

Package ‘lme4’

May 8, 2026

Version 2.0-1

Title Linear Mixed-Effects Models using 'Eigen' and S4

Description Fit linear and generalized linear mixed-effects models. The models and their components are represented using S4 classes and methods. The core computational algorithms are implemented using the 'Eigen' C++ library for numerical linear algebra and 'RcppEigen' `` glue".

Depends R (>= 3.6), Matrix, methods, stats

LinkingTo Matrix (>= 1.5-0), Rcpp (>= 0.10.5), RcppEigen (>= 0.3.3.9.4)

Imports MASS, Rdpack, boot, graphics, grid, lattice, minqa (>= 1.1.15), nlme (>= 3.1-123), nloptr (>= 1.0.4), parallel, reformulas (>= 0.4.3.1), rlang, splines, utils

Suggests HSAUR3, MEMSS, car, dfoptim, gamm4, ggplot2, glmmTMB, knitr, merDeriv, mgcv, mlmRev, numDeriv, optimx (>= 2013.8.6), pbkrtest, rmarkdown, rr2, semEff, statmod, testthat (>= 0.8.1), tibble

Enhances DHARMA, performance

RdMacros Rdpack

VignetteBuilder knitr

LazyData yes

License GPL (>= 2)

URL <https://github.com/lme4/lme4/>

BugReports <https://github.com/lme4/lme4/issues>

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Douglas Bates [aut] (ORCID: <<https://orcid.org/0000-0001-8316-9503>>),
Martin Maechler [aut] (ORCID: <<https://orcid.org/0000-0002-8685-9910>>),
Ben Bolker [cre, aut] (ORCID: <<https://orcid.org/0000-0002-2127-0443>>),

Steven Walker [aut] (ORCID: <<https://orcid.org/0000-0002-4394-9078>>),
 Rune Haubo Bojesen Christensen [ctb] (ORCID:
 <<https://orcid.org/0000-0002-4494-3399>>),
 Henrik Singmann [ctb] (ORCID: <<https://orcid.org/0000-0002-4842-3657>>),
 Bin Dai [ctb],
 Fabian Scheipl [ctb] (ORCID: <<https://orcid.org/0000-0001-8172-3603>>),
 Gabor Grothendieck [ctb],
 Peter Green [ctb] (ORCID: <<https://orcid.org/0000-0002-0238-9852>>),
 John Fox [ctb],
 Alexander Bauer [ctb],
 Pavel N. Krivitsky [ctb, cph] (ORCID:
 <<https://orcid.org/0000-0002-9101-3362>>, shared copyright on
 simulate.formula),
 Emi Tanaka [ctb] (ORCID: <<https://orcid.org/0000-0002-1455-259X>>),
 Mikael Jagan [aut] (ORCID: <<https://orcid.org/0000-0002-3542-2938>>),
 Ross D. Boylan [ctb] (ORCID: <<https://orcid.org/0009-0003-4123-8090>>),
 Anna Ly [aut] (ORCID: <<https://orcid.org/0000-0002-0210-0342>>)

Maintainer Ben Bolker <bbolker+lme4@gmail.com>

Repository CRAN

Date/Publication 2026-03-05 17:30:02 UTC

Contents

lme4-package	4
allFit	5
Arabidopsis	7
bootMer	9
cake	12
cbpp	13
checkConv	15
confint.merMod	16
convergence	18
culcitalogreg	21
culcitalvolume	22
devcomp	24
devfun2	24
drop1.merMod	25
dummy	27
Dyestuff	28
factorize	29
fixef	29
fortify	30
getDoublevertDefault	31
getME	32
getReCovs	35
GHrule	36
glmer	37

glmer.nb	40
glmerLaplaceHandle	42
glmFamily	43
glmFamily-class	43
golden-class	44
gopherdat2	45
GQdk	46
grouseticks	47
hatvalues.merMod	48
influence.merMod	49
InstEval	52
isNewMerMod	53
isREML	54
isSingular	55
lme4_testlevel	57
lmer	57
lmerControl	60
lmList	67
lmList4-class	68
lmResp	69
lmResp-class	70
merMod-class	71
merPredD	75
merPredD-class	76
mkMerMod	76
mkNewReTrms	77
mkRespMod	79
mkSimulateTemplate	79
mkVarCorr	80
modular	81
namedList	85
NelderMead	86
NelderMead-class	87
ngrps	89
nlformula	90
nlmer	91
nloptwrap	93
Pastes	95
Penicillin	96
plot.merMod	97
plots.thpr	100
predict.merMod	101
profile-methods	103
prt-utilities	106
pvalues	109
ranef	110
refit	113
refitML	114

reformulas_imports	115
rePCA	116
rePos	117
rePos-class	117
residuals.merMod	118
salamander	119
schizophrenia	120
sigma	121
simulate.formula	122
simulate.merMod	123
sleepstudy	126
summary	127
toenail	128
troubleshooting	129
VarCorr	130
vcconv	132
vcov.merMod	133
VerbAgg	135

Index	137
--------------	------------

lme4-package	<i>Linear, generalized linear, and nonlinear mixed models</i>
--------------	---

Description

lme4 provides functions for fitting and analyzing mixed models: linear ([lmer](#)), generalized linear ([glmer](#)) and nonlinear ([nlmer](#).)

Differences between nlme and lme4

lme4 covers approximately the same ground as the earlier **nlme** package. The most important differences are:

- **lme4** uses modern, efficient linear algebra methods as implemented in the Eigen package, and uses reference classes to avoid undue copying of large objects; it is therefore likely to be faster and more memory-efficient than **nlme**.
- **lme4** includes generalized linear mixed model (GLMM) capabilities, via the [glmer](#) function.
- **lme4** does *not* currently implement **nlme**'s features for modeling heteroscedasticity and correlation of residuals.
- **lme4** does not currently offer the same flexibility as **nlme** for composing complex variance-covariance structures, but it does implement crossed random effects in a way that is both easier for the user and much faster.
- **lme4** offers built-in facilities for likelihood profiling and parametric bootstrapping.

- **lme4** is designed to be more modular than **nlme**, making it easier for downstream package developers and end-users to re-use its components for extensions of the basic mixed model framework. It also allows more flexibility for specifying different functions for optimizing over the random-effects variance-covariance parameters.
- **lme4** is not (yet) as well-documented as **nlme**.

Differences between current (1.0.+) and previous versions of lme4

- `[gn]lmer` now produces objects of class `merMod` rather than class `mer` as before
- the new version uses a combination of S3 and reference classes (see [ReferenceClasses](#), [merPredD-class](#), and [lmResp-class](#)) as well as S4 classes; partly for this reason it is more interoperable with **nlme**
- The internal structure of `[gn]lmer` is now more modular, allowing finer control of the different steps of argument checking; construction of design matrices and data structures; parameter estimation; and construction of the final `merMod` object (see [modular](#))
- profiling and parametric bootstrapping are new in the current version
- the new version of **lme4** does *not* provide an `mcmc` (post-hoc MCMC sampling) method, because this was deemed to be unreliable. Alternatives for computing p-values include parametric bootstrapping ([bootMer](#)) or methods implemented in the **pbkrtest** package and leveraged by the **lmerTest** package and the `Anova` function in the **car** package (see [pvalues](#) for more details).

Caveats and trouble-shooting

- Some users who have previously installed versions of the **RcppEigen** and **minqa** packages may encounter segmentation faults (!!); the solution is to make sure to re-install these packages before installing **lme4**. (Because the problem is not with the explicit *version* of the packages, but with running packages that were built with different versions of **Rcpp** in conjunction with each other, simply making sure you have the latest version, or using `update.packages`, will not necessarily solve the problem; you must actually re-install the packages. The problem is most likely with **minqa**.)

allFit

Refit a fitted model with all available optimizers to check convergence

Description

Attempt to re-fit a `[g]lmer` model with a range of optimizers. The default is to use all known optimizers for R that satisfy the requirements (i.e. they do not require functions and allow box constraints: see ‘optimizer’ in [lmerControl](#)). These optimizers fall in four categories; (i) built-in (`minqa::bobyqa`, `lme4::Nelder_Mead`, `nlminbwrap`), (ii) wrapped via `optimx` (most of `optimx`’s optimizers that allow box constraints require an explicit gradient function to be specified; the two provided here are the base R functions that can be accessed via `optimx`), (iii) wrapped via `nloptr` (see examples for the list of options), (iv) ‘`doptim::nmkb`’ (via the (unexported) `nmkbw` wrapper: this appears as ‘`nmkbw`’ in `meth.tab`)

Usage

```
allFit(object, meth.tab = NULL, data = NULL,
       verbose = TRUE,
       show.meth.tab = FALSE,
       maxfun = 1e5,
       parallel = c("no", "multicore", "snow"),
       ncpus = getOption("allFit.ncpus", 1L), cl = NULL,
       catch.errs = TRUE,
       start_from_mle = TRUE)
```

Arguments

object	a fitted model.
meth.tab	a matrix (or data.frame) with columns. method the name of a specific optimization method to pass to the optimizer (leave blank for built-in optimizers). optimizer the optimizer function to use.
data	data to be included with result (for later debugging etc.)
verbose	(logical) report progress in detail?
show.meth.tab	(logical) return table of methods?
maxfun	passed as part of optCtrl to set the maximum number of function evaluations: this is <i>automatically</i> converted to the correct specification (e.g. maxfun, maxfeval, maxit, etc.) for each optimizer.
parallel	The type of parallel operation to be used (if any). If missing, the default is taken from the option "allFit.ncpus" (and if that is not set, "no").
ncpus	integer: number of processes to be used in parallel operation. Use options(allFit.ncpus=X) to set the default value to X for the duration of an R session.
cl	An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the allFit call.
catch.errs	(logical) wrap model fits in tryCatch clause to skip over errors? (catch.errs=FALSE is probably only useful for debugging).
start_from_mle	(logical) if TRUE, initialize the refitting process using starting values from the model's MLE.

Details

- Needs packages optimx, and dfoptim to use all optimizers.
- If you are using parallel="snow" (e.g. when running in parallel on Windows), you will need to set up a cluster yourself and run clusterEvalQ(cl, library("lme4")) before calling allFit to make sure that the lme4 package is loaded on all of the workers.
- Control arguments in control\$optCtrl that are unused by a particular optimizer will be *silently* ignored (in particular, the maxfun specification is only respected by bobyqa, Nelder_Mead, and nmkbw).

- Because `allFit` works by calling `update`, it may be fragile if the original model call contains references to variables, especially if they were originally defined in other environments or no longer exist when `allFit` is called.

Value

an object of type `allFit`, which is a list of fitted `merMod` objects (unless `show.meth.tab` is specified, in which case a data frame of methods is returned). The summary method for this class extracts tables with a variety of useful information about the different fits (see examples).

See Also

`slice,slice2D` from the **bbmle** package

Examples

```
if (interactive()) {
  library(lme4)
  gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
              data = cbpp, family = binomial)
  ## show available methods
  allFit(show.meth.tab=TRUE)
  gm_all <- allFit(gm1)
  ss <- summary(gm_all)
  ss$which.OK          ## logical vector: which optimizers worked?
  ## the other components only contain values for the optimizers that worked
  ss$l1lik             ## vector of log-likelihoods
  ss$fixef             ## table of fixed effects
  ss$sdcor             ## table of random effect SDs and correlations
  ss$theta             ## table of random effects parameters, Cholesky scale
}
## Not run:
## Parallel examples for Windows
nc <- max(detectCores() - 1L, 1L, na.rm = TRUE)
optCls <- makeCluster(nc)
clusterEvalQ(optCls,library("lme4"))
### not necessary here because using a built-in
## data set, but in general you should clusterExport() your data
clusterExport(optCls, "cbpp")
system.time(af1 <- allFit(m0, parallel = 'snow',
                        ncpus = nc, cl=optCls))

stopCluster(optCls)

## End(Not run)
```

Description

Data on genetic variation in responses to fertilization and simulated herbivory in *Arabidopsis*

Usage

```
data("Arabidopsis")
```

Format

A data frame with 625 observations on the following 8 variables.

reg region: a factor with 3 levels NL (Netherlands), SP (Spain), SW (Sweden)

popu population: a factor with the form n.R representing a population in region R

gen genotype: a factor with 24 (numeric-valued) levels

rack a nuisance factor with 2 levels, one for each of two greenhouse racks

nutrient fertilization treatment/nutrient level (1, minimal nutrients or 8, added nutrients)

amd simulated herbivory or "clipping" (apical meristem damage): unclipped (baseline) or clipped

status a nuisance factor for germination method (Normal, Petri.Plate, or Transplant)

total.fruits total fruit set per plant (integer)

Source

(Banta et al. 2010)

References

Banta JA, Stevens MHH, Pigliucci M (2010). "A comprehensive test of the 'limiting resources' framework applied to plant tolerance to apical meristem damage." *Oikos*, **119**(2), 359-369. doi:10.1111/j.16000706.2009.17726.x.

Examples

```
data(Arabidopsis)
summary(Arabidopsis[, "total.fruits"])
table(gsub("[0-9].", "", levels(Arabidopsis[, "popu"])))
library(lattice)
stripplot(log(total.fruits+1) ~ amd|nutrient, data = Arabidopsis,
          groups = gen,
          strip=strip.custom(strip.names=c(TRUE,TRUE)),
          type=c('p','a'), ## points and panel-average value --
          ## see ?panel.xyplot
          scales=list(x=list(rot=90)),
          main="Panel: nutrient, Color: genotype")
```

bootMer

*Model-based (Semi-)Parametric Bootstrap for Mixed Models***Description**

Perform model-based (Semi-)parametric bootstrap for mixed models.

Usage

```
bootMer(x, FUN, nsim = 1, seed = NULL, use.u = FALSE, re.form=NA,
        type = c("parametric", "semiparametric"),
        verbose = FALSE, .progress = "none", PBargs = list(),
        parallel = c("no", "multicore", "snow"),
        ncpus = getOption("boot.ncpus", 1L), cl = NULL)
```

Arguments

<code>x</code>	a fitted merMod object: see lmer , glmer , etc.
<code>FUN</code>	a function taking a fitted merMod object as input and returning the <i>statistic</i> of interest, which must be a (possibly named) numeric vector.
<code>nsim</code>	number of simulations, positive integer; the bootstrap B (or R).
<code>seed</code>	optional argument to set.seed .
<code>use.u</code>	logical, indicating whether the spherical random effects should be simulated / bootstrapped as well. If TRUE, they are not changed, and all inference is conditional on these values. If FALSE, new normal deviates are drawn (see Details).
<code>re.form</code>	formula, NA (equivalent to <code>use.u=FALSE</code>), or NULL (equivalent to <code>use.u=TRUE</code>): alternative to <code>use.u</code> for specifying which random effects to incorporate. See simulate.merMod for details.
<code>type</code>	character string specifying the type of bootstrap, "parametric" or "semiparametric"; partial matching is allowed.
<code>verbose</code>	logical indicating if progress should print output
<code>.progress</code>	character string - type of progress bar to display. Default is "none"; the function will look for a relevant *ProgressBar function, so "txt" will work in general; "tk" is available if the tk package is loaded; or "win" on Windows systems. Progress bars are disabled (with a message) for parallel operation.
<code>PBargs</code>	a list of additional arguments to the progress bar function (the package authors like <code>list(style=3)</code>).
<code>parallel</code>	The type of parallel operation to be used (if any). If missing, the default is taken from the option "boot.parallel" (and if that is not set, "no").
<code>ncpus</code>	integer: number of processes to be used in parallel operation: typically one would choose this to be the number of available CPUs.
<code>cl</code>	An optional parallel or snow cluster for use if <code>parallel = "snow"</code> . If not supplied, a cluster on the local machine is created for the duration of the boot call.

Details

The semi-parametric variant is only partially implemented, and we only provide a method for `lmer` and `glmer` results.

Information about warning and error messages incurred during the bootstrap returns can be retrieved via the attributes

bootFail number of failures (errors)

boot.fail.msgs error messages

boot.all.msgs messages, warnings, and error messages

e.g. `attr("boot.fail.msgs")` to retrieve error messages

The working name for `bootMer()` was “`simulestimate()`”, as it is an extension of `simulate` (see [simulate.merMod](#)), but we want to emphasize its potential for valid inference.

- If `use.u` is `FALSE` and `type` is “`parametric`”, each simulation generates new values of both the “*spherical*” random effects u and the i.i.d. errors ϵ , using `rnorm()` with parameters corresponding to the fitted model x .
- If `use.u` is `TRUE` and `type`==“`parametric`”, only the i.i.d. errors (or, for GLMMs, response values drawn from the appropriate distributions) are resampled, with the values of u staying fixed at their estimated values.
- If `use.u` is `TRUE` and `type`==“`semiparametric`”, the i.i.d. errors are sampled from the distribution of (response) residuals. (For GLMMs, the resulting sample will no longer have the same properties as the original sample, and the method may not make sense; a warning is generated.) The semiparametric bootstrap is currently an experimental feature, and therefore may not be stable.
- The case where `use.u` is `FALSE` and `type`==“`semiparametric`” is not implemented; Morris (2002) suggests that resampling from the estimated values of u is not good practice.

Value

an object of S3 class “`boot`”, compatible with `boot` package’s `boot()` result. (See Details for information on how to retrieve information about errors during bootstrapping.)

Note

If you are using `parallel="snow"`, you will need to run `clusterEvalQ(cl, library("lme4"))` before calling `bootMer` to make sure that the `lme4` package is loaded on all of the workers; you may additionally need to use `clusterExport` if you are using a summary function that calls any objects from the environment.

References

- Davison AC, Hinkley DV (1997). *Bootstrap methods and their application*. Cambridge University Press.
- Morris JS (2002). “The BLUPs are not “best” when it comes to bootstrapping.” *Statistics & Probability Letters*, **56**(4), 425-430. ISSN 0167-7152, doi:10.1016/S01677152(02)00041X.

See Also

- [confint.merMod](#), for a more specific approach to bootstrap confidence intervals on parameters.
- [refit\(\)](#), or [PBmodcomp\(\)](#) from the [pbrktest](#) package, for parametric bootstrap comparison of models.
- [boot\(\)](#), and then [boot.ci](#), from the [boot](#) package.
- [profile-methods](#), for likelihood-based inference, including confidence intervals.
- [pvalues](#), for more general approaches to inference and p-value computation in mixed models.

Examples

```

if (interactive()) {
  fm01ML <- lmer(Yield ~ 1|Batch, Dyestuff, REML = FALSE)
  ## see ?"profile-methods"
  mySumm <- function(.) { s <- sigma(.)
    c(beta=getME(., "beta"), sigma = s, sig01 = unname(s * getME(., "theta"))) }
  (t0 <- mySumm(fm01ML)) # just three parameters
  ## alternatively:
  mySumm2 <- function(.) {
    c(beta=fixef(.),sigma=sigma(.), sig01=sqrt(unlist(VarCorr(.))))
  }

  set.seed(101)
  ## 3.8s (on a 5600 MIPS 64bit fast(year 2009) desktop "AMD Phenom(tm) II X4 925"):
  system.time( boo01 <- bootMer(fm01ML, mySumm, nsim = 100) )

  ## to "look" at it
  if (requireNamespace("boot")) {
    boo01
    ## note large estimated bias for sig01
    ## (~30% low, decreases _slightly_ for nsim = 1000)

    ## extract the bootstrapped values as a data frame ...
    head(as.data.frame(boo01))

    ## ----- Bootstrap-based confidence intervals -----

    ## warnings about "Some ... intervals may be unstable" go away
    ## for larger bootstrap samples, e.g. nsim=500

    ## intercept
    (bCI.1 <- boot::boot.ci(boo01, index=1, type=c("norm", "basic", "perc")))# beta

    ## Residual standard deviation - original scale:
    (bCI.2 <- boot::boot.ci(boo01, index=2, type=c("norm", "basic", "perc"))
    ## Residual SD - transform to log scale:
    (bCI.2L <- boot::boot.ci(boo01, index=2, type=c("norm", "basic", "perc"),
      h = log, hdot = function(.) 1/., hinv = exp))

    ## Among-batch variance:

```

```

(bCI.3 <- boot::boot.ci(boo01, index=3, type=c("norm", "basic", "perc"))) # sig01

confint(boo01)
confint(boo01, type="norm")
confint(boo01, type="basic")

## Graphical examination:
plot(boo01, index=3)

## Check stored values from a longer (1000-replicate) run:
(load(system.file("testdata", "boo01L.RData", package="lme4")))# "boo01L"
plot(boo01L, index=3)
mean(boo01L$t[, "sig01"]==0) ## note point mass at zero!
}
}

```

cake

Breakage Angle of Chocolate Cakes

Description

Data on the breakage angle of chocolate cakes made with three different recipes and baked at six different temperatures. This is a split-plot design with the recipes being whole-units and the different temperatures being applied to sub-units (within replicates). The experimental notes suggest that the replicate numbering represents temporal ordering.

Format

A data frame with 270 observations on the following 5 variables.

replicate a factor with levels 1 to 15

recipe a factor with levels A, B and C

temperature an ordered factor with levels 175 < 185 < 195 < 205 < 215 < 225

angle a numeric vector giving the angle at which the cake broke.

temp numeric value of the baking temperature (degrees F).

Details

The replicate factor is nested within the recipe factor, and temperature is nested within replicate.

Source

Original data were presented in Cook (1938), and reported in Cochran and Cox (1957, p. 300). Also cited in Lee, Nelder and Pawitan (2006).

References

- Cook FE (1938). *Chocolate cake, I. Optimum baking temperature*. Master's Thesis, Iowa State College.
- Cochran WG, Cox GM (1957). *Experimental Designs*, 2nd edition. John Wiley & Sons, New York.
- Lee Y, Nelder JA, Pawitan Y (2006). *Generalized linear models with random effects: unified analysis via H-likelihood*. Chapman and Hall/CRC, Boca Raton.

Examples

```
str(cake)
## 'temp' is continuous, 'temperature' an ordered factor with 6 levels

(fm1 <- lmer(angle ~ recipe * temperature + (1|recipe:replicate), cake, REML= FALSE))
(fm2 <- lmer(angle ~ recipe + temperature + (1|recipe:replicate), cake, REML= FALSE))
(fm3 <- lmer(angle ~ recipe + temp          + (1|recipe:replicate), cake, REML= FALSE))

## and now "choose" :
anova(fm3, fm2, fm1)
```

cbpp

Contagious bovine pleuropneumonia

Description

Contagious bovine pleuropneumonia (CBPP) is a major disease of cattle in Africa, caused by a mycoplasma. This dataset describes the serological incidence of CBPP in zebu cattle during a follow-up survey implemented in 15 commercial herds located in the Boji district of Ethiopia. The goal of the survey was to study the within-herd spread of CBPP in newly infected herds. Blood samples were quarterly collected from all animals of these herds to determine their CBPP status. These data were used to compute the serological incidence of CBPP (new cases occurring during a given time period). Some data are missing (lost to follow-up).

There are two datasets, although their precise origins are unclear. The dataset used in the classical lme4 examples, cbpp, matches the file maintained by the package authors. The extended dataset, cbpp2, corresponds to Table 1 of (Lesnoff et al. 2004), with the exception of herd 6, which is a known typographical error in the original paper.

Format

cbpp is a data frame with 56 observations on the following 4 variables.

herd A factor identifying the herd (1 to 15).

incidence The number of new serological cases for a given herd and time period.

size A numeric vector describing herd size at the beginning of a given time period.

period A factor with levels 1 to 4.

The extended version, `cbpp2`, has the additional variables:

`herd` A factor identifying the herd (1 to 15).

`treatment` A factor referring to the control measure used to manage CBPP.

- Complete = complete isolation or antibiotic treatment,
- Partial/null = partial/null isolation and no antibiotic treatment,
- Unknown = strategy remained.

`avg_size` The average number of animals housed in a della (a temporary paddock used for holding cattle on the farm).

Details

Serological status was determined using a competitive enzyme-linked immuno-sorbent assay (cELISA).

Source

(Lesnoff et al. 2004)

References

Lesnoff M, Laval G, Bonnet P, Abdicho S, Workalemahu A, Kifle D, Peyraud A, Lancelot R, Thi-aucourt F (2004). “Within-herd spread of contagious bovine pleuropneumonia in Ethiopian high-lands.” *Preventive Veterinary Medicine*, **64**(1), 27-40. ISSN 0167-5877, doi:10.1016/j.prevetmed.2004.03.005.

Examples

```
## response as a matrix
(m1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             family = binomial, data = cbpp))
## response as a vector of probabilities and usage of argument "weights"
m1p <- glmer(incidence / size ~ period + (1 | herd), weights = size,
             family = binomial, data = cbpp)
## Confirm that these are equivalent:
stopifnot(all.equal(fixef(m1), fixef(m1p), tolerance = 1e-5),
          all.equal(ranef(m1), ranef(m1p), tolerance = 1e-5))

## GLMM with individual-level variability (accounting for overdispersion)
cbpp$obs <- 1:nrow(cbpp)
(m2 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd) + (1|obs),
             family = binomial, data = cbpp))

## Fitting the model for cbpp2
gm1 <- glmer(incidence/size ~ period + treatment + avg_size + (1 | herd),
             family = binomial,
             data = cbpp2, weights = size,
             control = glmerControl(optimizer="bobyqa"))
## Adding an observation-level random effect
cbpp2 <- transform(cbpp2, obs=factor(seq(nrow(cbpp2))))
## Herd and observation-level REs (below causes singular fit issues)
gm2 <- update(gm1, .~.(1|obs))
## observation-level REs only (no singular fit issue)
```

```
gm3 <- update(gm1, .~.-(1|herd)+(1|obs))
```

 checkConv

Extended Convergence Checking

Description

Primarily internal code for checking optimization convergence, see [convergence](#) for a more detailed discussion.

Usage

```
checkConv(derivs, coefs, ctrl, lbound, ubound,
          debug = FALSE, nobs = NULL, ndim = NULL)
```

Arguments

derivs	typically the "derivs" attribute of <code>optimizeLmer()</code> ; with "gradients" and possibly "Hessian" component
coefs	current coefficient estimates
ctrl	list of lists, each with action character strings specifying what should happen when a check triggers, and tol numerical tolerances, as is the result of <code>lmerControl()\$checkConv</code> .
lbound, ubound	vectors of lower and upper bounds <i>for random-effects parameters only</i> (length is taken to determine number of RE parameters)
debug	enable debugging output, useful if some checks are on "ignore", but would "trigger"
nobs	number of observations (needed to determine whether to do convergence checks for large data)
ndim	number of dimensions for the variance-covariance matrix of random effects (needed to determine whether to do the gradient/hessian checks for large non-linear optimization problems)

Value

A result list containing

code	The return code for the check
messages	A character vector of warnings and messages generated by the check

See Also

[convergence](#)

 confint.merMod

Compute Confidence Intervals for Parameters of a [ng]lmer Fit

Description

Compute confidence intervals on the parameters of a `*lmer()` model fit (of class `"merMod"`).

Usage

```
## S3 method for class 'merMod'
confint(object, parm, level = 0.95,
        method = c("profile", "Wald", "boot"), zeta,
        nsim = 500,
        boot.type = c("perc", "basic", "norm"),
        FUN = NULL, quiet = FALSE,
        oldNames, signames = TRUE,
        boot.scale = c("sdcor", "vcov"),
        ...)
## S3 method for class 'thpr'
confint(object, parm, level = 0.95,
        zeta, non.mono.tol = 1e-2, ...)
```

Arguments

<code>object</code>	a fitted [ng]lmer model or profile
<code>parm</code>	parameters for which intervals are sought. Specified by an integer vector of positions, character vector of parameter names, or (unless doing parametric bootstrapping with a user-specified bootstrap function) <code>"theta_"</code> or <code>"beta_"</code> to specify variance-covariance or fixed effects parameters only: see the which parameter of profile .
<code>level</code>	confidence level < 1, typically above 0.90.
<code>method</code>	a character string determining the method for computing the confidence intervals.
<code>zeta</code>	(for <code>method = "profile"</code> only:) likelihood cutoff (if not specified, as by default, computed from <code>level</code>).
<code>nsim</code>	number of simulations for parametric bootstrap intervals.
<code>FUN</code>	bootstrap function; if <code>NULL</code> , an internal function that returns the fixed-effect parameters as well as the random-effect parameters on the standard deviation/correlation scale will be used. See bootMer for details.
<code>boot.type</code>	bootstrap confidence interval type, as described in boot.ci . (Methods <code>'stud'</code> and <code>'bca'</code> are unavailable because they require additional components to be calculated.)
<code>quiet</code>	(logical) suppress messages about computationally intensive profiling?
<code>oldNames</code>	(logical) deprecated; has the same purpose as <code>signames</code> .


```

## --> ~ 2.2 seconds (2022)
set.seed(123) # (reproducibility when using bootstrap)
system.time(fm1B <- confint(fm1, method="boot", signames=FALSE,
                           .progress="txt", PBargs= list(style=3)))
## --> ~ 6.2 seconds (2022) and warning, messages
} else {
  load(system.file("testdata", "confint_ex.rda", package="lme4"))
}
fm1P
fm1B
} ## if interactive && testlevel>=3

```

convergence

Assessing Convergence for Fitted Models

Description

[g]lmer fits may produce convergence warnings; these do **not** necessarily mean the fit is incorrect (see “Theoretical details” below). The following steps are recommended assessing and resolving convergence warnings (also see examples below):

- double-check the model specification and the data
- adjust stopping (convergence) tolerances for the nonlinear optimizer, using the `optCtrl` argument to `[g]lmerControl` (see “Convergence controls” below)
- center and scale continuous predictor variables (e.g. with `scale`)
- double-check the Hessian calculation with the more expensive Richardson extrapolation method (see examples)
- restart the fit from the reported optimum, or from a point perturbed slightly away from the reported optimum
- use `allFit` to try the fit with all available optimizers (e.g. several different implementations of BOBYQA and Nelder-Mead, L-BFGS-B from `optim`, `nlminb`, ...). While this will of course be slow for large fits, we consider it the gold standard; if all optimizers converge to values that are practically equivalent, then we would consider the convergence warnings to be false positives.

Details

Convergence controls:

- the controls for the `nloptwrap` optimizer (the default for `lmer`) are
 - `ftol_abs`** (default `1e-6`) stop on small change in deviance
 - `ftol_rel`** (default `0`) stop on small relative change in deviance
 - `xtol_abs`** (default `1e-6`) stop on small change of parameter values
 - `xtol_rel`** (default `0`) stop on small relative change of parameter values
 - `maxeval`** (default `1000`) maximum number of function evaluations
- Changing `ftol_abs` and `xtol_abs` to stricter values (e.g. `1e-8`) is a good first step for resolving convergence problems, at the cost of slowing down model fits.

- the controls for `minqa::bobyqa` (default for `glmer` first-stage optimization) are
 - rhobeg** (default $2e-3$) initial radius of the trust region
 - rhoend** (default $2e-7$) final radius of the trust region
 - maxfun** (default 10000) maximum number of function evaluations
 rhoend, which describes the scale of parameter uncertainty on convergence, is approximately analogous to `xtol_abs`.
- the controls for `Nelder_Mead` (default for `glmer` second-stage optimization) are
 - FtolAbs** (default $1e-5$) stop on small change in deviance
 - FtolRel** (default $1e-15$) stop on small relative change in deviance
 - XtolRel** (default $1e-7$) stop on small change of parameter values
 - maxfun** (default 10000) maximum number of function evaluations

Theoretical issues: `lme4` uses general-purpose nonlinear optimizers (e.g. Nelder-Mead or Powell's BOBYQA method) to estimate the variance-covariance matrices of the random effects. Assessing the convergence of such algorithms reliably is difficult. For example, evaluating the **Karush-Kuhn-Tucker conditions** (convergence criteria which reduce in simple cases to showing that the gradient is zero and the Hessian is positive definite) is challenging because of the difficulty of evaluating the gradient and Hessian.

We (the `lme4` authors and maintainers) are still in the process of finding the best strategies for testing convergence. Some of the relevant issues are

- the gradient and Hessian are the basic ingredients of KKT-style testing, but (at least for now) `lme4` estimates them by finite-difference approximations which are sometimes unreliable.
- The Hessian computation in particular represents a difficult tradeoff between computational expense and accuracy. At present the Hessian computations used for convergence checking (and for estimating standard errors of fixed-effect parameters for GLMMs) follow the **ordinal** package in using a naive but computationally cheap centered finite difference computation (with a fixed step size of 10^{-4}). A more reliable but more expensive approach is to use **Richardson extrapolation**, as implemented in the **numDeriv** package.
- it is important to scale the estimated gradient at the estimate appropriately; two reasonable approaches are
 1. scale gradients by the inverse Cholesky factor of the Hessian, equivalent to scaling gradients by the estimated Wald standard error of the estimated parameters. `lme4` uses this approach; it requires the Hessian to be estimated (although the Hessian is required for **reliable estimation of the fixed-effect standard errors for GLMMs** in any case).
 2. use unscaled gradients on the random-effects parameters, since these are essentially already unitless (for LMMs they are scaled relative to the residual variance; for GLMMs they are scaled relative to the sampling variance of the conditional distribution); for GLMMs, scale fixed-effect gradients by the standard deviations of the corresponding input variable
- Exploratory analyses suggest that (1) the naive estimation of the Hessian may fail for large data sets (number of observations greater than approximately 10^5); (2) the magnitude of the scaled gradient increases with sample size, so that warnings will occur even for apparently well-behaved fits with large data sets.

See Also

[lmerControl](#), [isSingular](#)

Examples

```

if (interactive()) {
  fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)

  ## 1. decrease stopping tolerances
  strict_tol <- lmerControl(optCtrl=list(xtol_abs=1e-8, ftol_abs=1e-8))
  if (all(fm1@optinfo$optimizer=="nloptwrap")) {
    fm1.tol <- update(fm1, control=strict_tol)
  }

  ## 2. center and scale predictors:
  ss.CS <- transform(sleepstudy, Days=scale(Days))
  fm1.CS <- update(fm1, data=ss.CS)

  ## 3. recompute gradient and Hessian with Richardson extrapolation
  devfun <- update(fm1, devFunOnly=TRUE)
  if (isLMM(fm1)) {
    pars <- getME(fm1,"theta")
  } else {
    ## GLMM: requires both random and fixed parameters
    pars <- getME(fm1, c("theta","fixef"))
  }
  if (require("numDeriv")) {
    cat("hess:\n"); print(hess <- hessian(devfun, unlist(pars)))
    cat("grad:\n"); print(grad <- grad(devfun, unlist(pars)))
    cat("scaled gradient:\n")
    print(scgrad <- solve(chol(hess), grad))
  }
  ## compare with internal calculations:
  fm1@optinfo$derivs

  ## compute reciprocal condition number of Hessian
  H <- fm1@optinfo$derivs$Hessian
  Matrix::rcond(H)

  ## 4. restart the fit from the original value (or
  ## a slightly perturbed value):
  fm1.restart <- update(fm1, start=pars)
  set.seed(101)
  pars_x <- runif(length(pars),pars/1.01,pars*1.01)
  fm1.restart2 <- update(fm1, start=pars_x,
                        control=strict_tol)

  ## 5. try all available optimizers

  fm1.all <- allFit(fm1)
  ss <- summary(fm1.all)
  ss$ fixef          ## fixed effects
  ss$ llik           ## log-likelihoods
  ss$ sdcor          ## SDs and correlations
  ss$ theta          ## Cholesky factors
  ss$ which.OK       ## which fits worked

```

}

culcitalogreg	<i>Coral-eating seastar Culcita novaeguineae data (binary predation version)</i>
---------------	--

Description

This data shows the results from a block design experiment to record the interactions of ‘guard’ crustaceans, shrimp (*Alpheus lottini*) and crab (*Trapezia serenei*), that defend their coral host (*Pocillopora cf. meandrina*) from seastar predators (*Culcita novaeguineae*) (McKeon et al. 2012). The coral and their seastar predator were placed in large, octagonal, flow-through seawater tanks measuring approximately 0.5m deep and 2m in diameter. There were 10 tanks in total, each serving as an experimental block. Four treatment groups were tested: (1) no exosymbionts, (2) shrimp only, (3) crab only, and (4) both crab and shrimp present. Each treatment was replicated twice within each block.

Usage

```
data("culcitalogreg")
```

Format

A data frame with 80 observations on the following 9 variables.

`block` a numeric variable indicating the experimental block. There are 10 blocks in total, each corresponding to a large, octagonal, flow-through seawater tank approximately 0.5m deep and 2m in diameter.

`ttt` a number which represents the combinations of different symbionts.

- 1: no exosymbionts,
- 2: pair of *Alpheus lottini* only (Alpheus; shrimp),
- 3: pair of *Trapezia serenei* only (Trapezia; crab),
- 4: pair of *Alpheus lottini* and pair of *Trapezia serenei* (‘Alpheus and Trapezia’).

`predation` (binary) indicates whether the seastar predator consumed the coral. 0 = did not eat; 1 = ate.

`ttt.1` a relabelled version of `ttt` that explicitly indicates the corresponding treatment condition in plain language.

`crab` describes whether the crab was present in the experiment. C-: not present, C+: present.

`shrimp` describes whether the shrimp was present in the experiment. S-: not present, S+: present.

`ttt2` a relabelled version of `ttt` that uses letters (a, b, c, d) instead of numeric values (1, 2, 3, 4), respectively, to represent the treatment groups.

`crab2` a relabelling of `crab` where n corresponds to C- (no crab), and y corresponds to C+ (crab present).

`shrimp2` a relabelling of `shrimp` where n corresponds to S- (no shrimp), and y corresponds to S+ (shrimp present).

Details

Full details of the experiment (McKeon et al. 2012), is quoted below: "Twenty replicates in total were conducted for each exosymbiont treatment in a temporally blocked design with two replicates in each of ten temporal blocks. Trials were conducted in a large octagonal flow-through seawater tank approximately 0.5 m deep and 2 m across. The tank was divided into eight equal sections using plastic screening. Each section was provisioned with a seastar refugium constructed from concrete blocks. We placed Pocillopora colonies into the tank from the field during mid- to late afternoon. To minimize variation driven by search time, a single Culcita was placed directly on top of the coral colony. The following morning, we measured the coral size (length, width, height) and the feeding scars left by the Culcita (length, width, depth)."

Source

(McKeon et al. 2012)

References

McKeon CS, Stier AC, McIlroy SE, Bolker BM (2012). "Multiple defender effects: synergistic coral defense by mutualist crustaceans." *Oecologia*, **169**(4), 1095–1103. doi:10.1007/s00442012-22752.

See Also

The version which only contains predation counts but includes the amount of volume of the coral and the amount of volume lost, [culcitalvolume](#).

Examples

```
cul_mod <- glmer(predation ~ ttt2 + (1|block), data=culcitalogreg,  
                family = binomial(link = "logit"))
```

culcitalvolume *Coral-eating seastar Culcita novaeguineae data (volume loss version)*

Description

This is the same experiment outlined in [culcitalogreg](#) except this data set only contains values where predation occurred, and the original volume as well as volume lost was recorded.

Usage

```
data("culcitalvolume")
```

Format

A data frame with 50 observations on the following 7 variables.

`ttt` represents the combinations of different symbionts, which the treatment condition is explicitly written in plain language.

`volume` describes the size of the coral, measured in cm^3 .

`predvolume` the amount of volume loss from the coral, measured in cm^3 .

`crab` describes whether the crab was present in the experiment. n: not present, y: present.

`shrimp` describes whether the shrimp was present in the experiment. n: not present, y: present.

`block` a numeric variable indicating the experimental block. There are 10 blocks in total, each corresponding to a large, octagonal, flow-through seawater tank approximately 0.5m deep and 2m in diameter.

`ttt2` a relabelled version of `ttt`.

- 1: no exosymbionts,
- 2: pair of *Alpheus lottini* only (Alpheus; shrimp),
- 3: pair of *Trapezia serenei* only (Trapezia; crab),
- 4: pair of *Alpheus lottini* and pair of *Trapezia serenei* ('Alpheus and Trapezia').

Source

(McKeon et al. 2012)

References

McKeon CS, Stier AC, McIlroy SE, Bolker BM (2012). "Multiple defender effects: synergistic coral defense by mutualist crustaceans." *Oecologia*, **169**(4), 1095–1103. doi:10.1007/s00442012-22752.

See Also

The version which contains whether or not predation occurred, [culcitalogreg](#).

Examples

```
## Modifying to create a new response variable
vdata <- transform(culcitalvolume,
                  propeaten = predvolume/volume,
                  tvol = log(predvolume))
## One-way analysis
(cvm1 <- lmer(tvol ~ ttt2 + (1|block), data = vdata))
(cvm2 <- lmer(propeaten ~ ttt2 + (1|block), data = vdata))
## Two-way analysis
(cvm3 <- lmer(tvol ~ crab*shrimp + (1|block), data = vdata))
(cvm4 <- lmer(propeaten ~ crab*shrimp + (1|block), data = vdata))
```

devcomp	<i>Extract the deviance component list</i>
---------	--

Description

Return the deviance component list

Usage

```
devcomp(x)
```

Arguments

x a fitted model of class `merMod`

Details

A fitted model of class `merMod` has a `devcomp` slot as described in the value section.

Value

a list with components

`dims` a named integer vector of various dimensions

`cmp` a named numeric vector of components of the deviance

Note

This function is deprecated, use `getME(., "devcomp")`

devfun2	<i>Deviance Function in Terms of Standard Deviations/Correlations</i>
---------	---

Description

The deviance is profiled with respect to the fixed-effects parameters but not with respect to sigma; that is, the function takes parameters for the variance-covariance parameters and for the residual standard deviation. The random-effects variance-covariance parameters are on the standard deviation/correlation scale, not the theta (Cholesky factor) scale.

Usage

```
devfun2(fm, useSc = if(isLMM(fm)) TRUE else NA,
        scale = c("sdcor", "varcov"), ...)
```

Arguments

fm	a fitted model inheriting from class "merMod".
useSc	(logical) indicating whether a scale parameter has been in the model or should be used.
scale	a character string indicating the scale of the argument of the deviance function.
...	arguments passed to the internal profnames function (signames=TRUE to use old-style .sigxx names, FALSE uses (sd_corlxx); also prefix=c("sd", "cor"))

Value

Returns a function that takes a vector of standard deviations and correlations and returns the deviance (or REML criterion). The function has additional attributes

optimum a named vector giving the parameter values at the optimum

basedev the deviance at the optimum, (i.e., *not* the REML criterion).

thopt the optimal variance-covariance parameters on the "theta" (Cholesky factor) scale

stderr standard errors of fixed effect parameters

Note

Even if the original model was fitted using REML=TRUE as by default with `lmer()`, this returns the deviance, i.e., the objective function for maximum (log) likelihood (ML).

For the REML objective function, use `getME(fm, "devfun")` instead.

Examples

```
m1 <- lmer(Reaction~Days+(Days|Subject),sleepstudy)
dd <- devfun2(m1, useSc=TRUE)
pp <- attr(dd, "optimum")
## extract variance-covariance and residual std dev parameters
sigpars <- pp[grepl("^\\.sig", names(pp))]
all.equal(unname(dd(sigpars)), deviance(refitML(m1)))
```

drop1.merMod

Drop all possible single fixed-effect terms from a mixed effect model

Description

Drop allowable single terms from the model: see `drop1` for details of how the appropriate scope for dropping terms is determined.

Usage

```
## S3 method for class 'merMod'
drop1(object, scope, scale = 0,
       test = c("none", "Chisq", "user"),
       k = 2, trace = FALSE, sumFun, ...)
```

Arguments

object	a fitted merMod object.
scope	a formula giving the terms to be considered for adding or dropping.
scale	Currently ignored (included for S3 method compatibility)
test	should the results include a test statistic relative to the original model? The χ^2 test is a likelihood-ratio test, which is approximate due to finite-size effects.
k	the penalty constant in AIC
trace	print tracing information?
sumFun	a summary <code>function</code> to be used when <code>test=="user"</code> . It must allow arguments <code>scale</code> and <code>k</code> , but these may be ignored (e.g. swallowed by <code>...</code> , see the examples). The first two arguments must be <code>object</code> , the full model fit, and <code>objectDrop</code> , a reduced model. If <code>objectDrop</code> is missing, <code>sumFun(*)</code> should return a vector with the appropriate length and names (the actual contents are ignored).
...	other arguments (ignored)

Details

drop1 relies on being able to find the appropriate information within the environment of the formula of the original model. If the formula is created in an environment that does not contain the data, or other variables passed to the original model (for example, if a separate function is called to define the formula), then drop1 will fail. A workaround (see example below) is to manually specify an appropriate environment for the formula.

Value

An object of class `anova` summarizing the differences in fit between the models.

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
## likelihood ratio tests
drop1(fm1, test="Chisq")
## use Kenward-Roger corrected F test, or parametric bootstrap,
## to test the significance of each dropped predictor
if (require(pbkrtest) && packageVersion("pbkrtest") >= "0.3.8") {
  KRSumFun <- function(object, objectDrop, ...) {
    krnames <- c("ndf", "ddf", "Fstat", "p.value", "F.scaling")
    r <- if (missing(objectDrop)) {
      setNames(rep(NA, length(krnames)), krnames)
    } else {
      krtest <- KRmodcomp(object, objectDrop)
      unlist(krtest$stats[krnames])
    }
    attr("method") <- c("Kenward-Roger via pbkrtest package")
    r
  }
  drop1(fm1, test="user", sumFun=KRSumFun)
```

```

    if(lme4::testLevel() >= 3) { ## takes about 16 sec
      nsim <- 100
      PBSumFun <- function(object, objectDrop, ...) {
pbnames <- c("stat", "p.value")
r <- if (missing(objectDrop)) {
  setNames(rep(NA, length(pbnames)), pbnames)
} else {
  pbtest <- PBmodcomp(object, objectDrop, nsim=nsim)
  unlist(pbtest$test[2, pbnames])
}
attr(r, "method") <- c("Parametric bootstrap via pbkrtest package")
r
  }
  system.time(drop1(fm1, test="user", sumFun=PBSumFun))
}
}
## workaround for creating a formula in a separate environment
createFormula <- function(resp, fixed, rand) {
  f <- reformulate(c(fixed, rand), response=resp)
  ## use the parent (createModel) environment, not the
  ## environment of this function (which does not contain 'data')
  environment(f) <- parent.frame()
  f
}
createModel <- function(data) {
  mf.final <- createFormula("Reaction", "Days", "(Days|Subject)")
  lmer(mf.final, data=data)
}
drop1(createModel(data=sleepstudy))

```

 dummy

Dummy variables (experimental)

Description

Largely a wrapper for `model.matrix` that accepts a factor, `f`, and returns a dummy matrix with `nlevels(f)-1` columns (the first column is dropped by default). Useful whenever one wishes to avoid the behaviour of `model.matrix` of always returning an `nlevels(f)`-column matrix, either by the addition of an intercept column, or by keeping one column for all levels.

Usage

```
dummy(f, levelsToKeep)
```

Arguments

<code>f</code>	An object coercible to factor .
<code>levelsToKeep</code>	An optional character vector giving the subset of <code>levels(f)</code> to be converted to dummy variables.

Value

A `model.matrix` with dummy variables as columns.

Examples

```
data(Orthodont, package="nlme")
lmer(distance ~ age + (age|Subject) +
      (0+dummy(Sex, "Female")|Subject), data = Orthodont)
```

Dyestuff

Yield of dyestuff by batch

Description

The Dyestuff data frame provides the yield of dyestuff (Naphthalene Black 12B) from 5 different preparations from each of 6 different batches of an intermediate product (H-acid). The Dyestuff2 data were generated data in the same structure but with a large residual variance relative to the batch variance.

Format

Data frames, each with 30 observations on the following 2 variables.

Batch a factor indicating the batch of the intermediate product from which the preparation was created.

Yield the yield of dyestuff from the preparation (grams of standard color).

Details

The Dyestuff data are described in Davies and Goldsmith (1972) as coming from “an investigation to find out how much the variation from batch to batch in the quality of an intermediate product (H-acid) contributes to the variation in the yield of the dyestuff (Naphthalene Black 12B) made from it. In the experiment six samples of the intermediate, representing different batches of works manufacture, were obtained, and five preparations of the dyestuff were made in the laboratory from each sample. The equivalent yield of each preparation as grams of standard colour was determined by dye-trial.”

The Dyestuff2 data are described in Box and Tiao (1973) as illustrating “the case where between-batches mean square is less than the within-batches mean square. These data had to be constructed for although examples of this sort undoubtedly occur in practice, they seem to be rarely published.”

References

- Davies OL, Goldsmith PL (eds.) (1972). *Statistical Methods in Research and Production*, 4th edition. Oliver and Boyd. Section 6.4.
- Box GEP, Tiao GC (1973). *Bayesian Inference in Statistical Analysis*. Addison-Wesley. Section 5.1.2.

Examples

```

require(lattice)
str(Dyestuff)
dotplot(reorder(Batch, Yield) ~ Yield, Dyestuff,
        ylab = "Batch", jitter.y = TRUE, aspect = 0.3,
        type = c("p", "a"))
dotplot(reorder(Batch, Yield) ~ Yield, Dyestuff2,
        ylab = "Batch", jitter.y = TRUE, aspect = 0.3,
        type = c("p", "a"))
(fm1 <- lmer(Yield ~ 1|Batch, Dyestuff))
(fm2 <- lmer(Yield ~ 1|Batch, Dyestuff2))

```

factorize

*Attempt to convert grouping variables to factors***Description**

If variables within a data frame are not factors, try to convert them. Not intended for end-user use; this is a utility function that needs to be exported, for technical reasons.

Usage

```
factorize(x, frloc, char.only=FALSE)
```

Arguments

x a formula
frloc a data frame
char.only (logical) convert only character variables to factors?

Value

a copy of the data frame with factors converted

fixef

*Extract fixed-effects estimates***Description**

Extract the fixed-effects estimates

Usage

```

## S3 method for class 'merMod'
fixef(object, add.dropped=FALSE, noScale = NULL, ...)

```

Arguments

object	any fitted model object from which fixed effects estimates can be extracted.
add.dropped	for models with rank-deficient design matrix, reconstitute the full-length parameter vector by adding NA values in appropriate locations?
noScale	(logical) some users may use autoscaling. If so, the default returns the scaled parameters. If noScale = TRUE, then fixef returns the non-scaled parameters instead.
...	optional additional arguments. Currently none are used in any methods.

Details

Extract the estimates of the fixed-effects parameters from a fitted model.

Value

a named, numeric vector of fixed-effects estimates.

Examples

```
fixef(lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy))
fm2 <- lmer(Reaction ~ Days + Days2 + (1|Subject),
           data=transform(sleepstudy, Days2=Days))
fixef(fm2, add.dropped=TRUE)
## first two parameters are the same ...
stopifnot(all.equal(fixef(fm2, add.dropped=TRUE)[1:2],
                   fixef(fm2)))
```

fortify

add information to data based on a fitted model

Description

fortify adds information to data based on a fitted model; getData retrieves data as specified in the data argument

Usage

```
fortify.merMod(model, data = getData(model),
              ...)
## S3 method for class 'merMod'
getData(object)
```

Arguments

model	fitted model
object	fitted model
data	original data set, if needed
...	additional arguments

Details

- `fortify` is defined in the **ggplot2** package, q.v. for more details. `fortify` is *not* defined here, and `fortify.merMod` is defined as a function rather than an S3 method, to avoid (1) inducing a dependency on **ggplot2** or (2) masking methods from **ggplot2**. This feature is both experimental and semi-deprecated, as the help page for `fortify` itself says: “Rather than using this function, I now recommend using the `broom` package, which implements a much wider range of methods. `fortify` may be deprecated in the future.” The `broom.mixed` package is recommended for mixed models in general.
- `getData` is a bare-bones implementation; it relies on a `data` argument having been specified and the data being available in the environment of the formula. Unlike the functions in the `nlme` package, it does not do anything special with `na.action` or `subset`.

Examples

```
fm1 <- lmer(Reaction~Days+(1|Subject),sleepstudy)
names(fortify.merMod(fm1))
```

`getDoublevertDefault` *Retrieve default double-vertical-bar-expansion method*

Description

Retrieve information about whether random effect terms specified with double bars (`|`) should be split semantically into separate, intercept-free terms, or encoded with a `diag()` special

Usage

```
getDoublevertDefault()
```

Details

To set the global behaviour retrieved by `getDoublevertDefault()` to `"diag_special"`, run `options(lme4.doublevert.default = "diag_special")`

The default value of `"split"` is backward compatible with the behaviour of `lme4 < 2.0`; double bars do *not* work as typically expected with factors as varying terms. That is, if `g` is a factor then `(1 + g | f)` will expand (with method `"split"`) to `(1 | f) + (0 + g | f)`; the second term will *not* produce a covariance matrix where the contrasts generated by the factor are independent.

`"diag_special"` is probably the generally preferred option; by transforming `(1+a+b | f)` into `diag(1+a+b | f)`, it ensures that variation across groups in the contrasts coded by the factor will be independent. This matches the results of specifying a double-bar term in `afex::mixed()`, or `glmmTMB`.

Note that if `f` is a factor, even with `"diag_special"` set, models containing `(0+f || g)` vs. `(f || g)` will give different results. `(0+f || g)` creates a set of dummy variables that are indicator variables for each level of the factor, so the random effect term will describe variation in the intercept for each level of the factor. With default treatment contrasts set, `(f || g)` (equivalent to `(1+f || g)` because R automatically adds the intercept term) will estimate variation first in the intercept in the baseline level of the factor, then in the differences between observations in the baseline and successive levels of the factor (see [contr.treatment](#)).

Value

a (1-element) character vector suitable for reformulas::findbars_x specifying whether to expand random-effects terms containing double bars (||) as a set of independent random effects terms ("split": i.e., $(1+a+b||f)$ becomes $(1|f) + (0+a|f) + (0+b|f)$) or whether to specify that the covariance matrix is diagonal ("diag_special": i.e., $\text{diag}(1+a+b|f)$).

getME	<i>Extract or Get Generalized Components from a Fitted Mixed Effects Model</i>
-------	--

Description

Extract (or “get”) “components” – in a generalized sense – from a fitted mixed-effects model, i.e., (in this version of the package) from an object of class “merMod”.

Usage

```
getME(object, name, ...)

## S3 method for class 'merMod'
getME(object,
       name = c("X", "Z", "Zt", "Ztlist", "mmList", "y", "mu", "u", "b",
                "Gp", "Tp", "L", "Lambda", "Lambdat", "Lind", "Tlist",
                "A", "RX", "RZX", "sigma", "flist",
                "fixef", "beta", "theta", "ST", "par", "REML", "is_REML",
                "n_rtrms", "n_rfacs", "N", "n", "p", "q",
                "p_i", "l_i", "q_i", "k", "m_i", "m",
                "cnms", "devcomp", "offset", "lower", "devfun", "devarg",
                "glmer.nb.theta"),
       ...)
```

Arguments

object	a fitted mixed-effects model of class “merMod”, i.e., typically the result of <code>lmer()</code> , <code>glmer()</code> or <code>nlmer()</code> .
name	a character vector specifying the name(s) of the “component”. If <code>length(name) > 1</code> or if <code>name = "ALL"</code> , a named <code>list</code> of components will be returned. Possible values are: <ul style="list-style-type: none"> “X”: fixed-effects model matrix “Z”: random-effects model matrix “Zt”: transpose of random-effects model matrix. Note that the structure of Zt has changed since lme4.0; to get a backward-compatible structure, use <code>do.call(Matrix::rBind, getME(., "Ztlist"))</code> “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": 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, beginning with 0 and including a final element giving the total number of random effects
 "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 number of thetas.
 "L": sparse Cholesky factor of the penalized random-effects model.
 "Lambda": relative covariance factor Λ of the random effects.
 "Lambdat": transpose Λ' of Λ above.
 "Lind": index vector for inserting elements of θ into the nonzeros of Λ .
 "Tlist": vector of template matrices from which the blocks of Λ are generated.
 "A": Scaled sparse model matrix (class "dgCMatrix") for the unit, orthogonal random effects, U , equal to `getME(., "Zt") %*% getME(., "Lambdat")`
 "RX": Cholesky factor for the fixed-effects parameters
 "RZX": cross-term in the full Cholesky factor
 "sigma": residual standard error; note that `sigma(object)` is preferred.
 "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": lower triangular entries of Cholesky factors of relative random effect covariance matrices, stored in column-major order.
 "par": relative random effect covariance parameter estimates.
 "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 .
 "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 `dims@nth` below.
 "cnms": the "component names", a [list](#).
 "REML": 0 indicates the model was fitted by maximum likelihood, any other positive integer indicates fitting by restricted maximum likelihood
 "is_REML": same as the result of `isREML(.)`
 "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 twice the log determinant of L
ldRX2 twice the log determinant of RX
wrss weighted residual sum of squares
ussq squared length of u
pwrss penalized weighted residual sum of squares, "wrss + ussq"
drsum sum of residual deviance (GLMMs only)
REML REML criterion at optimum (LMMs fit by REML only)
dev deviance criterion at optimum (models fit by ML only)
sigmaML ML estimate of residual standard deviation
sigmaREML REML estimate of residual standard deviation
tolPwrss tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares (GLMMs only)

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`.)

```

"devfun": deviance function.
"devarg": deviance function argument, the concatenation of getME(object,
"par") and (if object is a glmerMod object) getME(object, "beta").
"glmer.nb.theta": negative binomial  $\theta$  parameter, only for glmer.nb.
"ALL": get all of the above as a list.
... currently unused in lme4, potentially further arguments in methods.

```

Details

The goal is to provide “everything a user may want” from a fitted “merMod” 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 “merMod” objects, such as `ranef`, `fixef`, `vcov`, etc.: see `methods(class="merMod")`

Examples

```

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

(fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy))
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))

```

Description

Returns a list with one covariance object for each random effect in the model. If the object is an old-style (pre-version 2.0.0) fitted model, it automatically converts all terms to `Covariance.us` objects (except for terms including double-bar ‘||’ notation, which are converged to `Covariance.diag` objects).

Usage

```
getReCovs(object)
```

Arguments

object an R object of class `merMod`, i.e., as resulting from `lmer()`, or `glmer()`, etc.

Value

a list of covariance objects

Examples

```
m1 <- lmer(Reaction ~ Days + (Days || Subject), sleepstudy)
sapply(getReCovs(m1), "class")
```

GHrule

Univariate Gauss-Hermite quadrature rule

Description

Create a univariate Gauss-Hermite quadrature rule.

Usage

```
GHrule(ord, asMatrix = TRUE)
```

Arguments

ord scalar integer between 1 and 100 - the order, or number of nodes and weights, in the rule. When the function being multiplied by the standard normal density is a polynomial of order $2k - 1$ the rule of order k integrates the product exactly.

asMatrix logical scalar - should the result be returned as a matrix. If FALSE a data frame is returned. Defaults to TRUE.

Details

This version of Gauss-Hermite quadrature provides the node positions and weights for a scalar integral of a function multiplied by the standard normal density.

Originally based on package **SparseGrid**'s `hiddenGQN()`, then on **fastGHQuad**'s `gaussHermiteData(.)`.

Value

a matrix (or data frame, if `asMatrix` is false) with `ord` rows and three columns which are `z` the node positions, `w` the weights and `ldnorm`, the logarithm of the normal density evaluated at the nodes.

References

Liu Q, Pierce DA (1994). "A note on Gauss—Hermite quadrature." *Biometrika*, **81**(3), 624–629. doi:10.2307/2337136.

See Also

a different interface is available via `GQdk()`.

Examples

```
(r5 <- GHrule( 5, asMatrix=FALSE))
(r12 <- GHrule(12, asMatrix=FALSE))

## second, fourth, sixth, eighth and tenth central moments of the
## standard Gaussian N(0,1) density:
ps <- seq(2, 10, by = 2)
cbind(p = ps, "E[X^p]" = with(r5, sapply(ps, function(p) sum(w * z^p)))) # p=10 is wrong for 5-rule
p <- 1:15
GQ12 <- with(r12, sapply(p, function(p) sum(w * z^p)))
cbind(p = p, "E[X^p]" = zapsmall(GQ12))
## standard R numerical integration can do it too:
intL <- lapply(p, function(p) integrate(function(x) x^p * dnorm(x),
                                       -Inf, Inf, rel.tol=1e-11))

integR <- sapply(intL, `[`, "value")
cbind(p, "E[X^p]" = integR)# no zapsmall() needed here
all.equal(GQ12, integR, tol=0)# => shows small difference
stopifnot(all.equal(GQ12, integR, tol = 1e-10))
(xactMom <- cumprod(seq(1,13, by=2)))
stopifnot(all.equal(xactMom, GQ12[2*(1:7)], tol=1e-14))
## mean relative errors :
mean(abs(GQ12 [2*(1:7)] / xactMom - 1)) # 3.17e-16
mean(abs(integR[2*(1:7)] / xactMom - 1)) # 9.52e-17 {even better}
```

glmer

Fitting Generalized Linear Mixed-Effects Models

Description

Fit a generalized linear mixed-effects model (GLMM). Both fixed effects and random effects are specified via the model formula.

Usage

```
glmer(formula, data = NULL, family = gaussian
      , control = glmerControl()
      , start = NULL
      , verbose = 0L
      , nAGQ = 1L
      , subset, weights, na.action, offset, contrasts = NULL
```

```
, mustart, etastart
, devFunOnly = FALSE)
```

Arguments

formula	a two-sided linear formula object describing both the fixed-effects and random-effects part of the model, with the response on the left of a <code>~</code> operator and the terms, separated by <code>+</code> operators, on the right. Random-effects terms are distinguished by vertical bars (" <code> </code> ") separating expressions for design matrices from grouping factors.
data	an optional data frame containing the variables named in <code>formula</code> . By default the variables are taken from the environment from which <code>lmer</code> is called. While <code>data</code> is optional, the package authors <i>strongly</i> recommend its use, especially when later applying methods such as <code>update</code> and <code>drop1</code> to the fitted model (<i>such methods are not guaranteed to work properly if data is omitted</i>). If <code>data</code> is omitted, variables will be taken from the environment of <code>formula</code> (if specified as a formula) or from the parent frame (if specified as a character vector).
family	a GLM family, see glm and family .
control	a list (of correct class, resulting from <code>lmerControl()</code> or <code>glmerControl()</code> respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the <code>*lmerControl</code> documentation for details.
start	a numeric vector or a named list with optional components, one named <code>par</code> or <code>theta</code> and another named <code>fixef</code> or <code>beta</code> , giving starting values for covariance parameters and fixed effect coefficients, respectively. Numeric <code>start</code> is equivalent to <code>list(par = start)</code> . Parameters corresponding to unstructured covariance matrices are on the scale of the Cholesky factor of the relative covariance matrix. By default, all relative covariance matrices are identity matrices and all fixed effect coefficients are zero. If two optimization steps are performed, as by default, then the first step uses starting values <code>par</code> and the second step uses starting values <code>c(newpar, fixef)</code> , where <code>newpar</code> is the optimum obtained in the first step. See modular for details about finer control of optimization.
verbose	integer scalar. If <code>> 0</code> verbose output is generated during the optimization of the parameter estimates. If <code>> 1</code> verbose output is generated during the individual penalized iteratively reweighted least squares (PIRLS) steps.
nAGQ	integer scalar - the number of points per axis for evaluating the adaptive Gauss-Hermite approximation to the log-likelihood. Defaults to 1, corresponding to the Laplace approximation. Values greater than 1 produce greater accuracy in the evaluation of the log-likelihood at the expense of speed. A value of zero uses a faster but less exact form of parameter estimation for GLMMs by optimizing the random effects and the fixed-effects coefficients in the penalized iteratively reweighted least squares step. (See Details .)
subset	an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

weights	an optional vector of ‘prior weights’ to be used in the fitting process. Should be NULL or a numeric vector.
na.action	a function that indicates what should happen when the data contain NAs. The default action (<code>na.omit</code> , inherited from the ‘factory fresh’ value of <code>getOption("na.action")</code>) strips any observations with any missing values in any variables.
offset	this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more <code>offset</code> terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See <code>model.offset</code> .
contrasts	an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .
mustart	optional starting values on the scale of the conditional mean, as in <code>glm</code> ; see there for details.
etastart	optional starting values on the scale of the unbounded predictor as in <code>glm</code> ; see there for details.
devFunOnly	logical - return only the deviance evaluation function. Note that because the deviance function operates on variables stored in its environment, it may not return <i>exactly</i> the same values on subsequent calls (but the results should always be within machine tolerance).

Details

Fit a generalized linear mixed model, which incorporates both fixed-effects parameters and random effects in a linear predictor, via maximum likelihood. The linear predictor is related to the conditional mean of the response through the inverse link function defined in the GLM family.

The expression for the likelihood of a mixed-effects model is an integral over the random effects space. For a linear mixed-effects model (LMM), as fit by `lmer`, this integral can be evaluated exactly. For a GLMM the integral must be approximated. The most reliable approximation for GLMMs is adaptive Gauss-Hermite quadrature, at present implemented only for models with a single scalar random effect. The `nAGQ` argument controls the number of nodes in the quadrature formula. A model with a single, scalar random-effects term could reasonably use up to 25 quadrature points per scalar integral (although the code allows for up to 100 quadrature points).

Value

An object of class `merMod` (more specifically, an object of *subclass* `glmerMod`) for which many methods are available (e.g. `methods(class="merMod")`)

Note

In earlier version of the `lme4` package, a `method` argument was used. Its functionality has been replaced by the `nAGQ` argument.

See Also

`lmer` (for details on formulas and parameterization); `glm` for Generalized Linear Models (*without* random effects). `nlmer` for nonlinear mixed-effects models.

`glmer.nb` to fit negative binomial GLMMs.

Examples

```

## generalized linear mixed model
library(lattice)
xyplot(incidence/size ~ period|herd, cbpp, type=c('g','p','l'),
       layout=c(3,5), index.cond = function(x,y)max(y))
(gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             data = cbpp, family = binomial))
## using nAGQ=0 only gets close to the optimum
(gm1a <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             cbpp, binomial, nAGQ = 0))
## using nAGQ = 9 provides a better evaluation of the deviance
## Currently the internal calculations use the sum of deviance residuals,
## which is not directly comparable with the nAGQ=0 or nAGQ=1 result.
## 'verbose = 1' monitors iteratin a bit; (verbose = 2 does more):
(gm1a <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             cbpp, binomial, verbose = 1, nAGQ = 9))

## GLMM with individual-level variability (accounting for overdispersion)
## For this data set the model is the same as one allowing for a period:herd
## interaction, which the plot indicates could be needed.
cbpp$obs <- 1:nrow(cbpp)
(gm2 <- glmer(cbind(incidence, size - incidence) ~ period +
             (1 | herd) + (1|obs),
             family = binomial, data = cbpp))
anova(gm1,gm2)

## glmer and glm log-likelihoods are consistent
gm1Devfun <- update(gm1,devFunOnly=TRUE)
gm0 <- glm(cbind(incidence, size - incidence) ~ period,
          family = binomial, data = cbpp)
## evaluate GLMM deviance at RE variance=theta=0, beta=(GLM coeffs)
gm1Dev0 <- gm1Devfun(c(0,coef(gm0)))
## compare
stopifnot(all.equal(gm1Dev0,c(-2*logLik(gm0))))
## the toenail oncholysis data from Backer et al 1998
## these data are notoriously difficult to fit
## Not run:
if (require("HSAUR3")) {
  gm2 <- glmer(outcome~treatment*visit+(1|patientID),
              data=toenail,
              family=binomial,nAGQ=20)
}

## End(Not run)

```

Description

Fits a generalized linear mixed-effects model (GLMM) for the negative binomial family, building on `glmer`, and initializing via `theta.ml` from **MASS**.

Usage

```
glmer.nb(..., interval = log(th) + c(-3, 3),
          tol = 5e-5, verbose = FALSE, nb.control = NULL,
          initCtrl = list(limit = 20, eps = 2*tol, trace = verbose,
                          theta = NULL))
```

Arguments

<code>...</code>	arguments as for <code>glmer(.)</code> such as <code>formula</code> , <code>data</code> , <code>control</code> , etc, but <i>not</i> <code>family</code> !
<code>interval</code>	interval in which to start the optimization. The default is symmetric on log scale around the initially estimated <code>theta</code> .
<code>tol</code>	tolerance for the optimization via <code>optimize</code> .
<code>verbose</code>	<code>logical</code> indicating how much progress information should be printed during the optimization. Use <code>verbose = 2</code> (or larger) to enable <code>verbose=TRUE</code> in the <code>glmer()</code> calls.
<code>nb.control</code>	optional <code>list</code> , like the output of <code>glmerControl()</code> , used in <code>refit(*, control = control.nb)</code> during the optimization (<code>control</code> , if included in <code>...</code> , will be used in the initial-stage <code>glmer(..., family=poisson)</code> fit, and passed on to the later optimization stages as well)
<code>initCtrl</code>	<i>(experimental, do not rely on this:)</i> a <code>list</code> with named components as in the default, passed to <code>theta.ml</code> (package MASS) for the initial value of the negative binomial parameter <code>theta</code> . May also include a <code>theta</code> component, in which case the initial estimation step is skipped

Value

An object of class `glmerMod`, for which many methods are available (e.g. `methods(class="glmerMod")`), see `glmer`.

Note

For historical reasons, the shape parameter of the negative binomial and the random effects parameters in our (G)LMM models are both called `theta` (θ), but are unrelated here.

The negative binomial θ can be extracted from a fit `g <- glmer.nb()` by `getME(g, "glmer.nb.theta")`.

Parts of `glmer.nb()` are still experimental and methods are still missing or suboptimal. In particular, there is no inference available for the dispersion parameter θ , yet.

To fit a negative binomial model with *known* overdispersion parameter (e.g. as part of a model comparison exercise, use `glmer` with the `negative.binomial` family from the **MASS** package, e.g. `glmer(..., family=MASS::negative.binomial(theta=1.75))`).

See Also

[glmer](#); from package [MASS](#), [negative.binomial](#) (which we re-export currently) and [theta.ml](#), the latter for initialization of optimization.

The ‘Details’ of [pnbinom](#) for the definition of the negative binomial distribution.

Examples

```
set.seed(101)
dd <- expand.grid(f1 = factor(1:3),
                 f2 = LETTERS[1:2], g=1:9, rep=1:15,
                 KEEP.OUT.ATTRS=FALSE)
summary(mu <- 5*(-4 + with(dd, as.integer(f1) + 4*as.numeric(f2))))
dd$y <- rnbinom(nrow(dd), mu = mu, size = 0.5)
str(dd)
require("MASS")## and use its glm.nb() - as indeed we have zero random effect:
## Not run:
m.glm <- glm.nb(y ~ f1*f2, data=dd, trace=TRUE)
summary(m.glm)
m.nb <- glmer.nb(y ~ f1*f2 + (1|g), data=dd, verbose=TRUE)
m.nb
## The neg.binomial theta parameter:
getME(m.nb, "glmer.nb.theta")
LL <- logLik(m.nb)
## mixed model has 1 additional parameter (RE variance)
stopifnot(attr(LL,"df")==attr(logLik(m.glm),"df")+1)
plot(m.nb, resid(.) ~ g)# works, as long as data 'dd' is found

## End(Not run)
```

glmerLaplaceHandle *Handle for glmerLaplace*

Description

Handle for calling the glmerLaplace C++ function. Not intended for routine use.

Usage

```
glmerLaplaceHandle(pp, resp, nAGQ, tol, maxit, verbose)
```

Arguments

pp	merPredD object
resp	lmResp object
nAGQ	see glmer
tol	tolerance
maxit	maximum number of pwrss iterations
verbose	display optimizer progress

Value

Value of the objective function

glmFamily	<i>Generator object for the glmFamily class</i>
-----------	---

Description

The generator object for the [glmFamily](#) reference class. Such an object is primarily used through its new method.

Usage

```
glmFamily(...)
```

Arguments

... Named argument (see Note below)

Methods

new(family=family) Create a new [glmFamily](#) object

Note

Arguments to the new method must be named arguments.

See Also

[glmFamily](#)

glmFamily-class	<i>Class "glmFamily" - a reference class for family</i>
-----------------	---

Description

This class is a wrapper class for [family](#) objects specifying a distribution family and link function for a generalized linear model ([glm](#)). The reference class contains an external pointer to a C++ object representing the class. For common families and link functions the functions in the family are implemented in compiled code so they can be accessed from other compiled code and for a speed boost.

Extends

All reference classes extend and inherit methods from "[envRefClass](#)".

Note

Objects from this reference class correspond to objects in a C++ class. Methods are invoked on the C++ class using the external pointer in the Ptr field. When saving such an object the external pointer is converted to a null pointer, which is why there is a redundant field ptr that is an active-binding function returning the external pointer. If the Ptr field is a null pointer, the external pointer is regenerated for the stored family field.

See Also

[family](#), [glmFamily](#)

Examples

```
str(glmFamily$new(family=poisson()))
```

golden-class

Class "golden" and Generator for Golden Search Optimizer Class

Description

"golden" is a reference class for a golden search scalar optimizer, for a parameter within an interval. golden() is the generator for the "golden" class. The optimizer uses reverse communications.

Usage

```
golden(...)
```

Arguments

... (partly optional) arguments passed to [new\(\)](#) must be named arguments. lower and upper are the bounds for the scalar parameter; they must be finite.

Extends

All reference classes extend and inherit methods from "[envRefClass](#)".

Examples

```
showClass("golden")
```

```
golden(lower= -100, upper= 1e100)
```

gopherdat2

Gopher tortoises shell remains

Description

Data of fresh Gopher tortoise shell remains (Ozgul et al. 2009).

Usage

```
data("gopherdat2")
```

Format

A data frame with 30 observations on the following 7 variables.

`Site` a factor representing the sampling site. There are 10 levels:

- BS (Big Shoals State Park).
- CB (Camp Blanding Wildlife Management Area).
- Cent (privately-owned property in central Florida).
- CF (Cecil Field/Branan Field Wildlife and Environmental Area).
- FC (Fort Cooper State Park).
- FE (Flying Eagle Wildlife Management Area).
- GH (Gold Head Branch State Park).
- Old (Perry Oldenburg Wildlife and Environmental Area).
- Ord (Ordway-Swisher Biological Station).
- TE (Tenoroc Fish Management Area).

`year` a numeric vector of the sampling year.

`shells` a numeric vector of the number of shells found.

`Area` a numeric vector representing site area (units unknown).

`density` a numeric vector representing population density.

`prev` a numeric vector representing the seroprevalence (frequency of antibodies to disease) of *M. agassizi*.

Details

Details of the study is described in (Ozgul et al. 2009) as follows: "The fieldwork was conducted between 2003 and 2006, during late spring/summer (May–September). Systematic surveys were conducted to locate tortoise burrows and shell remains from deceased tortoises at study sites and consisted of a line of four to eight observers spaced 10 m apart walking parallel transects across the study area."

Source

(Ozgul et al. 2009)

References

Ozgul A, Oli MK, Bolker BM, Perez-Heydrich C (2009). “Upper respiratory tract disease, force of infection, and effects on survival of gopher tortoises.” *Ecological Applications*, **19**(3), 786-798. doi:10.1890/080219.1.

Examples

```
## Simple model gives a singular fit:
gopher_glmer <- glmer(shells ~ factor(year) + prev + offset(log(Area))
  + (1|Site), data = gopherdat2, family = "poisson")
## The site-level variance for this model is indeed zero:
VarCorr(gopher_glmer)
## So a Poisson GLM gives the same answer here:
gopher_glm <- glm(shells ~ factor(year) + prev + offset(log(Area)),
  data = gopherdat2, family = "poisson")
all.equal(fixef(gopher_glmer), coef(gopher_glm))
```

GQdk

Sparse Gaussian / Gauss-Hermite Quadrature grid

Description

Generate the sparse multidimensional Gaussian quadrature grids.

Currently unused. See [GHrule\(\)](#) for the version currently in use in package **lme4**.

Usage

```
GQdk(d = 1L, k = 1L)
GQN
```

Arguments

d	integer scalar - the dimension of the function to be integrated with respect to the standard d-dimensional Gaussian density.
k	integer scalar - the order of the grid. A grid of order k provides an exact result for a polynomial of total order of $2k - 1$ or less.

Value

GQdk() returns a matrix with $d + 1$ columns. The first column is the weights and the remaining d columns are the node coordinates.

GQN is a [list](#) of lists, containing the non-redundant quadrature nodes and weights for integration of a scalar function of a d-dimensional argument with respect to the density function of the d-dimensional Gaussian density function.

The outer list is indexed by the dimension, d, in the range of 1 to 20. The inner list is indexed by k, the order of the quadrature.

Note

GQN contains only the non-redundant nodes. To regenerate the whole array of nodes, all possible permutations of axes and all possible combinations of ± 1 must be applied to the axes. This entire array of nodes is exactly what `GQdk()` reproduces.

The number of nodes gets very large very quickly with increasing d and k . See the charts at <http://www.sparse-grids.de>.

Examples

```
GQdk(2,5) # 53 x 3
```

```
GQN[[3]][[5]] # a 14 x 4 matrix
```

grouseticks

Data on red grouse ticks from Elston et al. 2001

Description

Number of ticks on the heads of red grouse chicks sampled in the field (`grouseticks`) and an aggregated version (`grouseticks_agg`); see original source for more details

Usage

```
data(grouseticks)
```

Format

INDEX (factor) chick number (observation level)
TICKS number of ticks sampled
BROOD (factor) brood number
HEIGHT height above sea level (meters)
YEAR year (-1900)
LOCATION (factor) geographic location code
cHEIGHT centered height, derived from HEIGHT
meanTICKS mean number of ticks by brood
varTICKS variance of number of ticks by brood

Details

`grouseticks_agg` is just a brood-level aggregation of the data

Source

Robert Moss, via David Elston

References

Elston DA, Moss R, Boulinier T, Arrowsmith C, Lambin X (2001). "Analysis of aggregation, a worked example: numbers of ticks on red grouse chicks." *Parasitology*, **122**(5), 563–569. doi:10.1017/S0031182001007740.

Examples

```

if (interactive()) {
data(grouseticks)
## Figure 1a from Elston et al
par(las=1,bty="l")
tvec <- c(0,1,2,5,20,40,80)
pvec <- c(4,1,3)
with(grouseticks_agg,plot(1+meanTICKS~HEIGHT,
                          pch=pvec[factor(YEAR)],
                          log="y",axes=FALSE,
                          xlab="Altitude (m)",
                          ylab="Brood mean ticks"))

axis(side=1)
axis(side=2,at=tvec+1,label=tvec)
box()
abline(v=405,lty=2)
## Figure 1b
with(grouseticks_agg,plot(varTICKS~meanTICKS,
                          pch=4,
                          xlab="Brood mean ticks",
                          ylab="Within-brood variance"))
curve(1*x,from=0,to=70,add=TRUE)
## Model fitting
form <- TICKS~YEAR+HEIGHT+(1|BROOD)+(1|INDEX)+(1|LOCATION)
(full_mod1 <- glmer(form, family="poisson",data=grouseticks))
}

```

hatvalues.merMod

Diagonal elements of the hat matrix

Description

Returns the values on the diagonal of the hat matrix, which is the matrix that transforms the response vector (minus any offset) into the fitted values (minus any offset). Note that this method should only be used for linear mixed models. It is not clear if the hat matrix concept even makes sense for generalized linear mixed models.

Usage

```

## S3 method for class 'merMod'
hatvalues(model, fullHatMatrix = FALSE, ...)

```

Arguments

model An object of class `merMod`.
 fullHatMatrix Return full hat matrix (not just diagonal values)?
 ... Not currently used

Value

The diagonal elements of the hat matrix.

Examples

```
m <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
hatvalues(m)
```

influence.merMod *Influence Diagnostics for Mixed-Effects Models*

Description

These functions compute deletion influence diagnostics for linear (fit by `lmer`) and generalized linear mixed-effects models (fit by `glmer`). The main functions are methods for the `influence` generic function. Other functions are provided for computing `dfbeta`, `dfbetas`, `cooks.distance`, and influence on variance-covariance components based on the objects computed by `influence.merMod`

Usage

```
## S3 method for class 'merMod'
influence(model, groups, data, maxfun = 1000,
          do.coef = TRUE, start = NULL,
          parallel = c("no", "multicore", "snow"),
          ncpus = getOption("influence.ncpus", 1L), cl = NULL, ncores, ...)
## S3 method for class 'influence.merMod'
cooks.distance(model, ...)
## S3 method for class 'influence.merMod'
dfbeta(model, which = c("fixed", "var.cov"), ...)
## S3 method for class 'influence.merMod'
dfbetas(model, ...)
```

Arguments

model in the case of `influence.merMod`, a model of class "merMod"; in the case of `cooks.distance`, `dfbeta`, or `dfbetas`, an object returned by `influence.merMod`

groups a character vector containing the name of a grouping factor or names of grouping factors; if more than one name is supplied, then groups are defined by all combinations of levels of the grouping factors that appear in the data. If omitted, then each individual row of the data matrix is treated as a "group" to be deleted in turn.

data	an optional data frame with the data to which model was fit; influence.merMod can usually retrieve the data used to fit the model, unless it can't be found in the current environment, so it's usually unnecessary to supply this argument.
maxfun	The maximum number of function evaluations (for influence.merMod) to perform after deleting each group; the defaults are large enough so that the iterations will typically continue to convergence. Setting to maxfun=20 for an lmer model or 100 for a glmer model will typically produce a faster reasonable approximation. An even smaller value can be used if interest is only in influence on the fixed effects.
which	if "fixed.effects" (the default), return influence on the fixed effects; if "var.cov", return influence on the variance-covariance components.
do.coef	if FALSE, skip potentially time-consuming computations, returning just a list containing hat values.
start	starting value for new fits (set to optimal values from original fit by default)
parallel	The type of parallel operation to be used (if any). If "multicore" (and R is running on a Unix-alike), then mclapply is used; if "snow", then parLapply is used.
ncpus	number of processes to be used in parallel operation.
cl	An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the call.
ncores	deprecated: equivalent to parallel = "snow" and ncpus = ncores; please use parallel, ncpus, and cl instead.
...	ignored.

Details

influence.merMod start with the estimated variance-covariance components from model and then refit the model omitting each group in turn, not necessarily iterating to completion. For example, maxfun=20 takes up to 20 function evaluations step away from the ML or REML solution for the full data, which usually provides decent approximations to the fully iterated estimates.

The other functions are methods for the `dfbeta`, `dfbetas`, and `cooks.distance` generics, to be applied to the "influence.merMod" object produced by the influence function; the dfbeta methods can also return influence on the variance-covariance components.

Value

influence.merMod returns objects of class "influence.merMod", which contain the following elements:

"fixed.effects" the estimated fixed effects for the model.

"fixed.effects[-groups]" a matrix with columns corresponding to the fixed-effects coefficients and rows corresponding to groups, giving the estimated fixed effects with each group deleted in turn; *groups* is formed from the name(s) of the grouping factor(s).

"var.cov.comps" the estimated variance-covariance parameters for the model.

"var.cov.comps[-groups]" a matrix with the estimated covariance parameters (in columns) with each group deleted in turn.

"vcov" The estimated covariance matrix of the fixed-effects coefficients.

"vcov[-groups]" a list each of whose elements is the estimated covariance matrix of the fixed-effects coefficients with one group deleted.

"groups" a character vector giving the names of the grouping factors.

"deleted" the possibly composite grouping factor, each of whose elements is deleted in turn.

"converged" for influence.merMod, a logical vector indicating whether the computation converged for each group.

"function.ivals" for influence.merMod, a vector of the number of function evaluations performed for each group.

For plotting "influence.merMod" objects, see [infIndexPlot](#).

Author(s)

J. Fox; parallel extensions by H. Bengtsson

References

Fox J, Weisberg S (2019). *An R companion to applied regression*, 3rd edition. Sage.

See Also

[infIndexPlot](#), [influence.measures](#)

Examples

```
if (interactive()) {
  fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
  inf_fm1 <- influence(fm1, "Subject")
  if (require("car")) {
    infIndexPlot(inf_fm1)
  }
  dfbeta(inf_fm1)
  dfbetas(inf_fm1)
  gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
              data = cbpp, family = binomial)
  inf_gm1 <- influence(gm1, "herd", maxfun=100)
  gm1.11 <- update(gm1, subset = herd != 11) # check deleting herd 11
  if (require("car")) {
    infIndexPlot(inf_gm1)
    compareCoefs(gm1, gm1.11)
  }
  if(packageVersion("car") >= "3.0.10") {
    dfbeta(inf_gm1)
    dfbetas(inf_gm1)
  }
}
```

Description

University lecture evaluations by students at ETH Zurich, anonymized for privacy protection. This is an interesting “medium” sized example of a *partially* nested mixed effect model.

Format

A data frame with 73421 observations on the following 7 variables.

`s` a factor with levels 1:2972 denoting individual students.

`d` a factor with 1128 levels from 1:2160, denoting individual professors or lecturers.

`studage` an ordered factor with levels 2 < 4 < 6 < 8, denoting student’s “age” measured in the *semester* number the student has been enrolled.

`lectage` an ordered factor with 6 levels, 1 < 2 < ... < 6, measuring how many semesters back the lecture rated had taken place.

`service` a binary factor with levels 0 and 1; a lecture is a “service”, if held for a different department than the lecturer’s main one.

`dept` a factor with 14 levels from 1:15, using a random code for the department of the lecture.

`y` a numeric vector of *ratings* of lectures by the students, using the discrete scale 1:5, with meanings of ‘poor’ to ‘very good’.

Each observation is one student’s rating for a specific lecture (of one lecturer, during one semester in the past).

Details

The main goal of the survey is to find “the best liked prof”, according to the lectures given. Statistical analysis of such data has been the basis for a (student) jury selecting the final winners.

The present data set has been anonymized and slightly simplified on purpose.

Examples

```
str(InstEval)
```

```
head(InstEval, 16)
```

```
xtabs(~ service + dept, InstEval)
```

`isNewMerMod`*Test if a 'merMod' Object Has Current Representation*

Description

`isNewMerMod` tests if a `merMod` object has the latest representation (in other words, if the object could have been generated using the current version of **lme4**). The representation of a `merMod` object was last changed in **lme4** version 2.0-0.

`forceNewMerMod` updates a `merMod` object so that it has the latest representation. It is useful in code that might handle objects generated by old versions of **lme4**.

`anyStructured` tests if a `merMod` object specifies random effects with structured covariance matrices.

Usage

```
isNewMerMod(object)
forceNewMerMod(object, reference = object)
anyStructured(object)
```

Arguments

`object`, `reference`
`merMod` objects.

Value

`isNewMerMod` and `anyStructured` return TRUE or FALSE.

`forceNewMerMod` returns the argument if it is “current” and a modified copy of the argument otherwise.

See Also

Virtual class [Covariance](#).

Examples

```
fm1 <- lmer(Reaction ~ Days + us(Days | Subject), sleepstudy)
fm2 <- lmer(Reaction ~ Days + diag(Days | Subject), sleepstudy)
stopifnot(isNewMerMod(fm1), identical(fm1, forceNewMerMod(fm1)),
          !anyStructured(fm1), anyStructured(fm2))
```

isREML

*Check characteristics of models***Description**

Check characteristics of models: whether a model fit corresponds to a linear (LMM), generalized linear (GLMM), or nonlinear (NLMM) mixed model, and whether a linear mixed model has been fitted by REML or not (`isREML(x)` is always FALSE for GLMMs and NLMMs).

Usage

```
isREML(x, ...)
```

```
isLMM(x, ...)
```

```
isNLMM(x, ...)
```

```
isGLMM(x, ...)
```

Arguments

`x` a fitted model.
`...` additional, optional arguments. (None are used in the `merMod` methods)

Details

These are generic functions. At present the only methods are for mixed-effects models of class `merMod`.

Value

a logical value

See Also

`getME`

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
            data = cbpp, family = binomial)
nm1 <- nlmer(circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
            Orange, start = c(Asym = 200, xmid = 725, scal = 350))

isLMM(fm1)
isGLMM(gm1)
## check all :
```

```
is.MM <- function(x) c(LMM = isLMM(x), GLMM= isGLMM(x), NLMM= isNLMM(x))
stopifnot(cbind(is.MM(fm1), is.MM(gm1), is.MM(nm1))
  == diag(rep(TRUE,3)))
```

isSingular

Test Fitted Model for (Near) Singularity

Description

Evaluates whether a fitted mixed model is (almost / near) singular, i.e., the parameters are on the boundary of the feasible parameter space: variances of one or more linear combinations of effects are (close to) zero.

Usage

```
isSingular(object, tol = getSingTol())
getSingTol()
```

Arguments

object	a fitted merMod object (result of lmer or glmer) or a Covariance object.
tol	numerical tolerance for detecting singularity.

Details

Complex mixed-effect models (i.e., those with a large number of variance-covariance parameters) frequently result in *singular* fits, i.e. estimated variance-covariance matrices with less than full rank. Less technically, this means that some "dimensions" of the variance-covariance matrix have been estimated as exactly zero. For scalar random effects such as intercept-only models, or 2-dimensional random effects such as intercept+slope models, singularity is relatively easy to detect because it leads to random-effect variance estimates of (nearly) zero, or estimates of correlations that are (almost) exactly -1 or 1. However, for more complex models (variance-covariance matrices of dimension ≥ 3) singularity can be hard to detect; models can often be singular without any of their individual variances being close to zero or correlations being close to ± 1 .

This function performs a simple test to determine whether any of the random effects covariance matrices of a fitted model are singular. The [rePCA](#) method provides more detail about the singularity pattern, showing the standard deviations of orthogonal variance components and the mapping from variance terms in the model to orthogonal components (i.e., eigenvector/rotation matrices).

While singular models are statistically well defined (it is theoretically sensible for the true maximum likelihood estimate to correspond to a singular fit), there are real concerns that (1) singular fits correspond to overfitted models that may have poor power; (2) chances of numerical problems and mis-convergence are higher for singular models (e.g. it may be computationally difficult to compute profile confidence intervals for such models); (3) standard inferential procedures such as Wald statistics and likelihood ratio tests may be inappropriate.

There is not yet consensus about how to deal with singularity, or more generally to choose which random-effects specification (from a range of choices of varying complexity) to use. Some proposals include:

- avoid fitting overly complex models in the first place, i.e. design experiments/restrict models *a priori* such that the variance-covariance matrices can be estimated precisely enough to avoid singularity (Matuschek et al 2017)
- use some form of model selection to choose a model that balances predictive accuracy and overfitting/type I error (Bates et al 2015, Matuschek et al 2017)
- “keep it maximal”, i.e. fit the most complex model consistent with the experimental design, removing only terms required to allow a non-singular fit (Barr et al. 2013), or removing further terms based on p-values or AIC
- use a partially Bayesian method that produces maximum *a posteriori* (MAP) estimates using *regularizing* priors to force the estimated random-effects variance-covariance matrices away from singularity (Chung et al 2013, **blme** package)
- use a fully Bayesian method that both regularizes the model via informative priors and gives estimates and credible intervals for all parameters that average over the uncertainty in the random effects parameters (Gelman and Hill 2006, McElreath 2015; **MCMCglmm**, **rstanarm** and **brms** packages)

`getSingTol()` is a utility function that just returns the default tolerance for testing singularity (1e-4, in the current version); if the user has set `options(lme4.singular.tolerance=...)`, it will retrieve this value instead.

Value

a logical value

References

- Barr DJ, Levy R, Scheepers C, Tily HJ (2013). “Random effects structure for confirmatory hypothesis testing: Keep it maximal.” *Journal of Memory and Language*, **68**(3), 255–278. doi:10.1016/j.jml.2012.11.001.
- Bates D, Kliegl R, Vasishth S, Baayen H (2015). “Parsimonious Mixed Models.” *arXiv preprint*. <https://arxiv.org/abs/1506.04967>.
- Chung Y, Rabe-Hesketh S, Dorie V, Gelman A, Liu J (2013). “A nondegenerate penalized likelihood estimator for variance parameters in multilevel models.” *Psychometrika*, **78**(4), 685–709. doi:10.1007/S1133601393282.
- Gelman A, Hill J (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.
- Matuschek H, Kliegl R, Vasishth S, Baayen H, Bates D (2017). “Balancing Type I error and power in linear mixed models.” *Journal of Memory and Language*, **94**, 305-315. ISSN 0749-596X, doi:10.1016/j.jml.2017.01.001.
- McElreath R (2015). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. Chapman and Hall/CRC.

See Also

[getME](#), [rePCA](#)

lme4_testlevel	<i>Detect testing level for lme4 examples and tests</i>
----------------	---

Description

Reads the environment variable LME4_TEST_LEVEL to determine which tests and examples to run

Usage

```
lme4_testlevel()
```

Value

a numeric value: 1 for standard/'light' testing, larger values for more testing. Defaults to 1 if the environment variable is not set.

lmer	<i>Fit Linear Mixed-Effects Models</i>
------	--

Description

Fit a linear mixed-effects model (LMM) to data, via restricted maximum likelihood (REML) or maximum likelihood.

Usage

```
lmer(formula, data = NULL, REML = TRUE, control = lmerControl(),
      start = NULL, verbose = 0L, subset, weights, na.action,
      offset, contrasts = NULL, devFunOnly = FALSE)
```

Arguments

- | | |
|---------|--|
| formula | <p>a two-sided linear formula object describing both the fixed-effects and random-effects part of the model, with the response on the left of a \sim operator and the terms, separated by + operators, on the right. Random-effects terms are distinguished by vertical bars () separating expressions for design matrices from grouping factors. By default, non-scalar random effects (where the design matrix has more than one column, e.g. (1+x f)) are fitted with <i>unstructured</i> (general positive semidefinite) covariance matrices.</p> <ul style="list-style-type: none"> • Two vertical bars () can be used to specify multiple uncorrelated random effects for the same grouping variable. With default settings, the -syntax works only for design matrices containing numeric (continuous) predictors; to fit models with independent categorical effects, use <code>diag(f g)</code> or set <code>options(lme4.doublevert.default = "diag_special")</code> (see getDoublevertDefault). • Tags preceding a random effect term specify covariance structure: |
|---------|--|

	<ul style="list-style-type: none"> - us (default: <code>us(f g)</code> is equivalent to <code>(f g)</code>): unstructured, positive semi-definite - diag: diagonal (all correlations set to zero). Specify <code>diag(f g, hom = TRUE)</code> to fit a homogeneous diagonal covariance matrix - cs: compound symmetric (all pairwise correlations set identical). Specify <code>cs(f g, hom = TRUE)</code> for homogeneous variances. - ar1: autoregressive order 1. Note that AR1 models are homogeneous by default; specify <code>ar1(f g, hom = FALSE)</code> for heterogeneous variances.
data	an optional data frame containing the variables named in <code>formula</code> . By default the variables are taken from the environment from which <code>lmer</code> is called. While <code>data</code> is optional, the package authors <i>strongly</i> recommend its use, especially when later applying methods such as <code>update</code> and <code>drop1</code> to the fitted model (<i>such methods are not guaranteed to work properly if data is omitted</i>). If <code>data</code> is omitted, variables will be taken from the environment of <code>formula</code> (if specified as a formula) or from the parent frame (if specified as a character vector).
REML	logical scalar - Should the estimates be chosen to optimize the REML criterion (as opposed to the log-likelihood)?
control	a list (of correct class, resulting from <code>lmerControl()</code> or <code>glmerControl()</code> respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the <code>*lmerControl</code> documentation for details.
start	a numeric vector or a named list with one optional component named <code>par</code> or <code>theta</code> , giving starting values for covariance parameters. Numeric <code>start</code> is equivalent to <code>list(par = start)</code> . Parameters corresponding to unstructured covariance matrices are on the scale of the Cholesky factor of the relative covariance matrix. By default, all relative covariance matrices are identity matrices.
verbose	integer scalar. If > 0 verbose output is generated during the optimization of the parameter estimates. If > 1 verbose output is generated during the individual penalized iteratively reweighted least squares (PIRLS) steps.
subset	an optional expression indicating the subset of the rows of <code>data</code> that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be <code>NULL</code> or a numeric vector. Prior weights are <i>not</i> normalized or standardized in any way. In particular, the diagonal of the residual covariance matrix is the squared residual standard deviation parameter <code>sigma</code> times the vector of inverse weights. Therefore, if the weights have relatively large magnitudes, then in order to compensate, the <code>sigma</code> parameter will also need to have a relatively large magnitude.
na.action	a function that indicates what should happen when the data contain NAs. The default action (<code>na.omit</code> , inherited from the 'factory fresh' value of <code>getOption("na.action")</code>) strips any observations with any missing values in any variables.
offset	this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be <code>NULL</code> or a numeric vector of length

	equal to the number of cases. One or more <code>offset</code> terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See <code>model.offset</code> .
<code>contrasts</code>	an optional list. See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .
<code>devFunOnly</code>	logical - return only the deviance evaluation function. Note that because the deviance function operates on variables stored in its environment, it may not return <i>exactly</i> the same values on subsequent calls (but the results should always be within machine tolerance).

Details

- If the `formula` argument is specified as a character vector, the function will attempt to coerce it to a formula. However, this is not recommended (users who want to construct formulas by pasting together components are advised to use `as.formula` or `reformulate`); model fits will work but subsequent methods such as `drop1`, `update`, etc. may fail.
- When handling perfectly collinear predictor variables (i.e. fixed-effect design matrices of less than full rank), `[gn]lmer` is not as sophisticated as modeling frameworks such as `lm` and `glm`. While it does automatically drop collinear variables (with a message rather than a warning), it does not automatically fill in NA values for the dropped coefficients; these can be added via `fixef(fitted.model, add.dropped=TRUE)`. This information can also be retrieved via `attr(getME(fitted.model, "X"), "col.dropped")`.
- the deviance function returned when `devFunOnly` is TRUE takes a single numeric vector argument which defines the scaled variance-covariance matrices of the random effects.
 - In the case of unstructured covariances, this vector is directly mapped to the `theta` vector, which represents the unique non-zero values in the Cholesky factor of the (scaled) covariance matrix. For models with only simple (intercept-only) random effects, `par` (and thus `theta`) is a vector of the standard deviations of the random effects. For more complex or multiple random effects, running `getME(. , "par")` or (equivalently) `getME(. , "theta")` to retrieve the `theta` vector for a fitted model and examining the names of the vector is probably the easiest way to determine the correspondence between the elements of the `theta` vector and elements of the lower triangles of the Cholesky factors of the random effects.
 - For structured covariances, the `getTheta` method translates the parameter vector to the `theta` (Cholesky-factor element) scale for internal use. The parameter vector is usually composed of a set of standard-deviation values (one if `hom = TRUE` or many if `hom = FALSE`), followed by one or more parameters that determine the correlation matrix.

Value

An object of class `merMod` (more specifically, an object of *subclass* `lmerMod`), for which many methods are available (e.g. `methods(class="merMod")`)

Note

In earlier version of the **lme4** package, a `method` argument was used. Its functionality has been replaced by the `REML` argument.

Also, `lmer(.)` allowed a `family` argument (to effectively switch to `glmer(.)`). This has been deprecated in summer 2013, and been disabled in spring 2019.

See Also

[lm](#) for linear models; [glmer](#) for generalized linear; and [nlmer](#) for nonlinear mixed models.
[plot.merMod](#) for plot diagnostics.

Examples

```
## linear mixed models - reference values from older code
(fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy))
summary(fm1) # (with its own print method; see class?merMod % ./merMod-class.Rd
plot(fm1) # plotting the model diagnostics; see ?plot.merMod

str(terms(fm1))
stopifnot(identical(terms(fm1, fixed.only=FALSE),
                    terms(model.frame(fm1))))
attr(terms(fm1, FALSE), "dataClasses") # fixed.only=FALSE needed for dataCl.

## Maximum Likelihood (ML), and "monitor" iterations via 'verbose':
fm1_ML <- update(fm1, REML=FALSE, verbose = 1)
(fm2 <- lmer(Reaction ~ Days + (Days || Subject), sleepstudy))
anova(fm1, fm2)
sm2 <- summary(fm2)
print(fm2, digits=7, ranef.comp="Var") # the print.merMod() method
print(sm2, digits=3, corr=FALSE) # the print.summary.merMod() method

## Fit sex-specific variances by constructing numeric dummy variables
## for sex and sex:age; in this case the estimated variance differences
## between groups in both intercept and slope are zero ...
data(Orthodont, package="nlme")
Orthodont$nsex <- as.numeric(Orthodont$Sex=="Male")
Orthodont$nsexage <- with(Orthodont, nsex*age)
lmer(distance ~ age + (age|Subject) + (0+nsex|Subject) +
      (0 + nsexage|Subject), data=Orthodont)
```

lmerControl

Control of Mixed Model Fitting

Description

Construct control structures for mixed model fitting. All arguments have defaults, and can be grouped into

- general control parameters, most importantly optimizer, further restart_edge, etc;
- model- or data-checking specifications, in short “checking options”, such as check.nobs.vs.rankZ, or check.rankX (currently not for nlmerControl);
- all the parameters to be passed to the optimizer, e.g., maximal number of iterations, passed via the optCtrl list argument.

Usage

```

lmerControl(optimizer = "nloptwrap",

  restart_edge = TRUE,
  boundary.tol = 1e-5,
  calc.derivs = NULL,
  use.last.params = FALSE,
  sparseX = FALSE,
  standardize.X = FALSE,
  autoscale = NULL,
  ## input checking options
  check.nobs.vs.rankZ = "ignore",
  check.nobs.vs.nlev = "stop",
  check.nlev.gtreq.5 = "ignore",
  check.nlev.gtr.1 = "stop",
  check.nobs.vs.nRE= "stop",
  check.rankX = c("message+drop.cols", "silent.drop.cols", "warn+drop.cols",
                  "stop.deficient", "ignore"),
  check.scaleX = c("warning","stop","silent.rescale",
                  "message+rescale","warn+rescale", "ignore"),
  check.formula.LHS = "stop",
  ## convergence checking options
  check.conv.nobsmax = 1e4,
  check.conv.nparmax = 10,
  check.conv.grad = .makeCC("warning", tol = 2e-3, relTol = NULL),
  check.conv.singular = .makeCC(action = "message", tol = getSingTol()),
  check.conv.hess = .makeCC(action = "warning", tol = 1e-6),
  ## optimizer args
  optCtrl = list(),
  mod.type = "lmer"
)

glmerControl(optimizer = c("bobyqa", "Nelder_Mead"),
  restart_edge = FALSE,
  boundary.tol = 1e-5,
  calc.derivs = NULL,
  use.last.params = FALSE,
  sparseX = FALSE,
  standardize.X = FALSE,
  autoscale = NULL,
  ## input checking options
  check.nobs.vs.rankZ = "ignore",
  check.nobs.vs.nlev = "stop",
  check.nlev.gtreq.5 = "ignore",
  check.nlev.gtr.1 = "stop",
  check.nobs.vs.nRE= "stop",
  check.rankX = c("message+drop.cols", "silent.drop.cols", "warn+drop.cols",
                  "stop.deficient", "ignore"),

```

```

check.scaleX = c("warning", "stop", "silent.rescale",
                "message+rescale", "warn+rescale", "ignore"),
check.formula.LHS = "stop",
## convergence checking options
check.conv.nobsmax = 1e4,
check.conv.nparmax = 20,
check.conv.grad = .makeCC("warning", tol = 2e-3, relTol = NULL),
check.conv.singular = .makeCC(action = "message", tol = getSingTol()),
check.conv.hess = .makeCC(action = "warning", tol = 1e-6),
## optimizer args
optCtrl = list(),
mod.type = "glmer",
tolPwrss = 1e-7,
compDev = TRUE,
nAGQ@initStep = TRUE,
check.response.not.const = "stop"
)

nlmerControl(optimizer = "Nelder_Mead", tolPwrss = 1e-10,
            optCtrl = list())

.makeCC(action, tol, relTol, ...)

```

Arguments

optimizer character - name of optimizing function(s). A [character](#) vector or list of functions: length 1 for `lmer` or `glmer`, possibly length 2 for `glmer`). Built-in optimizers are `"Nelder_Mead"`, `"bobyqa"` (from the [minqa](#) package), `"nlminbwrap"` (using base R's `nlminb`) and the default for `lmerControl()`, `"nloptwrap"`. Any minimizing function that allows box constraints can be used provided that it

- (1) takes input parameters `fn` (function to be optimized), `par` (starting parameter values), `lower` and `upper` (parameter bounds) and `control` (control parameters, passed through from the `control` argument) and
- (2) returns a list with (at least) elements `par` (best-fit parameters), `fval` (best-fit function value), `conv` (convergence code, equal to zero for successful convergence) and (optionally) `message` (informational message, or explanation of convergence failure).

Special provisions are made for [bobyqa](#), [Nelder_Mead](#), and optimizers wrapped in the [optimx](#) package; to use the [optimx](#) optimizers (including L-BFGS-B from base [optim](#) and `nlminb`), pass the `method` argument to `optim` in the `optCtrl` argument (you may need to load the [optimx](#) package manually using `library(optimx)`). For `glmer`, if `length(optimizer)==2`, the first element will be used for the preliminary (random effects parameters only) optimization, while the second will be used for the final (random effects plus fixed effect parameters) phase. See [modular](#) for more information on these two phases.

If `optimizer` is `NULL` (at present for `lmer` only), all of the model structures will be set up, but no optimization will be done (e.g. parameters will all be returned

	as NA).
<code>calc.derivs</code>	(logical) compute gradient and Hessian of nonlinear optimization solution? Default NULL value says these derivatives should be calculated if and only if derivative-based convergence checks are going to be done (i.e. when <code>nobs(model) < check.conv.nobsmax</code> and <code>ndim(model) < check.conv.nparmax</code> where <code>ndim</code> refers to the dimensions of the variance-covariance matrix)
<code>use.last.params</code>	(logical) should the last value of the parameters evaluated (TRUE), rather than the value of the parameters corresponding to the minimum deviance, be returned? This is a "backward bug-compatibility" option; use TRUE only when trying to match previous results.
<code>sparseX</code>	(logical) should a sparse model matrix be used for the fixed-effects terms? Currently inactive.
<code>restart_edge</code>	(logical) should the optimizer attempt a restart when it finds a solution at the boundary (i.e. zero random-effect variances or perfect +/-1 correlations)? (Currently only implemented for <code>ImerControl</code> .)
<code>boundary.tol</code>	numeric - within what distance of a boundary should the boundary be checked for a better fit? (Set to zero to disable boundary checking.)
<code>tolPwrss</code>	numeric scalar - the tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares step.
<code>compDev</code>	logical scalar - should compiled code be used for the deviance evaluation during the optimization of the parameter estimates?
<code>nAGQ0initStep</code>	Run an initial optimization phase with <code>nAGQ = 0</code> . While the initial optimization usually provides a good starting point for subsequent fitting (thus increasing overall computational speed), setting this option to FALSE can be useful in cases where the initial phase results in bad fixed-effect estimates (seen most often in binomial models with <code>link="cloglog"</code> and offsets).
<code>check.nlev.gtreq.5</code>	character - rules for checking whether all random effects have ≥ 5 levels. See action.
<code>check.nlev.gtr.1</code>	character - rules for checking whether all random effects have > 1 level. See action.
<code>check.nobs.vs.rankZ</code>	character - rules for checking whether the number of observations is greater than (or greater than or equal to) the rank of the random effects design matrix (Z), usually necessary for identifiable variances. As for action, with the addition of "warningSmall" and "stopSmall", which run the test only if the dimensions of Z are $< 1e6$. <code>nobs > rank(Z)</code> will be tested for LMMs and GLMMs with estimated scale parameters; <code>nobs \geq rank(Z)</code> will be tested for GLMMs with fixed scale parameter. The rank test is done using the <code>method="qr"</code> option of the <code>rankMatrix</code> function.
<code>check.nobs.vs.nlev</code>	character - rules for checking whether the number of observations is less than (or less than or equal to) the number of levels of every grouping factor, usually necessary for identifiable variances. As for action. <code>nobs < nlevels</code> will be

- tested for LMMs and GLMMs with estimated scale parameters; `nobs<=nlevels` will be tested for GLMMs with fixed scale parameter.
- `check.nobs.vs.nRE` character - rules for checking whether the number of observations is greater than (or greater than or equal to) the number of random-effects levels for each term, usually necessary for identifiable variances. As for `check.nobs.vs.nlev`.
- `check.conv.nobsmax` threshold for the number of observations above which derivative-based convergence checks are skipped. (Set this argument to `Inf` to restore previous behaviour of always checking.)
- `check.conv.nparmax` threshold for the number of dimensions of the variance/covariance matrix above which derivative-based convergence checks are skipped. (Set this argument to `Inf` to restore previous behaviour of always checking.)
- `check.conv.grad` rules for checking the gradient of the deviance function for convergence. A list as returned by `.makeCC`, or a character string with only the action.
- `check.conv.singular` rules for checking for a singular fit, i.e. one where some parameters are on the boundary of the feasible space (for example, random effects variances equal to 0 or correlations between random effects equal to +/- 1.0); as for `check.conv.grad` above. The default is to use `isSingular(. , tol = *)`'s default.
- `check.conv.hess` rules for checking the Hessian of the deviance function for convergence.; as for `check.conv.grad` above.
- `check.rankX` character - specifying if `rankMatrix(X)` should be compared with `ncol(X)` and if columns from the design matrix should possibly be dropped to ensure that it has full rank. Sometimes needed to make the model identifiable. The options can be abbreviated; the three `"*.drop.cols"` options all do drop columns, `"stop.deficient"` gives an error when the rank is smaller than the number of columns where `"ignore"` does no rank computation, and will typically lead to less easily understandable errors, later.
- `check.scaleX` character - check for problematic scaling of columns of fixed-effect model matrix, e.g. parameters measured on very different scales.
- `check.formula.LHS` check whether specified formula has a left-hand side. Primarily for internal use within `simulate.merMod`; *use at your own risk* as it may allow the generation of unstable `merMod` objects
- `check.response.not.const` character - check that the response is not constant.
- `optCtrl` a [list](#) of additional arguments to be passed to the nonlinear optimizer (see [Nelder_Mead](#), [bobyqa](#)). In particular, both `Nelder_Mead` and `bobyqa` use `maxfun` to specify the maximum number of function evaluations they will try before giving up - in contrast to `optim` and `optimx`-wrapped optimizers, which use `maxit`. (Also see [convergence](#) for details of stopping tolerances for different optimizers.)

Note: All of `lmer()`, `glmer()` and `nlmer()` have an optional integer argument `verbose` which you should raise (to a positive value) in order to get diagnostic console output about the optimization progress.

<code>action</code>	character - generic choices for the severity level of any test, with possible values "ignore" : skip the test. "warning" : warn if test fails. "message" : print a message if test fails. "stop" : throw an error if test fails.
<code>tol</code>	(numeric) tolerance for checking the gradient, scaled relative to the curvature (i.e., testing the gradient on a scale defined by its Wald standard deviation)
<code>relTol</code>	(numeric) tolerance for the gradient, scaled relative to the magnitude of the estimated coefficient
<code>mod.type</code>	model type (for internal use)
<code>standardize.X</code>	scale columns of X matrix? (not yet implemented)
<code>autoscale</code>	if TRUE, automatically applies scaling for the continuous covariates, and then back-transforms the estimates back to the original scale to prevent numeric stability without user intervention. (NOTE: not compatible with <code>glmer</code> yet!)
<code>...</code>	other elements to include in check specification

Details

Note that (only!) the pre-fitting “checking options” (i.e., all those starting with `"check."` but *not* including the convergence checks (`"check.conv.*"`) or rank-checking (`"check.rank*"`) options) may also be set globally via `options`. In that case, (g)lmerControl will use them rather than the default values, but will *not* override values that are passed as explicit arguments.

For example, `options(lmerControl=list(check.nobs.vs.rankZ = "ignore"))` will suppress warnings that the number of observations is less than the rank of the random effects model matrix `Z`.

Value

The `*Control` functions return a list (inheriting from class `"merControl"`) containing

1. general control parameters, such as `optimizer`, `restart_edge`;
2. (currently not for `nlmerControl`!) `"checkControl"`, a `list` of data-checking specifications, e.g., `check.nobs.vs.rankZ`;
3. parameters to be passed to the optimizer, i.e., the `optCtrl` list, which may contain `maxiter`.

`.makeCC` returns a list containing the check specification (action, tolerance, and optionally relative tolerance).

See Also

`convergence` and `allFit()` which fits for a couple of optimizers; `nloptwrap` for the `lmerControl()` default optimizer.

Examples

```

str(lmerControl())
str(glmerControl())
## fit with default algorithm [nloptr version of BOBYQA] ...
fm0 <- lmer(Reaction ~ Days + ( 1 | Subject), sleepstudy)
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
## or with "bobyqa" (default 2013 - 2019-02) ...
fm1_bobyqa <- update(fm1, control = lmerControl(optimizer="bobyqa"))
## or with "Nelder-Mead" (the default till 2013) ...
fm1_NMead <- update(fm1, control = lmerControl(optimizer="Nelder-Mead"))
## or with the nlminb function used in older (<1.0) versions of lme4;
## this will usually replicate older results
if (require(optimx)) {
  fm1_nlminb <- update(fm1,
                      control = lmerControl(optimizer= "optimx",
                                             optCtrl = list(method="nlminb")))
  ## The other option here is method="L-BFGS-B".
}

## Or we can wrap base::optim():
optimwrap <- function(fn,par,lower,upper,control=list(),
                    ...) {
  if (is.null(control$method)) stop("must specify method in optCtrl")
  method <- control$method
  control$method <- NULL
  ## "Brent" requires finite upper values (lower bound will always
  ## be zero in this case)
  if (method=="Brent") upper <- pmin(1e4,upper)
  res <- optim(par=par, fn=fn, lower=lower,upper=upper,
              control=control,method=method,...)
  with(res, list(par = par,
                fval = value,
                feval= counts[1],
                conv = convergence,
                message = message))
}
fm0_brent <- update(fm0,
                  control = lmerControl(optimizer = "optimwrap",
                                       optCtrl = list(method="Brent")))

## You can also use functions (in addition to the lmerControl() default "NLOPT_BOBYQA")
## from the 'nloptr' package, see also '?nloptrwrap' :
if (require(nloptr)) {
  fm1_nloptr_NM <- update(fm1, control=lmerControl(optimizer="nloptrwrap",
                                                  optCtrl=list(algorithm="NLOPT_LN_NELDERMEAD")))
  fm1_nloptr_COBYLA <- update(fm1, control=lmerControl(optimizer="nloptrwrap",
                                                      optCtrl=list(algorithm="NLOPT_LN_COBYLA",
                                                                    xtol_rel=1e-6,
                                                                    xtol_abs=1e-10,
                                                                    ftol_abs=1e-10)))
}
## other algorithm options include NLOPT_LN_SBPLX

```

lmList

*Fit List of lm or glm Objects with a Common Model***Description**

Fit a list of `lm` or `glm` objects with a common model for different subgroups of the data.

Usage

```
lmList(formula, data, family, subset, weights, na.action,
       offset, pool = !isGLM || .hasScale(family2char(family)),
       warn = TRUE, ...)
```

Arguments

formula	a linear <code>formula</code> object of the form $y \sim x_1 + \dots + x_n \mid g$. In the formula object, y represents the response, x_1, \dots, x_n the covariates, and g the grouping factor specifying the partitioning of the data according to which different <code>lm</code> fits should be performed.
family	an optional <code>family</code> specification for a generalized linear model (<code>glm</code>).
data	an optional data frame containing the variables named in <code>formula</code> . By default the variables are taken from the environment from which <code>lmer</code> is called. See <code>Details</code> .
subset	an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
weights	an optional vector of ‘prior weights’ to be used in the fitting process. Should be <code>NULL</code> or a numeric vector.
na.action	a function that indicates what should happen when the data contain NAs. The default action (<code>na.omit</code> , inherited from the ‘factory fresh’ value of <code>getOption("na.action")</code>) strips any observations with any missing values in any variables.
offset	this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be <code>NULL</code> or a numeric vector of length equal to the number of cases. One or more <code>offset</code> terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See <code>model.offset</code> .
pool	logical scalar indicating if the variance estimate should pool the residual sums of squares. By default true if the model has a scale parameter (which includes all linear, <code>lmer()</code> , ones).
warn	indicating if errors in the single fits should signal a “summary” <code>warning</code> .
...	additional, optional arguments to be passed to the model function or family evaluation.

Details

- While data is optional, the package authors *strongly* recommend its use, especially when later applying methods such as `update` and `drop1` to the fitted model (*such methods are not guaranteed to work properly if data is omitted*). If data is omitted, variables will be taken from the environment of formula (if specified as a formula) or from the parent frame (if specified as a character vector).
- Since **lme4** version 1.1-16, if there are errors (see [stop](#)) in the single (`lm()` or `glm()`) fits, they are summarized to a warning message which is returned as attribute "warnMessage" and signalled as `warning()` when the `warn` argument is true.
In previous **lme4** versions, a general (different) warning had been signalled in this case.

Value

an object of class `lmList4` (see there, notably for the [methods](#) defined).

See Also

[lmList4](#)

Examples

```
fm.plm <- lmList(Reaction ~ Days | Subject, sleepstudy)
coef(fm.plm)
fm.2 <- update(fm.plm, pool = FALSE)
## coefficients are the same, "pooled or unpooled":
stopifnot( all.equal(coef(fm.2), coef(fm.plm)) )

(ci <- confint(fm.plm)) # print and rather *see* :
plot(ci)                # how widely they vary for the individuals
```

lmList4-class

Class "lmList4" of 'lm' Objects on Common Model

Description

Class "lmList4" is an S4 class with basically a [list](#) of objects of class `lm` with a common model (but different data); see `lmList()` which returns these.

Package **nlme**'s `lmList()` returns objects of S3 class "lmList" and provides methods for them, on which our methods partly build.

Objects from the Class

Objects can be created by calls of the form `new("lmList4", ...)` or, more commonly, by a call to `lmList()`.

Methods

A dozen [methods](#) are provided. Currently, S4 methods for [show](#), coercion ([as\(. , .\)](#)) and others inherited via "list", and S3 methods for [coef](#), [confint](#), [fitted](#), [fixef](#), [formula](#), [logLik](#), [pairs](#), [plot](#), [predict](#), [print](#), [qqnorm](#), [ranef](#), [residuals](#), [sigma](#), [summary](#), and [update](#).

sigma(object) returns the standard deviation $\hat{\sigma}$ (of the errors in the linear models), assuming a *common* variance σ^2 by pooling (even when `pool = FALSE` was used in the fit).

See Also

[lmList](#)

Examples

```
if(getRversion() >= "3.2.0") {
  (mm <- methods(class = "lmList4"))
  ## The S3 ("not S4") ones :
  mm[!attr(mm,"info")["isS4"]]
}
## For more examples:  example(lmList)  i.e., ?lmList
```

lmResp

Generator objects for the response classes

Description

The generator objects for the [lmResp](#), [lmerResp](#), [glmResp](#) and [nlsResp](#) reference classes. Such objects are primarily used through their new methods.

Usage

```
lmResp(...)
```

Arguments

... List of arguments (see Note).

Methods

`new(y=y)`: Create a new [lmResp](#) or [lmerResp](#) object.

`new(family=family, y=y)`: Create a new [glmResp](#) object.

`new(y=y, n1mod=n1mod, nlenv=nlenv, pnames=pnames, gam=gam)`: Create a new [nlsResp](#) object.

Note

Arguments to the new methods must be named arguments.

y the numeric response vector

family a [family](#) object

nlmod the nonlinear model function

nlenv an environment holding data objects for evaluation of nlmod

pnames a character vector of parameter names

gam a numeric vector - the initial linear predictor

See Also

[lmResp](#), [lmerResp](#), [glmResp](#), [nlsResp](#)

lmResp-class	<i>Reference</i>	<i>Classes</i>	<i>for</i>	<i>Response</i>	<i>Modules,</i>
	"(lm glm nls lmer)Resp"				

Description

Reference classes for response modules, including linear models, "lmResp", generalized linear models, "glmResp", nonlinear models, "nlsResp" and linear mixed-effects models, "lmerResp". Each reference class is associated with a C++ class of the same name. As is customary, the generator object for each class has the same name as the class.

Extends

All reference classes extend and inherit methods from "[envRefClass](#)". Furthermore, "glmResp", "nlsResp" and "lmerResp" all extend the "lmResp" class.

Note

Objects from these reference classes correspond to objects in C++ classes. Methods are invoked on the C++ classes using the external pointer in the ptr field. When saving such an object the external pointer is converted to a null pointer, which is why there are redundant fields containing enough information as R objects to be able to regenerate the C++ object. The convention is that a field whose name begins with an upper-case letter is an R object and the corresponding field whose name begins with the lower-case letter is a method. Access to the external pointer should be through the method, not through the field.

See Also

[lmer](#), [glmer](#), [nlmer](#), [merMod](#).

Examples

```

showClass("lmResp")
str(lmResp$new(y=1:4))
showClass("glmResp")
str(glmResp$new(family=poisson(), y=1:4))
showClass("nlsResp")
showClass("lmerResp")
str(lmerResp$new(y=1:4))

```

merMod-class

*Class "merMod" of Fitted Mixed-Effect Models***Description**

A mixed-effects model is represented as a `merPredD` object and a response module of a class that inherits from class `lmResp`. A model with a `lmerResp` response has class `lmerMod`; a `glmResp` response has class `glmerMod`; and a `nlsResp` response has class `nlmerMod`.

Usage

```

## S3 method for class 'merMod'
anova(object, ..., refit = TRUE, model.names=NULL)
## S3 method for class 'merMod'
as.function(x, ...)
## S3 method for class 'merMod'
coef(object, ...)
## S3 method for class 'merMod'
deviance(object, REML = NULL, ...)
REMLcrit(object)
## S3 method for class 'merMod'
extractAIC(fit, scale = 0, k = 2, ...)
## S3 method for class 'merMod'
family(object, ...)
## S3 method for class 'merMod'
formula(x, fixed.only = FALSE, random.only = FALSE, ...)
## S3 method for class 'merMod'
fitted(object, ...)
## S3 method for class 'merMod'
logLik(object, REML = NULL, ...)
## S3 method for class 'merMod'
nobs(object, ...)
## S3 method for class 'merMod'
ngrps(object, ...)
## S3 method for class 'merMod'
terms(x, fixed.only = TRUE, random.only = FALSE, ...)
## S3 method for class 'merMod'
model.frame(formula, fixed.only = FALSE, ...)

```

```

## S3 method for class 'merMod'
model.matrix(object,
  type = c("fixed", "random", "randomListRaw"), noScale = NULL, ...)
## S3 method for class 'merMod'
print(x, digits = max(3, getOption("digits") - 3),
  correlation = NULL, symbolic.cor = FALSE,
  signif.stars = getOption("show.signif.stars"),
  ranef.comp = "Std.Dev.",
  ranef.corr = any(ranef.comp == "Std.Dev."), ...)
## S3 method for class 'merMod'
update(object, formula., ..., evaluate = TRUE)
## S3 method for class 'merMod'
weights(object, type = c("prior", "working"), ...)

```

Arguments

<code>object</code>	an R object of class <code>merMod</code> , i.e., as resulting from <code>lmer()</code> , or <code>glmer()</code> , etc.
<code>x</code>	an R object of class <code>merMod</code> or <code>summary.merMod</code> .
<code>fit</code>	an R object of class <code>merMod</code> .
<code>formula</code>	in the case of <code>model.frame</code> , a <code>merMod</code> object.
<code>refit</code>	logical indicating if objects of class <code>lmerMod</code> should be refitted with ML before comparing models. The default is <code>TRUE</code> to prevent the common mistake of inappropriately comparing REML-fitted models with different fixed effects, whose likelihoods are not directly comparable.
<code>model.names</code>	character vectors of model names to be used in the anova table.
<code>scale</code>	Not currently used (see <code>extractAIC</code>).
<code>k</code>	see <code>extractAIC</code> .
<code>REML</code>	Logical. If <code>TRUE</code> , return the restricted log-likelihood rather than the log-likelihood. If <code>NULL</code> (the default), set <code>REML</code> to <code>isREML(object)</code> (see <code>isREML</code>).
<code>fixed.only</code>	logical indicating if only the fixed effects components (terms or formula elements) are sought. If <code>false</code> , all components, including random ones, are returned.
<code>random.only</code>	complement of <code>fixed.only</code> ; indicates whether random components only are sought. (Trying to specify <code>fixed.only</code> and <code>random.only</code> at the same time will produce an error.)
<code>correlation</code>	(logical) for <code>print.summary.merMod</code> , indicates whether the correlation matrix of the fixed-effects parameters should be printed. In the latter case, when <code>NULL</code> (the default), the correlation matrix is printed when it has been computed by <code>summary(.)</code> , and when $p \leq 12$, and the cutoff 12 may be modified by <code>options(lme4.summary.cor.max = <n>)</code>
<code>digits</code>	number of significant digits for printing
<code>symbolic.cor</code>	should a symbolic encoding of the fixed-effects correlation matrix be printed? If so, the <code>symnum</code> function is used.
<code>signif.stars</code>	(logical) should significance stars be used?
<code>ranef.comp</code>	character vector of length one or two, indicating if random-effects parameters should be reported on the variance and/or standard deviation scale.

formula.	see update.formula .
evaluate	see update .
type	For <code>weights()</code> , type of weights to be returned; either "prior" for the initially supplied weights or "working" for the weights at the final iteration of the penalized iteratively reweighted least squares algorithm (PIRLS). For <code>model.matrix()</code> , type of model matrix to return: one of "fixed" giving the fixed effects model matrix, "random" giving the random effects model matrix, or "randomListRaw" giving a list of the raw random effects model matrices associated with each random effects term.
ranef.corr	(logical) print correlations (rather than covariances) of random effects?
noScale	(logical) some users may use autoscaling. If so, the default returns the scaled parameters. If <code>noScale = TRUE</code> , then <code>fixef</code> returns the non-scaled parameters instead.
...	potentially further arguments passed from other methods.

Objects from the Class

Objects of class `merMod` are created by calls to [lmer](#), [glmer](#) or [nlmer](#).

S3 methods

The following S3 methods with arguments given above exist (this list is currently not complete):

- anova:** returns the sequential decomposition of the contributions of fixed-effects terms or, for multiple arguments, model comparison statistics. For objects of class `lmerMod` the default behavior is to refit the models with ML if fitted with `REML = TRUE`, this can be controlled via the `refit` argument. See also [anova](#).
- as.function:** returns the deviance function, the same as [lmer](#)(* , `devFunOnly=TRUE`), and [mkLmerDevfun\(\)](#) or [mkGlmerDevfun\(\)](#), respectively.
- coef:** Computes the sum of the random and fixed effects coefficients for each explanatory variable for each level of each grouping factor.
- extractAIC:** Computes the (generalized) Akaike An Information Criterion. If `isREML(fit)`, then `fit` is refitted using maximum likelihood.
- family:** [family](#) of fitted GLMM. (*Warning:* this accessor may not work properly with customized families/link functions.)
- fitted:** Fitted values, given the conditional modes of the random effects. For more flexible access to fitted values, use [predict.merMod](#).
- logLik:** Log-likelihood at the fitted value of the parameters. Note that for GLMMs, the returned value is only proportional to the log probability density (or distribution) of the response variable. See [logLik](#).
- model.frame:** returns the frame slot of [merMod](#).
- model.matrix:** returns the fixed effects model matrix.
- nobs, ngrps:** Number of observations and vector of the numbers of levels in each grouping factor. See [ngrps](#).
- update:** See [update](#).

Deviance and log-likelihood of GLMMs

One must be careful when defining the deviance of a GLM. For example, should the deviance be defined as minus twice the log-likelihood or does it involve subtracting the deviance for a saturated model? To distinguish these two possibilities we refer to absolute deviance (minus twice the log-likelihood) and relative deviance (relative to a saturated model, e.g. Section 2.3.1 in McCullagh and Nelder 1989).

With GLMMs however, there is an additional complication involving the distinction between the likelihood and the conditional likelihood. The latter is the likelihood obtained by conditioning on the estimates of the conditional modes of the spherical random effects coefficients, whereas the likelihood itself (i.e. the unconditional likelihood) involves integrating out these coefficients. The following table summarizes how to extract the various types of deviance for a `glmerMod` object:

	conditional	unconditional
relative	<code>deviance(object)</code>	NA in <code>lme4</code>
absolute	<code>object@resp\$aic()</code>	<code>-2*logLik(object)</code>

This table requires two caveats:

- If the link function involves a scale parameter (e.g. Gamma) then `object@resp$aic() - 2 * getME(object, "devcomp")$dims["useSc"]` is required for the absolute-conditional case.
- If adaptive Gauss-Hermite quadrature is used, then `logLik(object)` is currently only proportional to the absolute-unconditional log-likelihood.

For more information about this topic see the `misc/logLikGLMM` directory in the package source.

Slots

`resp`: A reference class object for an **lme4** response module (`lmResp-class`).

`Gp`: See `getME`.

`call`: The matched call.

`frame`: The model frame containing all of the variables required to parse the model formula.

`flist`: See `getME`.

`cnms`: See `getME`.

`lower`: See `getME`.

`theta`: Covariance parameter vector.

`beta`: Fixed effects coefficients.

`u`: Conditional model of spherical random effects coefficients.

`devcomp`: See `getME`.

`pp`: A reference class object for an **lme4** predictor module (`merPredD-class`).

`optinfo`: List containing information about the nonlinear optimization.

See Also

[lmer](#), [glmer](#), [nlmer](#), [merPredD](#), [lmerResp](#), [glmResp](#), [nlResp](#)

Other methods for `merMod` objects documented elsewhere include: [fortify.merMod](#), [drop1.merMod](#), [isLMM.merMod](#), [isGLMM.merMod](#), [isNLMM.merMod](#), [isREML.merMod](#), [plot.merMod](#), [predict.merMod](#), [profile.merMod](#), [ranef.merMod](#), [refit.merMod](#), [refitML.merMod](#), [residuals.merMod](#), [sigma.merMod](#), [simulate.merMod](#), [summary.merMod](#).

Examples

```
showClass("merMod")
methods(class="merMod")## over 30 (S3) methods available

m1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
print(m1, ranef.corr = TRUE) ## print correlations of REs
print(m1, ranef.corr = FALSE) ## print covariances of REs
```

merPredD

Generator object for the merPredD class

Description

The generator object for the `merPredD` reference class. Such an object is primarily used through its new method.

Usage

```
merPredD(...)
```

Arguments

... List of arguments (see Note).

Note

`merPredD(...)` is a short form of `new("merPredD", ...)` to create a new `merPredD` object and the ... must be named arguments, (`X`, `Zt`, `Lambdat`, `Lind`, `theta`, `n`):

X: dense model matrix for the fixed-effects parameters, to be stored in the `X` field.

Zt: transpose of the sparse model matrix for the random effects. It is stored in the `Zt` field.

Lambdat: transpose of the sparse lower triangular relative variance factor (stored in the `Lambdat` field).

Lind: integer vector of the same length as the `x` slot in the `Lambdat` field. Its elements should be in the range 1 to the length of the `theta` field.

theta: numeric vector of variance component parameters (stored in the `theta` field).

n: sample size, usually `nrow(X)`.

See Also

The class definition, [merPredD](#), also for examples.

merPredD-class	<i>Class "merPredD" - a Dense Predictor Reference Class</i>
----------------	---

Description

A reference class (see mother class definition "[envRefClass](#)" for a mixed-effects model predictor module with a dense model matrix for the fixed-effects parameters. The reference class is associated with a C++ class of the same name. As is customary, the generator object, [merPredD](#), for the class has the same name as the class.

Note

Objects from this reference class correspond to objects in a C++ class. Methods are invoked on the C++ class object using the external pointer in the `Ptr` field. When saving such an object the external pointer is converted to a null pointer, which is why there are redundant fields containing enough information as R objects to be able to regenerate the C++ object. The convention is that a field whose name begins with an upper-case letter is an R object and the corresponding field, whose name begins with the lower-case letter is a method. References to the external pointer should be through the method, not directly through the `Ptr` field.

See Also

[lmer](#), [glmer](#), [nlmer](#), [merPredD](#), [merMod](#).

Examples

```
showClass("merPredD")
pp <- slot(lmer(Yield ~ 1|Batch, Dyestuff), "pp")
stopifnot(is(pp, "merPredD"))
str(pp) # an overview of all fields and methods' names.
```

mkMerMod	<i>Create a 'merMod' Object</i>
----------	---------------------------------

Description

Create an object of (a subclass of) class [merMod](#) from the environment of the objective function and the value returned by the optimizer.

Usage

```
mkMerMod(rho, opt, reTrms, fr, mc, lme4conv = NULL)
```

Arguments

rho	the environment of the objective function
opt	the optimization result returned by the optimizer (a list : see lmerControl for required elements)
reTrms	random effects structure from the calling function (see mkReTrms for required elements)
fr	model frame (see model.frame)
mc	matched call from the calling function
lme4conv	lme4-specific convergence information (results of checkConv)

Value

an object from a class that inherits from [merMod](#).

mkNewReTrms	<i>Make new random effect terms for prediction</i>
-------------	--

Description

Rebuilds the random effect terms

Usage

```
mkNewReTrms(object, newdata, re.form=NULL,
             na.action=na.pass,
             allow.new.levels=FALSE,
             sparse = max(lengths(orig.random.levs)) > 100)
```

Arguments

object	a fitted merMod object
newdata	data frame for which to create new RE terms object
re.form	(formula, NULL, or NA) specify which random effects to condition on when predicting. If NULL, include all random effects; if NA or ~ 0 , include no random effects
na.action	function determining what should be done with missing values for fixed effects in newdata
allow.new.levels	(logical) if new levels (or NA values) in newdata are allowed. If FALSE (default), such new values in newdata will trigger an error; if TRUE, then the prediction will use the unconditional (population-level) values for data with previously unobserved levels (or NAs)
sparse	generate sparse contrast matrices?

Value

a [list](#) with components

Zt	transpose of the sparse model matrix for the random effects
theta	initial values of the covariance parameters
Lind	an integer vector of indices determining the mapping of the elements of the theta vector to the "x" slot of Lambdat
Gp	a vector indexing the association of elements of the conditional mode vector with random-effect terms; if nb is the vector of numbers of conditional modes per term (i.e. number of groups times number of effects per group), Gp is $c(0, \text{cumsum}(nb))$ (and conversely nb is $\text{diff}(Gp)$)
lower	lower bounds on the covariance parameters
Lambdat	transpose of the sparse relative covariance factor
flist	list of grouping factors used in the random-effects terms
cnms	a list of column names of the random effects according to the grouping factors
Ztlist	list of components of the transpose of the random-effects model matrix, separated by random-effects term
n1	names of the terms (in the same order as Zt, i.e. reflecting the <code>reorder.terms</code> argument)

Note

mkNewReTrms is used in the context of prediction, to generate a new "random effects terms" object from an already fitted model

See Also

Other utilities: [findbars](#), [mkRespMod](#), [nlformula](#), [nobars](#), [subbars](#). [getME](#) can retrieve these components from a fitted model, although their values and/or forms may be slightly different in the final fitted model from their original values as returned from [mkReTrms](#).

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
nd <- data.frame(Days = 5, Subject = "new")
try(mkNewReTrms(fm1, newdata = nd))
t1 <- mkNewReTrms(fm1, newdata = nd, allow.new.levels = TRUE)
```

mkRespMod	<i>Create an lmerResp, glmResp or nlsResp instance</i>
-----------	--

Description

Create an lmerResp, glmResp or nlsResp instance

Usage

```
mkRespMod(fr, REML = NULL, family = NULL, nlenv = NULL,
           nlmod = NULL, ...)
```

Arguments

fr	a model frame
REML	logical scalar, value of REML for an lmerResp instance
family	the optional glm family (glmResp only)
nlenv	the nonlinear model evaluation environment (nlsResp only)
nlmod	the nonlinear model function (nlsResp only)
...	where to look for response information if fr is missing. Can contain a model response, y, offset, offset, and weights, weights.

Value

an lmerResp or glmResp or nlsResp instance

See Also

Other utilities: [findbars](#), [mkReTrms](#), [nlformula](#), [nobars](#), [subbars](#)

mkSimulateTemplate	<i>Make templates suitable for guiding mixed model simulations</i>
--------------------	--

Description

Make data and parameter templates suitable for guiding mixed model simulations, by specifying a model formula and other information (EXPERIMENTAL). Most useful for simulating balanced designs and for getting started on unbalanced simulations.

Usage

```
mkParsTemplate(formula, data)
mkDataTemplate(formula, data, nGrps = 2, nPerGrp = 1, rfunc = NULL, ...)
```

Arguments

formula	A mixed model formula (see lmer).
data	A data frame containing the names in formula.
nGrps	Number of levels of a grouping factor.
nPerGrp	Number of observations per level.
rfunc	Function for generating covariate data (e.g. rnorm).
...	Additional parameters for rfunc.

See Also

These functions are designed to be used with [simulate.merMod](#).

mkVarCorr

Make Variance and Correlation Components of Covariance

Description

Computes from reCovs the variance and correlation components of random effect covariance matrices.

Usage

```
mkVarCorr(sc, cnms, nc = lengths(cnms), use.names = FALSE),
          theta, nms = names(cnms), reCovs = NULL, relReCovs = TRUE)
```

Arguments

sc	a numeric vector of length 1 giving the residual standard deviation for LMMs. Set to NULL for GLMMs.
cnms	a list of character vectors giving component dimnames, used only to determine nc and nms if those are missing.
nc	an integer vector giving component dimensions (number of columns).
theta	a numeric vector giving covariance parameters (lower triangular entries of relative Cholesky factors in column-major order), used only to determine reCovs when that is missing.
nms	a character vector giving component names.
reCovs	a list of Covariance objects.
relReCovs	a logical indicating if the components of reCovs represent relative covariance matrices (in which case they are scaled by sc^2).

Value

A list of matrices.

See Also

[VarCorr](#).

Description

Modular functions for mixed model fits

Usage

```
lFormula(formula, data = NULL, REML = TRUE,
  subset, weights, na.action, offset, contrasts = NULL,
  control = lmerControl(), ...)
```

```
mkLmerDevfun(fr, X, reTrms, REML = TRUE, start = NULL,
  verbose = 0, control = lmerControl(), ...)
```

```
optimizeLmer(devfun,
  optimizer = formals(lmerControl)$optimizer,
  restart_edge = formals(lmerControl)$restart_edge,
  boundary.tol = formals(lmerControl)$boundary.tol,
  start = NULL, verbose = 0L,
  control = list(), ...)
```

```
glFormula(formula, data = NULL, family = gaussian,
  subset, weights, na.action, offset, contrasts = NULL,
  start, mustart, etastart, control = glmerControl(), ...)
```

```
mkGlmDevfun(fr, X, reTrms, family,
  nAGQ = if (control$nAGQ@initStep) 0L else 1L,
  verbose = 0L, maxit = 100L, control = glmerControl(), ...)
```

```
optimizeGlm(devfun,
  optimizer = if (nAGQ > 0L) "Nelder_Mead" else "bobyqa",
  restart_edge = FALSE,
  boundary.tol = formals(glmerControl)$boundary.tol,
  verbose = 0L, control = list(),
  nAGQ = if (missing(stage) || stage == 1L) 0L else 1L, stage,
  start = NULL, ...)
```

```
updateGlmDevfun(devfun, reTrms, nAGQ = 1L)
```

Arguments

`formula` a two-sided linear formula object describing both the fixed-effects and random-effects parts of the model, with the response on the left of a `~` operator and the terms, separated by `+` operators, on the right. Random-effects terms are

	distinguished by vertical bars (" ") separating expressions for design matrices from grouping factors.
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. While data is optional, the package authors <i>strongly</i> recommend its use, especially when later applying methods such as update and drop1 to the fitted model (<i>such methods are not guaranteed to work properly if data is omitted</i>). If data is omitted, variables will be taken from the environment of formula (if specified as a formula) or from the parent frame (if specified as a character vector).
REML	(logical) indicating to fit restricted maximum likelihood model.
subset	an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
na.action	a function that indicates what should happen when the data contain NAs. The default action (na.omit, inherited from the 'factory fresh' value of getOption("na.action")) strips any observations with any missing values in any variables.
offset	this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset .
contrasts	an optional list . See the contrasts.arg of model.matrix.default .
control	a list giving for [g]lFormula: all options for running the model, see lmerControl ; for mkLmerDevfun, mkGlmDevfun: options for the inner optimization step; for optimizeLmer and optimizeGlm: control parameters for nonlinear optimizer (typically inherited from the ... argument to lmerControl).
fr	A model frame containing the variables needed to create an lmerResp or glmResp instance.
X	fixed-effects design matrix
reTrms	information on random effects structure (see mkReTrms).
start	starting values (see lmer ; for glFormula, should be just a numeric vector of fixed-effect coefficients)
verbose	print output?
maxit	maximal number of Pwrss update iterations.
devfun	a deviance function, as generated by mkLmerDevfun
nAGQ	number of Gauss-Hermite quadrature points
stage	optimization stage (1: nAGQ=0, optimize over theta only; 2: nAGQ possibly >0, optimize over theta and beta)

optimizer	<p>character - name of optimizing function(s). A character vector or list of functions: length 1 for lmer or glmer, possibly length 2 for glmer. The built-in optimizers are "Nelder-Mead" and "bobyqa" (from the minqa package). Any minimizing function that allows box constraints can be used provided that it</p> <ol style="list-style-type: none"> 1. takes input parameters fn (function to be optimized), par (starting parameter values), lower (lower bounds) and control (control parameters, passed through from the control argument) and 2. returns a list with (at least) elements par (best-fit parameters), fval (best-fit function value), conv (convergence code) and (optionally) message (informational message, or explanation of convergence failure). <p>Special provisions are made for bobyqa, Nelder-Mead, and optimizers wrapped in the optimx package; to use optimx optimizers (including L-BFGS-B from base optim and nlsminb), pass the method argument to optim in the control argument.</p> <p>For glmer, if <code>length(optimizer)==2</code>, the first element will be used for the preliminary (random effects parameters only) optimization, while the second will be used for the final (random effects plus fixed effect parameters) phase. See modular for more information on these two phases.</p>
restart_edge	see lmerControl
boundary.tol	see lmerControl
family	a GLM family; see glm and family .
mustart	optional starting values on the scale of the conditional mean; see glm for details.
etastart	optional starting values on the scale of the unbounded predictor; see glm for details.
...	other potential arguments; for <code>optimizeLmer</code> and <code>optimizeGlmmer</code> , these are passed to internal function <code>optwrap</code> , which has relevant parameters <code>calc.derivs</code> and <code>use.last.params</code> (see lmerControl).

Details

These functions make up the internal components of an [gn]lmer fit.

- `[g]lFormula` takes the arguments that would normally be passed to `[g]lmer`, checking for errors and processing the formula and data input to create a list of objects required to fit a mixed model.
- `mk(Gl|L)merDevfun` takes the output of the previous step (minus the formula component) and creates a deviance function
- `optimize(Gl|L)mer` takes a deviance function and optimizes over theta (or over theta and beta, if `stage` is set to 2 for `optimizeGlmmer`)
- `updateGlmmerDevfun` takes the first stage of a GLMM optimization (with `nAGQ=0`, optimizing over theta only) and produces a second-stage deviance function
- `mkMerMod` takes the *environment* of a deviance function, the results of an optimization, a list of random-effect terms, a model frame, and a model all and produces a `[g]lmerMod` object.

Value

`lFormula` and `glFormula` return a list containing components:

fr model frame

X fixed-effect design matrix

reTrms list containing information on random effects structure: result of `mkReTrms`

REML (`lFormula` only): logical indicating if restricted maximum likelihood was used (Copy of argument.)

`mkLmerDevfun` and `mkGlmLmerDevfun` return a function to calculate deviance (or restricted deviance) as a function of the theta (random-effect) parameters. `updateGlmLmerDevfun` returns a function to calculate the deviance as a function of a concatenation of theta and beta (fixed-effect) parameters. These deviance functions have an environment containing objects required for their evaluation. CAUTION: The `environment` of functions returned by `mk(Gl|L)merDevfun` contains reference class objects (see `ReferenceClasses`, `merPredD-class`, `lmResp-class`), which behave in ways that may surprise many users. For example, if the output of `mk(Gl|L)merDevfun` is naively copied, then modifications to the original will also appear in the copy (and vice versa). To avoid this behavior one must make a deep copy (see `ReferenceClasses` for details).

`optimizeLmer` and `optimizeGlmLmer` return the results of an optimization.

Examples

```
### Fitting a linear mixed model in 4 modularized steps

## 1. Parse the data and formula:
lmod <- lFormula(Reaction ~ Days + (Days|Subject), sleepstudy)
names(lmod)
## 2. Create the deviance function to be optimized:
(devfun <- do.call(mkLmerDevfun, lmod))
ls(environment(devfun)) # the environment of 'devfun' contains objects
                        # required for its evaluation
## 3. Optimize the deviance function:
opt <- optimizeLmer(devfun)
opt[1:3]
## 4. Package up the results:
mkMerMod(environment(devfun), opt, lmod$reTrms, fr = lmod$fr)

### Same model in one line
lmer(Reaction ~ Days + (Days|Subject), sleepstudy)

### Fitting a generalized linear mixed model in six modularized steps

## 1. Parse the data and formula:
glmod <- glFormula(cbind(incidence, size - incidence) ~ period + (1 | herd),
                  data = cbpp, family = binomial)
#... see what've got :
str(glmod, max=1, give.attr=FALSE)
## 2. Create the deviance function for optimizing over theta:
```

```

(devfun <- do.call(mkGlmerDevfun, glmod))
ls(environment(devfun)) # the environment of devfun contains lots of info
## 3. Optimize over theta using a rough approximation (i.e. nAGQ = 0):
(opt <- optimizeGlmer(devfun))
## 4. Update the deviance function for optimizing over theta and beta:
(devfun <- updateGlmerDevfun(devfun, glmod$reTrms))
## 5. Optimize over theta and beta:
opt <- optimizeGlmer(devfun, stage=2)
str(opt, max=1) # seeing what we'got
## 6. Package up the results:
(fMod <- mkMerMod(environment(devfun), opt, glmod$reTrms, fr = glmod$fr))

### Same model in one line
fM <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
            data = cbpp, family = binomial)
all.equal(fMod, fM, check.attributes=FALSE, tolerance = 1e-12)
# ---- -- even tolerance = 0 may work

```

namedList

Self-naming list function

Description

this function takes a list of arguments and combines them into a list; any *unnamed* arguments are automatically named to match their symbols. The `tibble::lst()` function offers similarly functionality.

Usage

```
namedList(...)
```

Arguments

... comma-separated arguments

Examples

```

a <- 1
b <- 2
c <- 3
str(namedList(a, b, d = c))

```

NelderMead

*Nelder-Mead Optimization of Parameters, Possibly (Box) Constrained***Description**

Nelder-Mead optimization of parameters, allowing optimization subject to box constraints (contrary to the default, `method = "Nelder-Mead"`, in R's `optim()`), and using reverse communications.

Usage

```
Nelder_Mead(fn, par, lower = rep.int(-Inf, n), upper = rep.int(Inf, n),
            control = list())
```

Arguments

<code>fn</code>	a function of a single numeric vector argument returning a numeric scalar.
<code>par</code>	numeric vector of starting values for the parameters.
<code>lower</code>	numeric vector of lower bounds (elements may be <code>-Inf</code>).
<code>upper</code>	numeric vector of upper bounds (elements may be <code>Inf</code>).
<code>control</code>	a named list of control settings. Possible settings are <ul style="list-style-type: none"> iprint numeric scalar - frequency of printing evaluation information. Defaults to 0 indicating no printing. maxfun numeric scalar - maximum number of function evaluations allowed (default: 10000). FtolAbs numeric scalar - absolute tolerance on change in function values (default: 1e-5) FtolRel numeric scalar - relative tolerance on change in function values (default: 1e-15) XtolRel numeric scalar - relative tolerance on change in parameter values (default: 1e-7) MinfMax numeric scalar - maximum value of the minimum (default: <code>.Machine\$double.xmin</code>) xst numeric vector of initial step sizes to establish the simplex - all elements must be non-zero (default: <code>rep(0.02, length(par))</code>) xt numeric vector of tolerances on the parameters (default: <code>xst*5e-4</code>) verbose numeric value: 0=no printing, 1=print every 20 evaluations, 2=print every 10 evaluations, 3=print every evaluation. Sets 'iprint', if specified, but does not override it. warnOnly a logical indicating if non-convergence (codes -1,-2,-3) should not <code>stop(.)</code>, but rather only call <code>warning</code> and return a result which might be inspected. Defaults to <code>FALSE</code>, i.e., stop on non-convergence.

Value

a [list](#) with components

fval	numeric scalar - the minimum function value achieved
par	numeric vector - the value of x providing the minimum
convergence	integer valued scalar, if not 0, an error code: -4 nm_evals: maximum evaluations reached -3 nm_forced: ? -2 nm_nofeasible: cannot generate a feasible simplex -1 nm_x0notfeasible: initial x is not feasible (?) 0 successful convergence
message	a string specifying the kind of convergence.
control	the list of control settings after substituting for defaults.
feval	the number of function evaluations.

See Also

The [NelderMead](#) class definition and generator function.

Examples

```
fr <- function(x) { ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
p0 <- c(-1.2, 1)

oo <- optim(p0, fr) ## also uses Nelder-Mead by default
o. <- Nelder_Mead(fr, p0)
o.1 <- Nelder_Mead(fr, p0, control=list(verbose=1))# -> some iteration output
stopifnot(identical(o.[1:4], o.1[1:4]),
           all.equal(o.$par, oo$par, tolerance=1e-3))# diff: 0.0003865

o.2 <- Nelder_Mead(fr, p0, control=list(verbose=3, XtolRel=1e-15, FtolAbs= 1e-14))
all.equal(o.2[-5],o.1[-5], tolerance=1e-15)# TRUE, unexpectedly
```

NelderMead-class

Class "NelderMead" of Nelder-Mead optimizers and its Generator

Description

Class "NelderMead" is a reference class for a Nelder-Mead simplex optimizer allowing box constraints on the parameters and using reverse communication.

The NelderMead() function conveniently generates such objects.

Usage

```
NelderMead(...)
```

Arguments

```
...          Argument list (see Note below).
```

Methods

```
NelderMead$new(lower, upper, xst, x0, xt) Create a new NelderMead object
```

Extends

All reference classes extend and inherit methods from "[envRefClass](#)".

Note

This is the default optimizer for the second stage of [glmer](#) and [nlmer](#) fits. We found that it was more reliable and often faster than more sophisticated optimizers.

Arguments to `NelderMead()` and the `new` method must be named arguments:

lower numeric vector of lower bounds - elements may be `-Inf`.

upper numeric vector of upper bounds - elements may be `Inf`.

xst numeric vector of initial step sizes to establish the simplex - all elements must be non-zero.

x0 numeric vector of starting values for the parameters.

xt numeric vector of tolerances on the parameters.

References

Based on code in the `NLOpt` collection.

See Also

[Nelder_Mead](#), the typical "constructor". Further, [glmer](#), [nlmer](#)

Examples

```
showClass("NelderMead")
```

ngrps	<i>Number of Levels of a Factor or a "merMod" Model</i>
-------	---

Description

Returns the number of levels of a [factor](#) or a set of factors, currently e.g., for each of the grouping factors of [lmer\(\)](#), [glmer\(\)](#), etc.

Usage

```
ngrps(object, ...)
```

Arguments

object	an R object, see Details.
...	currently ignored.

Details

Currently there are methods for objects of class [merMod](#), i.e., the result of [lmer\(\)](#) etc, and [factor](#) objects.

Value

The number of levels (of a factor) or vector of number of levels for each “grouping factor” of a

Examples

```
ngrps(factor(seq(1,10,2)))
ngrps(lmer(Reaction ~ 1|Subject, sleepstudy))

## A named vector if there's more than one grouping factor :
ngrps(lmer(strength ~ (1|batch/cask), Pastes))
## cask:batch      batch
##           30      10

methods(ngrps) # -> "factor" and "merMod"
```

nlformula

*Manipulate a Nonlinear Model Formula***Description**

Check and manipulate the formula for a nonlinear model, such as specified in [nlmer](#).

Usage

```
nlformula(mc)
```

Arguments

mc matched call from the calling function, typically [nlmer\(\)](#). Should have arguments named

formula: a formula of the form `resp ~ nlmod ~ meform` where `resp` is an expression for the response, `nlmod` is the nonlinear model expression and `meform` is the mixed-effects model formula. `resp` can be omitted when, e.g., optimizing a design.

data: a data frame in which to evaluate the model function

start: either a numeric vector containing initial estimates for the nonlinear model parameters or a list with components

nlpars: the initial estimates of the nonlinear model parameters

theta: the initial estimates of the variance component parameters

Details

The model formula for a nonlinear mixed-effects model is of the form `resp ~ nlmod ~ mixed` where `resp` is an expression (usually just a name) for the response, `nlmod` is the call to the nonlinear model function, and `mixed` is the mixed-effects formula defining the linear predictor for the parameter matrix. If the formula is to be used for optimizing designs, the `resp` part can be omitted.

Value

a list with components

"respMod" a response module of class "[nlsResp](#)"

"frame" the model frame, including a terms attribute

"X" the fixed-effects model matrix

"reTrms" the random-effects terms object

See Also

Other utilities: [findbars](#), [mkRespMod](#), [mkReTrms](#), [nobars](#), [subbars](#)

nlmer

*Fitting Nonlinear Mixed-Effects Models***Description**

Fit a nonlinear mixed-effects model (NLMM) to data, via maximum likelihood.

Usage

```
nlmer(formula, data = NULL, control = nlmerControl(),
      start = NULL, verbose = 0L, nAGQ = 1L, subset, weights, na.action,
      offset, contrasts = NULL, devFunOnly = FALSE)
```

Arguments

formula	a three-part “nonlinear mixed model” formula, of the form <code>resp ~ Nonlin(...)</code> <code>~ fixed + random</code> , where the third part is similar to the RHS formula of, e.g., <code>lmer</code> . Currently, the <code>Nonlin(...)</code> formula part must not only return a numeric vector, but also must have a “gradient” attribute, a <code>matrix</code> . The functions <code>SSbiexp</code> , <code>SSlogis</code> , etc, see <code>selfStart</code> , provide this (and more). Alternatively, you can use <code>deriv()</code> to automatically produce such functions or expressions.
data	an optional data frame containing the variables named in <code>formula</code> . By default the variables are taken from the environment from which <code>lmer</code> is called. While <code>data</code> is optional, the package authors <i>strongly</i> recommend its use, especially when later applying methods such as <code>update</code> and <code>drop1</code> to the fitted model (<i>such methods are not guaranteed to work properly if data is omitted</i>). If <code>data</code> is omitted, variables will be taken from the environment of <code>formula</code> (if specified as a formula) or from the parent frame (if specified as a character vector).
control	a list (of correct class, resulting from <code>lmerControl()</code> or <code>glmerControl()</code> respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the <code>*lmerControl</code> documentation for details.
start	starting estimates for the nonlinear model parameters, as a named numeric vector or as a list with components nlpars required numeric vector of starting values for the nonlinear model parameters theta optional numeric vector of starting values for the covariance parameters
verbose	integer scalar. If > 0 verbose output is generated during the optimization of the parameter estimates. If > 1 verbose output is generated during the individual PIRLS steps (PIRLS aka PRSS, e.g. in the C++ sources).
nAGQ	integer scalar - the number of points per axis for evaluating the adaptive Gauss-Hermite approximation to the log-likelihood. Defaults to 1, corresponding to the Laplace approximation. Values greater than 1 produce greater accuracy in the evaluation of the log-likelihood at the expense of speed. A value of zero uses a faster but less exact form of parameter estimation for GLMMs by optimizing

	the random effects and the fixed-effects coefficients in the penalized iteratively reweighted least squares (PIRLS) step.
subset	an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
na.action	a function that indicates what should happen when the data contain NAs. The default action (<code>na.omit</code> , inherited from the 'factory fresh' value of <code>getOption("na.action")</code>) strips any observations with any missing values in any variables.
offset	this can be used to specify an <i>a priori</i> known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more <code>offset</code> terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See <code>model.offset</code> .
contrasts	an optional <code>list</code> . See the <code>contrasts.arg</code> of <code>model.matrix.default</code> .
devFunOnly	logical - return only the deviance evaluation function. Note that because the deviance function operates on variables stored in its environment, it may not return <i>exactly</i> the same values on subsequent calls (but the results should always be within machine tolerance).

Details

Fit nonlinear mixed-effects models, such as those used in population pharmacokinetics.

Note

Adaptive Gauss-Hermite quadrature ($nAGQ > 1$) is not currently implemented for `nlmer`. Several other methods, such as simulation or prediction with new data, are unimplemented or very lightly tested.

A method argument was used in earlier versions of the **lme4** package. Its functionality has been replaced by the `nAGQ` argument.

Examples

```
## nonlinear mixed models --- 3-part formulas ---
## 1. basic nonlinear fit. Use stats::SSlogis for its
## implementation of the 3-parameter logistic curve.
## "SS" stands for "self-starting logistic", but the
## "self-starting" part is not currently used by nlmer ... 'start' is
## necessary
startvec <- c(Asym = 200, xmid = 725, scal = 350)
(nm1 <- nlmer(circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
             Orange, start = startvec))
## 2. re-run with "quick and dirty" PIRLS step
(nm1a <- update(nm1, nAGQ = 0L))
```

```

## 3. Fit the same model with a user-built function:
## a. Define formula
nform <- ~Asym/(1+exp((xmid-input)/scal))
## b. Use deriv() to construct function:
nfun <- deriv(nform,namevec=c("Asym","xmid","scal"),
             function.arg=c("input","Asym","xmid","scal"))
nm1b <- update(nm1,circumference ~ nfun(age, Asym, xmid, scal) ~ Asym | Tree)

## 4. User-built function without using deriv():
##   derivatives could be computed more efficiently
##   by pre-computing components, but these are essentially
##   the gradients as one would derive them by hand
nfun2 <- function(input, Asym, xmid, scal) {
  value <- Asym/(1+exp((xmid-input)/scal))
  grad <- cbind(Asym=1/(1+exp((xmid-input)/scal)),
               xmid=-Asym/(1+exp((xmid-input)/scal))^2*1/scal*
                 exp((xmid-input)/scal),
               scal=-Asym/(1+exp((xmid-input)/scal))^2*
                 -(xmid-input)/scal^2*exp((xmid-input)/scal))
  attr(value,"gradient") <- grad
  value
}
stopifnot(all.equal(attr(nfun(2,1,3,4),"gradient"),
                    attr(nfun(2,1,3,4),"gradient")))
nm1c <- update(nm1,circumference ~ nfun2(age, Asym, xmid, scal) ~ Asym | Tree)

```

nloptwrap

Wrappers for additional optimizers

Description

Wrappers to allow use of alternative optimizers, from the NLOpt library (via **nloptr**) or elsewhere, for the nonlinear optimization stage.

Usage

```

nloptwrap(par, fn, lower, upper, control = list(), ...)
nlminbwrap(par, fn, lower, upper, control = list(), ...)

```

Arguments

par	starting parameter vector
fn	objective function
lower, upper	numeric vector of lower and upper bounds.
control	list of control parameters, corresponding to <code>optCtrl = *</code> , e.g., in <code>lmerControl()</code> . For <code>nloptwrap</code> , see <code>defaultControl</code> in ‘Details’.
...	additional arguments to be passed to objective function

Details

Using alternative optimizers is an important trouble-shooting tool for mixed models. These wrappers provide convenient access to the optimizers provided by Steven Johnson's NLOpt library (via the **nloptr** R package), and to the **nlminb** optimizer from base R. **nlminb** is also available via the **optimx** package; this wrapper provides access to `nlminb()` without the need to install/link the package, and without the additional post-fitting checks that are implemented by **optimx** (see examples below).

One important difference between the **nloptr**-provided implementation of BOBYQA and the **minqa**-provided version accessible via `optimizer="bobyqa"` is that it provides simpler access to optimization tolerances. **bobyqa** provides only the `rhoend` parameter (“[t]he smallest value of the trust region radius that is allowed”), while **nloptr** provides a more standard set of tolerances for relative or absolute change in the objective function or the parameter values (`ftol_rel`, `ftol_abs`, `xtol_rel`, `xtol_abs`).

The default (empty) control list corresponds to the following settings:

`nlminbwrap`: control exactly corresponds to `nlminb()`'s defaults, see there.

`nloptwrap`: `environment(nloptwrap)$defaultControl` contains the defaults, notably `algorithm = "NLOPT_LN_BOBYQA"`.

`nloptr::nloptr.print.options()` shows and explains the many possible algorithm and options.

Value

<code>par</code>	estimated parameters
<code>fval</code>	objective function value at minimum
<code>feval</code>	number of function evaluations
<code>conv</code>	convergence code (0 if no error)
<code>message</code>	convergence message

Author(s)

Gabor Grothendieck (`nlminbwrap`)

Examples

```
library(lme4)
ls.str(environment(nloptwrap)) # 'defaultControl' algorithm "NLOPT_LN_BOBYQA"
## Note that 'optimizer = "nloptwrap"' is now the default for lmer() :
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
## tighten tolerances
fm1B <- update(fm1, control= lmerControl(optCtrl = list(xtol_abs=1e-8, ftol_abs=1e-8)))
## run for longer (no effect in this case)
fm1C <- update(fm1, control = lmerControl(optCtrl = list(maxeval=10000)))

logLik(fm1B) - logLik(fm1) ## small difference in log likelihood
c(logLik(fm1C) - logLik(fm1)) ## no difference in LL
## Nelder-Mead
fm1_nloptr_NM <- update(fm1, control=
```

```

      lmerControl(optimizer = "nloptwrap",
                  optCtrl = list(algorithm = "NLOPT_LN_NELDERMEAD"))
## other nlopt algorithm options include NLOPT_LN_COBYLA, NLOPT_LN_SBPLX, see
if(interactive())
  nloptr::nloptr.print.options()

fm1_nlminb <- update(fm1, control=lmerControl(optimizer = "nlminbwrap"))
if (require(optimx)) { ## the 'optimx'-based nlminb :
  fm1_nlminb2 <- update(fm1, control=
    lmerControl(optimizer = "optimx",
                optCtrl = list(method="nlminb", kkt=FALSE)))
  cat("Likelihood difference (typically zero): ",
      c(logLik(fm1_nlminb) - logLik(fm1_nlminb2)), "\n")
}

```

Pastes

Paste strength by batch and cask

Description

Strength of a chemical paste product; its quality depending on the delivery batch, and the cask within the delivery.

Format

A data frame with 60 observations on the following 4 variables.

`strength` paste strength.

`batch` delivery batch from which the sample was sample. A factor with 10 levels: 'A' to 'J'.

`cask` cask within the delivery batch from which the sample was chosen. A factor with 3 levels: 'a' to 'c'.

`sample` the sample of paste whose strength was assayed, two assays per sample. A factor with 30 levels: 'A:a' to 'J:c'.

Details

The data are described in Davies and Goldsmith (1972) as coming from “ deliveries of a chemical paste product contained in casks where, in addition to sampling and testing errors, there are variations in quality between deliveries . . . As a routine, three casks selected at random from each delivery were sampled and the samples were kept for reference. . . . Ten of the delivery batches were sampled at random and two analytical tests carried out on each of the 30 samples”.

References

Davies OL, Goldsmith PL (eds.) (1972). *Statistical Methods in Research and Production*, 4th edition. Oliver and Boyd. Section 6.5.

Examples

```

str(Pastes)
require(lattice)
dotplot(cask ~ strength | reorder(batch, strength), Pastes,
        strip = FALSE, strip.left = TRUE, layout = c(1, 10),
        ylab = "Cask within batch",
        xlab = "Paste strength", jitter.y = TRUE)
## Modifying the factors to enhance the plot
Pastes <- within(Pastes, batch <- reorder(batch, strength))
Pastes <- within(Pastes, sample <- reorder(reorder(sample, strength),
        as.numeric(batch)))
dotplot(sample ~ strength | batch, Pastes,
        strip = FALSE, strip.left = TRUE, layout = c(1, 10),
        scales = list(y = list(relation = "free")),
        ylab = "Sample within batch",
        xlab = "Paste strength", jitter.y = TRUE)
## Four equivalent models differing only in specification
(fm1 <- lmer(strength ~ (1|batch) + (1|sample), Pastes))
(fm2 <- lmer(strength ~ (1|batch/cask), Pastes))
(fm3 <- lmer(strength ~ (1|batch) + (1|batch:cask), Pastes))
(fm4 <- lmer(strength ~ (1|batch/sample), Pastes))
## fm4 results in redundant labels on the sample:batch interaction
head(ranef(fm4)[[1]])
## compare to fm1
head(ranef(fm1)[[1]])
## This model is different and NOT appropriate for these data
(fm5 <- lmer(strength ~ (1|batch) + (1|cask), Pastes))

L <- getME(fm1, "L")
Matrix::image(L, sub = "Structure of random effects interaction in pastes model")

```

 Penicillin

Variation in penicillin testing

Description

Six samples of penicillin were tested using the *B. subtilis* plate method on each of 24 plates. The response is the diameter (mm) of the zone of inhibition of growth of the organism.

Format

A data frame with 144 observations on the following 3 variables.

diameter diameter (mm) of the zone of inhibition of the growth of the organism.

plate assay plate. A factor with levels 'a' to 'x'.

sample penicillin sample. A factor with levels 'A' to 'F'.

Details

The data are described in Davies and Goldsmith (1972) as coming from an investigation to “assess the variability between samples of penicillin by the *B. subtilis* method. In this test method a bulk-inoculated nutrient agar medium is poured into a Petri dish of approximately 90 mm. diameter, known as a plate. When the medium has set, six small hollow cylinders or pots (about 4 mm. in diameter) are cemented onto the surface at equally spaced intervals. A few drops of the penicillin solutions to be compared are placed in the respective cylinders, and the whole plate is placed in an incubator for a given time. Penicillin diffuses from the pots into the agar, and this produces a clear circular zone of inhibition of growth of the organisms, which can be readily measured. The diameter of the zone is related in a known way to the concentration of penicillin in the solution.”

References

Davies OL, Goldsmith PL (eds.) (1972). *Statistical Methods in Research and Production*, 4th edition. Oliver and Boyd. Section 6.6.

Examples

```
str(Penicillin)
require(lattice)
dotplot(reorder(plate, diameter) ~ diameter, Penicillin, groups = sample,
        ylab = "Plate", xlab = "Diameter of growth inhibition zone (mm)",
        type = c("p", "a"), auto.key = list(columns = 3, lines = TRUE,
        title = "Penicillin sample"))
(fm1 <- lmer(diameter ~ (1|plate) + (1|sample), Penicillin))

L <- getME(fm1, "L")
Matrix::image(L, main = "L",
              sub = "Penicillin: Structure of random effects interaction")
```

plot.merMod

Diagnostic Plots for 'merMod' Fits

Description

diagnostic plots for merMod fits. Some of these plots may be supplemented or superseded by functions from other packages. See the *See Also* section below for recommended alternatives.

Usage

```
## S3 method for class 'merMod'
plot(x,
     form = resid(., type = "pearson") ~ fitted(.), abline,
     id = NULL, idLabels = NULL, grid, ...)
## S3 method for class 'merMod'
qqmath(x, data = NULL, id = NULL, idLabels = NULL, ...)
```

Arguments

x	a fitted [ng]lmer model.
form	an optional formula specifying the desired type of plot. Any variable present in the original data frame used to obtain x can be referenced. In addition, x itself can be referenced in the formula using the symbol ". ". Conditional expressions on the right of a operator can be used to define separate panels in a lattice display. Default is <code>resid(., type = "pearson") ~ fitted(.)</code> , corresponding to a plot of the standardized residuals versus fitted values.
abline	an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.
id	an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized, or normalized residuals. Observations with absolute standardized (normalized) residuals greater than the $1 - value/2$ quantile of the standard normal distribution are identified in the plot using <code>idLabels</code> . If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.
idLabels	an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the observations identified according to <code>id</code> . If given as a vector, it is converted to character and used to label the observations identified according to <code>id</code> . If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified observations. Default is the interaction of all the grouping variables in the data frame. The special formula <code>idLabels=~.obs</code> will label the observations according to observation number.
data	ignored: required for S3 method compatibility.
grid	an optional logical value indicating whether a grid should be added to plot. Default depends on the type of lattice plot used: if <code>xyplot</code> defaults to TRUE, else defaults to FALSE.
...	optional arguments passed to the lattice plot function.

Details

Diagnostic plots for the linear mixed-effects fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a | operator) always implies that different panels are used for each level of the conditioning factor, according to a lattice display. If `form` is a one-sided formula, histograms of the variable on the right hand side of the formula, before a | operator, are displayed (the lattice function `histogram` is used). If `form` is two-sided and both its left and right hand side variables are numeric, scatter plots are displayed (the lattice function `xyplot` is used). Finally, if `form` is two-sided and its left hand side variable is a factor, box-plots of the right hand side variable by the levels of the left hand side variable are displayed (the lattice function `bwplot` is used).

`qqmath` produces a Q-Q plot of the residuals (see [qqmath.ranef.mer](#) for Q-Q plots of the conditional mode values).

Author(s)

original version in **nlme** package by Jose Pinheiro and Douglas Bates.

See Also

- [check_model](#) in the **performance** package.
- **DHARMA** package, which evaluates model diagnostics through simulating residuals (see: [simulateResiduals](#)).
- [qqmath](#) in the **lattice** package.
- [influencePlot](#) in the **car** package.

Examples

```

data(Orthodont,package="nlme")
fm1 <- lmer(distance ~ age + (age|Subject), data=Orthodont)
## standardized residuals versus fitted values by gender
plot(fm1, resid(., scaled=TRUE) ~ fitted(.) | Sex, abline = 0)
## box-plots of residuals by Subject
plot(fm1, Subject ~ resid(., scaled=TRUE))
## observed versus fitted values by Subject
plot(fm1, distance ~ fitted(.) | Subject, abline = c(0,1))
## residuals by age, separated by Subject
plot(fm1, resid(., scaled=TRUE) ~ age | Sex, abline = 0)
## scale-location plot, with red smoothed line
scale_loc_plot <- function(m, line.col = "red", line.lty = 1,
                          line.lwd = 2) {
  plot(fm1, sqrt(abs(resid(.))) ~ fitted(.,
    type = c("p", "smooth"),
    par.settings = list(plot.line =
      list(alpha=1, col = line.col,
            lty = line.lty, lwd = line.lwd)))
}
scale_loc_plot(fm1)
## Q-Q plot
lattice::qqmath(fm1, id=0.05)
ggp.there <- "package:ggplot2" %in% search()
if (ggp.there || require("ggplot2")) {
  ## we can create the same plots using ggplot2 and the fortify() function
  fm1F <- fortify.merMod(fm1)
  ggplot(fm1F, aes(.fitted, .resid)) + geom_point(colour="blue") +
    facet_grid(. ~ Sex) + geom_hline(yintercept=0)
  ## note: Subjects are ordered by mean distance
  ggplot(fm1F, aes(Subject,.resid)) + geom_boxplot() + coord_flip()
  ggplot(fm1F, aes(.fitted,distance)) + geom_point(colour="blue") +
    facet_wrap(~Subject) +geom_abline(intercept=0,slope=1)
  ggplot(fm1F, aes(age,.resid)) + geom_point(colour="blue") + facet_grid(.~Sex) +
    geom_hline(yintercept=0)+ geom_line(aes(group=Subject),alpha=0.4) +
    geom_smooth(method="loess")
  ## (warnings about loess are due to having only 4 unique x values)
  if(!ggp.there) detach("package:ggplot2")
}

```

Description

Xyplot, Densityplot, and Pairs plot methods for a mixed-effects model profile.

xyplot() draws “zeta diagrams”, also visualizing confidence intervals and their asymmetry.

densityplot() draws the profile densities.

splom() draws profile pairs plots. Contours are for the marginal two-dimensional regions (i.e. using $df = 2$).

Usage

```
## S3 method for class 'thpr'
xyplot(x, data = NULL,
       levels = sqrt(qchisq(pmax.int(0, pmin.int(1, conf)), df = 1)),
       conf = c(50, 80, 90, 95, 99)/100,
       absVal = FALSE, scales=NULL,
       which = 1:nptot, ...)
```

```
## S3 method for class 'thpr'
densityplot(x, data, npts = 201, upper = 0.999, ...)
```

```
## S3 method for class 'thpr'
splom(x, data,
      levels = sqrt(qchisq(pmax.int(0, pmin.int(1, conf)), 2)),
      conf = c(50, 80, 90, 95, 99)/100, which = 1:nptot,
      draw.lower = TRUE, draw.upper = TRUE, ...)
```

Arguments

x	a mixed-effects profile, i.e., of class "thpr", typically resulting from <code>profile(fm)</code> where <code>fm</code> is a fitted model from <code>lmer</code> (or its generalizations).
data	unused - only for compatibility with generic.
npts	the number of points to use for the <code>densityplot()</code> .
upper	a number in $(0, 1)$ to specify upper (and lower) boundaries as $\pm qnorm(upper)$.
levels	the contour levels to be shown; usually derived from <code>conf</code> .
conf	numeric vector of confidence levels to be shown as contours.
absVal	logical indicating if <code>abs(.)</code> olute values should be plotted, often preferred for confidence interval visualization.
scales	plotting options to be passed to <code>xyplot</code>
which	integer or character vector indicating which parameters to profile: default is all parameters (see <code>profile-methods</code> for details).

draw.lower (logical) draw lower-triangle (zeta scale) panels?
 draw.upper (logical) draw upper-triangle (standard dev/cor scale) panels?
 ... further arguments passed to `xyplot`, `densityplot`, or `splom` from package **lattice**, respectively.

Value

xyplot: a density plot, a "trellis" object (**lattice** package) which when `print()`ed produces plots on the current graphic device.

densityplot: a density plot, a "trellis" object, see above.

splom: a pairs plot, aka **scatterplot matrix**, a "trellis" object, see above.

See Also

`profile`, notably for an example.

Examples

```
## see example("profile.merMod")
```

predict.merMod *Predictions from a model at new data values*

Description

The `predict` method for `merMod` objects, i.e. results of `lmer()`, `glmer()`, etc.

Usage

```
## S3 method for class 'merMod'
predict(object, newdata = NULL, newparams = NULL,
        re.form = NULL,
        random.only=FALSE, terms = NULL,
        type = c("link", "response"), allow.new.levels = FALSE,
        na.action = na.pass,
        se.fit = FALSE,
        ...)
```

Arguments

object a fitted model object
 newdata data frame for which to evaluate predictions.
 newparams new parameters to use in evaluating predictions, specified as in the `start` parameter for `lmer` or `glmer` – a list with components `theta` and/or (for GLMMs) `beta`.

re.form	(formula, NULL, or NA) specify which random effects to condition on when predicting. If NULL, include all random effects; if NA or ~ 0 , include no random effects.
random.only	(logical) ignore fixed effects, making predictions only using random effects?
terms	a terms object - unused at present.
type	character string - either "link", the default, or "response" indicating the type of prediction object returned.
allow.new.levels	logical if new levels (or NA values) in newdata are allowed. If FALSE (default), such new values in newdata will trigger an error; if TRUE, then the prediction will use the unconditional (population-level) values for data with previously unobserved levels (or NAs).
na.action	function determining what should be done with missing values for fixed effects in newdata. The default is to predict NA: see na.pass .
se.fit	(Experimental) A logical value indicating whether the standard errors should be included or not. Default is FALSE.
...	optional additional parameters. None are used at present.

Details

- If any random effects are included in re.form (i.e. it is not ~ 0 or NA), newdata *must* contain columns corresponding to all of the grouping variables and random effects used in the original model, even if not all are used in prediction; however, they can be safely set to NA in this case.
- There is no option for computing standard errors of predictions because it is difficult to define an efficient method that incorporates uncertainty in the variance parameters; we recommend [bootMer](#) for this task.

Value

a numeric vector of predicted values

Examples

```
(gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 |herd), cbpp, binomial))
str(p0 <- predict(gm1))           # fitted values
str(p1 <- predict(gm1,re.form=NA)) # fitted values, unconditional (level-0)
newdata <- with(cbpp, expand.grid(period=unique(period), herd=unique(herd)))
str(p2 <- predict(gm1,newdata))   # new data, all RE
str(p3 <- predict(gm1,newdata,re.form=NA)) # new data, level-0
str(p4 <- predict(gm1,newdata,re.form= ~(1|herd))) # explicitly specify RE
stopifnot(identical(p2, p4))
```

profile-methods *Profile method for merMod objects*

Description

Methods for `profile()` of [ng]lmer fitted models.

The `log()` method and the more flexible `logProf()` utility transform a lmer profile into one where logarithms of standard deviations are used, while `varianceProf` converts from the standard-deviation to the variance scale; see Details.

Usage

```
## S3 method for class 'merMod'
profile(fitted, which = NULL, alphamax = 0.01,
        maxpts = 100, delta = NULL,
        delta.cutoff = 1/8, verbose = 0, devtol = 1e-09,
        devmatchtol = 1e-5,
        maxmult = 10, startmethod = "prev", optimizer = NULL,
        control = NULL, signames = TRUE,
        parallel = c("no", "multicore", "snow"),
        ncpus = getOption("profile.ncpus", 1L), cl = NULL,
        prof.scale = c("sdcor", "varcov"),
        ...)
## S3 method for class 'thpr'
as.data.frame(x, ...)
## S3 method for class 'thpr'
log(x, base = exp(1))
logProf(x, base = exp(1), ranef = TRUE,
        sigIni = if(ranef) "sig" else "sigma")
varianceProf(x, ranef = TRUE)
```

Arguments

<code>fitted</code>	a fitted model, e.g., the result of <code>lmer(...)</code> .
<code>which</code>	<p>NULL value, integer or character vector indicating which parameters to profile: default (NULL) is all parameters. For integer, i.e., indexing, the parameters are ordered as follows:</p> <ol style="list-style-type: none"> (1) random effects (theta) parameters; these are ordered as in <code>getME(., "theta")</code>, i.e., as the lower triangle of a matrix with standard deviations on the diagonal and correlations off the diagonal. (2) residual standard deviation (or scale parameter for GLMMs where appropriate). (3) fixed effect (beta) parameters. <p>Alternatively, which may be a character, containing "beta_" or "theta_" denoting the fixed or random effects parameters, respectively, or also containing parameter names, such as ".sigma" or "(Intercept)".</p>

alphamax	a number in (0,1), such that 1 - alphamax is the maximum alpha value for likelihood ratio confidence regions; used to establish the range of values to be profiled.
maxpts	maximum number of points (in each direction, for each parameter) to evaluate in attempting to construct the profile.
delta	stepping scale for deciding on next point to profile. The code uses the local derivative of the profile at the current step to establish a change in the focal parameter that will lead to a step of delta on the square-root-deviance scale. If NULL, the delta.cutoff parameter will be used to determine the stepping scale.
delta.cutoff	stepping scale (see delta) expressed as a fraction of the target maximum value of the profile on the square-root-deviance scale. Thus a delta.cutoff setting of 1/n will lead to a profile with approximately 2*n calculated points for each parameter (i.e., n points in each direction, below and above the estimate for each parameter).
verbose	level of output from internal calculations.
devtol	tolerance for fitted deviances less than baseline (supposedly minimum) deviance.
devmatchtol	tolerance for match between original deviance computation and value returned from auxiliary deviance function
maxmult	maximum multiplier of the original step size allowed, defaults to 10.
startmethod	method for picking starting conditions for optimization (STUB).
optimizer	(character or function) optimizer to use (see lmer for details); default is to use the optimizer from the original model fit.
control	a list of options controlling the profiling (see lmerControl): default is to use the control settings from the original model fit.
signames	logical indicating if abbreviated names of the form .sigNN should be used; otherwise, names are more meaningful (but longer) of the form (sd cor)_(effects) (group). Note that some code for profile transformations (e.g., log() or varianceProf) depends on signames==TRUE.
...	potential further arguments for various methods.
x	an object of class thpr (i.e., output of profile)
base	the base of the logarithm. Defaults to natural logarithms.
ranef	logical indicating if the sigmas of the random effects should be log() transformed as well. If false, only σ (standard deviation of errors) is transformed.
sigIni	character string specifying the initial part of the sigma parameters to be log transformed.
parallel	The type of parallel operation to be used (if any). If missing, the default is taken from the option "profile.parallel" (and if that is not set, "no").
ncpus	integer: number of processes to be used in parallel operation: typically one would choose this to be the number of available CPUs.
cl	An optional parallel or snow cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the profile call.
prof.scale	whether to profile on the standard deviation-correlation scale ("sdcor") or on the variance-covariance scale ("varcov")

Details

The `log` method and the more flexible `logProf()` function transform the profile into one where $\log(\sigma)$ is used instead of σ . By default all sigmas including the standard deviations of the random effects are transformed i.e., the methods return a profile with all of the `.sigNN` parameters replaced by `.lsigNN`. If `ranef` is `false`, only `".sigma"`, the standard deviation of the errors, is transformed (as it should never be zero, whereas random effect standard deviations (`.sigNN`) can be reasonably be zero).

The forward and backward splines for the log-transformed parameters are recalculated. Note that correlation parameters are not handled sensibly at present (i.e., they are logged rather than taking a more applicable transformation such as an arc-hyperbolic tangent, $\text{atanh}(x) = \log((1+x)/(1-x))/2$).

The `varianceProf` function works similarly, including non-sensibility for correlation parameters, by squaring all parameter values, changing the names by appending `sq` appropriately (e.g. `.sigNN` to `.sigsqNN`). Setting `prof.scale="varcov"` in the original `profile()` call is a more computationally intensive, but more correct, way to compute confidence intervals for covariance parameters.

Methods for function `profile` (package `stats`), here for profiling (fitted) mixed effect models.

Value

`profile(<merMod>)` returns an object of S3 class `"thpr"`, which is `data.frame`-like. Notable methods for such a profile object `confint()`, which returns the confidence intervals based on the profile, and three plotting methods (which require the `lattice` package), `xyplot`, `densityplot`, and `splom`.

In addition, the `log()` (see above) and `as.data.frame()` methods can transform `"thpr"` objects in useful ways.

See Also

The plotting methods `xyplot` etc, for class `"thpr"`.

For (more expensive) alternative confidence intervals: `bootMer`.

Examples

```
fm01ML <- lmer(Yield ~ 1|Batch, Dyestuff, REML = FALSE)
system.time(
  tpr <- profile(fm01ML, optimizer="Nelder-Mead", which="beta_")
)## fast; as only *one* beta parameter is profiled over -> 0.09s (2022)

## full profiling (default which means 'all') needs longer:
system.time( tpr <- profile(fm01ML, signames=FALSE) )
## ~ 0.26s (2022) + possible warning about convergence
(confint(tpr) -> CIpr)
# too much precision (etc). but just FYI:
trgt <- array(c(12.19854, 38.22998, 1486.451,
               84.06305, 67.6577, 1568.548), dim = 3:2)
stopifnot(all.equal(trgt, unname(CIpr), tol = .0001)) # had 3.1e-7

if (interactive()) {
  library("lattice")
}
```

```

xyplot(tpr)
xyplot(tpr, absVal=TRUE) # easier to see conf.int.s (and check symmetry)
xyplot(tpr, conf = c(0.95, 0.99), # (instead of all five 50, 80,...)
      main = "95% and 99% profile() intervals")
xyplot(logProf(tpr, ranef=FALSE),
      main = expression("lmer profile()s"~~ log(sigma)*" (only log)"))
densityplot(tpr, main="densityplot( profile(lmer(..)) )")
densityplot(varianceProf(tpr), main=" varianceProf( profile(lmer(..)) )")
splom(tpr)
splom(logProf(tpr, ranef=FALSE))
doMore <- lme4:::testLevel() > 2
if(doMore) { ## not typically, for time constraint reasons
  ## Batch and residual variance only
  system.time(tpr2 <- profile(fm01ML, which=1:2)) # , optimizer="Nelder_Mead" gives warning
  print( xyplot(tpr2) )
  print( xyplot(log(tpr2)) )# log(sigma) is better
  print( xyplot(logProf(tpr2, ranef=FALSE)) )

  ## GLMM example
  gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
    data = cbpp, family = binomial)
  ## running ~ 10 seconds on a modern machine {-> "verbose" while you wait}:
  print( system.time(pr4 <- profile(gm1, verbose=TRUE)) )
  print( xyplot(pr4, layout=c(5,1), as.table=TRUE) )
  print( xyplot(log(pr4), absVal=TRUE) ) # log(sigma_1)
  print( splom(pr4) )
  print( system.time( # quicker: only sig01 and one fixed effect
    pr2 <- profile(gm1, which=c("theta_", "period2"))))
  print( confint(pr2) )
  ## delta..: higher underlying resolution, only for 'sigma_1':
  print( system.time(
    pr4.hr <- profile(gm1, which="theta_", delta.cutoff=1/16)))
  print( xyplot(pr4.hr) )
}
} # only if interactive()

```

prt-utilities

Print and Summary Method Utilities for Mixed Effects

Description

The `print`, `summary` methods (including the `print` for the `summary()` result) in **lme4** are modular, using about ten small utility functions. Other packages, building on **lme4** can use the same utilities for ease of programming and consistency of output.

Notably see the Examples.

`llikAIC()` extracts the log likelihood, AIC, and related statics from a Fitted LMM.

`formatVC()` “format()”s the `VarCorr` matrix of the random effects – for `print()`ing and `show()`ing; it is also the “workhorse” of `.prt.VC()`, and returns a `character` matrix.

`.prt.*()` all use `cat` and `print` to produce output.

Usage

```

llikAIC(object, devianceFUN = devCrit, chkREML = TRUE,
        devcomp = object@devcomp)

methTitle(dims)

.prt.methTit(mtit, class)
.prt.family (famL)
.prt.resids (resids, digits, title = "Scaled residuals:", ...)
.prt.call (call, long = TRUE)
.prt.aictab (aictab, digits = 1)
.prt.grps (ngrps, nobs)
.prt.warn (optinfo, summary = FALSE, ...)

.prt.VC (varcor, digits, comp = "Std.Dev.", formatter = format, ...)
formatVC(varcor, digits = max(3, getOption("digits") - 2),
        corr_digits = max(2, digits-2),
        maxdim = 10,
        comp = "Std.Dev.", formatter = format,
        useScale = attr(varcor, "useSc"),
        ...)

```

Arguments

object	a LMM model fit
devianceFUN	the function to be used for computing the deviance; should not be changed for lme4 created objects.
chkREML	optional logical indicating if object maybe a REML fit.
devcomp	for lme4 always the equivalent of object@devcomp; here a list
dims	for lme4 always the equivalent of object@devcomp\$dims, a named vector or list with components "GLMM", "NLMM", "REML", and "nAGQ" of which the first two are logical scalars, and the latter two typically are FALSE or numeric .
mtit	the result of methTitle(object)
class	typically class (object).
famL	a list with components family and link, each a character string; note that standard R family objects can be used directly, as well.
resids	numeric vector of model residuals .
digits	non-negative integer of (significant) digits to print minimally.
title	character string.
...	optional arguments passed on, e.g., to residuals() .
call	the call of the model fit; e.g., available via (generic) function getCall() .
long	logical indicating if the output may be long, e.g., printing the control part of the call if there is one.
aictab	typically the AICTab component of the result of llikAIC().

varcor	typically the result of <code>VarCorr()</code> .
comp	optional character vector of length 1 or 2, containing "Std.Dev." and/or "Variance", indicating the columns to use.
formatter	a function used for formatting the numbers.
corr_digits	the number of significant digits for correlations.
maxdim	maximum dimensions (numbers of standard deviations/variances and number of rows of correlation matrices) to report per random effects term.
ngrps	integer (vector), typically the result of <code>ngrps(object)</code> .
nobs	integer; the number of observations, e.g., the result of <code>nobs</code> .
optinfo	typically object @ optinfo, the optimization infos, including warnings if there were.
summary	logical
useScale	(logical) whether the parent model estimates a scale parameter.

Value

`llikAIC()` returns a **list** with components

<code>logLik</code>	which is <code>logLik(object)</code> , and
<code>AICtab</code>	a "table" of AIC , BIC , <code>logLik</code> , deviance and <code>df.residual()</code> values.

Examples

```
## Create a few "lme4 standard" models -----
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
fmM <- update(fm1, REML=FALSE) # -> Maximum Likelihood
fmQ <- update(fm1, . ~ Days + (Days | Subject))

gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             data = cbpp, family = binomial)
gmA <- update(gm1, nAGQ = 5)

(lA1 <- llikAIC(fm1))
(lAM <- llikAIC(fmM))
(lAg <- llikAIC(gmA))

(m1 <- methTitle(fm1 @ devcomp $ dims))
(mM <- methTitle(fmM @ devcomp $ dims))
(mG <- methTitle(gm1 @ devcomp $ dims))
(mA <- methTitle(gmA @ devcomp $ dims))

.prt.methTit(m1, class(fm1))
.prt.methTit(mA, class(gmA))

.prt.family(gaussian())
.prt.family(binomial())
.prt.family( poisson())
```

```

.prt.resids(residuals(fm1), digits = 4)
.prt.resids(residuals(fmM), digits = 2)

.prt.call(getCall(fm1))
.prt.call(getCall(gm1))

.prt.aictab ( lA1 $ AICtab ) # REML
.prt.aictab ( lAM $ AICtab ) # ML --> AIC, BIC, ...

V1 <- VarCorr(fm1)
m <- formatVC(V1)

stopifnot(is.matrix(m), is.character(m), ncol(m) == 5)
print(m, quote = FALSE) ## prints all but the first line of .prt.VC() below:
.prt.VC( V1, digits = 4)
## Random effects:
## Groups   Name          Std.Dev. Corr
## Subject (Intercept) 24.740
##          Days         5.922  0.066
## Residual                25.592
p1 <- capture.output(V1)
p2 <- capture.output( print(m, quote=FALSE) )
pX <- capture.output( .prt.VC(V1, digits = max(3, getOption("digits")-2)) )
stopifnot(identical(p1, p2),
           identical(p1, pX[-1])) # [-1] : dropping 1st line

(Vq <- VarCorr(fmQ)) # default print()
print(Vq, comp = c("Std.Dev.", "Variance"))
print(Vq, comp = c("Std.Dev.", "Variance"), corr=FALSE)
print(Vq, comp = "Variance")

.prt.grps(ngroups = ngroups(fm1),
          nobs = nobs (fm1))
## --> Number of obs: 180, groups: Subject, 18

.prt.warn(fm1 @ optinfo) # nothing .. had no warnings
.prt.warn(fmQ @ optinfo) # (ditto)

```

Description

One of the most frequently asked questions about lme4 is "how do I calculate p-values for estimated parameters?" Previous versions of lme4 provided the `mcmc` function, which efficiently generated a Markov chain Monte Carlo sample from the posterior distribution of the parameters, assuming flat (scaled likelihood) priors. Due to difficulty in constructing a version of `mcmc` that was reliable even in cases where the estimated random effect variances were near zero (e.g. <https://>

stat.ethz.ch/pipermail/r-sig-mixed-models/2009q4/003115.html), `mcmcSamp` has been withdrawn (or more precisely, not updated to work with `lme4` versions $\geq 1.0.0$).

Many users, including users of the `aovlmer.fnc` function from the `languageR` package which relies on `mcmcSamp`, will be deeply disappointed by this lacuna. Users who need p-values have a variety of options. In the list below, the methods marked MC provide explicit model comparisons; CI denotes confidence intervals; and P denotes parameter-level or sequential tests of all effects in a model. The starred (*) suggestions provide finite-size corrections (important when the number of groups is < 50); those marked (+) support GLMMs as well as LMMs.

- likelihood ratio tests via `anova` or `drop1` (MC,+)
- profile confidence intervals via `profile.merMod` and `confint.merMod` (CI,+)
- parametric bootstrap confidence intervals and model comparisons via `bootMer` (or `PBmodcomp` in the `pbkrtest` package) (MC/CI,*,+)
- for random effects, simulation tests via the `RLRsim` package (MC,*)
- for fixed effects, F tests via Kenward-Roger approximation using `KRmodcomp` from the `pbkrtest` package (MC,*)
- `car::Anova` and `lmerTest::anova` provide wrappers for Kenward-Roger-corrected tests using `pbkrtest`: `lmerTest::anova` also provides t tests via the Satterthwaite approximation (P,*)
- `afex::mixed` is another wrapper for `pbkrtest` and `anova` providing "Type 3" tests of all effects (P,*,+)

`arm::sim`, or `bootMer`, can be used to compute confidence intervals on predictions.

For `glmer` models, the summary output provides p-values based on asymptotic Wald tests (P); while this is standard practice for generalized linear models, these tests make assumptions both about the shape of the log-likelihood surface and about the accuracy of a chi-squared approximation to differences in log-likelihoods.

When all else fails, don't forget to keep p-values in perspective: <https://phdcomics.com/comics/archive.php?comicid=905>

ranef

Extract the modes of the random effects

Description

A generic function to extract the conditional modes of the random effects from a fitted model object. For linear mixed models the conditional modes of the random effects are also the conditional means.

Usage

```
## S3 method for class 'merMod'
ranef(object, condVar = TRUE,
       drop = FALSE, which1 = names(ans), postVar = FALSE, ...)
## S3 method for class 'ranef.mer'
```

```

dotplot(x, data, main = TRUE, transf = I, level = 0.95, ...)
## S3 method for class 'ranef.mer'
qqmath(x, data, main = TRUE, level = 0.95, ...)
## S3 method for class 'ranef.mer'
as.data.frame(x, ...)

```

Arguments

object	an object of a class of fitted models with random effects, typically a merMod object.
condVar	a logical argument indicating if the conditional variance-covariance matrices of the random effects should be added as an attribute.
drop	should components of the return value that would be data frames with a single column, usually a column called '(Intercept)', be returned as named vectors instead?
whichel	character vector of names of grouping factors for which the random effects should be returned.
postVar	a (deprecated) synonym for condVar
x	a random-effects object (of class <code>ranef.mer</code>) produced by <code>ranef</code>
main	include a main title, indicating the grouping factor, on each sub-plot?
transf	transformation for random effects: for example, <code>exp</code> for plotting parameters from a (generalized) logistic regression on the odds rather than log-odds scale
data	This argument is required by the <code>dotplot</code> and <code>qqmath</code> generic methods, but is not actually used.
level	confidence level for confidence intervals
...	some methods for these generic functions require additional arguments.

Details

If grouping factor i has k levels and j random effects per level the i th component of the list returned by `ranef` is a data frame with k rows and j columns. If `condVar` is `TRUE` the "postVar" attribute is an array of dimension j by j by k (or a list of such arrays). The k th face of this array is a positive definite symmetric j by j matrix. If there is only one grouping factor in the model the variance-covariance matrix for the entire random effects vector, conditional on the estimates of the model parameters and on the data, will be block diagonal; this j by j matrix is the k th diagonal block. With multiple grouping factors the faces of the "postVar" attributes are still the diagonal blocks of this conditional variance-covariance matrix but the matrix itself is no longer block diagonal.

Value

- From `ranef`: An object of class `ranef.mer` composed of a list of data frames, one for each grouping factor for the random effects. The number of rows in the data frame is the number of levels of the grouping factor. The number of columns is the dimension of the random effect associated with each level of the factor.

If `condVar` is `TRUE` each of the data frames has an attribute called "postVar".

refit	<i>Refit a (merMod) Model with a Different Response</i>
-------	---

Description

Refit a model, possibly after modifying the response vector. This makes use of the model representation and directly goes to the optimization.

Usage

```
refit(object, newresp, ...)

## S3 method for class 'merMod'
refit(object, newresp = NULL, newweights = NULL,
       rename.response = FALSE,
       maxit = 100, ...)
```

Arguments

object	a fitted model, usually of class <code>lmerMod</code> , to be refit with a new response.
newresp	an (optional) numeric vector providing the new response, of the same length as the original response (see <code>Details</code> for information on NA handling). May also be a data frame with a single numeric column, e.g. as produced by <code>simulate(object)</code> .
newweights	an (optional) numeric vector of new weights
rename.response	when refitting the model, should the name of the response variable in the formula and model frame be replaced with the name of newresp?
maxit	scalar integer, currently only for GLMMs: the maximal number of Pwrss update iterations.
...	optional additional parameters. For the <code>merMod</code> method, <code>control</code> .

Details

Refit a model, possibly after modifying the response vector. This could be done using `update()`, but the `refit()` approach should be faster because it bypasses the creation of the model representation and goes directly to the optimization step.

Setting `rename.response = TRUE` may be necessary if one wants to do further operations (such as `update`) on the fitted model. However, the refitted model will still be slightly different from the equivalent model fitted via `update`; in particular, the `terms` component is not updated to reflect the new response variable, if it has a different name from the original.

If `newresp` has an `na.action` attribute, then it is assumed that NA values have already been removed from the numeric vector; this allows the results of `simulate(object)` to be used even if the original response vector contained NA values. Otherwise, the length of `newresp` must be the same as the *original* length of the response.

Value

an object like `x`, but fit to a different response vector Y .

See Also

[update.merMod](#) for more flexible and extensive model refitting; [refitML](#) for refitting a REML fitted model with maximum likelihood ('ML').

Examples

```
## Ex. 1: using refit() to fit each column in a matrix of responses -----
set.seed(101)
Y <- matrix(rnorm(1000),ncol=10)
## combine first column of responses with predictor variables
d <- data.frame(y=Y[,1],x=rnorm(100),f=rep(1:10,10))
## (use check.conv.grad="ignore" to disable convergence checks because we
## are using a fake example)
## fit first response
fit1 <- lmer(y ~ x+(1|f), data = d,
            control= lmerControl(check.conv.grad="ignore",
                                check.conv.hess="ignore"))
## combine fit to first response with fits to remaining responses
res <- c(fit1, lapply(as.data.frame(Y[,-1]), refit, object=fit1))

## Ex. 2: refitting simulated data using data that contain NA values -----
sleepstudyNA <- sleepstudy
sleepstudyNA$Reaction[1:3] <- NA
fm0 <- lmer(Reaction ~ Days + (1|Subject), sleepstudyNA)
## the special case of refitting with a single simulation works ...
ss0 <- refit(fm0, simulate(fm0))
## ... but if simulating multiple responses (for efficiency),
## need to use na.action=na.exclude in order to have proper length of data
fm1 <- lmer(Reaction ~ Days + (1|Subject), sleepstudyNA, na.action=na.exclude)
ss <- simulate(fm1, 5)
res2 <- refit(fm1, ss[[5]])
```

refitML

Refit a Model by Maximum Likelihood Criterion

Description

Refit a (`merMod`) model using the maximum likelihood criterion.

Usage

```
refitML(x, ...)
## S3 method for class 'merMod'
refitML(x, optimizer = "bobyqa", ...)
```

Arguments

x	a fitted model, usually of class " <code>lmerMod</code> ", to be refit according to the maximum likelihood criterion.
...	optional additional parameters. None are used at present.
optimizer	a string indicating the optimizer to be used.

Details

This function is primarily used to get a maximum likelihood fit of a linear mixed-effects model for an [anova](#) comparison.

Value

an object like x but fit by maximum likelihood

See Also

[refit](#) and [update.merMod](#) for more extensive refitting.

reformulas_imports *(deprecated) formula-processing functions*

Description

Formula-processing functions that have moved to reformulas

Usage

```
subbars(...)  
nobars(...)  
findbars(...)  
mkReTrms(...)  
isNested(...)  
expandDoubleVerts(...)  
reOnly(...)
```

Arguments

... arguments to be passed through to functions in the reformulas package

Details

All of these functions are now located in the reformulas package. The versions here are 'shims' that throw a warning and call the reformulas version. (reOnly will not be fully fixed in reformulas until version 0.4.5 is on CRAN.)

rePCA

PCA of random-effects covariance matrix

Description

PCA of random-effects variance-covariance estimates

Usage

```
rePCA(x)
```

Arguments

x a merMod object

Details

Perform a Principal Components Analysis (PCA) of the random-effects variance-covariance estimates from a fitted mixed-effects model. This allows the user to detect and diagnose overfitting problems in the random effects model (see Bates et al. 2015 for details).

Value

a prcomplist object

Author(s)

Douglas Bates

References

- Bates D, Kliegl R, Vasishth S, Baayen H (2015). "Parsimonious Mixed Models." *arXiv preprint*. <https://arxiv.org/abs/1506.04967>.

See Also

[isSingular](#)

Examples

```
fm1 <- lmer(Reaction~Days+(Days|Subject), sleepstudy)
rePCA(fm1)
```

rePos	<i>Generator object for the rePos (random-effects positions) class</i>
-------	--

Description

The generator object for the [rePos](#) class used to determine the positions and orders of random effects associated with particular random-effects terms in the model.

Usage

```
rePos(...)
```

Arguments

... Argument list (see Note).

Methods

`new(mer=mer)` Create a new [rePos](#) object.

Note

Arguments to the new methods must be named arguments. `mer`, an object of class "[merMod](#)", is the only required/expected argument.

See Also

[rePos](#)

rePos-class	<i>Class "rePos"</i>
-------------	----------------------

Description

A reference class for determining the positions in the random-effects vector that correspond to particular random-effects terms in the model formula

Extends

All reference classes extend and inherit methods from "[envRefClass](#)".

Examples

```
showClass("rePos")
```

residuals.merMod *residuals of merMod objects*

Description

residuals of merMod objects

Usage

```
## S3 method for class 'merMod'
residuals(object,
  type = if (isGLMM(object)) "deviance" else "response",
  scaled = FALSE, ...)

## S3 method for class 'lmResp'
residuals(object,
  type = c("working", "response", "deviance", "pearson", "partial"),
  ...)

## S3 method for class 'glmResp'
residuals(object,
  type = c("deviance", "pearson", "working", "response", "partial"),
  ...)
```

Arguments

object	a fitted [g]lmer (merMod) object
type	type of residuals
scaled	scale residuals by residual standard deviation (=scale parameter)?
...	additional arguments (ignored: for method compatibility)

Details

- The default residual type varies between lmerMod and glmerMod objects: they try to mimic [residuals.lm](#) and [residuals.glm](#) respectively. In particular, the default type is "response", i.e. (observed-fitted) for lmerMod objects vs. "deviance" for glmerMod objects. type="partial" is not yet implemented for either type.
- Note that the meaning of "pearson" residuals differs between [residuals.lm](#) and [residuals.lme](#). The former returns values scaled by the square root of user-specified weights (if any), but *not* by the residual standard deviation, while the latter returns values scaled by the estimated standard deviation (which will include the effects of any variance structure specified in the weights argument). To replicate lme behaviour, use type="pearson", scaled=TRUE.

salamander

Mountain dusky salamander mating

Description

S. Arnold and P. Verrell conducted an experiment at the University of Chicago to study breeding behaviours of mountain dusky salamanders. This mating data was used as an example in the "Generalized Linear Models" textbook (McCullagh and Nelder 1989).

Usage

```
data("salamander")
```

Format

A data frame with 360 observations on the following 8 variables.

Season represents the season Fall Summer of 1986.

Experiment experiment number 1, 2, 3.

TypeM representing the two types of male salamanders being studied. R stands for rough butt, and W stands for white side.

TypeF similar to the above except for female salamanders.

Cross represents the cross between a female and male type.

- RR = rough butt female crossed with rough butt male,
- RW = rough butt female crossed with white side male,
- WR = white side female crossed with rough butt male,
- WW = white side female crossed with white side male.

Male identification number of the male salamander.

Female identification number of the male salamander.

Mate represents whether mating has occurred. 1 = yes, 0 = no.

Details

In this example, every variable is either binary or a factor. However, most of the variables in this data set are treated like a numerical variable (i.e., Experiment, Male, Female, Mate), so we suggest turning them into factor variables before use. As outlined in McCullagh and Nelder (1989): This experiment, conducted by S. Arnold and P. Verrell at the University of Chicago, investigated whether geographically isolated populations of mountain dusky salamanders would interbreed. The goal was to see the difference in mating frequencies between the RW crosses compared to the WR crosses. Because each salamander was involved in multiple mating trials with different partners, the observations are not independent. Therefore, we use a model (see the *examples* section) that conditions on the specific male and female salamanders in the experiment.

Source

(McCullagh and Nelder 1989)

References

McCullagh P, Nelder JA (1989). *Generalized linear models*. Routledge.

Examples

```
## Making sure Male, Female, and Cross are treated as factors
salamander$Male <- factor(salamander$Male)
salamander$Female <- factor(salamander$Female)
salamander$Cross <- factor(salamander$Cross)
## Fitting the model described in 14.5.3 from McCullagh and Nelder
sal_mod <- glmer(Mate ~ (1|Female) + (1 | Male) + Cross, data = salamander,
                 family = binomial(link = "logit"))
```

schizophrenia

National Institute of Mental Health Schizophrenia Collaborative Study

Description

In the NIMH Schizophrenia Collaborative Study, patients were randomized to receive one of four medications, either placebo or one of three different anti-psychotic drugs. The protocol indicated subjects were to then be evaluated at weeks 0, 1, 3, 6 to assess severity of illness; additionally some measurements were made at weeks 2, 4, and 5.

Usage

```
data(schizophrenia)
```

Format

A data frame with 1603 observations on the following 9 variables.

`id` a numeric vector indicating the unique patient identifier.

`imps79` a numeric vector from item 79 on the Inpatient Multidimensional Psychiatric Scale indicating severity of illness. The scores were applied to be interpreted as follows:

- 1 = normal, not ill at all,
- 2 = borderline mentally ill,
- 3 = mildly ill,
- 4 = moderately ill,
- 5 = markedly ill,
- 6 = severely ill,
- 7 = among the most extremely ill.

`imps79b` a binary version of `imps79`.

`imps79o` an ordinally scaled version of `imps79`.

`int` a numeric vector of ones; used in stand-alone package to indicate intercept.

`TxDrug` a numeric vector indicating treatment with drug 1 or placebo 0.

Week a numeric vector indicating time, in weeks.
 SqrtWeek the square root of the Week variable.
 TxSWeek a variable representing the TxDrug * Week interaction.

Source

This data and most of this documentation was taken from the **mixor** package, which is no longer from CRAN. The references below mostly reference the deprecated **mixor** program.

References

- Hedeker D, Gibbons RD (1996). “MIXOR: a computer program for mixed-effects ordinal regression analysis.” *Computer methods and programs in biomedicine*, **49**(2), 157–176.
- Hedeker D, Gibbons RD (2006). *Longitudinal data analysis*. John Wiley & Sons.

Examples

```
schmod <- glmer(imps79 ~ TxDrug * Week + (1 | id),
               data = schizophrenia, family = Gamma(link = "log"))
```

sigma

Extract Residual Standard Deviation 'Sigma'

Description

Extract the estimated standard deviation of the errors, the “residual standard deviation” (also misnamed the “residual standard error”), from a fitted model of class `merMod`.

Usage

```
## S3 method for class 'merMod'
sigma(object, ...)
```

Arguments

object a fitted model.
 ... additional, optional arguments, passed from or to methods. (None currently in our two methods.)

Details

In general the dispersion parameter (which we call `sigma`) is the square root of the constant multiplier in the ‘variance function’ provided by the family functions.

- For Gaussian fits, `gaussian()$variance = rep.int(1, length(mu))` and it’s known that the variance is σ^2 . Hence, the constant multiplier of the variance function in this case is σ^2 , so we say `sigma` is $\sqrt{\sigma}$.

- For Gamma fits, `Gamma()$variance = mu^2` where `mu = scale*shape`. The known variance is $\frac{\mu^2}{\text{shape}}$. Thus, the constant multiplier here is $\frac{1}{\sqrt{\text{shape}}}$.
- Similarly, for Inverse Gaussian fits, we have `inverse.gaussian()$variance = mu^3`, with known variance $\frac{\mu^3}{\lambda}$; similarly, the constant multiplier here is $\frac{1}{\sqrt{\lambda}}$. λ is referred to as the shape parameter.

Value

The value differs based on the family (see details above for an explanation.)

- For Gaussian fitted models, this is the residual standard deviation.
- For Gamma and Inverse Gaussian fitted models, it represents the square root of the inverse of the shape parameter.
- For Binomial and Poisson fitted models, the value is always reported as 1, since their variance is determined entirely by the mean and there is no separate scale parameter.

See Also

Package **lme4** provides methods for mixed-effects models of class `merMod` and lists of linear models, `lmList4`.

Examples

```
methods(sigma)# from R 3.3.0 on, shows methods from pkgs 'stats' *and* 'lme4'
```

simulate.formula	<i>A simulate Method for formula objects that dispatches based on the Left-Hand Side</i>
------------------	--

Description

This method evaluates the left-hand side (LHS) of the given formula and dispatches it to an appropriate method based on the result by setting a nonce class name on the formula.

Usage

```
## S3 method for class 'formula'
simulate(object, nsim = 1, seed = NULL, ...,
         basis, newdata, data)
```

Arguments

object	a one- or two-sided formula .
nsim, seed	number of realisations to simulate and the random seed to use; see simulate
...	additional arguments to methods
basis	if given, overrides the LHS of the formula for the purposes of dispatching
newdata, data	if passed, the object's LHS is evaluated in this environment; at most one of the two may be passed.

Details

The dispatching works as follows:

1. If `basis` is not passed, and the formula has an LHS the expression on the LHS of the formula in the object is evaluated in the environment `newdata` or `data` (if given), in any case enclosed by the environment of `object`. Otherwise, `basis` is used.
2. The result is set as an attribute `".Basis"` on `object`. If there is no `basis` or LHS, it is not set.
3. The class vector of `object` has `c("formula_lhs_CLASS", "formula_lhs")` prepended to it, where `CLASS` is the class of the LHS value or `basis`. If LHS or `basis` has multiple classes, they are all prepended; if there is no LHS or `basis`, `c("formula_lhs_", "formula_lhs")` is.
4. `simulate` generic is evaluated on the new object, with all arguments passed on, excluding `basis`; if `newdata` or `data` are missing, they too are not passed on. The evaluation takes place in the parent's environment.

A "method" to receive a formula whose LHS evaluates to `CLASS` can therefore be implemented by a function `simulate.formula_lhs_CLASS()`. This function can expect a `formula` object, with additional attribute `.Basis` giving the evaluated LHS (so that it does not need to be evaluated again).

simulate.merMod

Simulate Responses From merMod Object

Description

Simulate responses from a "merMod" fitted model object, i.e., from the model represented by it.

Usage

```
## S3 method for class 'merMod'
simulate(object, nsim = 1, seed = NULL,
  use.u = FALSE, re.form = NA,
  newdata=NULL, newparams=NULL, family=NULL, cluster.rand=rnorm,
  allow.new.levels = FALSE, na.action = na.pass, ...)

.simulateFun(object, nsim = 1, seed = NULL, use.u = FALSE,
  re.form = NA,
  newdata=NULL, newparams=NULL,
  formula=NULL, family=NULL,
  cluster.rand=rnorm,
  weights=NULL, offset=NULL,
  allow.new.levels = FALSE, na.action = na.pass,
  cond.sim = TRUE, ...)
```

Arguments

object	(for simulate.merMod) a fitted model object or (for simulate.formula) a (one-sided) mixed model formula, as described for lmer .
nsim	positive integer scalar - the number of responses to simulate.
seed	an optional seed to be used in set.seed immediately before the simulation so as to generate a reproducible sample.
use.u	(logical) if TRUE, generate a simulation conditional on the current random-effects estimates; if FALSE generate new Normally distributed random-effects values. (Redundant with re.form, which is preferred: TRUE corresponds to re.form = NULL (condition on all random effects), while FALSE corresponds to re.form = ~0 (condition on none of the random effects).)
re.form	formula for random effects to condition on. If NULL, condition on all random effects; if NA or ~0, condition on no random effects. See Details.
newdata	data frame for which to evaluate predictions.
newparams	new parameters to use in evaluating predictions, specified as in the start parameter for lmer or glmer – a list with components theta and beta and (for LMMs or GLMMs that estimate a scale parameter) sigma
formula	a (one-sided) mixed model formula, as described for lmer .
family	a GLM family, as in glmer .
cluster.rand	Function that generates <i>standardized</i> random cluster effects. The function takes one argument, the number of random values to generate.
weights	prior weights, as in lmer or glmer .
offset	offset, as in glmer .
allow.new.levels	(logical) if FALSE (default), then any new levels (or NA values) detected in newdata will trigger an error; if TRUE, then the prediction will use the unconditional (population-level) values for data with previously unobserved levels (or NAs).
na.action	what to do with NA values in new data: see na.fail
cond.sim	(experimental) simulate the conditional distribution? if FALSE, simulate only random effects; do not simulate from the conditional distribution, rather return the predicted group-level values
...	optional additional arguments (none are used in .simulateFormula)

Details

- ordinarily `simulate` is used to generate new values from an existing, fitted model (`merMod` object): however, if `formula`, `newdata`, and `newparams` are specified, `simulate` generates the appropriate model structure to simulate from. `formula` must be a *one-sided* formula (i.e. with an empty left-hand side); in general, if `f` is a two-sided formula, `f[-2]` can be used to drop the LHS.
- The `re.form` argument allows the user to specify how the random effects are incorporated in the simulation. All of the random effects terms included in `re.form` will be *conditioned on* -

that is, the conditional modes of those random effects will be included in the deterministic part of the simulation. (If new levels are used (and `allow.new.levels` is TRUE), the conditional modes for these levels will be set to the population mode, i.e. values of zero will be used for the random effects.) Conversely, the random effect terms that are *not* included in `re.form` will be *simulated from* - that is, new values will be chosen for each group based on the estimated random-effects variances.

The default behaviour (using `re.form=NA`) is to condition on none of the random effects, simulating new values for all of the random effects.

- For Gaussian fits, `sigma` specifies the residual standard deviation; for Gamma fits, it specifies the square root of the dispersion parameter. The shape parameter can be calculated by $1/\text{sigma}(\text{fitted_model})^2$. For negative binomial fits, the overdispersion parameter is specified via the family, e.g. `simulate(..., family=negative.binomial(theta=1.5))`.
- For binomial models, `simulate.formula` looks for the binomial size first in the weights argument (if it's supplied), second from the left-hand side of the formula (if the formula has been specified in success/failure form), and defaults to 1 if neither of those have been supplied. Simulated responses will be given as proportions, unless the supplied formula has a matrix-valued left-hand side, in which case they will be given in matrix form. If a left-hand side is given, variables in that expression must be available in `newdata`.
- For negative binomial models, use the `negative.binomial` family (from the **MASS** package) and specify the overdispersion parameter via the `theta` (sic) parameter of the family function, e.g. `simulate(..., family=negative.binomial(theta=1))` to simulate from a geometric distribution (negative binomial with overdispersion parameter 1).
- `cluster.rand` allows one to test effects of departures from normality for the distribution of cluster random effects, for example by using a heavy-tailed distribution or a mixture distribution. One can also use a truncated normal to investigate true or false flagging rates; in that case the generated effects will not have a mean of 0 and standard deviation of 1; they are still considered standardized in that the simulation will multiply them by the estimated standard deviation of the cluster random effects.

See Also

[bootMer](#) for “simulestimate”, i.e., where each simulation is followed by refitting the model.

Examples

```
## test whether fitted models are consistent with the
## observed number of zeros in CBPP data set:
gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             data = cbpp, family = binomial)
gg <- simulate(gm1, 1000)
zeros <- sapply(gg, function(x) sum(x[, "incidence"] == 0))
plot(table(zeros))
abline(v = sum(cbpp$incidence == 0), col = 2)
##
## simulate from a non-fitted model; in this case we are just
## replicating the previous model, but starting from scratch
params <- list(theta = 0.5, beta = c(2, -1, -2, -3))
simdat <- with(cbpp, expand.grid(herd = levels(herd), period = factor(1:4)))
simdat$size <- 15
```

```

simdat$incidence <- sample(0:1,size=nrow(simdat),replace=TRUE)
form <- formula(gm1)[-2] ## RHS of equation only
simulate(form,newdata=simdat,family=binomial,
         newparams=params)
## simulate from negative binomial distribution instead
simulate(form,newdata=simdat,family=negative.binomial(theta=2.5),
         newparams=params)

```

sleepstudy

Reaction times in a sleep deprivation study

Description

The average reaction time per day (in milliseconds) for subjects in a sleep deprivation study.

Days 0-1 were adaptation and training (T1/T2), day 2 was baseline (B); sleep deprivation started after day 2.

Format

A data frame with 180 observations on the following 3 variables.

Reaction Average reaction time (ms)

Days Number of days of sleep deprivation

Subject Subject number on which the observation was made.

Details

These data are from the study described in Belenky et al. (2003), for the most sleep-deprived group (3 hours time-in-bed) and for the first 10 days of the study, up to the recovery period. The original study analyzed speed ($1/(\text{reaction time})$) and treated day as a categorical rather than a continuous predictor.

References

Belenky G, Wesensten NJ, Thorne DR, Thomas ML, Sing HC, Redmond DP, Russo MB, Balkin TJ (2003). "Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: a sleep dose-response study." *Journal of Sleep Research*, **12**(1), 1-12. doi:10.1046/j.13652869.2003.00337.x.

Examples

```

str(sleepstudy)
require(lattice)
xyplot(Reaction ~ Days | Subject, sleepstudy, type = c("g","p","r"),
       index = function(x,y) coef(lm(y ~ x))[1],
       xlab = "Days of sleep deprivation",
       ylab = "Average reaction time (ms)", aspect = "xy")
(fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy, subset=Days>=2))
## independent model
(fm2 <- lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy, subset=Days>=2))

```

summary

*Summary for a [ng]lmer Fit***Description**

`summary.merMod` Computes and returns a list of summary statistics of a fitted mixed-effects model of class `merMod`, as produced by `[ng]lmer` fits. The summary includes estimates for the fixed and random effects. `print.summary` controls the output for the summary method.

Usage

```
## S3 method for class 'merMod'
summary(object,
        correlation = (p <= getOption("lme4.summary.cor.max")),
        use.hessian = NULL, ...)
## S3 method for class 'summary.merMod'
print(x, digits = max(3, getOption("digits") - 3),
      correlation = NULL, symbolic.cor = FALSE,
      signif.stars = getOption("show.signif.stars"),
      ranef.comp = c("Variance", "Std.Dev."),
      ranef.corr = any(ranef.comp == "Std.Dev."), show.resids = TRUE, ...)
```

Arguments

<code>object</code>	a fitted <code>[ng]lmer</code> model.
<code>correlation</code>	(logical) indicates whether the correlation matrix should be computed and stored along with the covariance. For <code>print.summary.merMod</code> , indicates whether the correlation matrix of the fixed-effects parameters should be printed. In the latter case, when <code>NULL</code> (the default), the correlation matrix is printed when it has been computed by <code>summary(.)</code> , and when $p \leq 12$, and the cutoff 12 may be modified by <code>options(lme4.summary.cor.max = <n>)</code>
<code>use.hessian</code>	(logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects; see <code>vcov.merMod</code> for details
<code>x</code>	an R object of class <code>summary.merMod</code> .
<code>digits</code>	number of significant digits for printing.
<code>symbolic.cor</code>	should a symbolic encoding of the fixed-effects correlation matrix be printed? If so, the <code>symnum</code> function is used.
<code>signif.stars</code>	(logical) should significance stars be used?
<code>ranef.comp</code>	character vector of length one or two, indicating if random-effects parameters should be reported on the variance and/or standard deviation scale.
<code>ranef.corr</code>	(logical) print correlations (rather than covariances) of random effects?
<code>show.resids</code>	should the quantiles of the scaled residuals be printed?
<code>...</code>	potentially further arguments passed from other methods.

Value

An S3 object of class "summary.merMod", which is a list containing fixed and random effect estimates along with other standard model summary information.

See Also

- [summary](#) for the original function in base R,
- [lmer](#), and [glmer](#) for model fitting,
- [sigma.merMod](#) documents the result obtained in "Residual Std.Dev"; the name "Residual Std.Dev" is mostly applicable for [lmer](#) fits, but the meaning differs for [glmer](#) fits.

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
smry <- summary(fm1)
# Obtaining the variance-covariance matrix of the fixed effects
smry$vcov
# Obtaining the correlation matrix of fixed effects
smry$vcov@factors$correlation
```

toenail

Toenail onychomycosis data from dermatophyte infections

Description

Results of a clinical trial (De Backer et al. 1998) to compare two oral antifungal treatments: itraconazole and terbinafine to treat toenail onychomycosis causes by dermatophytes.

Usage

```
data("toenail")
```

Format

A data frame with 1908 observations on the following 8 variables.

`patientID` a unique identifier for each patient in the trial.

`outcome` a factor indicating the severity of the patient's onycholysis (separation of the nail plate from the nail bed), with levels none or mild and moderate or severe.

`treatment` a factor indicating the treatment used with levels itraconazole and terbinafine.

`time` exact timing of visit in months.

`visit` An integer represent the sequential visit number for each subject.

Details

The data in **lme4** was copied from the toenail data found in the **HSAUR3** package. From the dataset documentation from the **HSAUR3** package: There is a clinical trial (De Backer et al. 1998) to compare two competing oral antifungal treatments for toenail infection (dermatophyte onychomycosis). A total of 378 patients were randomly allocated into two treatment groups, one group receiving 250mg per day of terbinafine and the other group 200mg per day of itraconazole. Patients were evaluated at seven visits, intended to be at weeks 0, 4, 8, 12, 24, 36, and 48 for the degree of separation of the nail plate from the nail bed (onycholysis) dichotomized into moderate or severe and none or mild. But patients did not always arrive exactly at the scheduled time and the exact time in months that they did attend was recorded. The data is not balanced since not all patients attended for all seven planned visits.

Source

(De Backer et al. 1998)

References

De Backer M, De Vroey C, Lesaffre E, Scheys I, De Keyser P (1998). “Twelve weeks of continuous oral therapy for toenail onychomycosis caused by dermatophytes: A double-blind comparative trial of terbinafine 250 mg/day versus itraconazole 200 mg/day.” *Journal of the American Academy of Dermatology*, **38**(5, Supplement 2), S57-S63. ISSN 0190-9622, doi:[10.1016/S0190-9622\(98\)704864](https://doi.org/10.1016/S0190-9622(98)704864).

Examples

```
toemod <- glmer(outcome ~ time*treatment + (1 | patientID),
               data = toenail, family = binomial(link = "logit"))
```

troubleshooting	<i>Troubleshooting</i>
-----------------	------------------------

Description

This page attempts to summarize some of the common problems with fitting [gn]lmer models and how to troubleshoot them.

Most of the symptoms/diagnoses/workarounds listed below are due to various issues in the actual mixed model fitting process. You may run into problems due to multicollinearity or variables that are incorrectly typed (e.g. a variable is accidentally coded as character or factor rather than numeric). These problems can often be isolated by trying a `lm` or `glm` fit or attempting to construct the design matrix via `model.matrix()` (in each case with the random effects in your model excluded). If these tests fail then the problem is likely not specifically an `lme4` issue.

- failure to converge in (xxxx) evaluations The optimizer hit its maximum limit of function evaluations. To increase this, use the `optControl` argument of `[g]lmerControl` – for `Nelder_Mead` and `bobyqa` the relevant parameter is `maxfun`; for `optim` and `optimx`-wrapped optimizers, including `nlmminbwrap`, it's `maxit`; for `nloptwrap`, it's `maxeval`.

- Model failed to converge with `max|grad| ...`. The scaled gradient at the fitted (RE)ML estimates is worryingly large. Try
 - refitting the parameters starting at the current estimates: getting consistent results (with no warning) suggests a false positive
 - switching optimizers: getting consistent results suggests there is not really a problem; getting a similar log-likelihood with different parameter estimates suggests that the parameters are poorly determined (possibly the result of a misspecified or overfitted model)
 - compute values of the deviance in the neighbourhood of the estimated parameters to double-check that `lme4` has really found a local optimum.
- Hessian is numerically singular: parameters are not uniquely determined The Hessian (inverse curvature matrix) at the maximum likelihood or REML estimates has a very large eigenvalue, indicating that (within numerical tolerances) the surface is completely flat in some direction. The model may be misspecified, or extremely badly scaled (see "Model is nearly unidentifiable").
- Model is nearly unidentifiable ... Rescale variables? The Hessian (inverse curvature matrix) at the maximum likelihood or REML estimates has a large eigenvalue, indicating that the surface is nearly flat in some direction. Consider centering and/or scaling continuous predictor variables.
- Contrasts can be applied only to factors with 2 or more levels One or more of the categorical predictors in the model has fewer than two levels. This may be due to user error when converting these predictors to factors prior to modeling, or it may result from some factor levels being eliminated due to NAs in other predictors. Double-check the number of data points in each factor level to see which one is the culprit: `lapply(na.omit(df[,vars]), table)` (where `df` is the `data.frame` and `vars` are the column names of your predictor variables).

 VarCorr

Extract Variance and Correlation Components of Covariance

Description

This function calculates the estimated variances, standard deviations, and correlations between the random-effects terms in a mixed-effects model, of class `merMod` (linear, generalized or nonlinear). The within-group error variance and standard deviation are also calculated.

Usage

```
## S3 method for class 'merMod'
VarCorr(x, sigma = 1, ...)

## S3 method for class 'VarCorr.merMod'
as.data.frame(x, row.names = NULL, optional = FALSE,
              order = c("cov.last", "lower.tri"), ...)
## S3 method for class 'VarCorr.merMod'
print(x, digits = max(3L, getOption("digits") - 2L),
      comp = "Std.Dev.", formatter = format, ...)
```

Arguments

<code>x</code>	for <code>VarCorr</code> : a fitted model object, usually an object inheriting from class <code>merMod</code> . For <code>as.data.frame</code> , a <code>VarCorr.merMod</code> object returned from <code>VarCorr</code> .
<code>sigma</code>	an optional numeric value used as a multiplier for the standard deviations.
<code>digits</code>	an optional integer value specifying the number of digits.
<code>order</code>	arrange data frame with variances/standard deviations first and covariances/correlations last for each random effects term (" <code>cov.last</code> "), or in the order of the lower triangle of the variance-covariance matrix (" <code>lower.tri</code> ")?
<code>row.names, optional</code>	Ignored: necessary for the <code>as.data.frame</code> method.
<code>...</code>	Ignored for the <code>as.data.frame</code> method; passed to other <code>print()</code> methods for the <code>print()</code> method.
<code>comp</code>	a <code>character</code> vector, specifying the components to be printed; simply passed to <code>formatVC()</code> .
<code>formatter</code>	a <code>function</code> for formatting the numbers; simply passed to <code>formatVC()</code> .

Details

The `print` method for `VarCorr.merMod` objects has optional arguments `digits` (specify digits of precision for printing) and `comp`: the latter is a character vector with any combination of "Variance" and "Std.Dev.", to specify whether variances, standard deviations, or both should be printed.

Value

An object of class `VarCorr.merMod`. The internal structure of the object is a list of matrices, one for each random effects grouping term. For each grouping term, the standard deviations and correlation matrices for each grouping term are stored as attributes "`stddev`" and "`correlation`", respectively, of the variance-covariance matrix, and the residual standard deviation of LMMs is stored as attribute "`sc`".

Users may also access the attributes "`theta`" and "`rho`" (if applicable) which is explained in greater detail within the covariance structures vignette, and "`profpar`", which is a different parameterization of "`theta`" but suitable for profiling.

The `as.data.frame` method produces a combined data frame with one row for each variance or covariance parameter (and a row for the residual error term where applicable) and the following columns:

grp grouping factor
var1 first variable
var2 second variable (NA for variance parameters)
vcov variances or covariances
sdcor standard deviations or correlations

Author(s)

This is modeled after `VarCorr` from package `nlme`, by Jose Pinheiro and Douglas Bates.

See Also

- [lmer](#), [nlmer](#),
- The "Residual Std.Dev." output corresponds to [sigma.merMod](#).

Examples

```
data(Orthodont, package="nlme")
fm1 <- lmer(distance ~ age + (age|Subject), data = Orthodont)
print(vc <- VarCorr(fm1)) ## default print method: standard dev and corr
## both variance and std.dev.
print(vc, comp=c("Variance", "Std.Dev."), digits=2)
## variance only
print(vc, comp=c("Variance"))
## standard deviations only, but covariances rather than correlations
print(vc, corr = FALSE)
as.data.frame(vc)
as.data.frame(vc, order="lower.tri")
```

vcconv

*Convert between representations of (co-)variance structures***Description**

Convert between representations of (co-)variance structures (EXPERIMENTAL). See source code for details.

Usage

```
mlist2vec(L)
vec2mlist(v, n = NULL, symm = TRUE)
vec2STlist(v, n = NULL)
sdcor2cov(m)
cov2sdcor(V)
Vv_to_Cv(v, n = NULL, s = 1)
Sv_to_Cv(v, n = NULL, s = 1)
Cv_to_Vv(v, n = NULL, s = 1)
Cv_to_Sv(v, n = NULL, s = 1)
```

Arguments

L	List of symmetric, upper-triangular, or lower-triangular square matrices.
v	Concatenated vector containing the elements of the lower-triangle (including the diagonal) of a symmetric or triangular matrix.
n	Number of rows (and columns) of the resulting matrix.
symm	Return symmetric matrix if TRUE or lower-triangular if FALSE.
m	Standard deviation-correlation matrix.
V	Covariance matrix.
s	Scale parameter.

Details

- `m1ist2vec` Convert list of matrices to concatenated vector of lower triangles with an attribute that gives the dimension of each matrix in the original list. This attribute may be used to reconstruct the matrices. Returns a concatenation of the elements in one triangle of each matrix. An attribute "c1en" gives the dimension of each matrix.
- `vec2m1ist` Convert concatenated vector to list of matrices (lower triangle or symmetric). These matrices could represent Cholesky factors, covariance matrices, or correlation matrices (with standard deviations on the diagonal).
- `vec2ST1ist` Convert concatenated vector to list of ST matrices.
- `sdcor2cov` Standard deviation-correlation matrix to covariance matrix convert 'sdcor' format (std dev on diagonal, cor on off-diag) to and from variance-covariance matrix.
- `cov2sdcor` Covariance matrix to standard deviation-correlation matrix (i.e. standard deviations on the diagonal and correlations off the diagonal).
- `Vv_to_Cv` Variance-covariance to relative covariance factor. Returns a vector of elements from the lower triangle of a relative covariance factor.
- `Sv_to_Cv` Standard-deviation-correlation to relative covariance factor. Returns a vector of elements from the lower triangle of a relative covariance factor.
- `Cv_to_Vv` Relative covariance factor to variance-covariance. From unscaled Cholesky vector to (possibly scaled) variance-covariance vector. Returns a vector of elements from the lower triangle of a variance-covariance matrix.
- `Cv_to_Sv` Relative covariance factor to standard-deviation-correlation. From unscaled Chol to sdcor vector. Returns a vector of elements from the lower triangle of a standard-deviation-correlation matrix.

Value

(Co-)variance structure

Examples

```
vec2m1ist(1:6)
m1ist2vec(vec2m1ist(1:6)) # approximate inverse
```

vcov.merMod

Covariance matrix of estimated parameters

Description

Compute the variance-covariance matrix of estimated paramers. Optionally also computes correlations, or the full (joint) covariance matrix of the fixed-effect coefficients and the conditional modes of the random effects.

Usage

```
## S3 method for class 'merMod'
vcov(object, correlation = TRUE, sigm = sigma(object),
      use.hessian = NULL, full = FALSE, noScale = NULL, ...)
```

Arguments

object	an R object of class <code>merMod</code> , i.e., as resulting from <code>lmer()</code> , or <code>glmer()</code> , etc.
correlation	(logical) indicates whether the correlation matrix as well as the variance-covariance matrix is desired
sigm	the residual standard error; by default <code>sigma(object)</code> .
use.hessian	(logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects. See <i>Details</i> .
full	return the 'full' covariance matrix, i.e. the joint covariance matrix of the conditional distribution of conditional modes (as in <code>getME(., "b")</code>) and fixed-effect parameters. (<code>correlation</code> and <code>use.hessian</code> are <i>ignored</i> in this case.) Note that this option may be slow for models with large numbers of random-effect levels!
noScale	(logical) some users may use autoscaling. If so, the default returns the scaled parameters. If <code>noScale = TRUE</code> , then <code>vcov</code> returns the non-scaled parameters instead.
...	extra arguments for method compatibility (ignored)

Details

When `use.hessian = FALSE`, the code estimates the covariance matrix based on internal information about the inverse of the model matrix (see `getME(., "RX")`). This is exact for linear mixed models, but approximate for GLMMs. The default is to use the Hessian whenever the fixed effect parameters are arguments to the deviance function (i.e. for GLMMs with `nAGQ > 0`), and to use `getME(., "RX")` whenever the fixed effect parameters are profiled out (i.e. for GLMMs with `nAGQ = 0` or LMMs).

`use.hessian = FALSE` is backward-compatible with older versions of `lme4`, but may give less accurate SE estimates when the estimates of the fixed-effect (see `getME(., "beta")`) and random-effect (see `getME(., "theta")`) parameters are correlated.

However, `use.hessian = TRUE` is not always more accurate: for some numerically unstable fits, the approximation using `RX` is actually more reliable (because the Hessian has to be computed by a finite difference approximation, which is also error-prone): see e.g. <https://github.com/lme4/lme4/issues/720>

Value

a covariance matrix (sparse when `full = TRUE`)

Examples

```

fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
            data = cbpp, family = binomial)
(v1 <- vcov(fm1))
v2 <- vcov(fm1, correlation = TRUE)
# extract the hidden 'correlation' entry in @factors
as(v2, "corMatrix")
v3 <- vcov(gm1)
v3X <- vcov(gm1, use.hessian = FALSE)
all.equal(v3, v3X)
## full correlation matrix
cv <- vcov(fm1, full = TRUE)
image(cv, xlab = "", ylab = "",
      scales = list(y = list(labels = rownames(cv)),
                    at = seq(nrow(cv)),
                    x = list(labels = NULL)))

```

VerbAgg

*Verbal Aggression item responses***Description**

These are the item responses to a questionnaire on verbal aggression. These data are used throughout De Boeck and Wilson (2004) to illustrate various forms of item response models.

Format

A data frame with 7584 observations on the following 13 variables.

Anger the subject's Trait Anger score as measured on the State-Trait Anger Expression Inventory (STAXI)

Gender the subject's gender - a factor with levels M and F

item the item on the questionnaire, as a factor

resp the subject's response to the item - an ordered factor with levels no < perhaps < yes

id the subject identifier, as a factor

btype behavior type - a factor with levels curse, scold and shout

situ situation type - a factor with levels other and self indicating other-to-blame and self-to-blame

mode behavior mode - a factor with levels want and do

r2 dichotomous version of the response - a factor with levels N and Y

Source

Data originally from the UC Berkeley BEAR Center; original link is available at <https://web.archive.org/web/20221128003829/https://old.bear.berkeley.edu/page/materials-explanatory-item-respon> but the data are no longer accessible there.

References

De Boeck P, Wilson M (2004). *Explanatory item response models: A generalized linear and non-linear approach*. Springer.

Examples

```
str(VerbAgg)
## Show how r2 := h(resp) is defined:
with(VerbAgg, stopifnot( identical(r2, {
  r <- factor(resp, ordered=FALSE); levels(r) <- c("N","Y","Y"); r})))

xtabs(~ item + resp, VerbAgg)
xtabs(~ btype + resp, VerbAgg)
round(100 * ftable(prop.table(xtabs(~ situ + mode + resp, VerbAgg), 1:2), 1))
person <- unique(subset(VerbAgg, select = c(id, Gender, Anger)))
require(lattice)
densityplot(~ Anger, person, groups = Gender, auto.key = list(columns = 2),
  xlab = "Trait Anger score (STAXI)")

if(lme4:::testLevel() >= 3) { ## takes about 15 sec
  print(fmVA <- glmer(r2 ~ (Anger + Gender + btype + situ)^2 +
    (1|id) + (1|item), family = binomial, data =
    VerbAgg), corr=FALSE)
} ## testLevel() >= 3
if (interactive()) {
  ## much faster but less accurate
  print(fmVA0 <- glmer(r2 ~ (Anger + Gender + btype + situ)^2 +
    (1|id) + (1|item), family = binomial,
    data = VerbAgg, nAGQ=0L), corr=FALSE)
} ## interactive()
```

Index

- * **GLMM**
 - glmer, 37
 - glmer.nb, 40
- * **LMM**
 - lmer, 57
- * **NLMM**
 - nlmer, 91
- * **boundary**
 - isSingular, 55
- * **classes**
 - glmFamily, 43
 - glmFamily-class, 43
 - golden-class, 44
 - lmList4-class, 68
 - lmResp, 69
 - lmResp-class, 70
 - merMod-class, 71
 - merPredD, 75
 - merPredD-class, 76
 - NelderMead, 86
 - NelderMead-class, 87
 - rePos, 117
 - rePos-class, 117
- * **datasets**
 - Arabidopsis, 7
 - cake, 12
 - cbpp, 13
 - culcitalogreg, 21
 - culcitalvolume, 22
 - Dyestuff, 28
 - gopherdat2, 45
 - grouseticks, 47
 - InstEval, 52
 - Pastes, 95
 - Penicillin, 96
 - salamander, 119
 - schizophrenia, 120
 - sleepstudy, 126
 - toenail, 128
 - VerbAgg, 135
- * **htest**
 - bootMer, 9
- * **methods**
 - isSingular, 55
 - profile-methods, 103
 - ranef, 110
- * **misc**
 - drop1.merMod, 25
- * **models**
 - allFit, 5
 - bootMer, 9
 - factorize, 29
 - fixef, 29
 - glmer, 37
 - glmer.nb, 40
 - influence.merMod, 49
 - lmer, 57
 - lmList, 67
 - modular, 81
 - nlmer, 91
 - ranef, 110
 - VarCorr, 130
 - vcov.merMod, 133
- * **utilities**
 - devfun2, 24
 - factorize, 29
 - getME, 32
 - glmerLaplaceHandle, 42
 - isSingular, 55
 - mkNewReTrms, 77
 - prt-utilities, 106
 - .makeCC (lmerControl), 60
 - .prt.VC (prt-utilities), 106
 - .prt.aictab (prt-utilities), 106
 - .prt.call (prt-utilities), 106
 - .prt.family (prt-utilities), 106
 - .prt.grps (prt-utilities), 106
 - .prt.methTit (prt-utilities), 106

- .prt.resids (prt-utilities), 106
- .prt.warn (prt-utilities), 106
- .simulateFun (simulate.merMod), 123
- abs, 100
- AIC, 108
- allFit, 5, 18, 65
- anova, 73, 115
- anova.merMod (merMod-class), 71
- anyStructured (isNewMerMod), 53
- Arabidopsis, 7
- as, 69
- as.data.frame, 105
- as.data.frame.ranef.mer (ranef), 110
- as.data.frame.thpr (profile-methods), 103
- as.data.frame.VarCorr.merMod (VarCorr), 130
- as.formula, 59
- as.function.merMod (merMod-class), 71
- BIC, 108
- bobyqa, 62, 64, 83, 94
- boot, 10, 11
- boot.ci, 11, 16, 17
- bootMer, 5, 9, 16, 17, 102, 105, 110, 125
- cake, 12
- call, 107
- cat, 106
- cbpp, 13
- cbpp2 (cbpp), 13
- character, 16, 62, 106–108, 131
- check_model, 99
- checkConv, 15
- class, 10, 68, 107
- clusterExport, 10
- coef, 69
- coef.merMod (merMod-class), 71
- confint, 69, 105
- confint.merMod, 11, 16, 110
- confint.thpr (confint.merMod), 16
- contr.treatment, 31
- convergence, 15, 18, 64, 65
- cooks.distance, 49, 50
- cooks.distance.influence.merMod (influence.merMod), 49
- cooks.distance.merMod (influence.merMod), 49
- cov2sdcor (vcconv), 132
- Covariance, 53, 80
- culcitalogreg, 21, 22, 23
- culcitalvolume, 22, 22
- Cv_to_Sv (vcconv), 132
- Cv_to_Vv (vcconv), 132
- data.frame, 105
- densityplot, 101
- densityplot.thpr (plots.thpr), 100
- deriv, 91
- devcomp, 24
- devfun2, 24
- deviance.merMod (merMod-class), 71
- df.residual, 108
- df.residual.merMod (merMod-class), 71
- dfbeta, 49, 50
- dfbeta.influence.merMod (influence.merMod), 49
- dfbetas, 49, 50
- dfbetas.influence.merMod (influence.merMod), 49
- dgCMatrix, 33
- diagnostic.plots (plot.merMod), 97
- diagnostic_plots (plot.merMod), 97
- diagnosticPlots (plot.merMod), 97
- diagnostics (plot.merMod), 97
- dotplot, 112
- dotplot.ranef.mer (ranef), 110
- drop1, 25, 59, 91
- drop1.merMod, 25, 75
- dummy, 27
- Dyestuff, 28
- Dyestuff2 (Dyestuff), 28
- environment, 84
- envRefClass, 43, 44, 70, 76, 88, 117
- expandDoubleVerts (reformulas_imports), 115
- extractAIC, 72
- extractAIC.merMod (merMod-class), 71
- factor, 27, 89
- factorize, 29
- family, 38, 43, 44, 67, 70, 73, 83, 107
- family.merMod (merMod-class), 71
- findbars, 78, 79, 90
- findbars (reformulas_imports), 115
- fitted, 69

- fitted.merMod (merMod-class), 71
- fixed.effects (fixef), 29
- fixef, 29, 33, 35, 69
- forceNewMerMod (isNewMerMod), 53
- formatVC, 131
- formatVC (prt-utilities), 106
- formula, 67, 69, 122, 123
- formula.merMod (merMod-class), 71
- fortify, 30
- fortify.merMod, 75
- function, 26, 86, 102, 108, 131
- getCall, 35, 107
- getData (fortify), 30
- getDoublevertDefault, 31, 57
- getL (getME), 32
- getL, merMod-method (getME), 32
- getME, 24, 25, 32, 41, 56, 74, 78, 134
- getOption, 92
- getReCovs, 35
- getSingTol (isSingular), 55
- GHrule, 36, 46
- glFormula (modular), 81
- glm, 38, 39, 43, 59, 67, 83
- glmer, 4, 9, 10, 32, 34, 36, 37, 41, 42, 49, 60, 65, 70, 72, 73, 75, 76, 88, 89, 101, 124, 128, 134
- glmer.nb, 35, 39, 40
- glmerControl, 34, 38, 41, 58, 91
- glmerControl (lmerControl), 60
- glmerLaplaceHandle, 42
- glmerMod, 35
- glmerMod-class (merMod-class), 71
- glmFamily, 43, 43, 44
- glmFamily-class, 43
- glmResp, 69–71, 75, 82
- glmResp (lmResp), 69
- glmResp-class (lmResp-class), 70
- golden (golden-class), 44
- golden-class, 44
- gopherdat2, 45
- GQdk, 37, 46, 47
- GQN (GQdk), 46
- grouseticks, 47
- grouseticks_agg (grouseticks), 47
- hatvalues.merMod, 48
- infIndexPlot, 51
- influence, 49
- influence.measures, 51
- influence.merMod, 49
- influencePlot, 99
- InstEval, 52
- isGLMM (isREML), 54
- isGLMM.merMod, 75
- isLMM (isREML), 54
- isLMM.merMod, 75
- isNested (reformulas_imports), 115
- isNewMerMod, 53
- isNLMM (isREML), 54
- isNLMM.merMod, 75
- isREML, 34, 54, 72
- isREML.merMod, 75
- isSingular, 19, 55, 64, 116
- isSingular, Covariance.ar1-method (isSingular), 55
- isSingular, Covariance.cs-method (isSingular), 55
- isSingular, Covariance.diag-method (isSingular), 55
- isSingular, Covariance.us-method (isSingular), 55
- isSingular, merMod-method (isSingular), 55
- isSingular-methods (isSingular), 55
- lFormula (modular), 81
- library, 62
- list, 32, 34, 35, 38, 41, 46, 58, 64, 65, 68, 77, 78, 82, 87, 92, 93, 104, 107, 108
- llikAIC (prt-utilities), 106
- lm, 59, 60, 67, 68
- lme4 (lme4-package), 4
- lme4-package, 4
- lme4_testlevel, 57
- lmer, 4, 9, 10, 25, 32, 36, 39, 49, 57, 65, 67, 70, 72, 73, 75, 76, 80, 82, 89, 91, 100, 101, 103, 104, 124, 128, 132, 134
- lmerControl, 5, 15, 19, 38, 58, 60, 77, 82, 83, 91, 93, 104
- lmerMod, 113, 115
- lmerMod-class (merMod-class), 71
- lmerResp, 69–71, 75, 82
- lmerResp (lmResp), 69
- lmerResp-class (lmResp-class), 70
- lmList, 67, 68, 69

- lmList4, [68](#), [122](#)
- lmList4-class, [68](#)
- lmResp, [42](#), [69](#), [69](#), [70](#), [71](#)
- lmResp-class, [70](#)
- log, [105](#)
- log.thpr (profile-methods), [103](#)
- logical, [25](#), [41](#), [107](#)
- logLik, [69](#), [73](#), [108](#)
- logLik.merMod (merMod-class), [71](#)
- logProf (profile-methods), [103](#)
- matrix, [17](#), [91](#)
- mcmcscamp (pvalues), [109](#)
- merMod, [5](#), [16](#), [24](#), [25](#), [32](#), [36](#), [39](#), [49](#), [53](#), [54](#),
[59](#), [70](#), [72](#), [73](#), [76](#), [77](#), [89](#), [101](#), [111](#),
[117](#), [121–123](#), [127](#), [130](#), [131](#), [134](#)
- merMod (merMod-class), [71](#)
- merMod-class, [71](#)
- merPredD, [42](#), [71](#), [75](#), [75](#), [76](#)
- merPredD-class, [76](#)
- methods, [68](#), [69](#)
- methTitle (prt-utilities), [106](#)
- mkDataTemplate (mkSimulateTemplate), [79](#)
- mkGlmDevfun, [73](#)
- mkGlmDevfun (modular), [81](#)
- mkLmerDevfun, [73](#), [82](#)
- mkLmerDevfun (modular), [81](#)
- mkMerMod, [76](#), [83](#)
- mkNewReTrms, [77](#)
- mkParsTemplate (mkSimulateTemplate), [79](#)
- mkRespMod, [78](#), [79](#), [90](#)
- mkReTrms, [77–79](#), [82](#), [84](#), [90](#)
- mkReTrms (reformulas_imports), [115](#)
- mkSimulateTemplate, [79](#)
- mkVarCorr, [80](#)
- mlist2vec (vcconv), [132](#)
- model.frame, [77](#)
- model.frame.merMod (merMod-class), [71](#)
- model.matrix, [27](#), [28](#)
- model.matrix.default, [39](#), [82](#), [92](#)
- model.matrix.merMod (merMod-class), [71](#)
- model.offset, [39](#), [59](#), [67](#), [82](#), [92](#)
- modular, [5](#), [38](#), [62](#), [81](#), [83](#)
- na.fail, [124](#)
- na.pass, [102](#)
- name, [35](#)
- namedList, [85](#)
- negative.binomial, [41](#), [42](#)
- negative.binomial (glmer.nb), [40](#)
- Nelder_Mead, [62](#), [64](#), [83](#), [88](#)
- Nelder_Mead (NelderMead), [86](#)
- NelderMead, [86](#), [87](#), [88](#)
- NelderMead (NelderMead-class), [87](#)
- NelderMead-class, [87](#)
- new, [44](#), [88](#)
- ngrps, [73](#), [89](#), [108](#)
- ngrps.merMod (merMod-class), [71](#)
- nlformula, [78](#), [79](#), [90](#)
- nlmer, [4](#), [32](#), [39](#), [60](#), [65](#), [70](#), [73](#), [75](#), [76](#), [88](#), [90](#),
[91](#), [132](#)
- nlmerControl (lmerControl), [60](#)
- nlmerMod-class (merMod-class), [71](#)
- nlminb, [62](#), [83](#), [94](#)
- nlminbwrap, [62](#)
- nlminbwrap (nloptwrap), [93](#)
- nloptwrap, [62](#), [65](#), [93](#)
- nlsResp, [69–71](#), [75](#), [90](#)
- nlsResp (lmResp), [69](#)
- nlsResp-class (lmResp-class), [70](#)
- nobars, [78](#), [79](#), [90](#)
- nobars (reformulas_imports), [115](#)
- nobs, [108](#)
- nobs (merMod-class), [71](#)
- numeric, [107](#)
- offset, [39](#), [59](#), [67](#), [82](#), [92](#)
- optim, [62](#), [64](#), [83](#), [86](#)
- optimize, [41](#)
- optimizeGlm (modular), [81](#)
- optimizeLmer (modular), [81](#)
- options, [65](#)
- pairs, [69](#)
- Pastes, [95](#)
- PBmodcomp, [11](#)
- Penicillin, [96](#)
- plot, [69](#)
- plot.lmList (lmList), [67](#)
- plot.merMod, [60](#), [75](#), [97](#)
- plots.thpr, [100](#)
- pnbinom, [42](#)
- predict, [69](#), [101](#)
- predict.merMod, [73](#), [75](#), [101](#)
- print, [69](#), [101](#), [106](#), [131](#)
- print.merMod (merMod-class), [71](#)
- print.summary.merMod (summary), [127](#)
- print.VarCorr.merMod (VarCorr), [130](#)

- profile, [16](#), [17](#), [100](#), [101](#), [103](#), [105](#)
- profile(.), [17](#)
- profile-methods, [103](#)
- profile.merMod, [17](#), [75](#), [110](#)
- profile.merMod (profile-methods), [103](#)
- prt-utilities, [106](#)
- pvalues, [5](#), [11](#), [109](#)

- qqmath, [99](#), [112](#)
- qqmath.merMod (plot.merMod), [97](#)
- qqmath.ranef.mer, [98](#)
- qqmath.ranef.mer (ranef), [110](#)
- qqnorm, [69](#)
- qqplot (plot.merMod), [97](#)

- ranef, [35](#), [69](#), [110](#)
- ranef.merMod, [75](#)
- rankMatrix, [63](#), [64](#)
- ReferenceClasses, [5](#), [84](#)
- refit, [11](#), [41](#), [113](#), [115](#)
- refit.merMod, [75](#)
- refitML, [114](#), [114](#)
- refitML.merMod, [75](#)
- reformulas_imports, [115](#)
- reformulate, [59](#)
- REMLcrit (merMod-class), [71](#)
- reOnly (reformulas_imports), [115](#)
- rePCA, [55](#), [56](#), [116](#)
- rePos, [117](#), [117](#)
- rePos-class, [117](#)
- residuals, [69](#), [107](#)
- residuals.glm, [118](#)
- residuals.glmResp (residuals.merMod), [118](#)
- residuals.lm, [118](#)
- residuals.lme, [118](#)
- residuals.lmResp (residuals.merMod), [118](#)
- residuals.merMod, [75](#), [118](#)
- rnorm, [10](#), [80](#)

- salamander, [119](#)
- scale, [18](#)
- schizophrenia, [120](#)
- sdcor2cov (vcconv), [132](#)
- selfStart, [91](#)
- set.seed, [9](#), [124](#)
- show, [69](#), [106](#)
- show, lmList4-method (lmList4-class), [68](#)
- show, merMod-method (merMod-class), [71](#)
- show.merMod (merMod-class), [71](#)
- show.summary.merMod (summary), [127](#)
- sigma, [58](#), [69](#), [121](#), [134](#)
- sigma.merMod, [75](#), [128](#), [132](#)
- simulate, [113](#), [122](#), [123](#)
- simulate.formula, [122](#)
- simulate.merMod, [9](#), [10](#), [75](#), [80](#), [123](#)
- simulateResiduals, [99](#)
- sleepstudy, [126](#)
- splom, [101](#)
- splom.thpr (plots.thpr), [100](#)
- SSbiexp, [91](#)
- SSlogis, [91](#)
- stop, [68](#), [86](#)
- subbars, [78](#), [79](#), [90](#)
- subbars (reformulas_imports), [115](#)
- summary, [69](#), [106](#), [127](#), [128](#)
- summary.merMod, [75](#)
- Sv_to_Cv (vcconv), [132](#)
- symnum, [72](#), [127](#)

- terms, [102](#)
- terms.merMod (merMod-class), [71](#)
- theta.ml, [41](#), [42](#)
- toenail, [128](#)
- troubleshooting, [129](#)

- update, [59](#), [69](#), [73](#), [91](#), [113](#)
- update.formula, [73](#)
- update.merMod, [114](#), [115](#)
- update.merMod (merMod-class), [71](#)
- updateGlmDevfun (modular), [81](#)

- VarCorr, [80](#), [106](#), [108](#), [130](#), [131](#)
- varianceProf, [104](#)
- varianceProf (profile-methods), [103](#)
- vcconv, [132](#)
- vcov, [35](#)
- vcov.merMod, [127](#), [133](#)
- vcov.summary.merMod (vcov.merMod), [133](#)
- vec2mlist (vcconv), [132](#)
- vec2STlist (vcconv), [132](#)
- VerbAgg, [135](#)
- Vv_to_Cv (vcconv), [132](#)

- warning, [67](#), [68](#), [86](#)
- weights.merMod (merMod-class), [71](#)

- xyplot, [100](#), [101](#), [105](#)
- xyplot.thpr (plots.thpr), [100](#)