Package 'robustlmm'

February 3, 2019

Type Package

Title Robust Linear Mixed Effects Models

Version 2.3

Date 2019-02-03

Author Manuel Koller

Maintainer Manuel Koller <koller.manuel@gmail.com>

Description A method 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 (>= 1.0-13), R (>= 3.2.0)

Suggests digest, reshape2, microbenchmark

Imports ggplot2, lattice, nlme, methods, robustbase (>= 0.93), xtable, Rcpp (>= 0.12.2), fastGHQuad

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'

LinkingTo Rcpp, RcppEigen, robustbase, cubature (> 1.3-8)

Encoding UTF-8

RcppModules psi_function_module

SystemRequirements C++11

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-02-03 19:30:07 UTC

R topics documented:

`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

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

chgDefaults *Change default arguments*

Description

Change the default arguments for a PsiFunction instance.

Usage

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

Arguments

object	PsiFunction instance to convert
...	arguments to change

Note

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

Examples

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

compare *Create comparison charts for multiple fits*

Description

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

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.

Usage

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

## 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, ...)

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

## S3 method for class 'rlmerMod'
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.
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)
object	object

Value

getInfo returns alist 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)
```

---

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".

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

Usage

```
## S3 method for class 'rlmerMod'
getME(object,
      name = c("X", "Z", "Zt", "Ztlist", "y", "mu",
               "u", "b.s", "b", "Gp", "Tp", "Lambda",
               "Lambdat","A", "U_b", "Lind", "sigma",
               "flist", "beta", "theta", "n_rtrms",
               "n_rfacs", "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", "is_REML"), ...)

  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

y response vector

mu conditional mean of the response

u conditional mode of the "spherical" random effects variable

b.s synonym for "u"

b onditional 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

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

n_rtrms number of random-effects terms

n_rfacs number of distinct random-effects grouping factors

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

offset model offset

lower lower bounds on model parameters (random effects parameters only)

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

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

... potentially further arguments passed to and from methods; none here at the moment.

Details

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), getME(fm1, "beta"),
    check.attributes=FALSE, tolerance = 0))
```

```
## All that can be accessed [potentially ..]:
(nmME <- eval(formals(robustlmm:::getME.rlmerMod)$name))
% dont..
stopifnot(all.equal(theta(fm1), getME(fm1, "theta")))
```

---

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)
```

---

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, ...)

 ## 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.
`ask`	waits for user input before displaying each plot.
`...`	currently ignored.

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)
```

---

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.

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** `huberPsiRcpp` 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 smoothe 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; `PsiFunction` C++ class for a base class to create custom ψ-functions.

Examples

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

---

psi2propII *Convert to Propsal II weight function*

Description

Converts the PsiFunction instance into one that corresponds to Proposal II, i.e., a function of the squared weights. The other elements of the PsiFunction instance are adapted accordingly.

Usage

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

Arguments

object	PsiFunction instance to convert
...	optional, new default arguments passed to chgDefaults.

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 linear mixed models*

Description

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

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.

Usage

```
rlmer(formula, data, ..., method = "DAStau",
  rho.e = smoothPsi, rho.b = smoothPsi, rho.sigma.e,
  rho.sigma.b, rel.tol = 1e-08,
  max.iter = 40 * (r + 1)^2, verbose = 0, doFit = TRUE,
  init)

rlmerRcpp(formula, data, ..., method = "DAStau",
  rho.e = smoothPsi, rho.b = smoothPsi, 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.
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 a robust approach of fitting 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. By default, 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 used to specify the appropriate function.

- For simple variance components and the residual error scale use the function psi2propII to change the tuning parameters. The is similar to Proposal II 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 random effects modeled with correlation parameters (referred to as nondiagonal 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 nondiagonal are generally larger than for the simple case. As a rule of thumb, one may use the squared tuning parameters of the simple case for the nondiagonal case.

Tables of tuning factors are given in the vignette (vignette("rlmer")). For the smoothed Huber function the tuning parameters to get approximately 95% efficiency are $k = 2.28$ for simple variance components and $k = 5.11$ for variance components including correlation parameters.

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, to get higher efficiency one has to 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.

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")))

## new Rcpp implementation
system.time(print(rlmerRcpp(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
  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)
  rlmer(Reaction ~ Days + (Days|Subject), sleepstudy,
        rho.sigma.e = psi2propII(smoothPsi, k = 2.28),
        rho.sigma.b = chgDefaults(smoothPsi, k = 5.11, s=10))
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)
```

Index