAStau_k_0_5 (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_k_0_5_noAdj (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_k_2 (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_k_2_noAdj (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_k_5 (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_k_5_noAdj (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_lmerNoFit (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_noAdj (fitDatasets_lmer), 19
fitDatasets_rlmer_DAStau_sizeOBR (fitDatasets_lmer), 19
fitDatasets_rlmer_DASvar (fitDatasets_lmer), 19
fitDatasets_rlmer_ransac (fitDatasets_lmer), 19
fitDatasets_rlmer_ransac_bisq (fitDatasets_lmer), 19
fitDatasets_varComprob (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeS (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeS_2SGS (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeS_OGK (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeTau (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeTau_2SGS (fitDatasets_lmer), 19
fitDatasets_varComprob_compositeTau_OGK (fitDatasets_lmer), 19
fitDatasets_varComprob_S (fitDatasets_lmer), 19
fitDatasets_varComprob_S_2SGS (fitDatasets_lmer), 19
fitDatasets_varComprob_S_OGK (fitDatasets_lmer), 19
fitted, *76*
fitted.rlmerMod (rlmerMod-class), 75
fixef, *38*, *40*, *76*
fixef.rlmerMod (rlmerMod-class), 75
formula, *76*
formula.rlmerMod (rlmerMod-class), 75

generateAnovaDatasets, *9*, *16*, 25, *31*, *33*, *36*, *82*
generateLongitudinalDatasets, 28, *35*, *54*
generateMixedEffectDatasets, *16*, *27*, *31*, 32, *34*, *35*, *56*
generateRepeatedMeasuresDatasets, 34
generateSensitivityCurveDatasets, 35
geom_hline, *52*
geom_qq_line, *52*
getCall, *40*
getInfo, *76*
getInfo (compare), 11
getME, 37, *52*, *56*, *57*
getOption, *79*
ggplot, *52*
glmer, *39*
glmerControl, *39*

hatvalues, *6*, *15*
hatvalues.rlmerMod, 41
huberPsiRcpp (psi-functions), 65

implicitIF, *10*, *42*
implicitIF_full, *15*, 42, *43*
influence, *15*, *42*, *43*
influence.rlmerMod, 43
isGLMM, *76*
isGLMM.rlmerMod (rlmerMod-class), 75
isLMM, *76*
isLMM.rlmerMod (rlmerMod-class), 75
isNLMM, *76*
isNLMM.rlmerMod (rlmerMod-class), 75
isREML, *76*
isREML.rlmerMod (rlmerMod-class), 75

lapplyDatasets, *24*, 43
legend, *50*
list, *40*
lme4, *3*
lmer, *3*, *22*, *41*, *56*, *63*, *64*, *67*, *68*, *71*, *72*, *74–76*
lmerNoFit, *22*
lmerNoFit (rlmer), 70
lmrob, *46*, *47*
lmrob.control, *45*, *47*, *68*
loadAndMergePartialResults, 44, *58*

logLik, *76*
logLik.rlmerMod (rlmerMod-class), 75
lqmm, *24*
lqqPsi, *9*, 45, *46, 47*, *66–68*

makeBisquarePsi, *9*, 45
makeCluster, *58*
makeRobustbasePsi, *9*, *45, 46*, 46
match.arg, *85*
matplot, *50*
Mchi, *46, 47*
mergeProcessedFits, *44*, 48
merMod, *37*, *77*
model.frame, *76*
model.frame.rlmerMod (rlmerMod-class), 75
model.matrix, *76*
model.matrix.rlmerMod (rlmerMod-class), 75
Mpsi, *46, 47*
MrhoInf, *47*
Mwgt, *46, 47*

name, *40*
nobs, *76*
nobs.rlmerMod (rlmerMod-class), 75
numDeriv::jacobian, *42*

options, *79*
other, 48

partialMoment_standardNormal, 49
plot, *50*, *76*
plot,Rcpp_HuberPsi-method (plot-methods), 50
plot,Rcpp_PsiFunction-method (plot-methods), 50
plot,Rcpp_PsiFunctionToPropIIPsiFunctionWrapper-method (plot-methods), 50
plot,Rcpp_SmoothPsi-method (plot-methods), 50
plot-methods, 50
plot.rlmerMod, 51
plotLongitudinalBySubject, 52
predict, *76*
predict.merMod, *55*
predict.rlmerMod, 54
prepareMixedEffectDataset, *9*, *33*, 56, *82*
print.rlmerMod (rlmerMod-class), 75
print.rlmerMod_plots (plot.rlmerMod), 51
print.summary.rlmer (rlmerMod-class), 75
print.VarCorr.rlmerMod (rlmerMod-class), 75
print.xtable, *12*
print.xtable.comparison.table (compare), 11
proc.time, *64*
processDatasetsInParallel, *45*, 57
processFile, *58*, 59
processFit, *16*, *36*, *48*, *58, 59*, 60
psi-functions, 65
psi2propII, *65*, 66, *71*, *73, 74*
psi2propII,psi_func_rcpp-method (psi2propII), 66
psi2propII,Rcpp_SmoothPsi (psi2propII), 66
PsiFunction, *17, 18*
PsiFunction (psi-functions), 65

Qn, *68*

ranef, *40*, *76*
ranef.rlmerMod (rlmerMod-class), 75
ransac_basin_radius, 66
ransac_lme4, *22*, *24*, 67, *72*, *78*
resid, *3*
resid.rlmerMod (rlmerMod-class), 75
residuals, *76*
residuals.rlmerMod, 69
resolveCluster, *84*
rlmer, *3*, *22–24*, *37*, *40, 41*, *45–47*, *53, 54*, *63–65*, *68, 69*, 70, *72*, *75–80*
rlmer_ransac, *45*, *68, 69*, *72*, 77
rlmerMod, *37*
rlmerMod-class, 75
robustlmm (robustlmm-package), 3
robustlmm-options, 79
robustlmm-package, 3

saveDatasets, *58, 59*, 80
shortenLabelsKS2022, 81
show (other), 48
show,Rcpp_HuberPsi-method (other), 48
show,Rcpp_PsiFunction-method (other), 48
show,Rcpp_PsiFunctionToPropIIPsiFunctionWrapper-method (other), 48
show,Rcpp_SmoothPsi-method (other), 48

show,rlmerMod-method (rlmerMod-class), 75
show.rlmerMod (rlmerMod-class), 75
show.summary.rlmerMod (rlmerMod-class), 75
sigma, *76*
sigma.rlmerMod (rlmerMod-class), 75
SmoothPsi (psi-functions), 65
smoothPsi, *22, 23*, *71*
smoothPsi (psi-functions), 65
splitDatasets, *9*, *16*, *36*, 82
sprintf, *81*
summary, *3*, *76*
summary.rlmerMod (rlmerMod-class), 75
summary.summary.rlmerMod (rlmerMod-class), 75

terms, *76*
terms.rlmerMod (rlmerMod-class), 75
theta (getME), 37
title, *51*

uniroot, *8*
update, *76*
update.rlmerMod (rlmerMod-class), 75

VarCorr, *76*
VarCorr.rlmerMod (rlmerMod-class), 75
VarCorr.summary.rlmerMod (rlmerMod-class), 75
vcov, *5*, *7*, *10*, *15*, *40*, *56*, *76*, *85*
vcov.rlmerMod, 83
vcov.summary.rlmerMod (rlmerMod-class), 75
vcov_sandwich, *10*, *14*, *83, 84*, 84
viewCopyOfSimulationStudy, 85

weights, *76*
weights.rlmerMod (rlmerMod-class), 75

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