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").`

chgDefaults

Change default arguments

Description

Change the default arguments for a psi_func_cached object

Arguments

... arguments to change

Examples

```r
hPsi <- chgDefaults(huberPsi, k=2)
curve(huberPsi@psi(x), 0, 3)
curve(hPsi@psi(x), 0, 3, color="blue", add=TRUE)
```
**Description**

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

The functions xtable.compare.table and print.xtable.compare.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**

```r
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`.
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").
getME

Usage

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 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 "dgCMatrices") for the unit, orthogonal ran-
   dom 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)

### 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

```r
## 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(getME)$name))
% dont..
stopifnot(all.equal(theta(fm1), getME(fm1, "theta")))
```

---

### `plot.rlmerMod`

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

---

#### Description

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

#### Usage

```r
## 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 `lmer`; 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

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

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 `huberPsi` is identical to the one in the package `robustbase`. The psi slot equals the identity function within ±k (where k is the tuning parameter). Outside this interval it is equal to ±k. The rho slot equals the quadratic function within ±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_cached-class and psi_func-class for a more detailed description of the slots; psiFuncCached for a constructor function to create custom $\psi$-functions.

Examples

```r
plot(cPsi)
plot(huberPsi)
plot(smoothPsi)
curve(cPsi@psi(x), -3, 3)
curve(smoothPsi@psi(x, 1.345, 10), -3, 3, add=TRUE, col="red")
curve(huberPsi@psi(x, 1.345), -3, 3, add=TRUE, col="blue")
```

---

psi2propII  
*Convert to Proposal II weight function*

Description

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

Arguments

- **object**: psi_func object to convert
- **...**: optional, new default arguments passed to chgDefaults.

Examples

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

psiFuncCached constructor

Description

Create psi_func_cached object using cached numerical integration for E... slots.

Usage

psiFuncCached(rho, psi, wgt, Dwgt, Dpsi, name = NULL, 
...)

Arguments

rho    rho-function
psi    psi-function
wgt    wgt-function
Dwgt   derivative of weight function
Dpsi   derivative of psi
name   descriptor of this function family
...    default values for tuning constants

Value

psi_func_cached-class object

Warning

the E... slots will not be fully functional: they just return the value for the current defaults and ignore their arguments.

See Also

psi_func_cached-class

Examples

## re-define cPsi as psiFuncCached.
F0 <- function(x=1, .) rep.int(0, length(x))
F1 <- function(x=1, .) rep.int(1, length(x))
cPsi2 <- psiFuncCached(rho = function(x, .) x^2 / 2, 
    psi = function(x, .) x, 
    wgt = F1, Dwgt = F0, Dpsi = F1, 
    name = "classic (x^2/2)", 
    . = Inf ## dummy, need at least one parameter 
)stopifnot(all.equal(cPsi@Erho(), cPsi2@Erho()),
The class "psi_func_cached" is used to store $\psi$ (psi) functions for M-estimation. In particular, an object of the class contains $\rho(x)$ (rho), its derivative $\psi(x)$ (psi), the weight function $\psi(x)/x$, and first derivative of $\psi$, $D\psi = \psi'(x)$.

The expected values are calculated using numerical integration. For efficiency reasons, the results are cached. The current implementation only returns the expectations for the default values and ignores the arguments.

Objects from the Class

Objects can be created by calls of the form new("psi_func_cached", ...) but preferably by psiFuncCached(...).

Slots

name: Object of class "character", printable name of the psi function.
rho: Object of class "functionX", see "psi_func"
psi: Object of class "functionX", see "psi_func"
wgt: Object of class "functionX", see "psi_func"
Dwgt: Object of class "functionX", derivative of wgt
Dpsi: Object of class "functionX", see "psi_func"
tDef: Object of class "numeric", see "psi_func"
Erho: Object of class "functionXal", see "psi_func"
Epsi2: Object of class "functionXal", see "psi_func"
EDpsi: Object of class "functionXal", see "psi_func"
name: Object of class "character", see "psi_func"
xtras: Object of class "list", see "psi_func"

Extends

Class "psi_func", directly.
Methods

chgDefaults signature(object = "psi_func_cached"): The method is used to change the default values for the tuning parameters, and returns a psi_func-class object, a copy of input object with the slot tDefs possibly changed.

plot signature(object = "psi_func_cached"): Creates a plot of all the function slots.

psi2propII signature(object = "psi_func_cached"): Converts the psi_func object into a function that corresponds to Proposal II, i.e., a function of the squared weights. The other elements of the psi_func object are adapted accordingly.

show signature(object = "psi_func_cached"): Print function.

Warning

The E... slots will not be fully functional: they just return the value for the current defaults and ignore their arguments.

Author(s)

Manuel Koller

See Also

Inherits from psi_func, constructor psiFuncCached.

Examples

showClass("psi_func_cached")

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

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

## End(Not run)

### 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

```r
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)
```

```r
lmernofit(..., 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.
\texttt{rho.sigma.b} (optional) object of class \texttt{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 \texttt{psi2propII} to specify squared weights and custom tuning parameters or \texttt{chgDefaults} for regular weights for variance components including correlation parameters.

\texttt{rel.tol} relative tolerance used as criteria in the fitting process.

\texttt{max.iter} maximum number of iterations allowed.

\texttt{verbose} verbosity of output. Ranges from 0 (none) to 3 (a lot of output)

\texttt{dofit} logical scalar. When \texttt{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 \texttt{dofit = TRUE}, the default, the model is fit immediately.

\texttt{init} optional \texttt{lmerMod- or rlmerMod-object} to use for starting values, a list with elements \texttt{fixef}, \texttt{u}, \texttt{sigma}, \texttt{theta}, or a function producing an \texttt{lmerMod} object.

\texttt{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 \texttt{lmer} in the package \texttt{lme4}. The supported models are the same as for \texttt{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 \texttt{vignette("rlmer")}).

Models are specified using the \texttt{formula} argument, using the same syntax as for \texttt{lmer}. Additionally, one also needs to specify what robust scoring or weight functions are to be used (arguments starting with \texttt{rhoNe}). By default a smoothed version of the Huber function is used. Furthermore, the \texttt{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:

- \texttt{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.
- \texttt{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 \texttt{rho.e}, \texttt{rho.b}, \texttt{rho.sigma.e} and \texttt{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 \texttt{rho.sigma.e} and \texttt{rho.sigma.b} arguments, it is up to the user to specify the appropriate function.

- For simple variance components and the residual error scale use the function \texttt{psi2propII} to change the tuning parameters. This 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 \texttt{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 \texttt{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 (\texttt{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 \texttt{rho.b} (\texttt{rho.e}) but the argument \texttt{rho.sigma.b} (\texttt{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 \texttt{psi2propII} and \texttt{chgDefaults} functions.

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

**Value**

object of class \texttt{rlmerMod}.

**See Also**

\texttt{lmer}, \texttt{vignette(rlmer)}

**Examples**

```r
## 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,
```
## 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

*Topic classes
psi_func_cached-class, 11
rlmerMod-class, 16

*Topic models
compare, 3
rlmer, 13

*Topic utilities
chgDefaults, 2
compare, 3
getME, 4
psi2propII, 9
psiFuncCached, 10
chgDefaults, 2, 8, 9, 14, 15
cdgCMatrix, 5

extractAIC, 16
extractAIC.rlmerMod (rlmerMod-class), 16
family, 16
family.rlmerMod (rlmerMod-class), 16
fitted, 16
fitted.rlmerMod (rlmerMod-class), 16
fixef, 5, 6, 16
fixef.rlmerMod (rlmerMod-class), 16
formula, 16
formula.rlmerMod (rlmerMod-class), 16

getCall, 6
getInfo, 17
getInfo (compare), 3
getME, 4, 8

ggplot, 8
huberPsi (psi-functions), 8

isGLMM, 17
isGLMM.rlmerMod (rlmerMod-class), 16
isLMM, 17
isLMM.rlmerMod (rlmerMod-class), 16
isNLMM, 17
isNLMM.rlmerMod (rlmerMod-class), 16
isREML, 17
isREML.rlmerMod (rlmerMod-class), 16

lme4, 2
lmer, 2, 13–16
lmerNoFit (rlmer), 13
logLik, 17
logLik.rlmerMod (rlmerMod-class), 16

merMod, 4, 17
model.frame, 17
model.frame.rlmerMod (rlmerMod-class), 16
model.matrix, 17
model.matrix.rlmerMod (rlmerMod-class), 16

name, 6
nobs, 17
nobs.rlmerMod (rlmerMod-class), 16

plot, 17
plot.rlmerMod, 7
predict, 17
predict.rlmerMod (rlmerMod-class), 16
print, 17
print.rlmerMod (rlmerMod-class), 16
print.rlmerMod_plots (plot.rlmerMod), 7
print.summary.rlmer (rlmerMod-class), 16
print.Vacorr.rlmerMod
   (rlmerMod-class), 16
print.xtable, 3, 4
print.xtable.comparison.table
  (compare), 3
psi-functions, 8
psi2propII, 8, 9, 13–15
psi2propII, psi_func-method
  (psi2propII), 9
psi2propII, psi_func_cached-method
  (psi_func_cached-class), 11
psi_func, 11, 12
psi_func_cached-class, 11
psiFuncCached, 9, 10, 11, 12
ranef, 6, 17
ranef.rlmerMod (rlmerMod-class), 16
resid, 2
resid.rlmerMod (rlmerMod-class), 16
residuals, 17
residuals.rlmerMod, 12
rlmer, 2, 5, 8, 13, 16, 17
rlmerMod, 4, 5
rlmerMod-class, 16
robustlmm (robustlmm-package), 2
robustlmm-package, 2
show.rlmerMod-method (rlmerMod-class), 16
show.rlmerMod (rlmerMod-class), 16
show.summary.rlmerMod (rlmerMod-class), 16
sigma, 17
sigma.rlmerMod (rlmerMod-class), 16
smoothPsi (psi-functions), 8
summary, 2, 17
summary.rlmerMod (rlmerMod-class), 16
summary.summary.rlmerMod
  (rlmerMod-class), 16
terms, 17
terms.rlmerMod (rlmerMod-class), 16
theta (getME), 4
update, 17
update.rlmerMod (rlmerMod-class), 16
VarCorr, 17
VarCorr.rlmerMod (rlmerMod-class), 16
VarCorr.summary.rlmerMod
  (rlmerMod-class), 16
vcov, 6, 17
vcov.rlmerMod (rlmerMod-class), 16
vcov.summary.rlmerMod (rlmerMod-class), 16
weights, 17
weights.rlmerMod (rlmerMod-class), 16
xtable, 3, 4
xtable.comparison.table (compare), 3