Package: lme4 (via r-universe)

August 28, 2024

Version 1.1-35.6

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

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

Depends R (>= 3.6.0), Matrix, methods, stats

LinkingTo Rcpp (>= 0.10.5), RcppEigen (>= 0.3.3.9.4), Matrix (>= 1.2-3)

Imports graphics, grid, splines, utils, parallel, MASS, lattice, boot, nlme (>= 3.1-123), minqa (>= 1.1.15), nloptr (>= 1.0.4), reformulas (>= 0.3.0)

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

VignetteBuilder knitr

LazyData yes

License GPL (>=2)

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

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

Encoding UTF-8

Repository https://lme4.r-universe.dev

RemoteUrl https://github.com/lme4/lme4

RemoteRef HEAD

RemoteSha bfd7a44d0a718fff090412871504858559a0829f

Contents

lme4-package *Linear, generalized linear, and nonlinear mixed models*

Description

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

Differences between nlme and lme4

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

- **lme4** uses modern, efficient linear algebra methods as implemented in the Eigen package, and uses reference classes to avoid undue copying of large objects; it is therefore likely to be faster and more memory-efficient than **nlme**.
- **lme4** includes generalized linear mixed model (GLMM) capabilities, via the glmer function.
- **lme4** does *not* currently implement **nlme**'s features for modeling heteroscedasticity and correlation of residuals.
- **lme4** does not currently offer the same flexibility as **nlme** for composing complex variance-covariance structures, but it does implement crossed random effects in a way that is both easier for the user and much faster.
- **lme4** offers built-in facilities for likelihood profiling and parametric bootstrapping.
- **lme4** is designed to be more modular than **nlme**, making it easier for downstream package developers and end-users to re-use its components for extensions of the basic mixed model framework. It also allows more flexibility for specifying different functions for optimizing over the random-effects variance-covariance parameters.
- **lme4** is not (yet) as well-documented as **nlme**.

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

- [gn]lmer now produces objects of class merMod rather than class mer as before
- the new version uses a combination of S3 and reference classes (see ReferenceClasses, merPredD-class, and lmResp-class) as well as S4 classes; partly for this reason it is more interoperable with **nlme**
- The internal structure of [gn]lmer is now more modular, allowing finer control of the different steps of argument checking; construction of design matrices and data structures; parameter estimation; and construction of the final merMod object (see modular)
- profiling and parametric bootstrapping are new in the current version
- the new version of **lme4** does *not* provide an 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**.)

`allFit` *Refit a fitted model with all available optimizers*

Description

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

Usage

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

Arguments

`object`	a fitted model
`meth.tab`	a matrix (or data.frame) with columns **method** the name of a specific optimization method to pass to the optimizer (leave blank for built-in optimizers) **optimizer** the `optimizer` function to use
`data`	data to be included with result (for later debugging etc.)
`verbose`	logical: report progress in detail?
`show.meth.tab`	logical: return table of methods?

maxfun	passed as part of optCtrl to set the maximum number of function evaluations: this is *automatically* converted to the correct specification (e.g. maxfun, maxfeval, maxit, etc.) for each optimizer
parallel	The type of parallel operation to be used (if any). If missing, the default is taken from the option "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. Use options(allFit.ncpus=X) to set the default value to X for the duration of an R session.
cl	An optional **parallel** or **snow** cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the boot call.
catch.errs	(logical) Wrap model fits in tryCatch clause to skip over errors? (catch.errs=FALSE is probably only useful for debugging)

Details

- Needs packages optimx, and dfoptim to use all optimizers
- If you are using parallel="snow" (e.g. when running in parallel on Windows), you will need to set up a cluster yourself and run clusterEvalQ(cl,library("lme4")) before calling allFit to make sure that the lme4 package is loaded on all of the workers
- Control arguments in control$optCtrl that are unused by a particular optimizer will be *silently* ignored (in particular, the maxfun specification is only respected by bobyqa, Nelder_Mead, and nmkbw)
- Because allFit works by calling update, it may be fragile if the original model call contains references to variables, especially if they were originally defined in other environments or no longer exist when allFit is called.

Value

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

See Also

slice,slice2D from the **bbmle** package

Examples

```
if (interactive()) {
library(lme4)
  gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
               data = cbpp, family = binomial)
  ## show available methods
  allFit(show.meth.tab=TRUE)
  gm_all <- allFit(gm1)
  ss <- summary(gm_all)
  ss$which.OK            ## logical vector: which optimizers worked?
  ## the other components only contain values for the optimizers that worked
```

```
  ss$llik               ## vector of log-likelihoods
  ss$fixef              ## table of fixed effects
  ss$sdcor              ## table of random effect SDs and correlations
  ss$theta              ## table of random effects parameters, Cholesky scale
}
## Not run:
  ## Parallel examples for Windows
  nc <- detectCores()-1
  optCls <- makeCluster(nc, type = "SOCK")
  clusterEvalQ(optCls,library("lme4"))
  ### not necessary here because using a built-in
  ## data set, but in general you should clusterExport() your data
  clusterExport(optCls, "cbpp")
  system.time(af1 <- allFit(m0, parallel = 'snow',
                            ncpus = nc, cl=optCls))
  stopCluster(optCls)

## End(Not run)
```

`Arabidopsis` *Arabidopsis clipping/fertilization data*

Description

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

Usage

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

Format

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

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

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

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

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

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

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

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

`total.fruits` total fruit set per plant (integer)

Source

From Josh Banta

References

Joshua A. Banta, Martin H. H Stevens, and Massimo Pigliucci (2010) A comprehensive test of the 'limiting resources' framework applied to plant tolerance to apical meristem damage. *Oikos* **119**(2), 359–369; doi:10.1111/j.16000706.2009.17726.x

Examples

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

---

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

Description

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

Usage

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

Arguments

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.

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

bootFail number of failures (errors)

boot.fail.msgs error messages

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

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

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

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

Value

an object of S3 class "boot", compatible with **boot** package's boot() result. (See Details for information on how to retrieve information about errors during bootstrapping.)

Note

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

References

Davison, A.C. and Hinkley, D.V. (1997) *Bootstrap Methods and Their Application*. Cambridge University Press.

Morris, J. S. (2002). The BLUPs Are Not 'best' When It Comes to Bootstrapping. *Statistics & Probability Letters* **56**(4): 425–430. doi:10.1016/S0167-7152(02)00041-X.

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

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

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

## to "look" at it
if (requireNamespace("boot")) {
    boo01
    ## note large estimated bias for sig01
```

```
## (~30% low, decreases _slightly_ for nsim = 1000)

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

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

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

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

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

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


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

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

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

---

cake *Breakage Angle of Chocolate Cakes*

Description

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

Format

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

`replicate` a factor with levels 1 to 15

`recipe` a factor with levels A, B and C

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

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

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

Details

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

Source

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

References

Cook, F. E. (1938) *Chocolate cake, I. Optimum baking temperature*. Master's Thesis, Iowa State College.

Cochran, W. G., and Cox, G. M. (1957) *Experimental designs*, 2nd Ed. New York, John Wiley & Sons.

Lee, Y., Nelder, J. A., and Pawitan, Y. (2006) *Generalized linear models with random effects. Unified analysis via H-likelihood.* Boca Raton, Chapman and Hall/CRC.

Examples

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

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

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

---

cbpp *Contagious bovine pleuropneumonia*

Description

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

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

Lesnoff, M., Laval, G., Bonnet, P., Abdicho, S., Workalemahu, A., Kifle, D., Peyraud, A., Lancelot, R., Thiaucourt, F. (2004) Within-herd spread of contagious bovine pleuropneumonia in Ethiopian highlands. *Preventive Veterinary Medicine* **64**, 27–40.

Examples

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

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

---

checkConv *Extended Convergence Checking*

Description

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

Usage

```
checkConv(derivs, coefs, ctrl, lbound, 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 [ng]lmer Fit*

Description

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

Usage

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

Arguments

object	a fitted [ng]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.
FUN	bootstrap 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

```
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

```
if (interactive() || lme4_testlevel() >= 3) {
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
fm1W <- confint(fm1, method="Wald")# very fast, but not useful for "sigmas" = var-cov pars
fm1W
(fm2 <- lmer(Reaction ~ Days + (Days || Subject), sleepstudy))
(CI2 <- confint(fm2, maxpts = 8)) # method = "profile"; 8: to be much faster

if (lme4_testlevel() >= 3) {
  system.time(fm1P <- confint(fm1, method="profile", ## <- default
                              oldNames = FALSE))
  ## --> ~ 2.2 seconds (2022)
  set.seed(123) # (reproducibility when using bootstrap)
  system.time(fm1B <- confint(fm1, method="boot", oldNames=FALSE,
                              .progress="txt", PBargs= list(style=3)))
  ## --> ~ 6.2 seconds (2022) and warning, messages
} else {
    load(system.file("testdata","confint_ex.rda",package="lme4"))
}
fm1P
fm1B
} ## if interactive && testlevel>=3
```

---

convergence *Assessing Convergence for Fitted Models*

Description

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

- double-check the model specification and the data
- adjust stopping (convergence) tolerances for the nonlinear optimizer, using the optCtrl argument to [g]lmerControl (see "Convergence controls" below)
- center and scale continuous predictor variables (e.g. with scale)

- double-check the Hessian calculation with the more expensive Richardson extrapolation method (see examples)
- restart the fit from the reported optimum, or from a point perturbed slightly away from the reported optimum
- use `allFit` to try the fit with all available optimizers (e.g. several different implementations of BOBYQA and Nelder-Mead, L-BFGS-B from `optim`, `nlminb`, ...). While this will of course be slow for large fits, we consider it the gold standard; if all optimizers converge to values that are practically equivalent, then we would consider the convergence warnings to be false positives.

Details

Convergence controls:

- the controls for the `nloptwrap` optimizer (the default for `lmer`) are

  ftol_abs (default 1e-6) stop on small change in deviance

  ftol_rel (default 0) stop on small relative change in deviance

  xtol_abs (default 1e-6) stop on small change of parameter values

  xtol_rel (default 0) stop on small relative change of parameter values

  maxeval (default 1000) maximum number of function evaluations

  Changing `ftol_abs` and `xtol_abs` to stricter values (e.g. 1e-8) is a good first step for resolving convergence problems, at the cost of slowing down model fits.
- the controls for `minqa::bobyqa` (default for `glmer` first-stage optimization) are

  rhobeg (default 2e-3) initial radius of the trust region

  rhoend (default 2e-7) final radius of the trust region

  maxfun (default 10000) maximum number of function evaluations

  `rhoend`, which describes the scale of parameter uncertainty on convergence, is approximately analogous to `xtol_abs`.
- the controls for `Nelder_Mead` (default for `glmer` second-stage optimization) are

  FtolAbs (default 1e-5) stop on small change in deviance

  FtolRel (default 1e-15) stop on small relative change in deviance

  XtolRel (default 1e-7) stop on small change of parameter values

  maxfun (default 10000) maximum number of function evaluations

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

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

- the gradient and Hessian are the basic ingredients of KKT-style testing, but (at least for now) `lme4` estimates them by finite-difference approximations which are sometimes unreliable.

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

See Also

`lmerControl`, `isSingular`

Examples

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

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

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

## 3. recompute gradient and Hessian with Richardson extrapolation
devfun <- update(fm1, devFunOnly=TRUE)
if (isLMM(fm1)) {
    pars <- getME(fm1,"theta")
} else {
    ## GLMM: requires both random and fixed parameters
    pars <- getME(fm1, c("theta","fixef"))
}
if (require("numDeriv")) {
```

```
    cat("hess:\n"); print(hess <- hessian(devfun, unlist(pars)))
    cat("grad:\n"); print(grad <- grad(devfun, unlist(pars)))
    cat("scaled gradient:\n")
    print(scgrad <- solve(chol(hess), grad))
}
## compare with internal calculations:
fm1@optinfo$derivs

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

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

## 5. try all available optimizers

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

}
```

---

devcomp *Extract the deviance component list*

Description

Return the deviance component list

Usage

```
devcomp(x)
```

Arguments

x	a fitted model of class merMod

Details

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

Value

a list with components

dims	a named integer vector of various dimensions
cmp	a named numeric vector of components of the deviance

Note

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

---

devfun2 *Deviance Function in Terms of Standard Deviations/Correlations*

Description

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

Usage

```
devfun2(fm, useSc = if(isLMM(fm)) TRUE else NA,
        transfuns = list(from.chol = Cv_to_Sv,
                         to.chol = Sv_to_Cv,
                           to.sd = identity), ...)
```

Arguments

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

Value

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

optimum a named vector giving the parameter values at the optimum

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

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

stderr standard errors of fixed effect parameters

Note

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

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

Examples

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

---

drop1.merMod *Drop all possible single fixed-effect terms from a mixed effect model*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Examples

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

```
    ## environment of this function (which does not contain 'data')
    environment(f) <- parent.frame()
    f
}
createModel <- function(data) {
    mf.final <- createFormula("Reaction", "Days", "(Days|Subject)")
    lmer(mf.final, data=data)
}
drop1(createModel(data=sleepstudy))
```

---

dummy *Dummy variables (experimental)*

Description

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

Usage

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

Arguments

`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

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

Dyestuff *Yield of dyestuff by batch*

Description

The Dyestuff data frame provides the yield of dyestuff (Naphthalene Black 12B) from 5 different preparations from each of 6 different batchs 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 undoubtably 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

G.E.P. Box and G.C. Tiao, *Bayesian Inference in Statistical Analysis*, Addison-Wesley, (1973), section 5.1.2

Examples

```
require(lattice)
str(Dyestuff)
dotplot(reorder(Batch, Yield) ~ Yield, Dyestuff,
        ylab = "Batch", jitter.y = TRUE, aspect = 0.3,
        type = c("p", "a"))
dotplot(reorder(Batch, Yield) ~ Yield, Dyestuff2,
        ylab = "Batch", jitter.y = TRUE, aspect = 0.3,
```

```
        type = c("p", "a"))
(fm1 <- lmer(Yield ~ 1|Batch, Dyestuff))
(fm2 <- lmer(Yield ~ 1|Batch, Dyestuff2))
```

`expandDoubleVerts` *Expand terms with '||' notation into separate '|' terms*

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

```
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

```
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),
        family=gaussian)[[1]]
```

```
m1 <- lmer(y~f+(f|g),data=dd)
VarCorr(m1)
m2 <- lmer(y~f+(1|g) + (0+dummy(f,"b")|g) + (0+dummy(f,"c")|g),
             data=dd)
VarCorr(m2)
```

`factorize` *Attempt to convert grouping variables to factors*

Description

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

Usage

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

Arguments

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

Value

a copy of the data frame with factors converted

`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

```
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

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

---

fortify *add information to data based on a fitted model*

Description

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

Usage

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

Arguments

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

Details

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

Examples

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

---

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

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

## S3 method for class 'merMod'
getME(object,
      name = c("X", "Z", "Zt", "Ztlist", "mmList", "y", "mu", "u", "b",
               "Gp", "Tp", "L", "Lambda", "Lambdat", "Lind", "Tlist",
               "A", "RX", "RZX", "sigma", "flist",
               "fixef", "beta", "theta", "ST", "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(Matrix::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 Λ of the random effects.

"Lambdat": transpose Λ' of Λ above.

"Lind": index vector for inserting elements of θ into the nonzeros of Λ.

"Tlist": vector of template matrices from which the blocks of Λ are generated.

"A": Scaled sparse model matrix (class "dgCMatrix") for the unit, orthogonal random effects, U, equal to `getME(.,"Zt") %*% getME(.,"Lambdat")`

"RX": Cholesky factor for the fixed-effects parameters

"RZX": cross-term in the full Cholesky factor

"sigma": residual standard error; note that `sigma(object)` is preferred.

"flist": a list of the grouping variables (factors) involved in the random effect terms

"fixef": fixed-effects parameter estimates

"beta": fixed-effects parameter estimates (identical to the result of `fixef`, but without names)

"theta": 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, i.e., the same as `dims@nth` below.

"cnms": the "component names", a `list`.

"REML": 0 indicates the model was fitted by maximum likelihood, any other positive integer indicates fitting by restricted maximum likelihood

"is_REML": same as the result of isREML(.)

"devcomp": a list consisting of a named numeric vector, cmp, and a named integer vector, dims, describing the fitted model. The elements of cmp are:

ldL2 twice the log determinant of L

ldRX2 twice the log determinant of RX

wrss weighted residual sum of squares

ussq squared length of u

pwrss penalized weighted residual sum of squares, "wrss + ussq"

drsum sum of residual deviance (GLMMs only)

REML REML criterion at optimum (LMMs fit by REML only)

dev deviance criterion at optimum (models fit by ML only)

sigmaML ML estimate of residual standard deviation

sigmaREML REML estimate of residual standard deviation

tolPwrss tolerance for declaring convergence in the penalized iteratively weighted residual sum-of-squares (GLMMs only)

The elements of dims are:

N number of rows of X

n length of y

p number of columns of X

nmp n-p

nth length of theta

q number of columns of Z

nAGQ see glmer

compDev see glmerControl

useSc TRUE if model has a scale parameter

reTrms number of random effects terms

REML 0 indicates the model was fitted by maximum likelihood, any other positive integer indicates fitting by restricted maximum likelihood

GLMM TRUE if a GLMM

NLMM TRUE if an NLMM

"offset": model offset

"lower": lower bounds on random-effects model parameters (i.e, "theta" parameters). In order to constrain random effects covariance matrices to be semi-positive-definite, this vector is equal to 0 for elements of the theta vector corresponding to diagonal elements of the Cholesky factor, -Inf otherwise. (getME(.,"lower")==0 can be used as a test to identify diagonal elements, as in isSingular.)

"devfun": deviance function (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.

Value

Unspecified, as very much depending on the name.

See Also

getCall(). More standard methods for "merMod" objects, such as ranef, fixef, vcov, etc.: see methods(class="merMod")

Examples

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

(fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy))
Z <- getME(fm1, "Z")
stopifnot(is(Z, "CsparseMatrix"),
          c(180,36) == dim(Z),
   all.equal(fixef(fm1), b1 <- getME(fm1, "beta"),
      check.attributes=FALSE, tolerance = 0))

## A way to get *all* getME()s :
## internal consistency check ensuring that all work:
parts <- getME(fm1, "ALL")
str(parts, max=2)
stopifnot(identical(Z,  parts $ Z),
          identical(b1, parts $ beta))
```

GHrule *Univariate Gauss-Hermite quadrature rule*

Description

Create a univariate Gauss-Hermite quadrature rule.

Usage

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

Arguments

ord	scalar integer between 1 and 100 - the order, or number of nodes and weights, in the rule. When the function being multiplied by the standard normal density is a polynomial of order $2k - 1$ the rule of order k integrates the product exactly.
asMatrix	logical scalar - should the result be returned as a matrix. If FALSE a data frame is returned. Defaults to TRUE.

Details

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

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

Value

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

References

Qing Liu and Donald A. Pierce (1994). A Note on Gauss-Hermite Quadrature. *Biometrika* **81**(3), 624–629. doi:10.2307/2337136

See Also

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

Examples

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

## second, fourth, sixth, eighth and tenth central moments of the
## standard Gaussian N(0,1) density:
ps <- seq(2, 10, by = 2)
cbind(p = ps, "E[X^p]" = with(r5,  sapply(ps, function(p) sum(w * z^p)))) # p=10 is wrong for 5-rule
p <- 1:15
GQ12 <- with(r12, sapply(p, function(p) sum(w * z^p)))
cbind(p = p, "E[X^p]" = zapsmall(GQ12))
## standard R numerical integration can do it too:
intL <- lapply(p, function(p) integrate(function(x) x^p * dnorm(x),
                                        -Inf, Inf, rel.tol=1e-11))
integR <- sapply(intL, `[[`, "value")
cbind(p, "E[X^p]" = integR)# no zapsmall() needed here
all.equal(GQ12, integR, tol=0)# => shows small difference
stopifnot(all.equal(GQ12, integR, tol = 1e-10))
(xactMom <- cumprod(seq(1,13, by=2)))
stopifnot(all.equal(xactMom, GQ12[2*(1:7)], tol=1e-14))
## mean relative errors :
mean(abs(GQ12  [2*(1:7)] / xactMom - 1)) # 3.17e-16
mean(abs(integR[2*(1:7)] / xactMom - 1)) # 9.52e-17 {even better}
```

glmer *Fitting Generalized Linear Mixed-Effects Models*

Description

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

Usage

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

Arguments

formula	a two-sided linear formula object describing both the fixed-effects and random-effects part of the model, with the response on the left of a ~ 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).
family	a GLM family, see glm and family.
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, 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 penalized iteratively reweighted least squares (PIRLS) steps.

`nAGQ` integer scalar - the number of points per axis for evaluating the adaptive Gauss-Hermite approximation to the log-likelihood. Defaults to 1, corresponding to the Laplace approximation. Values greater than 1 produce greater accuracy in the evaluation of the log-likelihood at the expense of speed. A value of zero uses a faster but less exact form of parameter estimation for GLMMs by optimizing the random effects and the fixed-effects coefficients in the penalized iteratively reweighted least squares step. (See Details.)

`subset` an optional expression indicating the subset of the rows of `data` that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

`weights` an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.

`na.action` a function that indicates what should happen when the data contain `NA`s. 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`.

`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).

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"))

Note

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

See Also

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

glmer.nb to fit negative binomial GLMMs.

Examples

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

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

## glmer and glm log-likelihoods are consistent
gm1Devfun <- update(gm1,devFunOnly=TRUE)
gm0 <- glm(cbind(incidence, size - incidence) ~ period,
           family = binomial, data = cbpp)
## evaluate GLMM deviance at RE variance=theta=0, beta=(GLM coeffs)
gm1Dev0 <- gm1Devfun(c(0,coef(gm0)))
## compare
stopifnot(all.equal(gm1Dev0,c(-2*logLik(gm0))))
## the toenail oncholysis data from Backer et al 1998
```

```
## these data are notoriously difficult to fit
## Not run:
if (require("HSAUR3")) {
    gm2 <- glmer(outcome~treatment*visit+(1|patientID),
                 data=toenail,
                 family=binomial,nAGQ=20)
}

## End(Not run)
```

`glmer.nb` *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

```
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 the output of `glmerControl()`, used in `refit(*, control = control.nb)` during the optimization (`control`, if included in `...`, will be used in the initial-stage `glmer(...,family=poisson)` fit, and passed on to the later optimization stages as well)
`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

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

## End(Not run)
```

`glmerLaplaceHandle` *Handle for* `glmerLaplace`

Description

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

Usage

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

Arguments

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

Value

Value of the objective function

---

glmFamily *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 distibution 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)
```

GQdk *Sparse Gaussian / Gauss-Hermite Quadrature grid*

Description

Generate the sparse multidimensional Gaussian quadrature grids.

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

Usage

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

Arguments

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

Value

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

GQN is a list of lists, containing the non-redundant quadrature nodes and weights for integration of a scalar function of a d-dimensional argument with respect to the density function of the d-dimensional Gaussian density function.
The outer list is indexed by the dimension, d, in the range of 1 to 20. The inner list is indexed by k, the order of the quadrature.

Note

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

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

Examples

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

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

`grouseticks` *Data on red grouse ticks from Elston et al. 2001*

Description

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

Usage

```
data(grouseticks)
```

Format

`INDEX` (factor) chick number (observation level)

`TICKS` number of ticks sampled

`BROOD` (factor) brood number

`HEIGHT` height above sea level (meters)

`YEAR` year (-1900)

`LOCATION` (factor) geographic location code

`cHEIGHT` centered height, derived from `HEIGHT`

`meanTICKS` mean number of ticks by brood

`varTICKS` variance of number of ticks by brood

Details

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

Source

Robert Moss, via David Elston

References

Elston, D. A., R. Moss, T. Boulinier, C. Arrowsmith, and X. Lambin. 2001. "Analysis of Aggregation, a Worked Example: Numbers of Ticks on Red Grouse Chicks." Parasitology 122 (05): 563-569. doi:10.1017/S0031182001007740

Examples

```
if (interactive()) {
data(grouseticks)
## Figure 1a from Elston et al
par(las=1,bty="l")
tvec <- c(0,1,2,5,20,40,80)
pvec <- c(4,1,3)
with(grouseticks_agg,plot(1+meanTICKS~HEIGHT,
                  pch=pvec[factor(YEAR)],
                  log="y",axes=FALSE,
                  xlab="Altitude (m)",
                  ylab="Brood mean ticks"))
axis(side=1)
axis(side=2,at=tvec+1,label=tvec)
box()
abline(v=405,lty=2)
## Figure 1b
with(grouseticks_agg,plot(varTICKS~meanTICKS,
                  pch=4,
                  xlab="Brood mean ticks",
                  ylab="Within-brood variance"))
curve(1*x,from=0,to=70,add=TRUE)
## Model fitting
form <- TICKS~YEAR+HEIGHT+(1|BROOD)+(1|INDEX)+(1|LOCATION)
(full_mod1  <- glmer(form, family="poisson",data=grouseticks))
}
```

`hatvalues.merMod` *Diagonal elements of the hat matrix*

Description

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

Usage

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

Arguments

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

Value

The diagonal elements of the hat matrix.

Examples

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

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

Description

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

Usage

```
## S3 method for class 'merMod'
influence(model, groups, data, maxfun = 1000,
          do.coef = TRUE, ncores = getOption("mc.cores",1), start, ...)
## S3 method for class 'influence.merMod'
cooks.distance(model, ...)
## S3 method for class 'influence.merMod'
dfbeta(model, which = c("fixed", "var.cov"), ...)
## S3 method for class 'influence.merMod'
dfbetas(model, ...)
```

Arguments

`model`	in the case of `influence.merMod`, a model of class `"merMod"`; in the case of `cooks.distance`, `dfbeta`, or `dfbetas`, an object returned by `influence.merMod`
`groups`	a character vector containing the name of a grouping factor or names of grouping factors; if more than one name is supplied, then groups are defined by all combinations of levels of the grouping factors that appear in the data. If omitted, then each individual row of the data matrix is treated as a "group" to be deleted in turn.
`data`	an optional data frame with the data to which `model` was fit; `influence.merMod` can usually retrieve the data used to fit the model, unless it can't be found in the current environment, so it's usually unnecessary to supply this argument.
`maxfun`	The maximum number of function evaluations (for `influence.merMod`) to perform after deleting each group; the defaults are large enough so that the iterations will typically continue to convergence. Setting to `maxfun=20` for an `lmer` model

	or 100 for a glmer model will typically produce a faster reasonable approximation. An even smaller value can be used if interest is only in influence on the fixed effects.
which	if "fixed.effects" (the default), return influence on the fixed effects; if "var.cov", return influence on the variance-covariance components.
do.coef	if FALSE, skip potentially time-consuming computations, returning just a list containing hat values.
ncores	number of computational cores to use if run in parallel; directly passed to makeCluster() from R's **parallel** package.
start	starting value for new fits (set to optimal values from original fit by default)
...	ignored.

Details

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

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

Value

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

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

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

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

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

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

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

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

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

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

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

For plotting "influence.merMod" objects, see infIndexPlot.

Author(s)

J. Fox <jfox@mcmaster.ca>

References

Fox, J. and Weisberg, S. (2019) *An R Companion to Applied Regression*, Third Edition, Sage.

See Also

infIndexPlot, influence.measures

Examples

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

InstEval *University Lecture/Instructor Evaluations by Students at ETH*

Description

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

Format

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

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

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

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

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

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

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

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

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

Details

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

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

Examples

```
str(InstEval)

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

---

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, ...)
```

Arguments

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

Details

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

Value

a logical value

See Also

getME

Examples

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

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

---

isSingular *Test Fitted Model for (Near) Singularity*

Description

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

Usage

```
isSingular(x, tol = 1e-4)
```

Arguments

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

Details

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

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

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

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

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

Value

a logical value

References

Dale J. Barr, Roger Levy, Christoph Scheepers, and Harry J. Tily (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal; *Journal of Memory and Language* **68**(3), 255–278.

Douglas Bates, Reinhold Kliegl, Shravan Vasishth, and Harald Baayen (2015). *Parsimonious Mixed Models*; preprint (https://arxiv.org/abs/1506.04967).

Yeojin Chung, Sophia Rabe-Hesketh, Vincent Dorie, Andrew Gelman, and Jingchen Liu (2013). A nondegenerate penalized likelihood estimator for variance parameters in multilevel models; *Psychometrika* **78**, 685–709; doi:10.1007/S1133601393282.

Andrew Gelman and Jennifer Hill (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Hannes Matuschek, Reinhold Kliegl, Shravan Vasishth, Harald Baayen, and Douglas Bates (2017). Balancing type I error and power in linear mixed models. *Journal of Memory and Language* **94**, 305–315.

Richard McElreath (2015) *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. Chapman and Hall/CRC.

See Also

getME, rePCA

---

lme4_testlevel *Detect testing level for lme4 examples and tests*

Description

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

Usage

```
lme4_testlevel()
```

Value

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

---

lmer *Fit Linear Mixed-Effects Models*

Description

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

Usage

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

Arguments

formula	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 penalized iteratively reweighted least squares (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 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).

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), [gn]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 simple (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"))

Note

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

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

See Also

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

Examples

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

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

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

```
print(sm2, digits=3, corr=FALSE)       # the print.summary.merMod() method

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

`lmerControl` *Control of Mixed Model Fitting*

Description

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

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

Usage

```
lmerControl(optimizer = "nloptwrap",

    restart_edge = TRUE,
    boundary.tol = 1e-5,
    calc.derivs = TRUE,
    use.last.params = FALSE,
    sparseX = FALSE,
    standardize.X = 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 = "message", tol = formals(isSingular)$tol),
    check.conv.hess     = .makeCC(action = "warning", tol = 1e-6),
    ## optimizer args
    optCtrl = list(),
    mod.type = "lmer"
)

glmerControl(optimizer = c("bobyqa", "Nelder_Mead"),
    restart_edge = FALSE,
    boundary.tol = 1e-5,
    calc.derivs = TRUE,
    use.last.params = FALSE,
    sparseX = FALSE,
    standardize.X = 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 = "message", tol = formals(isSingular)$tol),
    check.conv.hess     = .makeCC(action = "warning", tol = 1e-6),
    ## optimizer args
    optCtrl = list(),
    mod.type = "glmer",
    tolPwrss = 1e-7,
    compDev = TRUE,
    nAGQ0initStep = TRUE,
    check.response.not.const = "stop"
 )

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

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

Arguments

optimizer	character - name of optimizing function(s). A `character` vector or list of functions: length 1 for `lmer` or `glmer`, possibly length 2 for `glmer`). Built-in optimizers are `"Nelder_Mead"`, `"bobyqa"` (from the **minqa** package), `"nlminbwrap"`

(using base R's nlminb) and the default for lmerControl(), "nloptwrap". Any minimizing function that allows box constraints can be used provided that it

(1) takes input parameters fn (function to be optimized), par (starting parameter values), lower and upper (parameter bounds) and control (control parameters, passed through from the control argument) and

(2) returns a list with (at least) elements par (best-fit parameters), fval (best-fit function value), conv (convergence code, equal to zero for successful convergence) and (optionally) message (informational message, or explanation of convergence failure).

Special provisions are made for bobyqa, Nelder_Mead, and optimizers wrapped in the **optimx** package; to use the **optimx** optimizers (including L-BFGS-B from base optim and nlminb), pass the method argument to optim in the optCtrl argument (you may need to load the **optimx** package manually using library(optimx)).

For glmer, if length(optimizer)==2, the first element will be used for the preliminary (random effects parameters only) optimization, while the second will be used for the final (random effects plus fixed effect parameters) phase. See modular for more information on these two phases.

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

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?

nAGQ0initStep — 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).

check.nlev.gtreq.5 — character - rules for checking whether all random effects have >= 5 levels. See action.

`check.nlev.gtr.1`
: character - rules for checking whether all random effects have > 1 level. See `action`.

`check.nobs.vs.rankZ`
: character - rules for checking whether the number of observations is greater than (or greater than or equal to) the rank of the random effects design matrix (Z), usually necessary for identifiable variances. As for `action`, with the addition of `"warningSmall"` and `"stopSmall"`, which run the test only if the dimensions of `Z` are < 1e6. `nobs > rank(Z)` will be tested for LMMs and GLMMs with estimated scale parameters; `nobs >= rank(Z)` will be tested for GLMMs with fixed scale parameter. The rank test is done using the `method="qr"` option of the `rankMatrix` function.

`check.nobs.vs.nlev`
: character - rules for checking whether the number of observations is less than (or less than or equal to) the number of levels of every grouping factor, usually necessary for identifiable variances. As for `action`. `nobs<nlevels` will be tested for LMMs and GLMMs with estimated scale parameters; `nobs<=nlevels` will be tested for GLMMs with fixed scale parameter.

`check.nobs.vs.nRE`
: character - rules for checking whether the number of observations is greater than (or greater than or equal to) the number of random-effects levels for each term, usually necessary for identifiable variances. As for `check.nobs.vs.nlev`.

`check.conv.grad`
: rules for checking the gradient of the deviance function for convergence. A list as returned by `.makeCC`, or a character string with only the action.

`check.conv.singular`
: rules for checking for a singular fit, i.e. one where some parameters are on the boundary of the feasible space (for example, random effects variances equal to 0 or correlations between random effects equal to +/- 1.0); as for `check.conv.grad` above. The default is to use `isSingular(.., tol = *)`'s default.

`check.conv.hess`
: rules for checking the Hessian of the deviance function for convergence.; as for `check.conv.grad` above.

`check.rankX`
: character - specifying if `rankMatrix(X)` should be compared with `ncol(X)` and if columns from the design matrix should possibly be dropped to ensure that it has full rank. Sometimes needed to make the model identifiable. The options can be abbreviated; the three `"*.drop.cols"` options all do drop columns, `"stop.deficient"` gives an error when the rank is smaller than the number of columns where `"ignore"` does no rank computation, and will typically lead to less easily understandable errors, later.

`check.scaleX`
: character - check for problematic scaling of columns of fixed-effect model matrix, e.g. parameters measured on very different scales.

`check.formula.LHS`
: check whether specified formula has a left-hand side. Primarily for internal use within `simulate.merMod`; *use at your own risk* as it may allow the generation of unstable `merMod` objects

`check.response.not.const`
: character - check that the response is not constant.

optCtrl	a list of additional arguments to be passed to the nonlinear optimizer (see Nelder_Mead, bobyqa). In particular, both Nelder_Mead and bobyqa use maxfun to specify the maximum number of function evaluations they will try before giving up - in contrast to optim and optimx-wrapped optimizers, which use maxit. (Also see convergence for details of stopping tolerances for different optimizers.) *Note:* All of lmer(), glmer() and nlmer() have an optional integer argument verbose which you should raise (to a positive value) in order to get diagnostic console output about the optimization progress.
action	character - generic choices for the severity level of any test, with possible values **"ignore":** skip the test. **"warning":** warn if test fails. **"message":** print a message if test fails. **"stop":** throw an error if test fails.
tol	(numeric) tolerance for checking the gradient, scaled relative to the curvature (i.e., testing the gradient on a scale defined by its Wald standard deviation)
relTol	(numeric) tolerance for the gradient, scaled relative to the magnitude of the estimated coefficient
mod.type	model type (for internal use)
standardize.X	scale columns of X matrix? (not yet implemented)
...	other elements to include in check specification

Details

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

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

Value

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

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

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

See Also

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

Examples

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

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

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

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

Description

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

Usage

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

Arguments

formula	a linear 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 (glm).
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. 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.
pool	logical scalar indicating if the variance estimate should pool the residual sums of squares. By default true if the model has a scale parameter (which includes all linear, lmer(), ones).
warn	indicating if errors in the single fits should signal a "summary" warning.
...	additional, optional arguments to be passed to the model function or family evaluation.

Details

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

  In previous **lme4** versions, a general (different) warning had been signalled in this case.

Value

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

See Also

`lmList4`

Examples

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

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

`lmList4-class` *Class "lmList4" of 'lm' Objects on Common Model*

Description

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

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

Objects from the Class

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

Methods

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

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

See Also

lmList

Examples

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

---

lmResp *Generator objects for the response classes*

Description

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

Usage

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

Arguments

...	List of arguments (see Note).

Methods

new(y=y): Create a new lmResp or lmerResp object.

new(family=family, y=y): Create a new glmResp object.

new(y=y, 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

---

lmResp-class *Reference Classes for Response Modules,* "(lm|glm|nls|lmer)Resp"

Description

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

Extends

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

Note

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

See Also

lmer, glmer, nlmer, merMod.

Examples

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

---

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

Description

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

Usage

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

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

## S3 method for class 'merMod'
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."),
      ranef.corr = any(ranef.comp == "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 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 $p <= 12$, and the cutoff 12 may be modified by options(lme4.summary.cor.max = <n>)
use.hessian	(logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects; see vcov.merMod for details
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 (PIRLS). 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.
ranef.corr	(logical) print correlations (rather than covariances) of random effects?
...	potentially further arguments passed from other methods.

Objects from the Class

Objects of class merMod are created by calls to lmer, glmer or nlmer.

S3 methods

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

anova: returns the sequential decomposition of the contributions of fixed-effects terms or, for multiple arguments, model comparison statistics. For objects of class lmerMod the default behavior is to refit the models with ML if fitted with REML = TRUE, this can be controlled via the refit argument. See also anova.

as.function: returns the deviance function, the same as lmer(*, devFunOnly=TRUE), and mkLmerDevfun() or mkGlmerDevfun(), respectively.

coef: Computes the sum of the random and fixed effects coefficients for each explanatory variable for each level of each grouping factor.

extractAIC: Computes the (generalized) Akaike An Information Criterion. If isREML(fit), then fit is refitted using maximum likelihood.

`family`: `family` of fitted GLMM. (*Warning:* this accessor may not work properly with customized families/link functions.)

`fitted`: Fitted values, given the conditional modes of the random effects. For more flexible access to fitted values, use `predict.merMod`.

`logLik`: Log-likelihood at the fitted value of the parameters. Note that for GLMMs, the returned value is only proportional to the log probability density (or distribution) of the response variable. See `logLik`.

`model.frame`: returns the `frame` slot of `merMod`.

`model.matrix`: returns the fixed effects model matrix.

`nobs`, `ngrps`: Number of observations and vector of the numbers of levels in each grouping factor. See `ngrps`.

`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.

`update`: See `update`.

Deviance and log-likelihood of GLMMs

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

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

	conditional	unconditional
relative	`deviance(object)`	NA in `lme4`
absolute	`object@resp$aic()`	`-2*logLik(object)`

This table requires two caveats:

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

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

Slots

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

`Gp`: See `getME`.

`call`: The matched call.

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

`flist`: See `getME`.

`cnms`: See `getME`.

`lower`: See `getME`.

`theta`: Covariance parameter vector.

`beta`: Fixed effects coefficients.

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

`devcomp`: See `getME`.

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

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

See Also

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

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

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

`merPredD` *Generator object for the* `merPredD` *class*

Description

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

Usage

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

Arguments

...	List of arguments (see Note).

Note

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

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

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

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

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

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

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

See Also

The class definition, `merPredD`, also for examples.

`merPredD-class` *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

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

---

mkMerMod *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.

---

mkRespMod *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

---

mkReTrms *Make Random Effect Terms: Create Z, Lambda, Lind, etc.*

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

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?
reorder.terms	arrange random effects terms in decreasing order of number of groups (factor levels)?
reorder.vars	arrange columns of individual random effects terms in alphabetical order?

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

Value

a list with components

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

Note

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

See Also

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

Examples

```
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` *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

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

```
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

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

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

optimizeLmer(devfun,
             optimizer    = formals(lmerControl)$optimizer,
```

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

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

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

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

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 **re**stricted 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,mkGlmerDevfun**:** options for the inner optimization step;

for optimizeLmer **and** optimizeGlmer**:** 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 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 (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.

restart_edge — see lmerControl

boundary.tol — see lmerControl

family — a GLM family; see glm and family.

mustart	optional starting values on the scale of the conditional mean; see glm for details.
etastart	optional starting values on the scale of the unbounded predictor; see glm for details.
...	other potential arguments; for optimizeLmer and optimizeGlmer, these are passed to internal function optwrap, which has relevant parameters calc.derivs and use.last.params (see lmerControl).

Details

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

- [g]lFormula takes the arguments that would normally be passed to [g]lmer, checking for errors and processing the formula and data input to create a list of objects required to fit a mixed model.
- mk(Gl|L)merDevfun takes the output of the previous step (minus the formula component) and creates a deviance function
- optimize(Gl|L)mer takes a deviance function and optimizes over theta (or over theta and beta, if stage is set to 2 for 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 mk(Gl|L)merDevfun contains reference class objects (see ReferenceClasses, merPredD-class, lmResp-class), which behave in ways that may surprise many users. For example, if the output of mk(Gl|L)merDevfun is naively copied, then modifications to the original will also appear in the copy (and vice versa). To avoid this behavior one must make a deep copy (see ReferenceClasses for details).

optimizeLmer and optimizeGlmer return the results of an optimization.

Examples

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

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


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


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

## 1.  Parse the data and formula:
glmod <- glFormula(cbind(incidence, size - incidence) ~ period + (1 | herd),
                   data = cbpp, family = binomial)
    #.... see what've got :
str(glmod, max=1, give.attr=FALSE)
## 2.  Create the deviance function for optimizing over theta:
(devfun <- do.call(mkGlmerDevfun, glmod))
ls(environment(devfun)) # the environment of devfun contains lots of info
## 3.  Optimize over theta using a rough approximation (i.e. nAGQ = 0):
(opt <- optimizeGlmer(devfun))
## 4.  Update the deviance function for optimizing over theta and beta:
(devfun <- updateGlmerDevfun(devfun, glmod$reTrms))
## 5.  Optimize over theta and beta:
opt <- optimizeGlmer(devfun, stage=2)
str(opt, max=1) # seeing what we'got
## 6.  Package up the results:
(fMod <- mkMerMod(environment(devfun), opt, glmod$reTrms, fr = glmod$fr))

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

`namedList` *Self-naming list function*

Description

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

Usage

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

Arguments

`...`	comma-separated arguments

Examples

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

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

Description

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

Usage

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

Arguments

`fn`	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 (default: 1e-5)

FtolRel numeric scalar - relative tolerance on change in function values (default:1e-15)

XtolRel numeric scalar - relative tolerance on change in parameter values (default: 1e-7)

MinfMax numeric scalar - maximum value of the minimum (default: .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 evalutions, 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 inspected. Defaults to `FALSE`, i.e., stop on non-convergence.

Value

a `list` with components

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

See Also

The `NelderMead` class definition and generator function.

Examples

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

oo  <- optim(p0, fr) ## also uses Nelder-Mead by default
o.  <- Nelder_Mead(fr, p0)
```

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

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

`NelderMead-class` *Class* "`NelderMead`" *of Nelder-Mead optimizers and its Generator*

Description

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

The `NelderMead()` function conveniently generates such objects.

Usage

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

Arguments

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

Methods

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

Extends

All reference classes extend and inherit methods from "`envRefClass`".

Note

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

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

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

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

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

x0 numeric vector of starting values for the parameters.

xt numeric vector of tolerances on the parameters.

References

Based on code in the NLopt collection.

See Also

Nelder_Mead, the typical "constructor". Further, glmer, nlmer

Examples

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

---

ngrps *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"
```

`nlformula` *Manipulate a Nonlinear Model Formula*

Description

Check and manipulate the formula for a nonlinear model, such as specified in `nlmer`.

Usage

```
nlformula(mc)
```

Arguments

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

Details

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

Value

a list with components

`"respMod"`	a response module of class `"nlsResp"`
`"frame"`	the model frame, including a terms attribute
`"X"`	the fixed-effects model matrix
`"reTrms"`	the random-effects terms object

See Also

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

nlmer *Fitting Nonlinear Mixed-Effects Models*

Description

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

Usage

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

Arguments

formula	a three-part "nonlinear mixed model" formula, of the form `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 (PIRLS aka PRSS, e.g. in the C++ sources).
nAGQ	integer scalar - the number of points per axis for evaluating the adaptive Gauss-Hermite approximation to the log-likelihood. Defaults to 1, corresponding to the Laplace approximation. Values greater than 1 produce greater accuracy in the evaluation of the log-likelihood at the expense of speed. A value of zero uses a faster but less exact form of parameter estimation for GLMMs by optimizing

the random effects and the fixed-effects coefficients in the penalized iteratively reweighted least squares (PIRLS) step.

`subset` an optional expression indicating the subset of the rows of `data` that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

`weights` an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.

`na.action` a function that indicates what should happen when the data contain `NA`s. 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`.

`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).

Details

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

Note

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

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

Examples

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

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

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

`nloptwrap` *Wrappers for additional optimizers*

Description

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

Usage

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

Arguments

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

Details

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

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

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

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

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

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

Value

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

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

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

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

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

---

nobars *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
```

Pastes *Paste strength by batch and cask*

Description

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

Format

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

`strength` paste strength.

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

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

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

Details

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

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

```
str(Pastes)
require(lattice)
dotplot(cask ~ strength | reorder(batch, strength), Pastes,
        strip = FALSE, strip.left = TRUE, layout = c(1, 10),
        ylab = "Cask within batch",
        xlab = "Paste strength", jitter.y = TRUE)
## Modifying the factors to enhance the plot
Pastes <- within(Pastes, batch <- reorder(batch, strength))
Pastes <- within(Pastes, sample <- reorder(reorder(sample, strength),
          as.numeric(batch)))
dotplot(sample ~ strength | batch, Pastes,
        strip = FALSE, strip.left = TRUE, layout = c(1, 10),
        scales = list(y = list(relation = "free")),
        ylab = "Sample within batch",
```

```
        xlab = "Paste strength", jitter.y = TRUE)
## Four equivalent models differing only in specification
(fm1 <- lmer(strength ~ (1|batch) + (1|sample), Pastes))
(fm2 <- lmer(strength ~ (1|batch/cask), Pastes))
(fm3 <- lmer(strength ~ (1|batch) + (1|batch:cask), Pastes))
(fm4 <- lmer(strength ~ (1|batch/sample), Pastes))
## fm4 results in redundant labels on the sample:batch interaction
head(ranef(fm4)[[1]])
## compare to fm1
head(ranef(fm1)[[1]])
## This model is different and NOT appropriate for these data
(fm5 <- lmer(strength ~ (1|batch) + (1|cask), Pastes))

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

`Penicillin` *Variation in penicillin testing*

Description

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

Format

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

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

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

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

Details

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

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

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

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

---

plot.merMod *Diagnostic Plots for 'merMod' Fits*

Description

diagnostic plots for merMod fits

Usage

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

Arguments

x	a fitted [ng]lmer model
form	an optional formula specifying the desired type of plot. Any variable present in the original data frame used to obtain x can be referenced. In addition, x itself can be referenced in the formula using the symbol ". ". Conditional expressions on the right of a | operator can be used to define separate panels in a lattice display. Default is 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 $1 - value/2$ 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 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.

`data` ignored: required for S3 method compatibility

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

See Also

`influencePlot` in the `car` package

Examples

```
data(Orthodont,package="nlme")
fm1 <- lmer(distance ~ age + (age|Subject), data=Orthodont)
## standardized residuals versus fitted values by gender
plot(fm1, resid(., scaled=TRUE) ~ fitted(.) | Sex, abline = 0)
## box-plots of residuals by Subject
plot(fm1, Subject ~ resid(., scaled=TRUE))
## observed versus fitted values by Subject
plot(fm1, distance ~ fitted(.) | Subject, abline = c(0,1))
## residuals by age, separated by Subject
```

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

`plots.thpr` *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

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

```
       which = 1:nptot, ...)

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

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

Arguments

x	a mixed-effects profile, i.e., of class "thpr", typically resulting from profile(fm) where fm is a fitted model from lmer (or its generalizations).
data	unused - only for compatibility with generic.
npts	the number of points to use for the densityplot().
upper	a number in $(0, 1)$ to specify upper (and lower) boundaries as +/- qnorm(upper).
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 abs(.)olute 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 **s**catter**plo**t **m**atrix, a "trellis" object, see above.

See Also

profile, notably for an example.

Examples

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

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

Description

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

Usage

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

Arguments

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

Details

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

Value

a numeric vector of predicted values

Examples

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

`profile-methods` *Profile method for merMod objects*

Description

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

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

Usage

```
## S3 method for class 'merMod'
profile(fitted, which = NULL, alphamax = 0.01,
 maxpts = 100, delta = NULL,
    delta.cutoff = 1/8, verbose = 0, devtol = 1e-09,
    devmatchtol = 1e-5,
    maxmult = 10, startmethod = "prev", optimizer = NULL,
    control=NULL, signames = TRUE,
    parallel = c("no", "multicore", "snow"),
    ncpus = getOption("profile.ncpus", 1L), cl = NULL,
    prof.scale = c("sdcor","varcov"),
    ...)
```

```
## S3 method for class 'thpr'
 as.data.frame(x, ...)
## S3 method for class 'thpr'
log(x, base = exp(1))
logProf(x, base = exp(1), ranef = TRUE,
           sigIni = if(ranef) "sig" else "sigma")
varianceProf(x, ranef = TRUE)
```

Arguments

`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.
`devmatchtol`	tolerance for match between original deviance computation and value returned from auxiliary deviance function
`maxmult`	maximum multiplier of the original step size allowed, defaults to 10.
`startmethod`	method for picking starting conditions for optimization (STUB).
`optimizer`	(character or function) optimizer to use (see `lmer` for details); default is to use the optimizer from the original model fit.

control	a list of options controlling the profiling (see lmerControl): default is to use the control settings from the original model fit.
signames	logical indicating if abbreviated names of the form .sigNN should be used; otherwise, names are more meaningful (but longer) of the form (sd|cor)_(effects)|(group). Note that some code for profile transformations (e.g., log() or varianceProf) depends on signames==TRUE.
...	potential further arguments for various methods.
x	an object of class thpr (i.e., output of profile)
base	the base of the logarithm. Defaults to natural logarithms.
ranef	logical indicating if the sigmas of the random effects should be log() transformed as well. If false, only σ (standard deviation of errors) is transformed.
sigIni	character string specifying the initial part of the sigma parameters to be log transformed.
parallel	The type of parallel operation to be used (if any). If missing, the default is taken from the option "profile.parallel" (and if that is not set, "no").
ncpus	integer: number of processes to be used in parallel operation: typically one would choose this to be the number of available CPUs.
cl	An optional **parallel** or **snow** cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the profile call.
prof.scale	whether to profile on the standard deviation-correlation scale ("sdcor") or on the variance-covariance scale ("varcov")

Details

The log method and the more flexible logProf() function transform the profile into one where $\log(\sigma)$ is used instead of σ. By default all sigmas including the standard deviations of the random effects are transformed i.e., the methods return a profile with all of the .sigNN parameters replaced by .lsigNN. If ranef is false, only ".sigma", the standard deviation of the errors, is transformed (as it should never be zero, whereas random effect standard deviations (.sigNN) can be reasonably be zero).
The forward and backward splines for the log-transformed parameters are recalculated. Note that correlation parameters are not handled sensibly at present (i.e., they are logged rather than taking a more applicable transformation such as an arc-hyperbolic tangent, atanh(x)=$\log((1+x)/(1-x))/2$).

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

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

Value

profile(<merMod>) returns an object of S3 class "thpr", which is data.frame-like. Notable methods for such a profile object confint(), which returns the confidence intervals based on the

profile, and three plotting methods (which require the **lattice** package), `xyplot`, `densityplot`, and `splom`.

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

See Also

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

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

Examples

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

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

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

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

```
  print( xyplot(log(pr4), absVal=TRUE) ) # log(sigma_1)
  print( splom(pr4) )
  print( system.time( # quicker: only sig01 and one fixed effect
      pr2 <- profile(gm1, which=c("theta_", "period2"))))
  print( confint(pr2) )
  ## delta..: higher underlying resolution, only for 'sigma_1':
  print( system.time(
      pr4.hr <- profile(gm1, which="theta_", delta.cutoff=1/16)))
  print( xyplot(pr4.hr) )
 }
} # only if interactive()
```

---

prt-utilities *Print and Summary Method Utilities for Mixed Effects*

Description

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

Notably see the Examples.

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

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

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

Usage

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

methTitle(dims)

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

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

Arguments

object	a LMM model fit
devianceFUN	the function to be used for computing the deviance; should not be changed for **lme4** created objects.
chkREML	optional logical indicating if object maybe a REML fit.
devcomp	for **lme4** always the equivalent of object@devcomp; here a list
dims	for **lme4** always the equivalent of object@devcomp$dims, a named vector or list with components "GLMM", "NLMM", "REML", and "nAGQ" of which the first two are logical scalars, and the latter two typically are FALSE or numeric.
mtit	the result of methTitle(object)
class	typically class(object).
famL	a list with components family and link, each a character string; note that standard R family objects can be used directly, as well.
resids	numeric vector of model residuals.
digits	non-negative integer of (significant) digits to print minimally.
title	character string.
...	optional arguments passed on, e.g., to residuals().
call	the call of the model fit; e.g., available via (generic) function getCall().
long	logical indicating if the output may be long, e.g., printing the control part of the call if there is one.
aictab	typically the AICtab component of the result of llikAIC().
varcor	typically the result of VarCorr().
comp	optional character vector of length 1 or 2, containing "Std.Dev." and/or "Variance", indicating the columns to use.
corr	logical indicating if correlations or covariances should be used for vector random effects.
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

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

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


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

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

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

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

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

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

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

V1 <- VarCorr(fm1)
m <- formatVC(V1)
stopifnot(is.matrix(m), is.character(m), ncol(m) == 4)
print(m, quote = FALSE) ## prints all but the first line of .prt.VC() below:
.prt.VC( V1, digits = 4)
## Random effects:
##  Groups   Name        Std.Dev. Corr
##  Subject  (Intercept) 24.740
##           Days         5.922   0.07
##  Residual             25.592
p1 <- capture.output(V1)
p2 <- capture.output( print(m, quote=FALSE) )
pX <- capture.output( .prt.VC(V1, digits = max(3, getOption("digits")-2)) )
stopifnot(identical(p1, p2),
          identical(p1, pX[-1])) # [-1] : dropping 1st line
```

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

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

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

---

pvalues *Getting p-values for fitted models*

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 generated a Markov chain Monte Carlo sample from the posterior distribution of the parameters, assuming flat (scaled likelihood) priors. Due to difficulty in constructing a version of 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 KRmodcomp from the pbkrtest package (MC,*)
- car::Anova and lmerTest::anova provide wrappers for Kenward-Roger-corrected tests using pbkrtest: lmerTest::anova also provides t tests via the Satterthwaite approximation (P,*)
- afex::mixed is another wrapper for pbkrtest and anova providing "Type 3" tests of all effects (P,*,+)

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

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

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

---

ranef *Extract the modes of the random effects*

Description

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

Usage

```
## S3 method for class 'merMod'
 ranef(object, condVar = TRUE,
       drop = FALSE, whichel = names(ans), postVar = FALSE, ...)
## S3 method for class 'ranef.mer'
 dotplot(x, data, main = TRUE, transf = I, level = 0.95, ...)
## S3 method for class 'ranef.mer'
 qqmath(x, data, main = TRUE, level = 0.95, ...)
## S3 method for class 'ranef.mer'
 as.data.frame(x, ...)
```

Arguments

object	an object of a class of fitted models with random effects, typically a merMod object.
condVar	a logical argument indicating if the conditional variance-covariance matrices of the random effects should be added as an attribute.
drop	should components of the return value that would be data frames with a single column, usually a column called '(Intercept)', be returned as named vectors instead?
whichel	character vector of names of grouping factors for which the random effects should be returned.
postVar	a (deprecated) synonym for condVar
x	a random-effects object (of class 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.
level	confidence level for confidence intervals
...	some methods for these generic functions require additional arguments.

Details

If grouping factor i has k levels and j random effects per level the 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 (or a list of such arrays). 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; 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

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

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

  - If there is a single random-effects term for a given grouping factor, this attribute is a three-dimensional array with symmetric faces; each face contains the variance-covariance matrix for a particular level of the grouping factor.
  - If there is more than one random-effects term for a given grouping factor (e.g. (1|f) + (0+x|f)), this attribute is a list of arrays as described above, one for each term.

  (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.

- From as.data.frame:

  This function converts the random effects to a "long format" data frame with columns

  grpvar grouping variable

  term random-effects term, e.g. "(Intercept)" or "Days"

  grp level of the grouping variable (e.g., which Subject)

  condval value of the conditional mean

  condsd conditional standard deviation

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

```
library(lattice) ## for dotplot, qqmath
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
fm2 <- lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy)
fm3 <- lmer(diameter ~ (1|plate) + (1|sample), Penicillin)
ranef(fm1)
str(rr1 <- ranef(fm1))
dotplot(rr1)  ## default
qqmath(rr1)
## specify free scales in order to make Day effects more visible
dotplot(rr1,scales = list(x = list(relation = 'free')))[["Subject"]]
## plot options: ... can specify appearance of vertical lines with
## lty.v, col.line.v, lwd.v, etc..
dotplot(rr1, lty = 3, lty.v = 2, col.line.v = "purple",
        col = "red", col.line.h = "gray")
ranef(fm2)
op <- options(digits = 4)
ranef(fm3, drop = TRUE)
options(op)
## as.data.frame() provides RE's and conditional standard deviations:
str(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, newweights = NULL,
      rename.response = FALSE,
      maxit = 100, ...)
```

Arguments

`object`	a fitted model, usually of class `lmerMod`, 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).
newweights	an (optional) numeric vector of new weights
rename.response	when refitting the model, should the name of the response variable in the formula and model frame be replaced with the name of newresp?
maxit	scalar integer, currently only for GLMMs: the maximal number of Pwrss update iterations.
...	optional additional parameters. For the 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 removed from the numeric vector; this allows the results of simulate(object) to be used even if the original response vector contained NA values. Otherwise, the length of newresp must be the same as the *original* length of the response.

Value

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

See Also

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

Examples

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

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

`refitML` *Refit a Model by Maximum Likelihood Criterion*

Description

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

Usage

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

Arguments

`x`	a fitted model, usually of class "`lmerMod`", 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.

`rePCA` *PCA of random-effects covariance matrix*

Description

PCA of random-effects variance-covariance estimates

Usage

```
rePCA(x)
```

Arguments

`x`	a `merMod` object

Details

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

Value

a `prcomplist` object

Author(s)

Douglas Bates

References

- Douglas Bates, Reinhold Kliegl, Shravan Vasishth, and Harald Baayen. Parsimonious Mixed Models. arXiv:1506.04967 [stat], June 2015. arXiv: 1506.04967.

See Also

`isSingular`

Examples

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

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

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

`residuals.merMod` *residuals of merMod objects*

Description

residuals of merMod objects

Usage

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

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

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

Arguments

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

Details

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

sigma *Extract Residual Standard Deviation 'Sigma'*

Description

Extract the estimated standard deviation of the errors, the "residual standard deviation" (also misnamed the "residual standard error"), from a fitted model.

Usage

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

Arguments

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

Details

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

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

simulate.formula *A* simulate *Method for* formula *objects that dispatches based on the Left-Hand Side*

Description

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

Usage

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

Arguments

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

Details

The dispatching works as follows:

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

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

---

simulate.merMod *Simulate Responses From* merMod *Object*

Description

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

Usage

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

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

Arguments

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

Details

- ordinarily `simulate` is used to generate new values from an existing, fitted model (`merMod` object): however, if `formula`, `newdata`, and `newparams` are specified, `simulate` generates the appropriate model structure to simulate from. `formula` must be a *one-sided* formula (i.e. with an empty left-hand side); in general, if `f` is a two-sided formula, `f[-2]` can be used to drop the LHS.
- The `re.form` argument allows the user to specify how the random effects are incorporated in the simulation. All of the random effects terms included in `re.form` will be *conditioned on* - that is, the conditional modes of those random effects will be included in the deterministic part of the simulation. (If new levels are used (and `allow.new.levels` is `TRUE`), the conditional modes for these levels will be set to the population mode, i.e. values of zero will be used for the random effects.) Conversely, the random effect terms that are *not* included in `re.form` will be *simulated from* - that is, new values will be chosen for each group based on the estimated random-effects variances.

  The default behaviour (using `re.form=NA`) is to condition on none of the random effects, simulating new values for all of the random effects.
- For Gaussian fits, `sigma` specifies the residual standard deviation; for Gamma fits, it specifies the 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=1.5))`.
- For binomial models, `simulate.formula` looks for the binomial size first in the `weights` argument (if it's supplied), second from the left-hand side of the formula (if the formula has been specified in success/failure form), and defaults to 1 if neither of those have been supplied. Simulated responses will be given as proportions, unless the supplied formula has a matrix-valued left-hand side, in which case they will be given in matrix form. If a left-hand side is given, variables in that expression must be available in `newdata`.
- For negative binomial models, use the `negative.binomial` family (from the **MASS** package) and specify the overdispersion parameter via the `theta` (sic) parameter of the family function, e.g. `simulate(...,family=negative.binomial(theta=1))` to simulate from a geometric distribution (negative binomial with overdispersion parameter 1).

See Also

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

Examples

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

```
params <- list(theta=0.5,beta=c(2,-1,-2,-3))
simdat <- with(cbpp,expand.grid(herd=levels(herd),period=factor(1:4)))
simdat$size <- 15
simdat$incidence <- sample(0:1,size=nrow(simdat),replace=TRUE)
form <- formula(gm1)[-2]  ## RHS of equation only
simulate(form,newdata=simdat,family=binomial,
    newparams=params)
## simulate from negative binomial distribution instead
simulate(form,newdata=simdat,family=negative.binomial(theta=2.5),
    newparams=params)
```

`sleepstudy` *Reaction times in a sleep deprivation study*

Description

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

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

Format

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

`Reaction` Average reaction time (ms)

`Days` Number of days of sleep deprivation

`Subject` Subject number on which the observation was made.

Details

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

References

Gregory Belenky, Nancy J. Wesensten, David R. Thorne, Maria L. Thomas, Helen C. Sing, Daniel P. Redmond, Michael B. Russo and Thomas J. Balkin (2003) Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: a sleep dose-response study. *Journal of Sleep Research* **12**, 1–12.

Examples

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

---

subbars *"Sub[stitute] 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

```
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

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

troubleshooting *Troubleshooting*

Description

This page attempts to summarize some of the common problems with fitting [gn]lmer models and how to troubleshoot them.

Most of the symptoms/diagnoses/workarounds listed below are due to various issues in the actual mixed model fitting process. You may run into problems due to multicollinearity or variables that are incorrectly typed (e.g. a variable is accidentally coded as character or factor rather than numeric). These problems can often be isolated by trying a lm or glm fit or attempting to construct the design matrix via model.matrix() (in each case with the random effects in your model excluded). If these tests fail then the problem is likely not specifically an lme4 issue.

- failure to converge in (xxxx) evaluations The optimizer hit its maximum limit of function evaluations. To increase this, use the optControl argument of [g]lmerControl – for Nelder_Mead and bobyqa the relevant parameter is maxfun; for optim and optimx-wrapped optimizers, including nlminbwrap, it's maxit; for nloptwrap, it's maxeval.
- Model failed to converge with max|grad| ... The scaled gradient at the fitted (RE)ML estimates is worryingly large. Try
  - refitting the parameters starting at the current estimates: getting consistent results (with no warning) suggests a false positive
  - switching optimizers: getting consistent results suggests there is not really a problem; getting a similar log-likelihood with different parameter estimates suggests that the parameters are poorly determined (possibly the result of a misspecified or overfitted model)
  - compute values of the deviance in the neighbourhood of the estimated parameters to double-check that lme4 has really found a local optimum.
- Hessian is numerically singular: parameters are not uniquely determined The Hessian (inverse curvature matrix) at the maximum likelihood or REML estimates has a very large eigenvalue, indicating that (within numerical tolerances) the surface is completely flat in some direction. The model may be misspecified, or extremely badly scaled (see "Model is nearly unidentifiable").
- Model is nearly unidentifiable ... Rescale variables? The Hessian (inverse curvature matrix) at the maximum likelihood or REML estimates has a large eigenvalue, indicating that the surface is nearly flat in some direction. Consider centering and/or scaling continuous predictor variables.
- Contrasts can be applied only to factors with 2 or more levels One or more of the categorical predictors in the model has fewer than two levels. This may be due to user error when converting these predictors to factors prior to modeling, or it may result from some factor levels being eliminated due to NAs in other predictors. Double-check the number of data points in each factor level to see which one is the culprit: lapply(na.omit(df[,vars]), table) (where df is the data.frame and vars are the column names of your predictor variables).

VarCorr *Extract Variance and Correlation Components*

Description

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

Usage

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

## S3 method for class 'VarCorr.merMod'
as.data.frame(x, row.names = NULL,
    optional = FALSE, order = c("cov.last", "lower.tri"), ...)
## S3 method for class 'VarCorr.merMod'
print(x, digits = max(3, getOption("digits") - 2),
    comp = "Std.Dev.", corr = any(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().
corr	(logical) print correlations (rather than covariances) of random effects?

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

var1 first variable

var2 second variable (`NA` for variance parameters)

vcov variances or covariances

sdcor standard deviations or correlations

Author(s)

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

See Also

`lmer`, `nlmer`

Examples

```
data(Orthodont, package="nlme")
fm1 <- lmer(distance ~ age + (age|Subject), data = Orthodont)
print(vc <- VarCorr(fm1))  ## default print method: standard dev and corr
## both variance and std.dev.
print(vc,comp=c("Variance","Std.Dev."), digits=2)
## variance only
print(vc, comp=c("Variance"))
## standard deviations only, but covariances rather than correlations
print(vc, corr = FALSE)
as.data.frame(vc)
as.data.frame(vc, order="lower.tri")
```

---

vcconv *Convert between representations of (co-)variance structures*

Description

Convert between representations of (co-)variance structures (EXPERIMENTAL). See source code for details.

Usage

```
mlist2vec(L)
vec2mlist(v, n = NULL, symm = TRUE)
vec2STlist(v, n = NULL)
sdcor2cov(m)
cov2sdcor(V)
Vv_to_Cv(v, n = NULL, s = 1)
Sv_to_Cv(v, n = NULL, s = 1)
Cv_to_Vv(v, n = NULL, s = 1)
Cv_to_Sv(v, n = NULL, s = 1)
```

Arguments

L	List of symmetric, upper-triangular, or lower-triangular square matrices.
v	Concatenated vector containing the elements of the lower-triangle (including the diagonal) of a symmetric or triangular matrix.
n	Number of rows (and columns) of the resulting matrix.
symm	Return symmetric matrix if TRUE or lower-triangular if FALSE.
m	Standard deviation-correlation matrix.
V	Covariance matrix.
s	Scale parameter.

Details

`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 sdcor vector. Returns a vector of elements from the lower triangle of a standard-deviation-correlation matrix.

Value

(Co-)variance structure

Examples

```
vec2mlist(1:6)
mlist2vec(vec2mlist(1:6)) # approximate inverse
```

`vcov.merMod` *Covariance matrix of estimated parameters*

Description

Compute the variance-covariance matrix of estimated paramers. Optionally also computes correlations, or the full (joint) covariance matrix of the fixed-effect coefficients and the conditional modes of the random effects.

Usage

```
## S3 method for class 'merMod'
vcov(object, correlation = TRUE, sigm = sigma(object),
    use.hessian = NULL, full = FALSE, ...)
```

Arguments

`object`	an R object of class `merMod`, i.e., as resulting from `lmer()`, or `glmer()`, etc.
`correlation`	(logical) indicates whether the correlation matrix as well as the variance-covariance matrix is desired
`sigm`	the residual standard error; by default `sigma(object)`.
`use.hessian`	(logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects. See *Details*.
`full`	return the 'full' covariance matrix, i.e. the joint covariance matrix of the conditional distribution of conditional modes (as in `getME(., "b")`) and fixed-effect parameters. (`correlation` and `use.hessian` are *ignored* in this case.) Note that this option may be slow for models with large numbers of random-effect levels!
`...`	extra arguments for method compatibility (ignored)

Details

When `use.hessian = FALSE`, the code estimates the covariance matrix based on internal information about the inverse of the model matrix (see `getME(.,"RX")`). This is exact for linear mixed models, but approximate for GLMMs. The default is to use the Hessian whenever the fixed effect parameters are arguments to the deviance function (i.e. for GLMMs with `nAGQ>0`), and to use `getME(.,"RX")` whenever the fixed effect parameters are profiled out (i.e. for GLMMs with `nAGQ==0` or LMMs).

use.hessian=FALSE is backward-compatible with older versions of lme4, but may give less accurate SE estimates when the estimates of the fixed-effect (see getME(.,"beta")) and random-effect (see getME(.,"theta")) parameters are correlated.

However, use.hessian=TRUE is not always more accurate: for some numerically unstable fits, the approximation using RX is actually more reliable (because the Hessian has to be computed by a finite difference approximation, which is also error-prone): see e.g. https://github.com/lme4/lme4/issues/720

Value

a covariance matrix (sparse when full=TRUE)

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
             data = cbpp, family = binomial)
(v1 <- vcov(fm1))
v2 <- vcov(fm1, correlation = TRUE)
# extract the hidden 'correlation' entry in @factors
as(v2, "corMatrix")
v3 <- vcov(gm1)
v3X <- vcov(gm1, use.hessian  = FALSE)
all.equal(v3, v3X)
## full correlatiom matrix
cv <- vcov(fm1, full = TRUE)
image(cv, xlab = "", ylab = "",
      scales = list(y = list(labels = rownames(cv)),
                    at = seq(nrow(cv)),
                    x = list(labels = NULL)))
```

VerbAgg *Verbal Aggression item responses*

Description

These are the item responses to a questionaire on verbal aggression. These data are used throughout De Boeck and Wilson (2004) to illustrate various forms of item response models.

Format

A data frame with 7584 observations on the following 13 variables.

Anger the subject's Trait Anger score as measured on the State-Trait Anger Expression Inventory (STAXI)

Gender the subject's gender - a factor with levels M and F

item the item on the questionaire, as a factor

resp the subject's response to the item - an ordered factor with levels no < perhaps < yes

`id` the subject identifier, as a factor

`btype` behavior type - a factor with levels `curse`, `scold` and `shout`

`situ` situation type - a factor with levels `other` and `self` indicating other-to-blame and self-to-blame

`mode` behavior mode - a factor with levels `want` and `do`

`r2` dichotomous version of the response - a factor with levels `N` and `Y`

Source

Data originally from the UC Berkeley BEAR Center; original link is available at https://web.archive.org/web/20221128003829/https://old.bear.berkeley.edu/page/materials-explanatory-item-respon but the data are no longer accessible there.

References

De Boeck and Wilson (2004), *Explanatory Item Response Models*, Springer.

Examples

```
str(VerbAgg)
## Show how  r2 := h(resp) is defined:
with(VerbAgg, stopifnot( identical(r2, {
     r <- factor(resp, ordered=FALSE); levels(r) <- c("N","Y","Y"); r})))

xtabs(~ item + resp, VerbAgg)
xtabs(~ btype + resp, VerbAgg)
round(100 * ftable(prop.table(xtabs(~ situ + mode + resp, VerbAgg), 1:2), 1))
person <- unique(subset(VerbAgg, select = c(id, Gender, Anger)))
require(lattice)
densityplot(~ Anger, person, groups = Gender, auto.key = list(columns = 2),
            xlab = "Trait Anger score (STAXI)")

if(lme4:::testLevel() >= 3) { ## takes about 15 sec
    print(fmVA <- glmer(r2 ~ (Anger + Gender + btype + situ)^2 +
      (1|id) + (1|item), family = binomial, data =
     VerbAgg), corr=FALSE)
} ## testLevel() >= 3
if (interactive()) {
## much faster but less accurate
    print(fmVA0 <- glmer(r2 ~ (Anger + Gender + btype + situ)^2 +
                             (1|id) + (1|item), family = binomial,
                         data = VerbAgg, nAGQ=0L), corr=FALSE)
} ## interactive()
```

Index