Package ‘lme4’

September 27, 2017

Version 1.1-14
Title Linear Mixed-Effects Models using 'Eigen' and S4
Contact LME4 Authors <lme4-authors@lists.r-forge.r-project.org>
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 (>= 3.0.2), Matrix (>= 1.1.1), methods, stats
LinkingTo Rcpp (>= 0.10.5), RcppEigen
Imports graphics, grid, splines, utils, parallel, MASS, lattice, nlme
(>= 3.1-123), minqa (>= 1.1.15), nloptr (>= 1.0.4)
Suggests knitr, boot, PKPDmodels, MEMSS, testthat (>= 0.8.1), ggplot2,
mlmRev, optimx (>= 2013.8.6), gamm4, pbkrtest, HSAUR2, numDeriv
VignetteBuilder knitr
LazyData yes
License GPL (>= 2)
BugReports https://github.com/lme4/lme4/issues
NeedsCompilation yes
Author Douglas Bates [aut],
    Martin Maechler [aut],
    Ben Bolker [aut, cre],
    Steven Walker [aut],
    Rune Haubo Bojesen Christensen [ctb],
    Henrik Singmann [ctb],
    Bin Dai [ctb],
    Gabor Grothendieck [ctb],
    Peter Green [ctb]
Maintainer Ben Bolker <bbolker+lme4@gmail.com>
Repository CRAN
Date/Publication 2017-09-27 08:28:13 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>lme4-package</td>
<td>3</td>
</tr>
<tr>
<td>Arabidopsis</td>
<td>5</td>
</tr>
<tr>
<td>bootMer</td>
<td>6</td>
</tr>
<tr>
<td>cake</td>
<td>9</td>
</tr>
<tr>
<td>cbpp</td>
<td>10</td>
</tr>
<tr>
<td>checkConv</td>
<td>11</td>
</tr>
<tr>
<td>confint.merMod</td>
<td>12</td>
</tr>
<tr>
<td>convergence</td>
<td>14</td>
</tr>
<tr>
<td>devcomp</td>
<td>16</td>
</tr>
<tr>
<td>drop1.merMod</td>
<td>17</td>
</tr>
<tr>
<td>dummy</td>
<td>19</td>
</tr>
<tr>
<td>Dyestuff</td>
<td>20</td>
</tr>
<tr>
<td>expandDoubleVerts</td>
<td>21</td>
</tr>
<tr>
<td>factorize</td>
<td>22</td>
</tr>
<tr>
<td>findbars</td>
<td>22</td>
</tr>
<tr>
<td>fixef</td>
<td>23</td>
</tr>
<tr>
<td>fortify</td>
<td>24</td>
</tr>
<tr>
<td>getME</td>
<td>25</td>
</tr>
<tr>
<td>GHRule</td>
<td>28</td>
</tr>
<tr>
<td>glmer</td>
<td>29</td>
</tr>
<tr>
<td>glmer.nb</td>
<td>32</td>
</tr>
<tr>
<td>glmerLaplaceHandle</td>
<td>34</td>
</tr>
<tr>
<td>glmFamily</td>
<td>35</td>
</tr>
<tr>
<td>glmFamily-class</td>
<td>35</td>
</tr>
<tr>
<td>golden-class</td>
<td>36</td>
</tr>
<tr>
<td>GQdk</td>
<td>37</td>
</tr>
<tr>
<td>grouseticks</td>
<td>38</td>
</tr>
<tr>
<td>hatvalues.merMod</td>
<td>39</td>
</tr>
<tr>
<td>InstEval</td>
<td>40</td>
</tr>
<tr>
<td>isNested</td>
<td>41</td>
</tr>
<tr>
<td>isREML</td>
<td>41</td>
</tr>
<tr>
<td>lmer</td>
<td>42</td>
</tr>
<tr>
<td>lmerControl</td>
<td>45</td>
</tr>
<tr>
<td>lmList</td>
<td>51</td>
</tr>
<tr>
<td>lmList4-class</td>
<td>52</td>
</tr>
<tr>
<td>lmResp</td>
<td>52</td>
</tr>
<tr>
<td>lmResp-class</td>
<td>53</td>
</tr>
<tr>
<td>merMod-class</td>
<td>54</td>
</tr>
<tr>
<td>merPredD</td>
<td>55</td>
</tr>
<tr>
<td>merPredD-class</td>
<td>59</td>
</tr>
<tr>
<td>mkdevfun</td>
<td>60</td>
</tr>
<tr>
<td>mkMerMod</td>
<td>61</td>
</tr>
<tr>
<td>mkRespMod</td>
<td>62</td>
</tr>
<tr>
<td>mkReTrms</td>
<td>62</td>
</tr>
<tr>
<td>mkSimulateTemplate</td>
<td>63</td>
</tr>
<tr>
<td>mkVarCorr</td>
<td>64</td>
</tr>
</tbody>
</table>
lme4-package

Description

lme4 provides functions for fitting and analyzing mixed models: linear (\texttt{lmer}), generalized linear (\texttt{glmer}) and nonlinear (\texttt{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 mcmcsamp (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.)
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.Dish, or Transplant)
- **totalNfruits**: total fruit set per plant (integer)

Source

From Josh Banta

References


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

```r
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

- `x` a fitted `merMod` object: see `lmer`, `glmer`, etc.
- `FUN` a function taking a fitted `merMod` object as input and returning the statistic of interest, which must be a (possibly named) numeric vector.
- `nsim` number of simulations, positive integer; the bootstrap \( B \) (or \( R \)).
- `seed` optional argument to `set.seed`.
- `use.u` 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).
- `re.form` formula, NA (equivalent to `use.u=FALSE`), or NULL (equivalent to `use.u=TRUE`): alternative to `use.u` for specifying which random effects to incorporate. See `simulate.merMod` for details.
- `type` character string specifying the type of bootstrap, "parametric" or "semiparametric"; partial matching is allowed.
- `verbose` logical indicating if progress should print output
- `.progress` 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 `tcltk` package is loaded; or "win" on Windows systems. Progress bars are disabled (with a message) for parallel operation.
- `PBargs` a list of additional arguments to the progress bar function (the package authors like `list(style=3)`).
- `parallel` 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").
- `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 boot call.
Details

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

The working name for bootMer() was “simulestimate()”, as it is an extension of simulate (see `simulate.lmerMod`), 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 $\epsilon$, 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.

Note

If you are using `parallel="snow"`, you will need to run `clusterEvalQ(cl1,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


See Also

- `confint.merMod`, for a more specific approach to bootstrap confidence intervals on parameters.
- `refit()`, or `PBmodcomp()` from the `pbkrtest` 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

```r
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

  ## Copy of unexported stats::format.perc helper function
  format.perc <- function(probs, digits) {
    paste(format(100 * probs, trim = TRUE,
                 scientific = FALSE, digits = digits),
         "%")
  }

  ## Extract all CIs (somewhat awkward)
  bCI.tab <- function(b, ind=length(b$to), type="perc", conf=0.95) {
    btab0 <- t(sapply(as.list(seq(ind)),
```
cake

function(i)
  boot::boot.ci(b,index=i,conf=conf, type=type)$percent)
btab <- btab[4:5]
rownames(btab) <- names(b$0)
a <- (1 - conf)/2
a <- c(a, 1 - a)
pct <- format.perc(a, 3)
colnames(btab) <- pct
return(btab)
}

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

## Check stored values from a longer (1000-replicate) run:
(load(system.file("testdata","boo01.RData", package="lme4")))
plot(boo01L, index=3)
mean(boo01L$t[,"sig01"]==0) ## note point mass at zero!
}

## if boot package available

---

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 Q to QU
- **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


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)

data(cbpp)

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).

Format

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.

Details

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

Examples

```r
## 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))
```

**Description**
Primarily internal code for checking optimization convergence, see convergence for a more detailed discussion.

**Usage**
checkConv(derivs, coefs, ctrl, lbound, debug = FALSE)

**Arguments**
- **derivs**: typically the "derivs" attribute of optimizeLmer(); 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
  `lmerControl()$checkConv`
- **lbound**: vector of lower bounds for random-effects parameters only (length is taken to determine number of RE parameters)
- **debug**: enable debugging output, useful if some checks are on "ignore", but would "trigger"
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 lmer Fit**

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

Usage
```r
## 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 = TRUE, ...)
## S3 method for class 'thpr'
confint(object, parm, level = 0.95,
  zeta, non mono.tol=1e-2,
...)
```

Arguments
- `object`: a fitted `lmer` model or profile
- `parm`: 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) "theta_" or "beta_" to specify variance-covariance or fixed effects parameters only: see the which parameter of `profile`.
- `level`: confidence level < 1, typically above 0.90.
- `method`: a `character` string determining the method for computing the confidence intervals.
- `zeta`: (for `method = "profile"` only:) likelihood cutoff (if not specified, as by default, computed from `level`).
- `nsim`: number of simulations for parametric bootstrap intervals.
bootstrapping function; if NULL, 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.

boot.type bootstrap confidence interval type, as described in `boot.ci`. (Methods ‘stud’ and ‘bca’ are unavailable because they require additional components to be calculated.)

quiet (logical) suppress messages about computationally intensive profiling?

oldNames (logical) use old-style names for variance-covariance parameters, e.g. ".sig01", rather than newer (more informative) names such as "sd_(Intercept)|Subject"? (See `signames` argument to `profile`).

non_mono.tol tolerance for detecting a non-monotonic profile and warning/falling back to linear interpolation

... additional parameters to be passed to `profile.merMod` or `bootMer`, respectively.

Details

Depending on the method specified, `confint()` computes confidence intervals by

"profile": computing a likelihood profile and finding the appropriate cutoffs based on the likelihood ratio test;

"Wald": approximating the confidence intervals (of fixed-effect parameters only; all variance-covariance parameters CIs will be returned as NA) based on the estimated local curvature of the likelihood surface;

"boot": performing parametric bootstrapping with confidence intervals computed from the bootstrap distribution according to `boot.type` (see `bootMer`, `boot.ci`).

Value

a numeric table (matrix with column and row names) of confidence intervals; the confidence intervals are computed on the standard deviation scale.

Note

The default method "profile" amounts to

```r
confint(profile(object, which=parm), signames=oldNames, ...),
level, zeta)
```

where the `profile` method `profile.merMod` does almost all the computations. Therefore it is typically advisable to store the `profile(.)` result, say in `pp`, and then use `confint(pp, level=*)` e.g., for different levels.
### Examples

```r
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
fmlw <- confint(fm1, method="Wald")# very fast, but ....
fmlw
testLevel <- if (nzchar(s <- Sys.getenv("LME4_TEST_LEVEL"))) as.numeric(s) else 1
if(interactive() || testLevel >= 3) {
  ## ~20 seconds, MacBook Pro laptop
  system.time(fm1p <- confint(fm1, method="profile", oldNames = FALSE))
  ## ~ 40 seconds
  system.time(fm1b <- confint(fm1b, method="boot",
                           .progress="txt", PBargs=list(style=3)))
} else
  load(system.file("testdata","confint_ex.rda",package="lme4"))
fm1p
fm1b
```

---

### Assessing Convergence for Fitted Models

#### Description

The `lme4` package 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 reliably whether such algorithms have converged is difficult. For example, evaluating the Karush-Kuhn-Tucker conditions (convergence criteria which in the simplest case of non-constrained optimization reduce 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 when they have to be estimated by finite differences (as in the case of `lme4`; direct computation of derivatives based on analytic expressions may eventually be available for some special classes, but we have not yet implemented them) they may not be sufficiently accurate for reliable convergence testing.

- 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. don’t scale random-effects (Cholesky) gradients, 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, or

2. scale gradients by the inverse Cholesky factor of the Hessian, equivalent to scaling by the estimated Wald standard error of the estimated parameters. The latter approach is used in the current version of **lme4**; it has the disadvantage that it requires us to estimate the Hessian (although the Hessian is required for reliable estimation of the fixed-effect standard errors for GLMMs in any case).

- 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.

If you do see convergence warnings, and want to trouble-shoot/double-check the results, the following steps are recommended (examples are given below):

- double-check the model specification and the data for mistakes
- center and scale continuous predictor variables (e.g. with **scale**)
- check for singularity: if any of the diagonal elements of the Cholesky factor are zero or very small, the convergence testing methods may be inappropriate (see examples)
- double-check the Hessian calculation with the more expensive Richardson extrapolation method (see examples)
- restart the fit from the apparent optimum, or from a point perturbed slightly away from the optimum
- try 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.

To quote Douglas Adams, **we apologize for the inconvenience**.

**See Also**

- **lmerControl**

**Examples**

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

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

## 2. check singularity
diag.vals <- getME(fm1,"theta")[getME(fm1,"lower") == 0]
any(diag.vals < 1e-6) # FALSE

## 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(scg <- solve(chol(hess), grad))
}
## compare with internal calculations:
fm1@optinfo$derivs

## 4. restart the fit from the original value (or
## a slightly perturbed value):
fml.restart <- update(fm1, start=pars)

## 5. try all available optimizers

source(system.file("utils", "allFit.R", package="lme4"))
fml.all <- allFit(fm1)
ss <- summary(fml.all)
ss$fixef  ## extract fixed effects
ss$lik    ## log-likelihoods
ss$sds     ## SDs and correlations
ss$theta  ## Cholesky factors
ss$which.OK  ## which fits worked

---

**devcomp**

*Extract the deviance component list*

### Description

Return the deviance component list

### Usage

```r
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"`.

### Description

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

### Usage

```r
## 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 \( \chi^2 \) 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 function to be used when `test="user"`. It must allow arguments `scale` and `k`, but these may be ignored (e.g. specified in `dots`). The first two arguments must be `object`, the full model fit, and `objectDrop`, a reduced model. If `objectDrop` is missing, `sumFun` should return a vector of 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

```r
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 {
      krttest <- KRModcomp(object,objectDrop)
      unlist(krttest$stats[krnames])
    }
    attr(r,"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
```
**dummy**

```r
## 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)
}
drop(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

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

### Arguments

- `f` An object coercible to `factor`.
- `levelsToKeep` An optional character vector giving the subset of `levels(f)` to be converted to dummy variables.

### Value

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

### Examples

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

Source

O.L. Davies and P.L. Goldsmith (eds), *Statistical Methods in Research and Production, 4th ed.*, Oliver and Boyd, (1972), section 6.4


Examples

```r
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,
```
Description

From the right hand side of a formula for a mixed-effects model, expand terms with the double vertical bar operator into separate, independent random effect terms.

Usage

```r
expandDoubleVerts(term)
```

Arguments

term a mixed-model formula

Value

the modified term

Note

Because `||` works at the level of formula parsing, it has no way of knowing whether a variable is a factor. It just takes the terms within a random-effects term and literally splits them into the intercept and separate no-intercept terms, e.g. `1+x+y|f` would be split into `(1|f) + (0+x|f) + (0+y|f)`. However, `||` will fail to break up factors into separate terms; the `dummy` function can be useful in this case, although it is not as convenient as `||`.

See Also

`formula`, `model.frame`, `model.matrix`, `dummy`.

Other utilities: `mkRespMod`, `mkReTrms`, `nlformula`, `nobars`, `subbars`

Examples

```r
m <- ~ x + (x || g)
expandDoubleVerts(m)
set.seed(101)
 dd <- expand.grid(f=factor(letters[1:3]),g=factor(1:200),rep=1:3)
dd$y <- simulate(-f + (1|g) + (0+dummy(f,"b")|g) + (0+dummy(f,"c")|g),
  newdata=dd,
  newparams=list(beta=rep(0,3),
                 theta=c(1,2,1),
                 sigma=1),
  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

**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**

```r
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

findbars

**Determine random-effects expressions from a formula**

**Description**

From the right hand side of a formula for a mixed-effects model, determine the pairs of expressions that are separated by the vertical bar operator. Also expand the slash operator in grouping factor expressions and expand terms with the double vertical bar operator into separate, independent random effect terms.

**Usage**

```r
findbars(term)
```

**Arguments**

- `term` a mixed-model formula
Value

pairs of expressions that were separated by vertical bars

Note

This function is called recursively on individual terms in the model, which is why the argument is called term and not a name like form, indicating a formula.

See Also

formula, model.frame, model.matrix.
Other utilities: mkRespMod, mkReTrms, nlformula, nobars, subbars

Examples

findbars(f1 <- Reaction ~ Days + (Days | Subject))
## => list(Days | Subject)
## These two are equivalent: % tests in ../inst/tests/test-doubleVertNotation.R
findbars(y ~ Days + (1 | Subject) + (0 + Days | Subject))
findbars(y ~ Days + (Days || Subject))
## => list of length 2: list(1 | Subject, 0 + Days | Subject)
findbars(~ 1 + (1 | batch / cask))
## => list of length 2: list(1 | cask:batch, 1 | batch)

fixef
Extract fixed-effects estimates

Description

Extract the fixed-effects estimates

Usage

## S3 method for class ‘merMod’
fixef(object, add.dropped=FALSE, ...)

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?
... 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

```r
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**

add information to data based on a fitted model

**Usage**

```r
fortify.merMod(model, data = getData(model),
               ...)```

**Arguments**

- `model` fitted model
- `data` original data set, if needed
- `...` additional arguments

**Details**

fortify is a function 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 is currently an experimental feature.
**getME**

Extract or Get Generalized Components from a Fitted Mixed Effects Model

**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**

```r
getME(object, name, ...)  
## S3 method for class 'merMod'
getME(object, name = c("X", "Z", "Zt", "Ztlist", "mmList", "y", "mu", "u", "b",  
"A", "RX", "RZX", "sigma", "flist",  
"fixef", "beta", "theta", "ST", "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", "glmer.nb.theta"),  
...)
```

**Arguments**

- `object` a fitted mixed-effects model of class "merMod", i.e., typically the result of `lmer()`, `glmer()` or `nlmer()`.
- `name` a character vector specifying the name(s) of the “component”. If `length(name) > 1` or if `name = "ALL"`, a named list of components will be returned. Possible values are:
  
  "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  
  do.call(Mat**:x**:rBind,getME(.,"Ztlist"))  
  "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 of the random effects.

"Lambdat": transpose of \( \Lambda \) above.

"Lind": index vector for inserting elements of \( \theta \) into the nonzeros of \( \Lambda \).

"Tlist": vector of template matrices from which the blocks of \( \Lambda \) 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": random-effects parameter estimates: these are parameterized as the relative Cholesky factors of each random effect term

"ST": A list of S and T factors in the TSST’ Cholesky factorization of the relative variance matrices of the random effects associated with each random-effects term. The unit lower triangular matrix, \( T \), and the diagonal matrix, \( S \), for each term are stored as a single matrix with diagonal elements from \( S \) and off-diagonal elements from \( T \).

"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

"cnms": the “component names”, a list.

"REML": \( \theta \) 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**: θ 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 model parameters (random effects parameters only).

"devfun": deviance function (so far only available for LMMs)

"glmer.nb.theta": negative binomial θ 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.
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 25 - 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 \texttt{SparseGrid}'s hidden \texttt{GQN}().

Value

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

See Also

a different interface is available via \texttt{GQdk}().

Examples

\begin{verbatim}
(r5 <- GHrule(5, asMatrix=FALSE))
## second, fourth, sixth, eighth and tenth central moments of the
## standard Gaussian density
with(r5, sapply(seq(2, 10, 2), function(p) sum(w * z^p)))
\end{verbatim}

\begin{verbatim}

\texttt{glmer} \hspace{1cm} \textit{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

\begin{verbatim}
\texttt{glmer(formula, data = \texttt{NULL}, family = \texttt{gaussian}, control = \texttt{glmerControl}(),}
\texttt{ start = \texttt{NULL}, verbose = \texttt{0L}, nAGQ = \texttt{1L}, subset, weights, na.action,}
\texttt{ offset, contrasts = \texttt{NULL}, mustart, etastart,}
\texttt{ devFunOnly = \texttt{FALSE}, \ldots})
\end{verbatim}

Arguments

\begin{verbatim}
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 \texttt{~} operator and the terms, separated by + operators, on the right. Random-effects terms are distinguished by vertical bars ("\mid") separating expressions for design matrices from grouping factors.
\end{verbatim}
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 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).

family
a GLM family, see glm and family.

control
a list (of correct class, resulting from lmerControl() or glmerControl() re-
spectively) containing control parameters, including the nonlinear optimizer to
be used and parameters to be passed through to the nonlinear optimizer, see the
*lmerControl documentation for details.

start
a named list of starting values for the parameters in the model, or a numeric
vector. A numeric start argument will be used as the starting value of theta.
If start is a list, the theta element (a numeric vector) is used as the starting
value for the first optimization step (default=1 for diagonal elements and 0 for
off-diagonal elements of the lower Cholesky factor); the fitted value of theta
from the first step, plus start[["fixef"]], are used as starting values for the
second optimization step. If start has both fixef and theta elements, the first
optimization step is skipped. For more details or finer control of optimization,
see modular.

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.

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
rewighted 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 de-
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 a priori 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.
mustart  optional starting values on the scale of the conditional mean, as in `glm`; see there for details.

etastart  optional starting values on the scale of the unbounded predictor as in `glm`; 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 exactly the same values on subsequent calls (but the results should always be within machine tolerance).

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

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.

Value

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

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

```r
## generalised 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),
    data = cbpp, family = 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.
```
(glm1a <- glmer(cbind/incidence, size - incidence) ~ period + (1 | herd),
cbpp, binomial, 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)
(glm2 <- glmer(cbind/incidence, size - incidence) ~ period +
  (1 | herd) + (1|obs),
  family = binomial, data = cbpp))
anova(glm1,glm2)

## glmer and glm log-likelihoods are consistent
 glm1Devfun <- update(glm1,devFunOnly=TRUE)
 glm0 <- glm(cbind/incidence, size - incidence) ~ period,
  family = binomial, data = cbpp)
## evaluate GLMM deviance at RE variance=theta=0, beta=(GLM coeffs)
 glm1Dev0 <- glm1Devfun(c(0,coef(glm0))))
## compare
 stopifnot(all.equal(glm1Dev0,c(-2*logLik(glm0))))
## the toenail oncholysis data from Backer et al 1998
## these data are notoriously difficult to fit
## Not run:
if (require("HSAUR2")) {
  gm2 <- glmer(outcome~treatment*visit+(1|patientID),
    data=toenail,
    family=binomial,nAGQ=20)
}
## End(Not run)

---

**Fitting Negative Binomial GLMMs**

### 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

```r
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

... arguments as for `glmer(.)` such as formula, data, control, etc, but *not* family!
interval  interval in which to start the optimization. The default is symmetric on log scale around the initially estimated theta.

tol  tolerance for the optimization via optimize.

verbose  logical indicating how much progress information should be printed during the optimization. Use verbose = 2 (or larger) to enable verbose=TRUE in the glmer() calls.

nb.control  optional list, like glmerControl(), used in refit(*, control = control.nb) during the optimization.

initCtrl  (experimental, do not rely on this:) a list with named components as in the default, passed to theta.ml (package MASS) for the initial value of the negative binomial parameter theta. May also include a theta 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  

```r
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)
```
glmerLaplaceHandle

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

Generator object for the glmFamily class

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

Class "glmFamily" - a reference class for family

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)
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

```r
gqdk(d = 1L, k = 1L)
QQN
```

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.

`QQN` 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

`QQN` contains only the non-redundant nodes. To regenerate the whole array of nodes, all possible permutations of axes and all possible combinations of \(\pm 1\) 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](http://www.sparse-grids.de).

Examples

```r
GQdk(2,5) # 53 x 3
QQN[[3]][[5]] # a 14 x 4 matrix
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

Examples

```r
data(grouseticks)
## Figure 1a from Elston et al
par(las=1,bty="1")
tvec <- c(0,1,2,5,20,40,80)
pvec <- c(4,1,3)
with(grouseticks_aggl.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(y=405,lty=2)
## Figure 1b
with(grouseticks_aggl.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_mod) <- glmer(form, family="poisson",data=grouseticks))
```

---

**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**

```r
## 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

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

**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**

```r
str(InstEval)
head(InstEval, 16)
xtabs(~ service + dept, InstEval)
```
isNested  

Is f1 nested within f2?

Description

Does every level of f1 occur in conjunction with exactly one level of f2? The function is based on converting a triplet sparse matrix to a compressed column-oriented form in which the nesting can be quickly evaluated.

Usage

isNested(f1, f2)

Arguments

f1  factor 1
f2  factor 2

Value

TRUE if factor 1 is nested within factor 2

Examples

with(Pastes, isNested(cask, batch)) ## => FALSE
with(Pastes, isNested(sample, batch)) ## => TRUE

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, ...)
isGLMM(x, ...)
lmer

**Fit Linear Mixed-Effects Models**

**Description**

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

**Usage**

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

**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**

```r
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)
isNLMM(nm1)
## 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)))
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 ~ 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. Two vertical bars (||) can be used to specify multiple uncorrelated random effects for the same grouping variable. (Because of the way it is implemented, the ||-syntax works only for design matrices containing numeric (continuous) predictors; to fit models with independent categorical effects, see dummy or the lmer_alt function from the afex package.)

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 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).

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 lmerControl() or glmerControl() respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the *lmerControl documentation for details.

start a named list of starting values for the parameters in the model. For lmer this can be a numeric vector or a list with one component named "theta".

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.

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. Prior weights are not normalized or standardized in any way. In particular, the diagonal of the residual covariance matrix is the squared residual standard deviation parameter sigma times the vector of inverse weights. Therefore, if the weights have relatively large magnitudes, then in order to compensate, the sigma 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 (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 a priori 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 offset. See the `contrasts.arg` of `model.matrix.default`.

- **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 exactly the same values on subsequent calls (but the results should always
  be within machine tolerance).

- ... other potential arguments. A method argument was used in earlier versions of
  the package. Its functionality has been replaced by the `REML` argument.

**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` may fail.

- When handling perfectly collinear predictor variables (i.e. design matrices of less than full
  rank), `lmer` is not quite as sophisticated as some simpler 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, representing the theta vector. This vector defines the scaled variance-covariance
  matrices of the random effects, in the Cholesky parameterization. For models with only sim-
  ple (intercept-only) random effects, theta is a vector of the standard deviations of the random
  effects. For more complex or multiple random effects, running `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.

**Value**

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

**See Also**

- `lm` for linear models; `glmer` for generalized linear; and `nlmer` for nonlinear mixed models.

**Examples**

```r
## 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

str(terms(fm1))
stopifnot(identical(terms(fm1, fixed.only=FALSE),
```
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 = "bobyqa",
             restart_edge = TRUE,
             boundary.tol = 1e-5,
             calc.derivs=TRUE,
             use.last.params=FALSE,
             sparseX = FALSE,
             ## 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.grad = .makeCC("warning", tol = 2e-3, relTol = NULL),
check.conv.singular = .makeCC(action = "ignore", tol = 1e-4),
check.conv.hess = .makeCC(action = "warning", tol = 1e-6),
## optimizer args
optCtrl = list()

glmerControl(optimizer = c("bobyqa", "Nelder_Mead"),
  restart_edge = FALSE,
  boundary.tol = 1e-5,
  calc.derivs=TRUE,
  use.last.params=FALSE,
  sparseX = FALSE,
  tolPwrss=1e-7,
  compDev=TRUE,
  nAGQ0initStep=TRUE,
  ## input checking options
check.nobs.vs.rankZ = "ignore",
check.nobs.vs.nlev = "stop",
check.nlev.gtr.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 = "warning",
check.formula.LHS = "stop",
check.response.not.const = "stop",
## convergence checking options
check.conv.grad = .makeCC("warning", tol = 1e-3, relTol = NULL),
check.conv.singular = .makeCC(action = "ignore", tol = 1e-4),
check.conv.hess = .makeCC(action = "warning", tol = 1e-6),
## optimizer args
optCtrl = list()

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. 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 (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 also need to load the optimx package manually using library(optimx) or require(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.

- **calc derivs**: logical - compute gradient and Hessian of nonlinear optimization solution?

- **use.last.params**: 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.

- **sparseX**: logical - should a sparse model matrix be used for the fixed-effects terms? Currently inactive.

- **restart_edge**: 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 lmerControl.)

- **boundary.tol**: numeric - within what distance of a boundary should the boundary be checked for a better fit? (Set to zero to disable boundary checking.)

- **tolPwrss**: numeric scalar - the tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares step.

- **compDev**: logical scalar - should compiled code be used for the deviance evaluation during the optimization of the parameter estimates?

- **nAGQ@initStep**: Run an initial optimization phase with nAGQ = 0. 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 link="cloglog" and offsets).
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check.nlev.gtr.5</code></td>
<td>rules for checking whether all random effects have ( \geq 5 ) levels. See action.</td>
</tr>
<tr>
<td><code>check.nlev.gtr.1</code></td>
<td>rules for checking whether all random effects have ( &gt; 1 ) level. See action.</td>
</tr>
<tr>
<td><code>check.nobs.vs.rankZ</code></td>
<td>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 &quot;warningSmall&quot; and &quot;stopSmall&quot;, which run the test only if the dimensions of ( Z ) are (&lt; 1\times 10^6). ( \text{nobs} &gt; \text{rank}(Z) ) will be tested for LMMs and GLMMs with estimated scale parameters; ( \text{nobs} \geq \text{rank}(Z) ) will be tested for GLMMs with fixed scale parameter. The rank test is done using the method=&quot;qr&quot; option of the \text{rankMatrix} function.</td>
</tr>
<tr>
<td><code>check.nobs.vs.nlev</code></td>
<td>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. ( \text{nobs}&lt;\text{nlevels} ) will be tested for LMMs and GLMMs with estimated scale parameters; ( \text{nobs}\leq\text{nlevels} ) will be tested for GLMMs with fixed scale parameter.</td>
</tr>
<tr>
<td><code>check.nobs.vs.nRE</code></td>
<td>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 <code>check.nobs.vs.nlev</code>.</td>
</tr>
<tr>
<td><code>check.conv.grad</code></td>
<td>rules for checking the gradient of the deviance function for convergence. A list as returned by \text{.makeCC}, or a character string with only the action.</td>
</tr>
<tr>
<td><code>check.conv.singular</code></td>
<td>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 <code>check.conv.grad</code> above.</td>
</tr>
<tr>
<td><code>check.conv.hess</code></td>
<td>rules for checking the Hessian of the deviance function for convergence.; as for <code>check.conv.grad</code> above.</td>
</tr>
<tr>
<td><code>check.rankX</code></td>
<td>specifying if \text{rankMatrix}(X) should be compared with ( \text{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 &quot;*.drop.cols&quot; options all do drop columns, &quot;stop.deficient&quot; gives an error when the rank is smaller than the number of columns where &quot;ignore&quot; does no rank computation, and will typically lead to less easily understandable errors, later.</td>
</tr>
<tr>
<td><code>check.scaleX</code></td>
<td>check for problematic scaling of columns of fixed-effect model matrix, e.g. parameters measured on very different scales.</td>
</tr>
<tr>
<td><code>check.formula.LHS</code></td>
<td>check whether specified formula has a left-hand side. Primarily for internal use</td>
</tr>
</tbody>
</table>
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.

action
character - generic choices for the severity level of any test. "ignore": skip the test. "warning": warn if test fails. "stop": throw an error if test fails.

tol
numeric - tolerance for check

relTol
numeric - tolerance for checking relative variation

... 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

Examples

str(lmerControl())
str(glmmerControl())

## Not run:
## fit with default "bobyqa" algorithm ...  
fm0 <- lmer(Reaction ~ Days + (1 | Subject), sleepstudy)
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
## or with "Nelder_Mead" (the previous default) ...

```r
defaultControl <- list(algorithm="NLOPT_LN_BOBYQA",
                        xtol_abs=1e-6, ftol_abs=1e-6, maxeval=1e5)
nloptrwrap <- function(fn,par,lb,ub, opts) {
  for (n in names(defaultControl))
    if (is.null(defaultControl[[n]]))
      defaultControl[[n]] <- nloptrwrap(0, par, 0, 0, nloptrwrap(0, par, 0, 0, opts=control,...)
  res <- nloptr(x0=par, eval_f=fn, lb=lower, ub=upper, opts=control,...)
  with(res, list(par = solution,
                  fval = objective,
                  feval = iterations,
                  conv = if (status>0) 0 else status,
                  message = message))
}
```

```r
fm1_nloptr <- update(fm1, control=lmerControl(optimizer=nloptrwrap))
fm1_nloptr_NM <- update(fm1, control=lmerControl(optimizer=nloptrwrap,
                                       optCtrl=list(algorithm="NLOPT_LN_NELDERMEAD"))
```

## other algorithm options include NLOPT_LN_COBYLA, NLOPT_LN_SBPLX

## End(Not run)
**lmList**

*Fit List of lm Objects with a Common Model*

**Description**

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

**Usage**

```
lmList(formula, data, family, subset, weights, na.action, offset, pool = TRUE, ...)
```

**Arguments**

- **formula**: a linear `formula` object of the form `y ~ x1+...+xn | g`. In the formula object, `y` represents the response, `x1,...,xn` the covariates, and `g` the grouping factor specifying the partitioning of the data according to which different `lm` fits should be performed.
- **family**: an optional family specification for a generalized linear model.
- **pool**: logical scalar, should the variance estimate pool the residual sums of squares
- **...**: additional, optional arguments to be passed to the model function or family evaluation.
- **data**: an optional data frame containing the variables named in `formula`. By default the variables are taken from the environment from which `lm` is called. 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 (`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 *a priori* 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`.

**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).

Value

an object of class \texttt{lmlist4} (see there, notably for the methods defined).

See Also

\texttt{lmlist4}

Examples

```r
fm.plm <- lmlist(Reaction ~ Days | Subject, sleepstudy)
coef(fm.plm)
fm.2 <- update(fm.plm, pool = FALSE)
## coefficients are the same, "pooled or unpoled":
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
```

\texttt{lmlist4-class} \hspace{1cm} \textit{Class "lmlist4" of \textquoteleft lm\textquoteright Objects on Common Model}

Description

Class "lmlist4" is an S4 class with basically a list of objects of class \texttt{lm} with a common model (but different data); see \texttt{lmlist()} which returns these.

Package \texttt{nlme}'s \texttt{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 \texttt{new("lmlist4", ...)} or, more commonly, by a call to \texttt{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.

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

See Also

\texttt{lmlist}
Examples

if(getRversion() >= "3.2.0") {
  (mm <- methods(class = "lmList4"))
  ## The S3 ("not S4") ones:
  mm[lattr(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, nlmod=nlmod, 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`
### Description

Reference classes for response modules, including linear models, "lm Resp", generalized linear models, "glm Resp", nonlinear models, "nls Resp" and linear mixed-effects models, "lmer Resp". 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, "glm Resp", "nls Resp" and "lmer Resp" all extend the "lm Resp" 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

```r
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))
```
Description
A mixed-effects model is represented as a merPred 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
```r
## S3 method for class 'merMod'
anova(object, ..., refit = TRUE, model.names=NULL)
## S3 method for class 'merMod'
coef(object, ...)
## S3 method for class 'merMod'
deviance(object, REML = NULL, ...)
REMLcrit(object)
## S3 method for class 'merMod'
evAIC(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'
vcov(object, correlation = TRUE, sigm = sigma(object),
use.hessian = NULL, ...)
## S3 method for class 'merMod'
model.frame(formula, fixed.only = FALSE, ...)
## S3 method for class 'merMod'
model.matrix(object, type = c("fixed", "random", "randomListRaw"), ...)
## 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.", ...)
```
summary(object, correlation = , 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."), show.resids = TRUE, ...)
## S3 method for class 'merMod'
update(object, formula, ..., evaluate = TRUE)
## S3 method for class 'merMod'
weights(object, type = c("prior", "working"), ...)

Arguments

object an R object of class merMod, i.e., as resulting from lmer(), or glmer(), etc.
x an R object of class merMod or summary.merMod, respectively, the latter resulting from summary(<merMod>).
fit an R object of class merMod.
formula in the case of model.frame, a merMod object.
refit logical indicating if objects of class lmerMod should be refitted with ML before comparing models. The default is TRUE to prevent the common mistake of inappropriately comparing REML-fitted models with different fixed effects, whose likelihoods are not directly comparable.
model.names character vectors of model names to be used in the anova table.
scale Not currently used (see extractAIC).
k see extractAIC.
REML Logical. If TRUE, return the restricted log-likelihood rather than the log-likelihood. If NULL (the default), set REML to isREML(object) (see isREML).
fixed.only logical indicating if only the fixed effects components (terms or formula elements) are sought. If false, all components, including random ones, are returned.
random.only complement of fixed.only; indicates whether random components only are sought. (Trying to specify fixed.only and random.only at the same time will produce an error.)
correlation (logical) for vcov, indicates whether the correlation matrix as well as the variance-covariance matrix is desired; for summary.merMod, indicates whether the correlation matrix should be computed and stored along with the covariance; for print.summary.merMod, indicates whether the correlation matrix of the fixed-effects parameters should be printed. In the latter case, when NULL (the default), the correlation matrix is printed when it has been computed by summary(.), and when $p <= 20$.
use.hessian (logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects, rather estimating based on internal information about the inverse of the model matrix (see getME(.,"RX")). 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.

sigm  the residual standard error; by default sigma(object).
digits number of significant digits for printing
symbolic.cor should a symbolic encoding of the fixed-effects correlation matrix be printed? If so, the symnum function is used.
signif.stars (logical) should significance stars be used?
ranef.comp character vector of length one or two, indicating if random-effects parameters should be reported on the variance and/or standard deviation scale.
show.resids should the quantiles of the scaled residuals be printed?
formula. see update.formula.
evaluate see update.
type For weights, 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. For model.matrix, 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).

... 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 merMod 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.

coeff: 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.

summary: Computes and returns a list of summary statistics of the fitted model, the amount of output can be controlled via the print method, see also summary.

print.summary: Controls the output for the summary method.

vcov: Calculate variance-covariance matrix of the fixed effect terms, see also vcov.

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:

<table>
<thead>
<tr>
<th>Conditional</th>
<th>Unconditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative</td>
<td>Deviance(object)</td>
</tr>
<tr>
<td>Absolute</td>
<td>object@resp$aic() - 2*logLik(object)</td>
</tr>
</tbody>
</table>

This table requires two caveats:

- If the link function involves a scale parameter (e.g. Gamma) then `object@resp$aic() - 2 * getME(object)`, `devcomp$ID$imagery` 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`).

See Also

`lmer`, `glmer`, `nlmer`, `merPredD`, `lmerResp`, `glmResp`, `nlsResp`

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

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

## -> example(lmer) for an example of vcov.merMod()
```

---

### `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**

*Class "merPredD" - a Dense Predictor Reference Class*

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

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

Create Deviance Evaluation Function from Predictor and Response Module

Description

From a merMod object create an R function that takes a single argument, which is the new parameter value, and returns the deviance.

Usage

mkdevfun(rho, nAGQ = 1L, maxit = 100, verbose = 0, control = list())

Arguments

rho an environment containing pp, a prediction module, typically of class merPredD and resp, a response module, e.g., of class lmerResp.
nAGQ scalar integer - the number of adaptive Gauss-Hermite quadrature points. A value of 0 indicates that both the fixed-effects parameters and the random effects are optimized by the iteratively reweighted least squares algorithm.
maxit scalar integer, currently only for GLMMs: the maximal number of Pwrss update iterations.
verbose scalar logical: print verbose output?
control list of control parameters, a subset of those specified by lmerControl (tolPwrss and compDev for GLMMs, tolPwrss for NLMMs)

Details

The function returned by mkdevfun evaluates the deviance of the model represented by the predictor module, pp, and the response module, resp.

For lmer model objects the argument of the resulting function is the variance component parameter, theta, with lower bound. For glmer or nlmer model objects with nAGQ = 0 the argument is also theta. However, when nAGQ > 0, the argument is c(theta, beta).

Value

A function of one numeric argument.

See Also

lmer, glmer and nlmer

Examples

(dd <- lmer(Yield ~ 1|Batch, Dyestuff, devFunOnly=TRUE))

dd(0.8)

minqa::bobyqa(1, dd, 0)
Create a 'merMod' Object

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.

Create an lmerResp, glmResp or nlsResp instance

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

Description

From the result of findbars applied to a model formula and the evaluation frame fr, create the model matrix Zt, etc, associated with the random-effects terms.

Usage

mkReTrms(bars, fr, drop.unused.levels=TRUE)

Arguments

bars
a list of parsed random-effects terms
fr
a model frame in which to evaluate these terms
drop.unused.levels
(logical) drop unused factor levels? (experimental)

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
mkSimulateTemplate

Make templates suitable for guiding mixed model simulations

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

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

data("Pixel", package="nlme")
mform <- pixel ~ day + I(day^2) + (day | Dog) + (1 | Side/Dog)
(bar.f <- findbars(mform)) # list with 3 terms
mf <- model.frame(subbars(mform), data=Pixel)
rt <- mkReTrms(bar.f, mf)
names(rt)

mkSimulateTemplate

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, cumsum(nb)) (and conversely nb is diff(Gp))

lower

lower bounds on the covariance parameters

Lambdat

transpose of the sparse relative covariance factor

dlist

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
See Also

These functions are designed to be used with `simulate.merMod`.

---

**mkVarCorr**

*Make Variance and Correlation Matrices from theta*

**Description**

Make variance and correlation matrices from theta

**Usage**

```r
mkVarCorr(sc, cnms, nc, theta, nms)
```

**Arguments**

- **sc**: scale factor (residual standard deviation).
- **cnms**: component names.
- **nc**: numeric vector: number of terms in each RE component.
- **theta**: theta vector (lower-triangle of Cholesky factors).
- **nms**: component names (FIXME: nms/cnms redundant: nms=names(cnms)?)

**Value**

A matrix

**See Also**

`VarCorr`

---

**modular**

*Modular Functions for Mixed Model Fits*

**Description**

Modular functions for mixed model fits
Usage

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

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

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

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

```r
mkGlmerDevfun(fr, X, reTrms, family, nAGQ = 1L,
verbose = 0L, maxit = 100L, control = glmerControl(), ...)
```

```r
optimizeGlmer(devfun, optimizer = "bobyqa",
    restart.edge = FALSE,
    boundary.tol = formals(glmerControl)$boundary.tol,
    verbose = 0L, control = list(),
    nAGQ = 1L, stage = 1, start = NULL, ...)
```

```r
updateGlmerDevfun(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 *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).

**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 a priori 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 character - name of optimizing function(s). A character vector or list of functions: length 1 for lmer or glm, 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

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).

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 `nlminb`), pass the method argument to `optim` in the control argument.

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.

Details

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

- `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.
- `mkGlLmerDevfun` takes the output of the previous step (minus the `formula` component) and creates a deviance function
- `optimizeGlLmer` takes a deviance function and optimizes over theta (or over theta and beta, if stage is set to 2 for `optimizeGlmer`
- `updateGlmerDevfun` 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 `mkGlmerDevfun` return a function to calculate deviance (or restricted deviance) as a function of the theta (random-effect) parameters. `updateGlmerDevfun` 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 `mkGlmerDevfun` 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 `mkGlmerDevfun` 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 `optimizeGlmer` return the results of an optimization.

**Examples**

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

# 1. Parse the data and formula:
mod <- lm(formula(Reaction ~ Days + (Days|Subject), sleepstudy)
names(mod)
# 2. Create the deviance function to be optimized:
(devfun <- do.call(mkLmerDevfun, mod))
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:
mod <- glm(formula(cbind(incidence, size - incidence) ~ period + (1 | herd),
data = cbpp, family = binomial)
names(mod)
# 2. Create the deviance function for optimizing over theta:
(devfun <- do.call(mkGlmerDevfun, mod))
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$reTrms))
# 5. Optimize over theta and beta:
opt <- optimizeGlmer(devfun, stage=2)
opt[1:3]
# 6. Package up the results:
mkMerMod(environment(devfun), opt, glmod$reTrms, fr = glmod$fr)

### Same model in one line
glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
data = cbpp, family = binomial)
```
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(f, par, lower = rep.int(-Inf, n), upper = rep.int(Inf, n),
control = list())

Arguments

f  a function of a single numeric vector argument returning a numeric scalar.
par numeric vector of starting values for the parameters.
lower numeric vector of lower bounds (elements may be -Inf).
upper numeric vector of upper bounds (elements may be Inf).
control a named list of control settings. Possible settings are
   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 (de-
dfault: 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 (de-
dfault: 1e-7)
   MinfMax numeric scalar - maximum value of the minimum (default: Machine$double.xmin)
   xst numeric vector of initial step sizes to establish the simplex - all elements
must be non-zero (default: rep(0.02,length(par)))
   xt numeric vector of tolerances on the parameters (default: xst*5e-4)
   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
stop(.), but rather only call warning and return a result which might in-
spected. Defaults to FALSE, i.e., stop on non-convergence.
NelderMead-class

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

```r
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. <- NelderMead(fr, p0)
o.1 <- NelderMead(fr, p0, control=list(verbose=1))# -> some iteration output
stopifnot(identical(o.1[1:4], o.1[1:4]),
          all.equal(o.1$par, oo$par, tolerance=1e-3))# diff: 0.0003865

o.2 <- NelderMead(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  

Number of Levels of a Factor or a "merMod" Model

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"
Check and manipulate the formula for a nonlinear model, such as specified in \texttt{nlmer}.

Usage

\texttt{nlformula(mc)}

Arguments

- \texttt{mc}: matched call from the calling function, typically \texttt{nlmer()}. Should have arguments named
- \texttt{formula}: a formula of the form \texttt{resp ~ nlmod ~ meform} where \texttt{resp} is an expression for the response, \texttt{nlmod} is the nonlinear model expression and \texttt{meform} is the mixed-effects model formula. \texttt{resp} can be omitted when, e.g., optimizing a design.
- \texttt{data}: a data frame in which to evaluate the model function
- \texttt{start}: either a numeric vector containing initial estimates for the nonlinear model parameters or a list with components
  - \texttt{nlpars}: the initial estimates of the nonlinear model parameters
  - \texttt{theta}: the initial estimates of the variance component parameters

Details

The model formula for a nonlinear mixed-effects model is of the form \texttt{resp ~ nlmod ~ mixed} where \texttt{resp} is an expression (usually just a name) for the response, \texttt{nlmod} is the call to the nonlinear model function, and \texttt{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 \texttt{resp} part can be omitted.

Value

A list with components

- "respMod": a response module of class \texttt{"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: \texttt{findbars}, \texttt{mkRespMod}, \texttt{mkReTrms}, \texttt{nobars}, \texttt{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 resp ~ Nonlin(...) ~ fixed + random, where the third part is similar to the RHS formula of, e.g., lmer. Currently, the Nonlin(...) formula part must not only return a numeric vector, but also must have a “gradient” attribute, a matrix. The functions SSbiexp, SSlogis, etc, see selfStart, provide this (and more). Alternatively, you can use deriv() to automatically produce such functions or expressions.

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 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).

control a list (of correct class, resulting from lmerControl() or glmerControl() respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the *lmerControl 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.

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.

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 de-
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 a priori 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.

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 exactly the same values on subsequent calls (but the results should always
be within machine tolerance).

... other potential arguments. A method argument was used in earlier versions of
the package. Its functionality has been replaced by the nAGQ argument.

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.

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)
(nml <- nlmer(circumference ~ SSlogis(age, Asym, xmid, scal) ~ Asym|Tree,
             Orange, start = startvec))
## 2. re-run with "quick and dirty" PIRLS step
(nmla <- update(nml, nAGQ = 0L))
## 3. Fit the same model with a user-built function:
### a. Define formula
\[
\text{nform} \leftarrow -\text{Asym}/(1+\exp((\text{xmid}-\text{input})/\text{scal}))
\]
### b. Use deriv() to construct function:
\[
\text{nfun} \leftarrow \text{deriv}(\text{nform}, \text{namevec}=c("\text{Asym","xmid","scal"}), \\
\text{function.arg}=c("\text{input","Asym","xmid","scal"}))
\]
\[
\text{nm1b} \leftarrow \text{update(nm1, circumference ~ nfun(age, Asym, xmid, scal) ~ Asym | Tree)}
\]

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

---

### nloptwrap

**Wrappers for additional optimizers**

### Description
Wrappers to allow use of alternative optimizers, from NLopt library or elsewhere, for 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**: vector of lower bounds
- **upper**: vector of upper bounds
- **control**: list of control parameters
- **...**: 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. minqa::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).

Value

- `par`: estimated parameters
- `fval`: objective function value at minimum
- `feval`: number of function evaluations
- `conv`: convergence code (0 if no error)
- `message`: convergence message

Author(s)

- Gabor Grothendieck (nlminbwrap)

Examples

```r
environment(nloptwrap)$defaultControl
library(lme4)
fm1 <- lmer(Reaction~Days+(Days|Subject), sleepstudy)
## BOBYQA (default)
fm1_nloptr <- update(fm1,control=lmerControl(optimizer="nloptwrap"))
## 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
fm1_nlminb <- update(fm1,control=lmerControl(optimizer="nlminbwrap"))
if (require(optimx)) {
  fm1_nlminb2 <- update(fm1,control=lmerControl(optimizer="optimx", 
                                        optCtrl=list(method="nlminb",kkt=FALSE)))
}
```
Omit terms separated by vertical bars in a formula

Description

Remove the random-effects terms from a mixed-effects formula, thereby producing the fixed-effects formula.

Usage

nobars(term)

Arguments

term the right-hand side of a mixed-model formula

Value

the fixed-effects part of the formula

Note

This function is called recursively on individual terms in the model, which is why the argument is called term and not a name like form, indicating a formula.

See Also

formula, model.frame, model.matrix.

Other utilities: findbars, mkRespMod, mkReTrms, nlformula, subbars

Examples

nobars(Reaction ~ Days + (Days|Subject)) ## => Reaction ~ Days

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”.

**Source**

O.L. Davies and P.L. Goldsmith (eds), *Statistical Methods in Research and Production, 4th ed.*, Oliver and Boyd, (1972), section 6.5

**Examples**

```r
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(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
```
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.”

Source

O.L. Davies and P.L. Goldsmith (eds), *Statistical Methods in Research and Production, 4th ed.*, Oliver and Boyd, (1972), section 6.6

Examples

```r
(fm5 <- lmer(strength ~ (1|batch) + (1|cask), Pastes))
L <- getME(fm1, "L")
Matrix::image(L, sub = "Structure of random effects interaction in pastes model")
```
plot.lmList4

plots for lmList4 objects

Description

diagnostic and confidence-interval plots for lmList fits

Usage

## S3 method for class 'lmList4'
plot(x, form, abline, id, idLabels, grid, ...)

## S3 method for class 'lmList4.confint'
plot(x, y, order, ...)

Arguments

x an object inheriting from class lmList, representing a list of lm objects with a common model.

y ignored: for agreement with generic method

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 Trellis display. Default is resid(.) ~ fitted(.) , corresponding to a plot of the standardized residuals (using a pooled estimate for the residual standard error) 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 residuals. Observations with absolute standardized residuals greater than the 1 − value/2 quantile of the standard normal distribution are identified in the plot using idLabels. 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 id. 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 getGroups(x).
grid

an optional logical value indicating whether a grid should be added to plot. Default depends on the type of Trellis plot used: if `xyplot` defaults to `TRUE`, else defaults to `FALSE`.

order

which coefficient to order the results by

...optional arguments passed to the Trellis plot function.

Details

The `plot` method for `lmlist4` objects is copied from that for `lmlist` objects; see `plot.lmlist` for details.

Author(s)

Original versions in `nlme` package by Jose Pinheiro and Douglas Bates

Examples

```r
fm.plm <- lmlist(Reaction ~ Days | Subject, sleepstudy)
## diagnostic plot: standardized residuals vs. fitted
plot(fm.plm, id=0.05)
ci <- confint(fm.plm)
## plot CIs, ordered by slope (coefficient 2)
plot(ci, order=2, ylab="Subject")
```

Description

diagnostic plots for `merMod` fits

Usage

```r
## 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, id = NULL, idLabels = NULL, ...)
```

Arguments

- `x` a fitted `nlmer` 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 `resid(., type = "pearson") ~ fitted(.)`, 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 \( 1 - \frac{\text{value}}{2} \) quantile of the standard normal distribution are identified in the plot using idLabels. 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 id. 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 idLabels=~.obs will label the observations according to observation number.

grid an optional logical value indicating whether a grid should be added to plot. Default depends on the type of lattice plot used: if xyplot 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 had 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

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

```r
plot(fm1, Subject ~ resid(. , scaled=TRUE))
```

## observed versus fitted values by Subject

```r
plot(fm1, distance ~ fitted(.) | Subject, abline = c(0,1))
```

## residuals by age, separated by Subject

```r
plot(fm1, resid(. , scaled=TRUE) ~ age | Sex, abline = 0)
```

```r
require("lattice")
qqmath(fm1, id=0.05)
```

```r
if (require("ggplot2")) {
  ## we can create the same plots using ggplot and the fortify() function
  fmIF <- fortify(fmI)
  ggplot(fmIF, aes(.fitted,.resid)) + geom_point(colour="blue") +
     facet_grid(~Sex) + geom_hline(yintercept=0)
  ## note: Subjects are ordered by mean distance
  ggplot(fmIF, aes(Subject,.resid)) + geom_boxplot() + coord_flip()
  ggplot(fmIF, aes(.fitted,distance))+ geom_point(colour="blue") +
     facet_wrap(~Subject) +geom_abline(intercept=0,slope=1)
  ggplot(fmIF, 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
  detach("package:ggplot2")
}
```

---

### Mixed-Effects Profile Plots (Regular / Density / Pairs)

## 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

```r
## 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, ...
)
```

```r
## S3 method for class 'thpr'
densityplot(x, data, ...)
```

```r
## 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 profile(fm)
where fm is a fitted model from lmer (or its generalizations).
data unused - only for compatibility with generic.
levels the contour levels to be shown; usually derived from conf.
conf numeric vector of confidence levels to be shown as contours.
absVal logical indicating if absolute values should be plotted, often preferred for
confidence interval visualization.
scales plotting options to be passed to xyplot
which integer or character vector indicating which parameters to profile: default is all
parameters (see profile-methods 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")
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, ReForm, REForm, REform, terms = NULL,
type = c("link", "response"), allow.new.levels = FALSE,
na.action = na.pass, ...)

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 for random effects to condition on. If NULL, include all random effects; if NA or ~0, include no random effects.
ReForm, REForm, REform allowed for backward compatibility: re.form is now the preferred argument name.
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.
...
optional additional parameters. None are used at present.

Details

- If any random effects are included in re.form (see below), 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

```r
(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))
```

### Description

Methods for `profile()` of `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

```r
## 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,
        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"),
        ...)  
```

```r
## S3 method for class 'thpr'
as.data.frame(x, ...)
```

```r
## 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

- **fitted**: a fitted model, e.g., the result of `lmer(...).
- **which**: 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:
  1. random effects (theta) parameters; these are ordered as in `getME(...,"theta")`, 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.

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)".

- **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.
- **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.
profile-methods

ranef logical indicating if the sigmas of the random effects should be $\log()$ transformed as well. If false, only $\sigma$ (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.

c1 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 $\sigma$. 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, $\tanh(x) = \log((1 + x)/(1 - x))/2$).

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

```r
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
## full profiling (default which means 'all') needs
## ~2.6s (on a 2010 Macbook Pro)
system.time( tpr <- profile(fm01ML))
## ~1s, + possible warning about bobyqa convergence
(confint(tpr) -> CIpr)

stopifnot(all.equal(unname(CIpr),
  array(c(12.1985292, 38.2299848, 1486.4515,
    84.0630513, 67.6576964, 1568.54849), dim = 3:2),
  tol= 1e-07))## 1.37e-9 (64b)

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(\(\sigma\))"),
  densityplot(tpr, main="densityplot( profile(lmer(\(\sigma\) ) )"),
  densityplot(varianceProf(tpr), main=" varianceProf( profile(lmer(\(\sigma\) ) )")

splom(tpr)

if(doMore) { # not typically, for time constraint reasons
  ## Batch and residual variance only
  system.time(tpr2 <- profile(fm01ML, which=1:2, optimizer="Nelder_Mead")
  print( xplot(tpr2) )
  print( xplot(log(tpr2)) )# log(sigma) is better
  print( xplot(logProf(tpr2, ranef=FALSE)) )

  ## GLMM example
  gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
    data = cbpp, family = binomial)
  ## running ~ 10-12 seconds on a modern machine {-> "verbose" while you wait):
  print( system.time(pr4 <- profile(gm1, verbose=TRUE)) )
  print( xplot(pr4, layout=c(5,1), as.table=TRUE) )
  print( xplot(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( xplot(pr4.hr) )
}
```
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 statistics 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

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

```r
methTitle(dims)
```

```r
.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, nos)
.prt.warn (optinfo, summary = FALSE, ...)```

```r
.prt.VC (varcor, digits, comp, formatter = format, ...)
formatVC(varcor, digits = max(3, getOption("digits") - 2),
        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`. 
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 `VarCorr()`.  
`comp`  
optional ...  
`formatter`  
a function used for formatting the numbers.  
`ngrps`  
integer (vector), typically the result of `ngrps(object)`.  
`nobs`  
integer; the number of observations, e.g., the result of `nobs`.  
`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

- `logLik` which is `logLik(object)`, and
- `AICtab` a “table” of AIC, BIC, logLik, deviance and `df.residual()` values.

### Examples

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

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

(1A1 <- llikAIC(fm1))
(1AM <- llikAIC(fmM))
(1Ag <- llikAIC(gmA))

(ml <- methTitle(fm1 @ devcomp $ dims))
```
pvalues

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 mcmcsamp function, which efficiently gener-
ated 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 `mcmcsamp` 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 >=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 `KModcomp` from the `pbkrtest` package (MC,*)
- `car::Anova` and `lmerTest::anova` provide wrappers for Kenward-Roger-corrected tests using `KModcomp` from the `pbkrtest` package (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: http://www.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 = FALSE,
       drop = FALSE, whichel = names(ans), postVar=FALSE, ...)

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

## S3 method for class 'ranef.mer'
qqmath(x, data, main=TRUE, ...)

Arguments

- **object**: an object of a class of fitted models with random effects, typically a `merMod` object.
- **condVar**: an optional 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 `ranef.mer`) produced by `ranef`.
- **main**: include a main title, indicating the grouping factor, on each sub-plot?
- **transf**: transformation for random effects: for example, `exp` for plotting parameters from a (generalized) logistic regression on the odds rather than log-odds scale.
- **data**: This argument is required by the `dotplot` and `qqmath` generic methods, but is not actually used.
- **...**: some methods for these generic functions require additional arguments.

Details

If grouping factor i has k levels and j random effects per level the ith 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. The kth 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 and this j by j matrix is the kth 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

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" which is a threedimensional array with symmetric faces; each face contains the variance-covariance matrix for a particular level of the grouping factor. (The name of this attribute is a historical artifact, and may be changed to condVar at some point in the future.)

When drop is TRUE any components that would be data frames of a single column are converted to named numeric vectors.

Note

To produce a (list of) "caterpillar plots" of the random effects apply dotplot to the result of a call to ranef with condVar = TRUE; qqmath will generate a list of Q-Q plots.

Examples

```
require(lattice)
fml <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
fml2 <- lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy)
fml3 <- lmer(diameter ~ (1|plate) + (1|sample), Penicillin)
ranef(fml)
str(rr1 <- ranef(fml1, condVar = TRUE))
dotplot(rr1) ## default
    ## specify free scales in order to make Day effects more visible
    dotplot(rr1, scales = list(x = list(relation = 'free')))["Subject"]
if(FALSE) { ##-- condVar=TRUE is not yet implemented for multiple terms -- FIXME
    str(ranef(fml2, condVar = TRUE))
}
op <- options(digits = 4)
rtranef(fml3, drop = TRUE)
options(op)
## extracting random effects and conditional standard deviations
dd <- as.data.frame(rr1)
if (require(ggplot2)) {
    ggplot(dd,aes(y=grp,x=condval))+
        geom_point()+facet_wrap(~term,scales="free_x")+
        geom_errorbarh(aes(xmin=condval-2*condsd,xmax=condval+2*condsd),
                        height=0)
}
```

---

**refit**

*Refit a (merMod) Model with a Different Response*

**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, rename.response = FALSE,
       maxit = 100, ...)

Arguments

object a fitted model, usually of class merMod, 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 Details for information on NA handling). May also
be a data frame with a single numeric column, e.g. as produced by simulate(object).
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 merMod method, control.

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 re-
moved 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 "1merMod", 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.

---

repos

Generator object for the rePos (random-effects positions) class

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

**Class** "rePos"

**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**

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

---

residuals.merMod

**residuals of merMod objects**

**Description**

residuals of merMod objects

**Usage**

```r
## 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]mer (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.

---

**sigma**

*Extract Residual Standard Deviation 'Sigma'*

Description

Extract the estimated standard deviation of the errors, the “residual standard deviation” (also mis-named the “residual standard error”), from a fitted model.

Usage

```r
sigma(object, ...)  
```

Arguments

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

Details

Package lme4 provides methods for mixed-effects models of class merMod and lists of linear models, lmlist4.

Value

Typically a number, the estimated standard deviation of the errors (“residual standard deviation”) for Gaussian models, and - less interpretably - the square root of the residual deviance per degree of freedom in more general models.

Examples

```r
methods(sigma)  
```

# R 3.3.0 on, shows methods from pkgs 'stats' *and* 'lme4'
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, ReForm, REform, REform,
          newdata=NULL, newparams=NULL, family=NULL,
          allow.new.levels = FALSE, na.action = na.pass, ...)

## S3 method for class 'formula'
simulate(object, nsim = 1, seed = NULL,
          family, weights=NULL, offset=NULL, ...)

.simulateFun(object, nsim = 1, seed = NULL, use.u = FALSE,
             re.form = NA, ReForm, REform, REform,
             newdata=NULL, newparams=NULL,
             formula=NULL, family=NULL, 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.
- **ReForm, REform, REform** deprecated: re.form is now the preferred argument name.
- **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
simulate.merMod

formula a (one-sided) mixed model formula, as described for `lmer`.
family a GLM family, as in `glmer`.
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 at present.

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.
- 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 shape parameter (the rate parameter for each observation i is calculated as shape/mean(i)). For negative binomial fits, the overdispersion parameter is specified via the family, e.g. `simulate(...) family=negative.binomial(theta=1)` to simulate from a geometric distribution (negative binomial with overdispersion parameter 1).
- 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))`.

See Also

`bootMer` for “simulestimate”, i.e., where each simulation is followed by refitting the model.
Examples

```r
## test whether fitted models are consistent with the observed number of zeros in CBPP data set:
gml <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             data = cbpp, family = binomial)
gg <- simulate(gml,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(gml)[-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 for subjects in a sleep deprivation study. On day 0 the subjects had their normal amount of sleep. Starting that night they were restricted to 3 hours of sleep per night. The observations represent the average reaction time on a series of tests given each day to each subject.

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 sleep-deprived group and for the first 10 days of the study, up to the recovery period.
References


Examples

```r
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))
(fm2 <- lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy))
```

---

**subbars**

"Substitute Bars"

**Description**

Substitute the '+' function for the '|' function in a mixed-model formula, recursively (hence the argument name term). This provides a formula suitable for the current `model.frame` function.

**Usage**

```r
subbars(term)
```

**Arguments**

- **term**
  - a mixed-model formula

**Value**

the formula with all | operators replaced by +

**See Also**

`formula`, `model.frame`, `model.matrix`.

Other utilities: `findbars`, `nobars`, `mkRespMod`, `mkReTrms`, `nlformula`.

**Examples**

```r
subbars(Reaction ~ Days + (Days|Subject)) ## => Reaction ~ Days + (Days + Subject)
```
Description

This page attempts to summarize some of the common problems with fitting \texttt{gnlmer} models and how to troubleshoot them.

- failure to converge in (xxxx) evaluations. The optimizer hit its maximum limit of function evaluations. To increase this, use the \texttt{optControl} argument of \texttt{glmerControl} – for \texttt{Nelder_Mead} and \texttt{bobyqa} the relevant parameter is \texttt{maxfun}; for \texttt{optim} and \texttt{optimx}-wrapped optimizers, including \texttt{nlminbwrap}, it’s \texttt{maxit}; for \texttt{nloptwrap}, it’s \texttt{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 \texttt{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: \texttt{lapply(NA.omit(dff[, vars]), table)} (where \texttt{df} is the \texttt{data.frame} and \texttt{vars} are the column names of your predictor variables).

VarCorr

\texttt{Extract Variance and Correlation Components}

Description

This function calculates the estimated variances, standard deviations, and correlations between the random-effects terms in a mixed-effects model, of class \texttt{merMod} (linear, generalized or nonlinear). The within-group error variance and standard deviation are also calculated.
Usage

```r
## S3 method for class 'merMod'
VarCorr(x, sigma=1, ...)
```

```r
## 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(3,getOption("digits") - 2),
      comp = "Std.Dev.", formatter = format, ...)
```

Arguments

- **x** for `VarCorr`: a fitted model object, usually an object inheriting from class `merMod`. For `as.data.frame`, a `VarCorr.merMod` object returned from `VarCorr`.
- **sigma** an optional numeric value used as a multiplier for the standard deviations.
- **digits** an optional integer value specifying the number of digits
- **order** arrange data frame with variances/standard deviations first and covariances/correlations last for each random effects term ("cov.last"), or in the order of the lower triangle of the variance-covariance matrix ("lower.tri")?
- **row.names**, optional
  - Ignored: necessary for the `as.data.frame` method.
- **...** Ignored for the `as.data.frame` method; passed to other `print()` methods for the `print()` method.
- **comp** a `character` vector, specifying the components to be printed; simply passed to `formatVC()`.
- **formatter** a function for formatting the numbers; simply passed to `formatVC()`.

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 is stored as attribute "sc" (for `glmer` fits, this attribute stores the scale parameter of the model).

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
vcconv

var1 first variable
var2 second variable (NA for variance parameters)
v cov variances or covariances
sd cor standard deviations or correlations

Author(s)

This is modeled after VarCorr from package nlme, by Jose Pinheiro and Douglas Bates.

See Also

lmer, nlmer

Examples

data(Orthodont, package="nlme")
f m1 <- lmer(distance ~ age + (age|Subject), data = Orthodont)
(vc <- VarCorr(f m1)) ## 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"))
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)
sd cor2cov(m)
cov2sd cor(V)
V v_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

- **mlist2vec**: 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 "clen" gives the dimension of each matrix.
- **vec2mlist**: 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).
- **vec2STlist**: 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 sd-cor vector. Returns a vector of elements from the lower triangle of a standard-deviation-correlation matrix.

Value

(Co-)variance structure

Examples

```r
vec2mlist(1:6)
mlist2vec(vec2mlist(1:6)) # approximate inverse
```
**Description**

These are the item responses to a questionnaire on verbal aggression. These data are used throughout De Boeck and Wilson, *Explanatory Item Response Models* (Springer, 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**

http://bear.soe.berkeley.edu/EIRM/

**References**


**Examples**

```r
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)
}

  # 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)
Index

*Topic classes
  glmFamily, 35
  glmFamily-class, 35
  golden-class, 36
  lmList4-class, 52
  lmResp, 53
  lmResp-class, 54
  merMod-class, 55
  merPredD, 59
  merPredD-class, 60
  NelderMead, 70
  NelderMead-class, 71
  rePos, 100
  rePos-class, 101

*Topic datasets
  Arabidopsis, 5
  cake, 9
  cbpp, 10
  Dyestuff, 20
  grouseticks, 38
  InstEval, 40
  Pastes, 79
  Penicillin, 81
  sleepstudy, 105
  Verbagg, 111

*Topic htest
  bootMer, 6

*Topic methods
  profile-methods, 88
  ranef, 95

*Topic misc
  drop1 merMod, 17

*Topic models
  bootMer, 6
  expandDoubleVerts, 21
  factorize, 22
  findbars, 22
  fixef, 23
  glmer, 29
  glmer.nb, 32
  lmer, 42
  lmList, 51
  mkdevfun, 61
  modular, 65
  nlmer, 75
  nobars, 79
  ranef, 95
  subbars, 106
  VarCorr, 107

*Topic utilities
  expandDoubleVerts, 21
  factorize, 22
  findbars, 22
  getME, 25
  glmerLaplaceHandle, 34
  mkReTrms, 63
  nobars, 79
  prt-utilities, 92
  subbars, 106
  .makeCC (lmerControl), 45
  .prt.VC (prt-utilities), 92
  .prt.aictab (prt-utilities), 92
  .prt.call (prt-utilities), 92
  .prt.family (prt-utilities), 92
  .prt.grps (prt-utilities), 92
  .prt.methTit (prt-utilities), 92
  .prt.resids (prt-utilities), 92
  .prt.warn (prt-utilities), 92
  .simulateFun (simulate.merMod), 103

abs, 86
AIC, 93
anova, 57, 99
anova.merMod (merMod-class), 55
Arabidopsis, 5
as, 52
as.data.frame, 90
as.data.frame.thpr (profile-methods), 88

113
as.data.frame.VarCorr.merMod (VarCorr), 107
as.formula, 44
as.function.merMod (merMod-class), 55

BIC, 93
bobyqa, 47, 49, 67
boot, 7
boot.ci, 7, 13
bootMer, 4, 6, 13, 88, 90, 95, 104
cake, 9
call, 93
cat, 92
cbpp, 10
character, 12, 92, 93, 108
checkConv, 11
class, 7, 52, 93
clusterExport, 7
coeff, 52
coeff.merMod (merMod-class), 55
confint, 52, 90
confint.merMod, 7, 12, 95
confint.thpr (confint.merMod), 12
convergence, 11, 12, 14, 49
cov2sdcor (vcconv), 109
Cv_to_Sv (vcconv), 109
Cv_to_Vv (vcconv), 109
data.frame, 90
densityplot, 86
densityplot.thpr (plots.thpr), 85
deriv, 75
devcomp, 16
deviance.merMod (merMod-class), 55
df.residual, 93
df.residual.merMod (merMod-class), 55
dgCMatrix, 26
dotplot, 97
dotplot.ranef.mer (ranef), 95
dropl, 17, 44, 75
dropl.merMod, 17, 59
dummy, 19, 21, 43
Dyestuff, 20
Dyestuff2 (Dyestuff), 20
evironment, 69
envRefClass, 35, 36, 54, 60, 72, 101
expandDoubleVerts, 21

extractAIC, 56
extractAIC.merMod (merMod-class), 55
factor, 19, 73
factorize, 22
family, 30, 35, 36, 53, 57, 68, 93
family.merMod (merMod-class), 55
findbars, 22, 63, 64, 74, 79, 106
fitted, 52
fitted.merMod (merMod-class), 55
fixed.effects (fixef), 23
fixef, 23, 26–28, 52
formatVC, 108
formatVC (prt-utilities), 92
formla, 21, 23, 31, 52, 79, 106
formula.merMod (merMod-class), 55
fortify, 24
fortify.merMod, 59
function, 61, 70, 87, 93, 108
gcall, 28, 93
ggetL (getME), 25
ggetL.merMod-method (getME), 25
ggetME, 17, 25, 33, 56–59, 64
ggetOption, 76
GHRule, 28, 37
glmFormula (modular), 65
glm, 30, 31, 35, 44, 68
glmer, 3, 4, 6, 7, 25, 27, 29, 32–34, 44, 54, 56,
57, 59–61, 72, 73, 87, 103, 104
glmer.nb, 27, 31, 32
glmerControl, 27, 30, 33, 43, 75
glmerControl (lmerControl), 45
glmerLaplaceHandle, 34
glmerMod-class (merMod-class), 55
glmFamily, 35, 35, 36
glmFamily-class, 35
glmResp, 53, 55, 59, 67
glmResp (lmResp), 53
glmResp-class (lmResp-class), 54
golden (golden-class), 36
golden-class, 36
GQdk, 29, 37, 37
GQN (GQdk), 37
grouseticks, 38
grouseticks_agg (grouseticks), 38
hatvalues.merMod, 39
InstEval, 40
isGLMM (isREML), 41
isGLMM.merMod, 59
isLM (isREML), 41
isLM.merMod, 59
isNested, 41
isNLMM (isREML), 41
isNLMM.merMod, 59
isREML, 26, 41, 56
isREML.merMod, 59

lFormula (modular), 65
library, 47
list, 25–27, 33, 37, 43, 49, 52, 62, 63, 67, 71, 76, 89, 92, 93
llLikAIC (prt-utilities), 92
lm, 44, 51, 52
lme4 (lme4-package), 3
lme4-package, 3
lmer, 3, 6, 7, 25, 31, 42, 54, 56, 57, 59–61, 64, 67, 73, 75, 86–89, 103, 104, 109
lmerControl, 11, 15, 30, 43, 45, 61, 62, 67, 68, 75, 89
lmerMod, 98, 99
lmerMod-class (merMod-class), 55
lmerResp, 53, 55, 59, 61, 67
lmerResp (lmResp), 53
lmerResp-class (lmResp-class), 54
lmList, 51, 52
lmList4, 52, 102
lmList4-class, 52
lmResp, 34, 53, 53, 55
lmResp-class, 54
log, 90
log.thpr (profile-methods), 88
logical, 33, 92
logLik, 52, 58, 93
logLik.merMod (merMod-class), 55
logProf (profile-methods), 88

matrix, 13, 65, 75
mcmcsamp (pvalues), 94
merMod, 4, 12, 16, 25, 31, 39, 42, 44, 54, 56, 58, 60–62, 73, 87, 96, 100, 102, 103, 107, 108
merMod (merMod-class), 55
merMod-class, 55
merPredD, 34, 55, 59, 59, 60, 61
merPredD-class, 60
methods, 52
methTitle (prt-utilities), 92
mkDataTemplate (mkSimulateTemplate), 64
mkdevfun, 61
mkGlmerDevfun (modular), 65
mkLmerDevfun, 67
mkLmerDevfun (modular), 65
mkMerMod, 62, 68
mkParseTemplate (mkSimulateTemplate), 64
mkResPMod, 21, 23, 62, 64, 74, 79, 106
mkReTrms, 21, 23, 62, 63, 63, 67, 68, 74, 79, 106
mkSimulateTemplate, 64
mkVarCorr, 65
mList2vec (vcConv), 109
model.frame, 21, 23, 62, 79, 106
model.frame.merMod (merMod-class), 55
model.matrix, 19, 21, 23, 79, 106
model.matrix.default, 30, 67, 76
model.matrix.merMod (merMod-class), 55
model.offset, 30, 44, 51, 67, 76
modular, 4, 30, 47, 65, 68
na.fail, 104
na.pass, 87
name, 28
negative.binomial, 33
negative.binomial (glmer.nb), 32
Nelder_Mead, 47, 49, 67, 72
Nelder_Mead (NelderMead), 70
NelderMead, 70, 71, 72
NelderMead (NelderMead-class), 71
NelderMead-class, 71
new, 36, 72
ngrps, 58, 73, 93
ngrps.merMod (merMod-class), 55
nlFormula, 21, 23, 63, 64, 74, 79, 106
nlmer, 3, 25, 31, 44, 54, 57, 59–61, 72, 74, 75, 109
nlmerControl (nlmerControl), 45
nlmerMod-class (merMod-class), 55
nlminb, 47, 68, 78
nlminbwrap (nloptwrap), 77
nloptwrap, 77
nlsResp, 53, 55, 59, 74
nlsResp (lmResp), 53
nlsResp-class (lmResp-class), 54
nobars, 21, 23, 63, 64, 74, 79, 106
nobs, 93
nobs (merMod-class), 55
numeric, 92
offset, 30, 43, 51, 67, 76
optim, 47, 49, 68, 70
optimize, 33
optimizeGlmer (modular), 65
optimizeLmer (modular), 65
options, 49
pairs, 52
Pastes, 79
PBmodcomp, 7
Penicillin, 81
plot, 52
plot.lmList, 83
plot.lmList(lmList), 51
plot.lmList4, 82
plot.merMod, 59, 83
plots.thpr, 85
pnbinom, 33
predict, 52, 87
predict.merMod, 57, 59, 87
print, 52, 86, 92, 108
print.merMod (merMod-class), 55
print.summary.merMod (merMod-class), 55
print.VarCorr.merMod (VarCorr), 107
profile, 12, 13, 86, 88, 90
profile(.), 13
profile-methods, 88
profile.merMod, 13, 59, 95
profile.merMod (profile-methods), 88
prt-utilities, 92
pvalues, 4, 7, 94
qamath, 97
qamath.merMod (plot.merMod), 83
qamath.ranef.mer, 84
qamath.ranef.mer (ranef), 95
qqnorm, 52
ranef, 27, 28, 52, 95
ranef.merMod, 59
rankMatrix, 48
ReferenceClasses, 4, 69
refit, 7, 33, 97, 100
refit.merMod, 59
refitML, 98, 99
refitML.merMod, 59
reformulate, 44
REMLcrit (merMod-class), 55
rePos, 100, 100
rePos-class, 101
require, 47
residuals, 52, 93
residuals.lm, 102
residuals.glm, 102
residuals.glmResp (residuals.merMod), 101
residuals.lm, 102
residuals.lme, 102
residuals.lmResp (residuals.merMod), 101
residuals.merMod, 59, 101
rnorm, 7, 64
scale, 15
sdcor2cov (vcconv), 109
selfStart, 75
set.seed, 6, 103
show, 52, 92
show.lmList4-method (lmList4-class), 52
show.merMod-method (merMod-class), 55
show.merMod (merMod-class), 55
show.summary.merMod (merMod-class), 55
sigma, 43, 52, 57, 102
sigma.merMod, 59
simulate, 98
simulate.formula (simulate.merMod), 103
simulate.merMod, 6, 7, 59, 65, 103
sleepstudy, 105
sploM, 86
sploM.thpr (plots.thpr), 85
SSbiexp, 75
SSlogis, 75
stop, 70
subbars, 21, 23, 63, 64, 74, 79, 106
summary, 52, 58, 92
summary.merMod, 59
summary.merMod (merMod-class), 55
summary.summary.merMod (merMod-class), 55
Sv_to_Cv (vcconv), 109
symnum, 57

terms, 87
terms.merMod (merMod-class), 55
theta.ml, 32, 33
troubleshooting, 107
update, 44, 52, 57, 58, 75, 98
update.formula, 57
update.merMod, 98, 100
update.merMod(merMod-class), 55
updateGlmerDevfun(modular), 65

VarCorr, 65, 92, 93, 107, 109
varianceProf, 89
varianceProf (profile-methods), 88
vcov, 109
vcov, 27, 28, 58
vcov.merMod (merMod-class), 55
vcov.summary.merMod (merMod-class), 55
vec2mlist (vcconv), 109
vec2STlist (vcconv), 109
VerbAgg, 111
Vv_to_Cv (vcconv), 109

warning, 70
weights.merMod (merMod-class), 55

xyplot, 86, 90
xyplot.thpr (plots.thpr), 85