Package: robustlmm (via r-universe)

October 10, 2024

Type Package

Title Robust Linear Mixed Effects Models

Version 3.3-1

Date 2023-12-10

Author Manuel Koller

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 (>= 1.1-9), Matrix (>= 1.6-2), R (>= 3.5.0)

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

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

Collate 'ghq.R' 'psiFunc2.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' 'generateMixedEffectDatasets.R' 'generateSensitivityCurveDatasets.R' 'manageDatasets.R' 'fitDatasets.R' 'processFit.R' 'processFile.R' 'simulationStudies.R' 'asymptoticEfficiency.R' 'emmeans.R'

LinkingTo Rcpp, robustbase, Matrix

Encoding UTF-8

RcppModules psi_function_module

RoxygenNote 7.2.3

Config/build/clean-inst-doc false

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-12-14 23:20:03 UTC

Contents

`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).

`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)))
```

`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)
```

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",
  ...,
  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_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_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.
init	optional argument passed on to rlmer.

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_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_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)))
```

`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))
```

---

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"),
  ...
)

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

`"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

`"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

"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")))
```

`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`

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. http://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)
```

---

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))
```

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
)

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', or a function producing an lmerMod object.
`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.

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.

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.

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)
```

---

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)))
```

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", "convergence.R",
    "robustnessDiagonal.R", "robustnessBlockDiagonal.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