Package 'lme4'

April 13, 2014

Version 1.1-6

Title Linear mixed-effects models using Eigen and S4

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

Contact LME4 Authors <lme4-authors@lists.r-forge.r-project.org>

Author Douglas Bates [aut], Martin Maechler [aut],Ben Bolker [aut, cre], Steven Walker [aut],Rune Haubo Bojesen Christensen [ctb],Henrik Singmann [ctb]

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 (>= 2.15.1), Matrix (>= 1.1.1), methods, stats, Rcpp (>= 0.10.5)

LinkingTo Rcpp, RcppEigen

Imports graphics, grid, splines, parallel, MASS, nlme, lattice, minqa (>= 1.1.15), RcppEigen (>= 0.3.2.0)

Suggests boot, PKPDmodels, MEMSS, testthat, ggplot2, mlmRev, optimx (>= 2013.8.6), plyr, reshape, pbkrtest

LazyData yes

License GPL (>= 2)

URL https://github.com/lme4/lme4/ http://lme4.r-forge.r-project.org/

BuildVignettes yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-04-13 15:59:30

R topics documented:

`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**.)

`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,
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).
type	character string specifying the type of bootstrap, "parametric" or "semiparametric"; partial matching is allowed.
verbose	logical indicating if progress should print output
.progress	character string - type of progress bar to display. Default is "none"; the function will look for a relevant *ProgressBar function, so "txt" will work in general; "tk" is available if the **tcltk** package is loaded; or "win" on Windows systems. Progress bars are disabled (with a message) for parallel operation.
PBargs	a list of additional arguments to the progress bar function (the package authors like list(style=3)).
parallel	The type of parallel operation to be used (if any). If missing, the default is taken from the option "boot.parallel" (and if that is not set, "no").
ncpus	integer: number of processes to be used in parallel operation: typically one would choose this to be the number of available CPUs.
cl	An optional **parallel** or **snow** cluster for use if parallel = "snow". If not supplied, a cluster on the local machine is created for the duration of the boot call.

Details

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

The working name for bootMer() was "simulestimate()", as it is an extension of simulate (see simulate.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.

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

```
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
require("boot") ## a recommended package, i.e. *must* be there
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 ------------

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

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

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

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

## Check stored values from a longer (1000-replicate) run:
load(system.file("testdata","boo01L.RData",package="lme4"))
plot(boo01L,index=3)
```

---

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))
```

`confint.merMod` *Compute confidence intervals on the parameters of an lme4 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 = "perc", quiet = FALSE,
oldNames = TRUE, ...)
```

Arguments

`object`	a fitted [ng]lmer model
`parm`	parameters (specified by integer position)
`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.
boot.type	bootstrap confidence interval type.
quiet	(logical) suppress messages about computationally intensive profiling?
oldNames	(logical) use old-style names for method="profile"? (See signames argument to profile).
...	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": approximate the confidence intervals (of fixed-effect parameters only) based on the estimated local curvature of the likelihood surface;

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

Value

a numeric table of confidence intervals

Examples

```
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
fm1W <- confint(fm1, method="Wald")# very fast, but ....
fm1W
testLevel <- if (nzchar(s <- Sys.getenv("LME4_TEST_LEVEL"))) as.numeric(s) else 1
if(interactive() || testLevel >= 3) {
 ## ~20 seconds, MacBook Pro laptop
 system.time(fm1P <- confint(fm1, method="profile", ## default
                             oldNames = FALSE))
 ## ~ 40 seconds
 system.time(fm1B <- confint(fm1,method="boot",
                             .progress="txt", PBargs=list(style=3)))
} else
  load(system.file("testdata","confint_ex.rda",package="lme4"))
fm1P
fm1B
```

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")`

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. specified in dots). The first two arguments must be object, the full model fit, and objectDrop, a reduced model. If objectDrop is missing, sumFun should return a vector of with the appropriate length and names (the actual contents are ignored).
...	other arguments (ignored)

Details

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

Value

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

Examples

```
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

See Also

formula, model.frame, model.matrix.

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

Examples

```
f <- y ~ x + (x || g)
# the right-hand side of f is,
f[[3]]
# the expanded right-hand side,
expandDoubleVerts(f[[3]])
```

`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 )
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, ...)
```

Arguments

object	any fitted model object from which fixed effects estimates can be extracted.
...	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))
```

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

Description

add information to data based on a fitted model

Usage

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

Arguments

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

Details

`fortify` is a function defined in the **ggplot2** package, q.v. for more details. `fortify` is *not* defined here, and `fortify.merMod` is defined as a function rather than an S3 method, to avoid (1) inducing a dependency on **ggplot2** or (2) masking methods from **ggplot2**. This is currently an experimental feature.

`getME` *Extract or Get Generalized Components from a Fitted Mixed Effects Model*

Description

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

Usage

```
getME(object,
      name = c("X", "Z", "Zt", "Ztlist", "y", "mu", "u", "b",
               "Gp", "Tp", "L", "Lambda", "Lambdat", "Lind",
               "A", "RX", "RZX", "sigma", "flist",
               "beta", "theta", "ST", "REML", "is_REML",
               "n_rtrms", "n_rfacs", "cnms", "devcomp", "offset", "lower"))
```

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`, 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

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.

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

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

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

REML 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 dispersion parameter), "reTrms" (number of random effects terms), "REML" ("TRUE" if model fit by REML), "GLMM" ("TRUE" if a GLMM), "NLMM" ("TRUE" if an NLMM).

offset model offset

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

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

## A way to get *all* getME()s :
getME.all <- function(obj) {
    sapply(eval(formals(getME)$name), getME, object=obj, simplify=FALSE)
}
## internal consistency check ensuring that all work:
parts <- getME.all(fm1)
str(parts, max=2)
```

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

Details

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

Value

a matrix 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.

Examples

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

---

glmer *Fit Generalized Linear Mixed-Effects Models*

Description

Fit a generalized linear mixed model (GLMM)

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

family	a GLM family, see glm and family.
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.

`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`.

`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.

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

`formula` a two-sided linear formula object describing both the fixed-effects and fixed-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).

`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.

`verbose` integer scalar. If `> 0` verbose output is generated during the optimization of the parameter estimates. If `> 1` verbose output is generated during the individual PIRLS steps.

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

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

`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 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 with a single grouping factor for the random effects is adaptive Gauss-Hermite quadrature. 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.

With vector-valued random effects the complexity of the Gauss-Hermite quadrature formulas increases dramatically with the dimension. For a 3-dimensional vector-valued random effect nAGQ=5 requires 93 evaluations of the GLM deviance per evaluation of the approximate GLMM deviance. For 20-dimensional evaluations of the GLM deviance per evaluation of the approximate GLMM deviance.

The default approximation is the Laplace approximation, corresponding to nAGQ=1.

Value

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

See Also

lmer (for details on formulas and parameterization); glm

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.
(gm1a <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
               cbpp, binomial, nAGQ = 9))

## GLMM with individual-level variability (accounting for overdispersion)
## For this data set the model is the same as one allowing for a period:herd
## interaction, which the plot indicates could be needed.
cbpp$obs <- 1:nrow(cbpp)
(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))))
```

`glmer.nb` *glmer() for Negative Binomial*

Description

glmer() for Negative Binomial

Usage

```
glmer.nb(..., interval = log(th) + c(-3, 3),
  verbose = FALSE)
```

Arguments

`...`	formula, data, etc: the arguments for `glmer(..)` (apart from `family`!).
`interval`	interval in which to start the optimization
`verbose`	logical indicating how much progress information should be printed.

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` *Generator object for the golden search optimizer class.*

Description

The generator objects for the `golden` class of a scalar optimizer for a parameter within an interval. The optimizer uses reverse communications.

Usage

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

Arguments

`...`	additional, optional arguments. None are used at present.

Methods

`new(lower=lower, upper=upper)` Create a new `golden` object.

Note

Arguments to the `new` methods must be named arguments. `lower` and `upper` are the bounds for the scalar parameter; they must be finite.

See Also

`golden`

golden-class *Class* "golden"

Description

A reference class for a golden search scalar optimizer using reverse communication.

Extends

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

Examples

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

GQdk *Sparse Gaussian Quadrature grid*

Description

Generate the sparse multidimensional Gaussian quadrature grids

Usage

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

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 multiplied by the

Value

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

Note

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)
```

GQN *Sparse Gauss-Hermite quadrature grids*

Description

GQN contains 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. These are stored in a list of lists. 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.

Format

A list of lists.

Note

These are 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. The function GQdk reproduces the entire array of nodes.

See Also

GQdk

Examples

```
GQN[[3]][[5]]
```

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. http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=82701.

Examples

```
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))
```

`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)))
```

---

lmer *Fit Linear Mixed-Effects Models*

Description

Fit a linear mixed-effects model (LMM) to data.

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.
data	an optional data frame containing the variables named in formula. By default the variables are taken from the environment from which lmer is called. While data is optional, the package authors *strongly* recommend its use, especially when later applying methods such as update and drop1 to the fitted model (*such methods are not guaranteed to work properly if* data *is omitted*). If data is omitted, variables will be taken from the environment of formula (if specified as a formula) or from the parent frame (if specified as a character vector).
REML	logical scalar - Should the estimates be chosen to optimize the REML criterion (as opposed to the log-likelihood)?
control	a list (of correct class, resulting from lmerControl() or glmerControl() respectively) containing control parameters, including the nonlinear optimizer to be used and parameters to be passed through to the nonlinear optimizer, see the *lmerControl documentation for details.
start	a named list of starting values for the parameters in the model. For lmer this can be a numeric vector or a list with one component named "theta".
verbose	integer scalar. If > 0 verbose output is generated during the optimization of the parameter estimates. If > 1 verbose output is generated during the individual PIRLS steps.
subset	an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector. Prior weights are *not* normalized or standardized in any way. In particular, the diagonal of the residual covariance matrix is the squared residual standard deviation parameter sigma times the vector of inverse weights. Therefore, if the weights have relatively large magnitudes, then in order to compensate, the sigma parameter will also need to have a relatively large magnitude.
na.action	a function that indicates what should happen when the data contain NAs. The default action (na.omit, inherited from the 'factory fresh' value of getOption("na.action")) strips any observations with any missing values in any variables.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.

devFunOnly — logical - return only the deviance evaluation function. Note that because the deviance function operates on variables stored in its environment, it may not return *exactly* the same values on subsequent calls (but the results should always be within machine tolerance).

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

Details

- If the formula argument is specified as a character vector, the function will attempt to coerce it to a formula. However, this is not recommended (users who want to construct formulas by pasting together components are advised to use as.formula or reformulate); model fits will work but subsequent methods such as drop1, update may fail.
- Unlike some simpler modeling frameworks such as lm and glm which automatically detect perfectly collinear predictor variables, [gn]lmer cannot handle design matrices of less than full rank. For example, in cases of models with interactions that have unobserved combinations of levels, it is up to the user to define a new variable (for example creating ab within the data from the results of interaction(a,b,drop=TRUE)).
- 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, for which many methods are available (e.g. methods(class="merMod"))

See Also

lm

Examples

```
## linear mixed models - reference values from older code
(fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy))
summary(fm1)# (with its own print method)

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.

fm1_ML <- update(fm1,REML=FALSE)
(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

(vv <- vcov.merMod(fm2, corr=TRUE))
as(vv, "corMatrix")# extracts the ("hidden") 'correlation' entry in @factors

## 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 = "bobyqa",
    restart_edge = TRUE,
    boundary.tol = 1e-5,
    calc.derivs=TRUE,
    use.last.params=FALSE,
    sparseX = FALSE,
    ## input checking options
    check.nobs.vs.rankZ = "warningSmall",
    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  = "warning",
    check.formula.LHS = "stop",
```

```
    ## convergence checking options
    check.conv.grad     = .makeCC("warning", tol = 2e-3, relTol = NULL),
    check.conv.singular = .makeCC(action = "ignore",  tol = 1e-4),
    check.conv.hess     = .makeCC(action = "warning", tol = 1e-6),
    ## optimizer args
    optCtrl = list())

glmerControl(optimizer = c("bobyqa", "Nelder_Mead"),
    restart_edge = FALSE,
    boundary.tol = 1e-5,
    calc.derivs=TRUE,
    use.last.params=FALSE,
    sparseX = FALSE,
    tolPwrss=1e-7,
    compDev=TRUE,
    ## input checking options
    check.nobs.vs.rankZ = "warningSmall",
    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  = "warning",
    check.formula.LHS = "stop",
    ## convergence checking options
    check.conv.grad     = .makeCC("warning", tol = 1e-3, relTol = NULL),
    check.conv.singular = .makeCC(action = "ignore",  tol = 1e-4),
    check.conv.hess     = .makeCC(action = "warning", tol = 1e-6),
    ## optimizer args
    optCtrl = list())

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

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

Arguments

`optimizer` character - name of optimizing function(s). A character vector or list of functions: length 1 for `lmer` or `glmer`, possibly length 2 for `glmer`). The built-in optimizers are `Nelder_Mead` and `bobyqa` (from the **minqa** package). Any minimizing function that allows box constraints can be used provided that it

(1) takes input parameters `fn` (function to be optimized), `par` (starting parameter values), `lower` (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, equal to zero for successful con-

vergence) and (optionally) message (informational message, or explanation of convergence failure).

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

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

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

use.last.params
logical - should the last value of the parameters evaluated (TRUE), rather than the value of the parameters corresponding to the minimum deviance, be returned? This is a "backward bug-compatibility" option; use TRUE only when trying to match previous results.

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

restart_edge logical - should the optimizer attempt a restart when it finds a solution at the boundary (i.e. zero random-effect variances or perfect +/-1 correlations)? (Currently only implemented for lmerControl.)

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

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

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

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.

`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

`optCtrl`
: a `list` of additional arguments to be passed to the nonlinear optimizer (see `Nelder_Mead`, `bobyqa`). In particular, both `Nelder_Mead` and `bobyqa` use `maxfun` to specify the maximum number of function evaluations they will try before giving up - in contrast to `optim` and `optimx`-wrapped optimizers, which use `maxit`.

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

`tol`
: numeric - tolerance for check

`relTol`
: numeric - tolerance for checking relative variation

`...`
: other elements to include in check specification

Details

Note that (only!) the "checking options", i.e., all those starting with `"check."`, may also be set 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).

Examples

```
str(lmerControl())
str(glmerControl())
## Not run:
    ## fit with default Nelder-Mead algorithm ...
    fm0 <- lmer(Reaction ~ Days + (1 | Subject), sleepstudy)
    fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
    ## or with minqa::bobyqa ...
    fm1_bobyqa <- update(fm1,control=lmerControl(optimizer="bobyqa"))
    ## or with the nlminb function used in older (<1.0) versions of lme4;
    ## this will usually replicate older results
    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 from the nloptr package.
```

```
    ## You must run library(nloptr) here ... it is commented out
    ## to avoid making lme4 dependent on nloptr
    defaultControl <- list(algorithm="NLOPT_LN_BOBYQA",
                           xtol_rel=1e-6,maxeval=1e5)
    nloptwrap2 <- function(fn,par,lower,upper,control=list(),...) {
        for (n in names(defaultControl))
            if (is.null(control[[n]])) control[[n]] <- defaultControl[[n]]
        res <- nloptr(x0=par,eval_f=fn,lb=lower,ub=upper,opts=control,...)
        with(res,list(par=solution,
                      fval=objective,
                      feval=iterations,
                      conv=if (status>0) 0 else status,
                      message=message))
    }
    fm1_nloptr <- update(fm1,control=lmerControl(optimizer="nloptwrap2"))
    fm1_nloptr_NM <- update(fm1,control=lmerControl(optimizer="nloptwrap2",
                                optCtrl=list(algorithm="NLOPT_LN_NELDERMEAD")))
    ## other algorithm options include NLOPT_LN_COBYLA, NLOPT_LN_SBPLX

## End(Not run)
```

`lmList` *List of lm Objects with a Common Model*

Description

List of lm Objects with a Common Model

Usage

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

Arguments

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

Details

- data should be a data frame (not, e.g. a groupedData object from the **nlme** package); use as.data.frame first to convert the data.
- 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).

---

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

Description

Class "lmList" is an S4 class with basically a list of objects of class lm with a common model.

Objects from the Class

Objects can be created by calls of the form new("lmList", ...) or, more commonly, by a call to 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 *Classes* "lmResp", "glmResp", "nlsResp" *and* "lmerResp"

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'
terms(x, fixed.only = TRUE, ...)
## S3 method for class 'merMod'
vcov(object, correlation = TRUE, sigm = sigma(object),
use.hessian = NULL, ...)

## S3 method for class 'merMod'
print(x, digits = max(3, getOption("digits") - 3),
      correlation = NULL, symbolic.cor = FALSE,
     signif.stars = getOption("show.signif.stars"), ranef.comp = "Std.Dev.", ...)

## 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."), show.resids = 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>).
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.
fixed.only	logical indicating if only the fixed effects terms are sought, defaults to true. If false, all terms, including random ones are returned.

correlation	(logical) for vcov, indicates whether the correlation matrix as well as the variance-covariance matrix is desired; for print.summary.merMod, indicates whether the correlation matrix of the fixed-effects parameters should be printed.
use.hessian	(logical) indicates whether to use the finite-difference Hessian of the deviance function to compute standard errors of the fixed effects, rather estimating based on internal information about the inverse of the model matrix (see getME(.,"RX")). The default is to to use the Hessian whenever the fixed effect parameters are arguments to the deviance function (i.e. for GLMMs with nAGQ>0), and to use getME(.,"RX") whenever the fixed effect parameters are profiled out (i.e. for GLMMs with nAGQ==0 or LMMs). use.hessian=FALSE is backward-compatible with older versions of lme4, but may give less accurate SE estimates when the estimates of the fixed-effect (see getME(.,"beta")) and random-effect (see getME(.,"theta")) parameters are correlated.
sigm	the residual standard error; by default sigma(object).
digits	number of significant digits for printing
symbolic.cor	should a symbolic encoding of the fixed-effects correlation matrix be printed? If so, the symnum function is used.
signif.stars	(logical) should significance stars be used?
ranef.comp	character vector of length one or two, indicating if random-effects parameters should be reported on the variance and/or standard deviation scale.
show.resids	should the quantiles of the scaled residuals be printed?
type	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.
...	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):

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

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

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

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 LM if fitted with REML = TRUE, this can be controlled via the refit argument. See also anova.

See Also

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

Examples

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

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

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

Description

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

Usage

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

Arguments

...	List of arguments (see Note).

Methods

new(X, Zt, Lambdat, Lind, theta, n): Create a new `merPredD` object

Note

Arguments to the `new` methods must be named arguments:

- 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

`merPredD`

merPredD-class *Class* "merPredD" *- a dense predictor reference class*

Description

A reference class 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.

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 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")
str(slot(lmer(Yield ~ 1|Batch, Dyestuff), "pp"))
```

mkdevfun *Create a deviance evaluation function from a predictor and a response module*

Description

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

Usage

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

Arguments

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

Details

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

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

Value

A function of one numeric argument.

See Also

lmer, glmer and nlmer

Examples

```
(dd <- lmer(Yield ~ 1|Batch, Dyestuff, devFunOnly=TRUE))
dd(0.8)
minqa::bobyqa(1, dd, 0)
```

---

mkMerMod *Create a merMod object*

Description

Create an object in a subclass of 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 value returned by the optimizer
reTrms	reTrms list from the calling function
fr	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 *Create Z, Lambda, Lind, etc.*

Description

From the result of findbars applied to a model formula and and the evaluation frame, create the model matrix, etc. associated with random-effects terms. See the description of the returned value for a detailed list.

Usage

```
mkReTrms(bars, fr)
```

Arguments

bars	a list of parsed random-effects terms
fr	a model frame in which to evaluate these terms

Value

a list with components

Zt	transpose of the sparse model matrix for the random effects
Lambdat	transpose of the sparse relative covariance factor
Lind	an integer vector of indices determining the mapping of the elements of the theta to the "x" slot of Lambdat
theta	initial values of the covariance parameters
lower	lower bounds on the covariance parameters
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

See Also

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

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,
    mustart, etastart, control = glmerControl(), ...)

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

optimizeGlmer(devfun, optimizer = "bobyqa",
    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 fixed-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).
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)
verbose	print output?
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)
names(glmod)
## 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$reTrms))
## 5.  Optimize over theta and beta:
opt <- optimizeGlmer(devfun, stage=2)
opt[1:3]
## 6.  Package up the results:
mkMerMod(environment(devfun), opt, glmod$reTrms, fr = glmod$fr)

### Same model in one line
glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
      data = cbpp, family = binomial)
```

`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*

Description

Number of levels of one or more grouping factor.

Usage

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

Arguments

object	An R object.
...	Currently ignored.

Details

Currently there are methods for merMod and factor objects.

Value

Number(s) of levels

Examples

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

`nlformula` *Manipulate a nonlinear model formula.*

Description

Check and manipulate the formula for a nonlinear model.

Usage

```
nlformula(mc)
```

Arguments

`mc`	matched call from the calling function. 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 *Fit Nonlinear Mixed-Effects Models*

Description

Fit a nonlinear mixed-effects model

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 `SSbiexp`, `SSlogis`, etc, see `selfStart` functions provide this (and more). Alternatively, you can use `deriv()` to automatically produce such functions or expressions.
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
...	other potential arguments. A `method` argument was used in earlier versions of the package. Its functionality has been replaced by the `nAGQ` argument.
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.
verbose	integer scalar. If `> 0` verbose output is generated during the optimization of the parameter estimates. If `> 1` verbose output is generated during the individual PIRLS steps.

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

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

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

na.action — a function that indicates what should happen when the data contain NAs. The 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.

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 derivs():
##    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)
```

---

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, ...)
```

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.
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).

Author(s)

original version in nlme package by Jose Pinheiro and Douglas Bates

Examples

```
data(Orthodont,package="nlme")
fm1 <- lmer(distance ~ age + (age|Subject), data=Orthodont)
## standardized residuals versus fitted values by gender
plot(fm1, resid(., scaled=TRUE) ~ fitted(.) | Sex, abline = 0)
## box-plots of residuals by Subject
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)
if (require(ggplot2)) {
    ## we can create the same plots using ggplot2 and the fortify() function
    fm1F <- fortify(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)
    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)), 1)),
       conf = c(50, 80, 90, 95, 99)/100,
       absVal = FALSE, which=1:nptot, ...)

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

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

Arguments

x	a mixed-effects profile, i.e., of class "thpr", typically resulting from profile(fm) where fm is a fitted model from lmer (or its generalizations).
data	unused - only for compatibility with generic.
levels	the contour levels to be shown; usually derived from conf.
conf	numeric vector of confidence levels to be shown as contours.
absVal	logical indicating if abs(.)olute values should be plotted, often preferred for confidence interval visualization.
which	integer or character vector indicating which parameters to profile: default is all parameters (see profile-methods for details).
...	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

predict method for merMod objects

Usage

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

Arguments

object	a fitted model object
newdata	data frame for which to evaluate predictions
newparams	new parameters to use in evaluating predictions, specified as in the start parameter for lmer or glmer – a list with components theta and/or (for GLMMs) beta
newX	new design matrix to use in evaluating predictions (alternative to newdata)
re.form	formula for random effects to condition on. If NULL, include all random effects; if NA or ~0, include no random effects
ReForm	allowed for backward compatibility: re.form is now the preferred argument name
REForm	allowed for backward compatibility: re.form is now the preferred argument name
REform	allowed for backward compatibility: re.form is now the preferred argument name
terms	a terms object - not used at present
type	character string - either "link", the default, or "response" indicating the type of prediction object returned
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	function determining what should be done with missing values for fixed effects in newdata. The default is to predict NA: see na.pass.
...	optional additional parameters. None are used at present.

Value

a numeric vector of predicted values

Note

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.

Examples

```
(gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 |herd), cbpp, binomial))
str(p0 <- predict(gm1))            # fitted values
str(p1 <- predict(gm1,ReFORM=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,ReFORM=NA)) # new data, level-0
str(p4 <- predict(gm1,newdata,ReFORM=~(1|herd))) # explicitly specify RE
```

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

Description

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

The `log()` method transforms a lmer profile to the scale of the logarithm of the standard deviation of the random effects, i.e. it returns a profile with all the .sigNN parameters replaced by .lsigNN. The forward and backward splines for these parameters are recalculated.

Usage

```
## S3 method for class 'merMod'
profile(fitted, which = 1:nptot, alphamax = 0.01,
maxpts = 100, delta = cutoff/8, verbose = 0, devtol = 1e-09,
        maxmult = 10, startmethod = "prev", optimizer = "bobyqa",
signames = TRUE, ...)
## S3 method for class 'thpr'
 as.data.frame(x, ...)
## S3 method for class 'thpr'
 log(x, base = exp(1))
```

Arguments

`fitted`	a fitted model, e.g., the result of `lmer(..)`.
`which`	integer or character vector indicating which parameters to profile: default 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. In addition, `which` may be a character, containing `"beta_"` or `"theta_"` which means the fixed and 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.
verbose	level of output from internal calculations.
devtol	tolerance for fitted deviances less than baseline (supposedly minimum) deviance.
maxmult	maximum multiplier of the original step size allowed, defaults to 10.
startmethod	method for picking starting conditions for optimization (STUB).
optimizer	(character or function) optimizer to use (see `lmer` for details).
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., `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.

Details

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

Value

`profile(<merMod>)` returns an object of S3 class `"thpr"`, `data.frame`-like. Methods for such a profile object are notably `confint()`, the three plots methods `xyplot`, `densityplot`, and `splom`.

Further, `log()` (see above) and `as.data.frame()`.

Methods

signature(fitted = \"merMod\") ...

See Also

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
## full profiling (default which means 'all) needs
## ~2.6s (on a 2010 Macbook Pro)
system.time( tpr  <- profile(fm01ML))
```

```
## ~1s, + possible warning about bobyqa convergence
(confint(tpr) -> CIpr)

stopifnot(all.equal(CIpr,
  array(c(12.1985292, 38.2299848, 1486.4515,
          84.0630513, 67.6576964, 1568.54849), dim = 3:2,
        dimnames = list(c(".sig01", ".sigma", "(Intercept)"),
                        c("2.5 %", "97.5 %"))),
                  tol= 1e-07))# 1.37e-9 {64b}

require(lattice)
xyplot(tpr)
xyplot(tpr, absVal=TRUE) # easier to see conf.int.s (and check symmetry)
densityplot(tpr, main="densityplot( profile(lmer(..)) )")
splom(tpr)
doMore <- lme4:::testLevel() > 1
if(doMore) { ## not typically, for time constraint reasons
 ## Batch and residual variance only
 system.time(tpr2 <- profile(fm01ML, which=1:2, optimizer="Nelder_Mead"))
 print( xyplot(tpr2) )
 print( xyplot(log(tpr2)) )# log(sigma) is better

 ## GLMM example
 gm1 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
     data = cbpp, family = binomial)
 ## running time ~9 seconds on a modern machine:
 print( system.time(pr4 <- profile(gm1)) )
 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) )
}
```

`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 (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 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.

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

---

ranef *Extract the modes of the random effects*

Description

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

Usage

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

Arguments

object	an object of a class of fitted models with random effects, typically a merMod object.
condVar	an optional logical argument indicating if the conditional variance-covariance matrices of the random effects should be added as an attribute.

drop	should components of the return value that would be data frames with a single column, usually a column called '(Intercept)', be returned as named vectors instead?
whichel	character vector of names of grouping factors for which the random effects should be returned.
postVar	a (deprecated) synonym for condVar
x	a random-effects object (of class ranef.mer) produced by ranef
main	include a main title, indicating the grouping factor, on each sub-plot?
data	This argument is required by the dotplot and qqmath generic methods, but is not actually used.
...	some methods for these generic functions require additional arguments.

Details

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

Value

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

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

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

Note

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

Examples

```
require(lattice)
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, condVar = TRUE))
dotplot(rr1)  ## default
## specify free scales in order to make Day effects more visible
dotplot(rr1,scales = list(x = list(relation = 'free')))[["Subject"]]
if(FALSE) { ##-- condVar=TRUE is not yet implemented for multiple terms -- FIXME
str(ranef(fm2, condVar = TRUE))
}
op <- options(digits = 4)
ranef(fm3, drop = TRUE)
options(op)
```

`refit` *Refit a model with a new response, by maximum likelihood criterion*

Description

Refit a model with a different response vector

Usage

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

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

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. Must be of the same length as the original response.
`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`?
`...`	optional additional parameters. None are used at present.

Details

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

Setting `rename.response` to `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.

Value

an object like x, but fit by maximum likelihood

Examples

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

---

refitML *Refit a model by maximum likelihood criterion*

Description

Refit a 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

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 error*

Description

Extract the residual standard error from a fitted model.

Usage

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

Arguments

object	a fitted model.
...	additional, optional arguments. (None are used in the merMod method)

Details

This is a generic function. At present the only methods are for mixed-effects models of class merMod.

Value

the residual standard error as a scalar

simulate.merMod *Simulate responses from a merMod object*

Description

Simulate responses from the model represented by a "merMod" fitted model object.

Usage

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

Arguments

object	(for simulate.merMod) a fitted model object or (for simulate.formula) a (one-sided) mixed model formula, as described for lmer
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, include all random effects; if NA or ~0, include no random effects.
ReForm	allowed for backward compatibility: re.form is now the preferred argument name
REForm	allowed for backward compatibility: re.form is now the preferred argument name
REform	allowed for backward compatibility: re.form is now the preferred argument name
newdata	data frame for which to evaluate predictions
newparams	new parameters to use in evaluating predictions, specified as in the start parameter for lmer or glmer – a list with components theta and beta and (for LMMs or GLMMs that estimate a scale parameter) sigma
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
...	optional additional arguments: none are used at present

Details

- ordinarily simulate is used to generate new values from an existing, fitted model (merMod object): however, if formula, newdata, and newparams are specified, simulate generates the appropriate model structure to simulate from

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$incidence==0),col=2)
##
## simulate from a non-fitted model; in this case we are just
## replicating the previous model, but
params <- list(theta=0.5,beta=c(2,-1,-2,-3))
simdat <- with(cbpp,expand.grid(herd=levels(herd),period=factor(1:4)))
simdat$size <- 15
simdat$incidence <- sample(0:1,size=nrow(simdat),replace=TRUE)
form <- formula(gm1)[-2]
simulate(form,newdata=simdat,family=binomial,
    newparams=params)
```

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

Description

The average reaction time per day for subjects in a sleep deprivation study. On day 0 the subjects had their normal amount of sleep. Starting that night they were restricted to 3 hours of sleep per night. The observations represent the average reaction time on a series of tests given each day to each subject.

Format

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

`Reaction` Average reaction time (ms)

`Days` Number of days of sleep deprivation

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

Details

These data are from the study described in Belenky et al. (2003), for the sleep-deprived group and for the first 10 days of the study, up to the recovery period.

References

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

---

subbars *"Sub[stitute] Bars"*

Description

Substitute the '+' function for the '|' function in a mixed-model formula. 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 +

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`, `nobars`

Examples

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

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, rdig=3)
## S3 method for class 'VarCorr.merMod'
 as.data.frame(x, row.names = NULL, optional =
FALSE, ...)
```

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.
rdig	an optional integer value specifying the number of digits used to represent correlation estimates.
row.names	Ignored: necessary for the as.data.frame method.
optional	Ignored: necessary for the as.data.frame method.
...	Ignored: necessary for the as.data.frame method.

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)
(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"))
as.data.frame(vc)
```

---

varianceProf *Transform to the variance scale*

Description

Transform a mixed-effects profile to the variance scale

Usage

```
varianceProf(pr)
```

Arguments

pr	a mixed-effects model profile

Value

a transformed mixed-effects model profile

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, *Explanatory Item Response Models* (Springer, 2004) to illustrate various forms of item response models.

Format

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

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

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

`item` the item on the 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

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

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)
}
                       ## much faster but less accurate
print(fmVA0 <- glmer(r2 ~ (Anger + Gender + btype + situ)^2 +
                    (1|id) + (1|item), family = binomial, data =
                    VerbAgg, nAGQ=0L), corr=FALSE)
```

Index