Syntax for using lme in R
Stat 8311, Fall 2002

Sanford Weisberg

November 20, 2002

The best source for using lme in R is José Pinheiro and Douglas Bates (2000), *Mixed-Effects Models in S and S-Plus*, New York: Springer. This note summarizes the syntax that is used with this program. More information is available in the on-line help: enter the following commands:

```
> library(nlme) # load the library
> help.start()  # start the help system in a web browser
```

Then, select the link for "packages" and then the link for "nlme". This will give you a listing of all the items included in the nlme package. Some of the help pages for important methods are included at the end of this document.

The package uses Trellis graphics, and to get these in R, you may need to enter the two commands library(lattice) and library(grid).

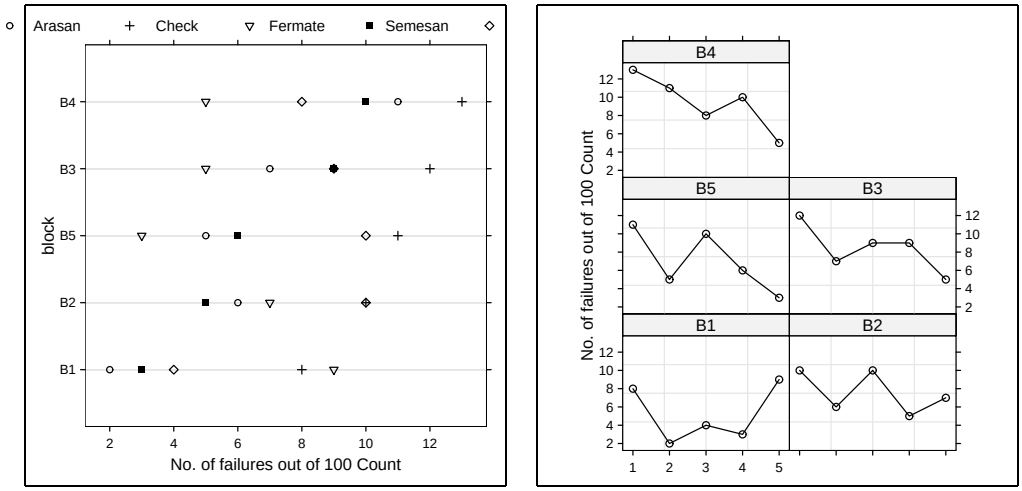
1 Grouped-data objects

Many methods in R, such as fitting linear models with lm, are easier to use if you collect your data into a data frame. A data frame usually consists of columns of data, all the the same length, with a label for each column and possibly a label for each row. Data objects in lme can benefit from a bit more structure than is available from a data frame, and so there is a class of objects called groupedData objects. A groupedData object consists of: (1) data collected into a data frame; (2) a display formula that is used to draw a default plot of the data; (3) labels and units that are used to annotate graphs, and (4) a few other arguments that will be discussed shortly.

For example, Snedecor and Cochran (1980, 7th Ed), p. 256, report a randomized block design that compares five treatments for soybean seeds in each of five blocks. The response is the number of seeds out of 100 that fail to germinate. We can create the grouped data item as follows:

```
> block <- rep(paste("B",1:5,sep=""),rep(5,5))
> trt <- rep(c("Check","Arasan","Spergon","Semesan","Fermate"),5)
> y <- c(8,2,4,3,9,10,6,10,5,7,12,7,9,9,5,13,11,8,10,5,11,5,10,6,3)
> sc256 <- groupedData(y~trt|block,
+       data=data.frame(block=block,trt=trt,y=y),
+       labels=list(y="No. of failures out of 100"),
+       units=list(y="Count"))
> plot(sc256)
```

The display formula generally has three pieces: a response variable, to the left of the ~, in this case equal to y; predictor to the right of ~, which will often be the name of a single variable or factor. If this is an



a. Trt is a factor. b. Trt is continuous.

Figure 1: Default plots for groupedData objects.

equation, like $M \sim L + W + D$, the the right hand side is converted to a variable by adding $L + W + D$. The third piece is to the right of the vertical bar "|" and is the conditioning variable: a separate graph is drawn for each level of this variable. In this case, a separate plot is drawn for each block, and within each block we get a dot plot, with a different symbol for each treatment. If the variable trt had been continuous, we would get a scatterplot for each block:

```
> block <- rep(paste("B",1:5,sep=""),rep(5,5))
> faketrt <- rep(1:5,5)
> y <- c(8,2,4,3,9,10,6,10,5,7,12,7,9,9,5,13,11,8,10,5,11,5,10,6,3)
> fakesc256 <- groupedData(y~trt|block,
+         data=data.frame(block=block,trt=faketrt,y=y),
+         labels=list(y="No. of failures out of 100"),
+         units=list(y="Count"))
> plot(fakesc256)
```

Three additional arguments to groupedData further control the plotting for more complex models. *This arguments to do effect the fitting of models.* If you have a variable in the data frame that is constant within levels of the grouping variable, it should be declared as an outer factor. For example, if the blocks B1, B2 and B3 were in one location and blocks B4 and B5 were in a second location, we might have a variable location. Including outer = ~ location in the call to groupedData will guarantee that the blocks are grouped by location in the graph. You can also put the outer argument in the plot command, so

```
plot(Soybean, outer= ~ Year*Variety)
```

will plot a groupedData object called Soybean arranging plots so that each Variety appears in the same row. The call

```
plot(Soybean, outer= ~ Variety*Year)
```

rearranges the plot so all the plots in a row are for the same year.

There are too many options to groupedData to be described here. Chapter 2 of Pinheiro and Bates gives a much more complete description.

In a sense, creating groupedData objects is *optional*. You can run lme using a data frame. All that you lose is the default plotting methods.

1.1 Factors and variables

In the randomized block example given above, the variable y consists of numbers, and so y is assumed to be a variable. In contrast, both block and trt are text variables, and so by default these are assumed to be *unordered factors*. Unless you change it, the default parameterization for a factor with t levels is based on the identity matrix I_t. If an intercept is included in the model, then the basis is obtained by deleting the first column from I_t.

If a numeric variable actually gives the levels of a factor, then you must explicitly declare that the variable is a factor. We can do this for the fake data as follows:

```
> y <- c(8,2,4,3,9,10,6,10,5,7,12,7,9,9,5,13,11,8,10,5,11,5,10,6,3)
> fakeblock <- rep(1:5,rep(5,5))  # numeric blocks
> faketrt <- rep(1:5,5)  # numeric treatments
> fake.frame <- data.frame(block=fakeblock,trt=faketrt,y=y)
> fake.frame$block <- factor(fake.frame$block, ordered=FALSE)
> fake.frame$trt <- factor(fake.frame$trt, ordered=TRUE)
> fakesc256 <- groupedData(y~trt|block,
+         data=fake.frame,
+         labels=list(y="No. of failures out of 100"),
+         units=list(y="Count"))
```

In the above, the data frame is explicitly created, and then the fake blocking and treatment variables are converted to factors. The blocking variable is made into an unordered factor, while the treatment variable is declared to be ordered, so it will be defined using orthogonal polynomials. You can also set the way that factors are defined; see the attached documentation for factor.

2 The call to lme

The primary function needed for fitting linear mixed effects models is lme. The required items in the call to this function are: (1) a formula for the fixed effects; (2) a formula for the random effects; (3) the name of a data frame or groupedData object to locate the data to use; and (4) either method="ml" for maximum likelihood estimation or method="reml", for restricted maximum likelihood estimation, with the latter the default if neither is specified. Additional arguments are also included, and are given in the function documentation. The models that can be fit in lme assume that there is a grouping variable, that the various levels of the grouping variable are independent, and within the ith level of the grouping variable the model is

$$Y_i = X_i\beta + Z_i u_i + \varepsilon_i, i = 1, \ldots, M \tag{1}$$

where M is the number of levels of the grouping variable, Y_i is the $m_i \times 1$ response for level i, X_i is an $m_i \times p$ matrix that describes the fixed effects for level i, β is a vector of parameters that is common to all levels of the grouping variable, Z_i is $m_i \times q$ full rank matrix that describes the design for the random effects for subject i, $u_i \sim \mathrm{N}(0, \Psi_i)$, $\varepsilon_i \sim \mathrm{N}(0, \sigma^2 I)$ and $u_i \perp\!\!\!\perp \varepsilon_i$. In many problems, the design is the same for each level of the grouping variable, but this is not required. Also, multiple levels of grouping are also possible, leading to more complex models that can be solved using lme.

2.1 Specifying the fixed effects

The fixed effects are specified with a formula that is similar to the formulas used in lm or in glm. The formula is of the form

$$\text{Left-side} \sim \text{right-side}$$

The left-side is the *response variable*. In the randomized block experiment described above, the response is y, but you can also use constructions like $\log(y)$ or $1/y$ or even more complicated forms. The left-side specifies the matrix X_i in (1), so it is the model for fixed-effects *within a level of the grouping variable*. For the randomized block experiment, the fixed effect formula is either y ~ trt or y ~ -1 + trt. The first of these parameterizes $X = (J_5, e_2, e_3, e_4, e_5)$ to have a grand mean plus the factor trt with in this case $5 - 1 = 4$ df. There are several ways of parameterizing the 4 df for treatments. In the second case, the "-1" explicitly excludes the intercept, and so $X = I_5$.

The quantities on the right side of the formula can consist a set of terms combined with "$+$" signs. The terms can be variables, functions of variables like $\log(x)$, or factors. Thus, y~x1+x2+x3 defines the X matrix to consist of an intercept and the three columns given by x_1, x_2, x_3. If x_3 were a factor, then there would be more than three columns, corresponding now to main effects of x_1, x_2 and x_3, as well as the grand mean. Interactions are indicated with a colon, and so Variety:Trt is the two-factor interaction of variety and trt. The construction Variety*Trt expands to Variety+Trt+Variety:Trt. There are additional short-hand constructions using powers for polynomials, parentheses for multiplying, and others, but these are not described here at length.

2.1.1 One-factor design

In a problem with a single random effect and the usual model $y_{ij} = \beta_0 + u_i + \varepsilon_{ij}$, the fixed effects formula is y ~ 1.

2.1.2 Randomized block design

Six randomly selected workers (blocks) operate each of three machines (treatment), and measure productivity once. The fixed-effects formula is y ~ machine.

2.1.3 Replicated block design

Six randomly selected workers (blocks) operate each of three machines (treatment), and measure productivity three times. The fixed-effects formula is unchanged from the last example, and is still y ~ machine. The random effects formula will be different.

2.1.4 Analysis of covariance

Dental measurements are taken on a random sample of 30 children every year from age 5 to age 10. Some children are boys, and some are girls. The random and grouping effect is the child. Within a child we have repeated measures over time. Sex is a covariate. The within-child fixed formula is y ~ time + sex, assuming a linear effect of time and an additive difference between boys and girls. More generally, we might need to fit y ~ g(time) + sex + sex:g(time), where the function of time might be a polynomial, or some other function.

2.1.5 Nested classification

A random sample of 10 dogs is studied. Each dog is injected with a dye, and concentration of the dye is determined at several time points at two locations on the dog. Thus, dog is the grouping factor, with

locations nested within dog. Time, or a function of time, is the repeated measure. The fixed effects formula for the response within locations nested within dogs is y ~ g(time). All the other effects appear in the random formula.

2.1.6 Split-plot experiment

We have six randomized blocks, each with three whole plots. Varieties of a crop are assigned to the whole plots. Each whole plot is subdivided into four subplots, and levels of nitrogen are assigned to subplots within whole plots. The fixed-effects formula is y ~ Variety * Nitrogen, where Nitrogen is an ordered factor to allow looking at polynomial effects. The specification of the split plot is done in the random effects formula.

2.2 Specifying the random effects

The syntax for specifying the random effects is necessarily complex because there are so many options for the form of a random effect. The general form seems to be:

random = list(effect1 =~ Z1, . . . , effectk =~ Zk)

We illustrate this in the following examples.

2.2.1 One-factor design

In a problem with a single random effect and the usual model $y_{ij} = \beta_0 + u_i + \varepsilon_{ij}$, the random effect is assumed to be distributed as $u \sin \mathrm{N}(0, \sigma_u^2 I)$. There is only one random effect parameter to estimate, and the formula is random = list(group =~ 1).

2.2.2 Randomized block design

Six randomly selected workers (blocks) operate each of three machines (treatment), and measure productivity once. The within block model is As with the last example, there is only one random-effects parameter σ_b^2, and the formula is random = list(block =~ 1).

2.2.3 Replicated block design

Six randomly selected workers (blocks) operate each of three machines (treatment), and measure productivity three times. The simplest treatment of this experiment is to assume that there are random effects between workers and between replications within a machine/worker. This specifies two variance parameters, and is given by random=list(worker=~ 1, replicate = ~ 1). A shorthand for this is random=~1|worker/replicate.

We have an alternative approach. Fit the model

$$y_{ijk} \quad = \quad \beta_j + b_{ij} + \varepsilon_{ijk}$$

and suppose that $\mathbf{b}_i \sim \mathrm{N}(0, \Psi)$, where Ψ is an unknown positive definite matrix. In this case, the Z_i matrix is $J_3 \otimes I_3$, meaning that we are fitting a separate parameter for each machine. The basic specification is random=list(worker=~ machine -1 which assumes that Ψ is an arbitrary positive definite matrix that must be estimated. Special forms for Ψ are available, as listed in Table 1. For example, to fit with compound symmetry, use random=list(worker=pdCompSymm(~ machine -1).

Table 1: Constructors for Ψ in lme models.

`pdSymm`	general positive-definite matrix, with no additional structure
`pdLogChol`	general positive-definite matrix, with no additional structure, using a log-Cholesky parameterization
`pdDiag`	diagonal
`pdIdent`	multiple of an identity
`pdCompSymm`	compound symmetry structure (constant diagonal and constant off-diagonal elements)
`pdBlocked`	block-diagonal matrix, with diagonal blocks of any "atomic" `pdMat` class
`pdNatural`	general positive-definite matrix in natural parameterization (i.e. parameterized in terms of standard deviations and correlations). The underlying coefficients are not unrestricted, so this class should NOT be used for optimization.

2.2.4 Analysis of covariance

Dental measurements are taken on a random sample of 30 children every year from age 5 to age 10. Some children are boys, and some are girls. The random and grouping effect is the child. Within a child we have repeated measures over time. One possible random formula for these data is random=list(Child=pdCompSymm(=~ factor(Age)-1)) to fit equal correlation between measurements within a child, or random=list(Child ~ 1) to fit a simple variance component for each child.

2.2.5 Nested classification

A random sample of 10 dogs is studied. Each dog is injected with a dye, and concentration of the dye is determined at several time points at two locations on the dog. Thus, dog is the grouping factor, with locations nested within dog. Time, or a function of time, is the repeated measure. The simplest formula for the random effects is random=list(dog = ~ 1, side = ~ 1), but more complex models are possible.

2.2.6 Split-plot experiment

We have six randomized blocks, each with three whole plots. Varieties of a crop are assigned to the whole plots. Each whole plot is subdivided into four subplots, and levels of nitrogen are assigned to subplots within whole plots. The random-effects formula is list(Block = ~, Variety = ~ 1), even though Variety is a fixed effects. The syntax assumes that elements in this list are nested within each other, and so this really means fit a term for Block and a term for Variety in block.

groupedData *Construct a groupedData Object*

Description

An object of the groupedData class is constructed from the formula and data by attaching the formula as an attribute of the data, along with any of outer, inner, labels, and units that are given. If order.groups is TRUE the grouping factor is converted to an ordered factor with the ordering determined by FUN. Depending on the number of grouping levels and the type of primary covariate, the returned object will be of one of three classes: nfnGroupedData - numeric covariate, single level of nesting; nffGroupedData - factor covariate, single level of nesting; and nmGroupedData - multiple levels of nesting. Several modeling and plotting functions can use the formula stored with a groupedData object to construct default plots and models.

Usage

```
groupedData(formula, data, order.groups, FUN, outer, inner,
 labels, units)
update(object, formula, data, order.groups, FUN,
outer, inner, labels, units, ...)
```

Arguments

object	an object inheriting from class groupedData.
formula	a formula of the form resp ~ cov | group where resp is the response, cov is the primary covariate, and group is the grouping factor. The expression 1 can be used for the primary covariate when there is no other suitable candidate. Multiple nested grouping factors can be listed separated by the / symbol as in fact1/fact2. In an expression like this the fact2 factor is nested within the fact1 factor.
data	a data frame in which the expressions in formula can be evaluated. The resulting groupedData object will consist of the same data values in the same order but with additional attributes.
order.groups	an optional logical value, or list of logical values, indicating if the grouping factors should be converted to ordered factors according to the function FUN applied to the response from each group. If multiple levels of grouping are present, this argument can be either a single logical value (which will be repeated for all grouping levels) or a list of logical values. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). Ordering within a level of grouping is done within the levels of the grouping factors which are outer to it. Changing the grouping factor to an ordered factor does not affect the ordering of the rows in the data frame but it does affect the order of the panels in a trellis display of the data or models fitted to the data. Defaults to TRUE.

`FUN` an optional summary function that will be applied to the values of the response for each level of the grouping factor, when `order.groups = TRUE`, to determine the ordering. Defaults to the `max` function.

`outer` an optional one-sided formula, or list of one-sided formulas, indicating covariates that are outer to the grouping factor(s). If multiple levels of grouping are present, this argument can be either a single one-sided formula, or a list of one-sided formulas. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. When plotting a groupedData object, the argument `outer = TRUE` causes the panels to be determined by the `outer` formula. The points within the panels are associated by level of the grouping factor. Defaults to `NULL`, meaning that no outer covariates are present.

`inner` an optional one-sided formula, or list of one-sided formulas, indicating covariates that are inner to the grouping factor(s). If multiple levels of grouping are present, this argument can be either a single one-sided formula, or a list of one-sided formulas. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). An inner covariate can change within the sets of rows defined by the grouping factor. An inner formula can be used to associate points in a plot of a groupedData object. Defaults to `NULL`, meaning that no inner covariates are present.

`labels` an optional list of character strings giving labels for the response and the primary covariate. The label for the primary covariate is named `x` and that for the response is named `y`. Either label can be omitted.

`units` an optional list of character strings giving the units for the response and the primary covariate. The units string for the primary covariate is named `x` and that for the response is named `y`. Either units string can be omitted.

`...` some methods for this generic require additional arguments. None are used in this method.

Value

an object of one of the classes `nfnGroupedData`, `nffGroupedData`, or `nmGroupedData`, and also inheriting from classes `groupedData` and `data.frame`.

Author(s)

Douglas Bates and Jose Pinheiro

References

Bates, D.M. and Pinheiro, J.C. (1997), "Software Design for Longitudinal Data", in "Modeling Longitudinal and Spatially Correlated Data: Methods, Applications and Future Directions", T.G. Gregoire (ed.), Springer-Verlag, New York.

Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at http://nlme.stat.wisc.edu/

See Also

formula, gapply, gsummary, lme

Examples

```
data(Orthodont)
Orth.new <-  # create a new copy of the groupedData object
  groupedData( distance ~ age | Subject,
              data = as.data.frame( Orthodont ),
              FUN = mean,
              outer = ~ Sex,
              labels = list( x = "Age",
                y = "Distance from pituitary to pterygomaxillary fissure" ),
              units = list( x = "(yr)", y = "(mm)") )

plot( Orth.new )         # trellis plot by Subject

formula( Orth.new )      # extractor for the formula
gsummary( Orth.new )     # apply summary by Subject
fm1 <- lme( Orth.new )   # fixed and groups formulae extracted from object
Orthodont2 <- update(Orthodont, FUN = mean)
```

---

factor *Factors*

Description

The function factor is used to encode a vector as a factor (the names category and enumerated type are also used for factors). If ordered is TRUE, the factor levels are assumed to be ordered. For compatibility with S there is also a function ordered.

is.factor, is.ordered, as.factor and as.ordered are the membership and coercion functions for these classes.

Usage

```
factor(x, levels = sort(unique(x), na.last = TRUE), labels = levels,
       exclude = NA, ordered = is.ordered(x))
ordered(x, ...)

is.factor(x)
is.ordered(x)

as.factor(x)
as.ordered(x)
```

Arguments

`x`	a vector of data, usually taking a small number of distinct values
`levels`	an optional vector of the values that `x` might have taken. The default is the set of values taken by `x`, sorted into increasing order.
`labels`	*either* an optional vector of labels for the levels (in the same order as `levels` after removing those in `exclude`), *or* a character string of length 1.
`exclude`	a vector of values to be excluded when forming the set of levels. This should be of the same type as `x`, and will be coerced if necessary.
`ordered`	logical flag to determine if the levels should be regarded as ordered (in the order given).
`...`	(in `ordered(.)`): any of the above, apart from `ordered` itself.

Details

The type of the vector `x` is not restricted.

Ordered factors differ from factors only in their class, but methods and the model-fitting functions treat the two classes quite differently.

The encoding of the vector happens as follows. First all the values in `exclude` are removed from `levels`. If `x[i]` equals `levels[j]`, then the `i`-th element of the result is `j`. If no match is found for `x[i]` in `levels`, then the `i`-th element of the result is set to `NA`.

Normally the 'levels' used as an attribute of the result are the reduced set of levels after removing those in `exclude`, but this can be altered by supplying `labels`. This should either be a set of new labels for the levels, or a character string, in which case the levels are that character string with a sequence number appended.

`factor(x)` applied to a factor is a no-operation unless there are unused levels: in that case, a factor with the reduced level set is returned. If `exclude` is used it should also be a factor with the same level set as `x` or a set of codes for the levels to be excluded.

The codes of a factor may contain `NA`. For a numeric `x`, set `exclude=NULL` to make `NA` an extra level (`"NA"`), by default the last level.

If `"NA"` is a level, the way to set a code to be missing is to use `is.na` on the left-hand-side of an assignment. Under those circumstances missing values are printed as `<NA>`.

Value

`factor` returns an object of class `"factor"` which has a set of numeric codes the length of `x` with a `"levels"` attribute of mode `character`. If `ordered` is true (or `ordered` is used) the result has class `c("ordered", "factor")`.

`is.factor` returns `TRUE` or `FALSE` depending on whether its argument is of type factor or not. Correspondingly, `is.ordered` returns `TRUE` when its argument is ordered and `FALSE` otherwise.

`as.factor` coerces its argument to a factor. It is an abbreviated form of `factor`.

`as.ordered(x)` returns `x` if this is ordered, and `ordered(x)` otherwise.

Warning

The interpretation of a factor depends on both the codes and the `"levels"` attribute. Be careful only to compare factors with the same set of levels (in the same order). In particular, `as.numeric` applied to a factor is meaningless, and may happen by implicit coercion.

The levels of a factor are by default sorted, but the sort order may well depend on the locale at the time of creation, and should not be assumed to be ASCII.

See Also

`gl` for construction of "balanced" factors and `C` for factors with specified contrasts. `levels` and `nlevels` for accessing the levels, and `codes` to get integer codes.

Examples

```
ff <- factor(substring("statistics", 1:10, 1:10), levels=letters)
ff
codes(ff)
factor(ff)# drops the levels that do not occur
factor(factor(letters[7:10])[2:3]) # exercise indexing and reduction
factor(letters[1:20], label="letter")

class(ordered(4:1))# "ordered", inheriting from "factor"

## suppose you want "NA" as a level, and to allowing missing values.
(x <- factor(c(1, 2, "NA"), exclude = ""))
is.na(x)[2] <- TRUE
x  # [1] 1    <NA> NA, <NA> used because NA is a level.
is.na(x)
# [1] FALSE  TRUE FALSE
```

---

lme *Linear Mixed-Effects Models*

Description

This generic function fits a linear mixed-effects model in the formulation described in Laird and Ware (1982) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances.

Usage

```
lme(fixed, data, random, correlation, weights, subset, method,
    na.action, control)
update(object, fixed, data, random, correlation, weights,
      subset, method, na.action, control, ...)
```

Arguments

`object`	an object inheriting from class `lme`, representing a fitted linear mixed-effects model.
`fixed`	a two-sided linear formula object describing the fixed-effects part of the model, with the response on the left of a `~` operator and the terms, separated by `+` operators, on the right, an `lmList` object, or a `groupedData` object. The method functions `lme.lmList` and `lme.groupedData` are documented separately.
`data`	an optional data frame containing the variables named in `fixed`, `random`, `correlation`, `weights`, and `subset`. By default the variables are taken from the environment from which `lme` is called.
`random`	optionally, any of the following: (i) a one-sided formula of the form `~x1+...+xn | g1/.../gm`, with `x1+...+xn` specifying the model for the random effects and `g1/.../gm` the grouping structure (`m` may be equal to 1, in which case no `/` is required). The random effects formula will be repeated for all levels of grouping, in the case of multiple levels of grouping; (ii) a list of one-sided formulas of the form `~x1+...+xn | g`, with possibly different random effects models for each grouping level. The order of nesting will be assumed the same as the order of the elements in the list; (iii) a one-sided formula of the form `~x1+...+xn`, or a `pdMat` object with a formula (i.e. a non-`NULL` value for `formula(object)`), or a list of such formulas or `pdMat` objects. In this case, the grouping structure formula will be derived from the data used to fit the linear mixed-effects model, which should inherit from class `groupedData`; (iv) a named list of formulas or `pdMat` objects as in (iii), with the grouping factors as names. The order of nesting will be assumed the same as the order of the order of the elements in the list; (v) an `reStruct` object. See the documentation on `pdClasses` for a description of the available `pdMat` classes. Defaults to a formula consisting of the right hand side of `fixed`.
`correlation`	an optional `corStruct` object describing the within-group correlation structure. See the documentation of `corClasses` for a description of the available `corStruct` classes. Defaults to `NULL`, corresponding to no within-group correlations.
`weights`	an optional `varFunc` object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to `varFixed`, corresponding to fixed variance weights. See the documentation on `varClasses` for a description of the available `varFunc` classes. Defaults to `NULL`, corresponding to homoscedastic within-group errors.
`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.
`method`	a character string. If `"REML"` the model is fit by maximizing the restricted log-likelihood. If `"ML"` the log-likelihood is maximized. Defaults to `"REML"`.
`na.action`	a function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `lme` to print an error message and terminate if there are any incomplete observations.

`control` a list of control values for the estimation algorithm to replace the default values returned by the function `lmeControl`. Defaults to an empty list.

`...` some methods for this generic require additional arguments. None are used in this method.

Value

an object of class `lme` representing the linear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `lmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixed.effects`, and `random.effects` can be used to extract some of its components.

Author(s)

Jose Pinheiro ⟨jose.pinheiro@pharma.novartis.com⟩ and Douglas Bates ⟨bates@stat.wisc.edu⟩

References

The computational methods are described in Bates, D.M. and Pinheiro (1998) and follow on the general framework of Lindstrom, M.J. and Bates, D.M. (1988). The model formulation is described in Laird, N.M. and Ware, J.H. (1982). The variance-covariance parameterizations are described in <Pinheiro, J.C. and Bates., D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littel, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at http://franz.stat.wisc.edu/pub/NLME/

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", Biometrics, 38, 963-974.

Lindstrom, M.J. and Bates, D.M. (1988) "Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data", Journal of the American Statistical Association, 83, 1014-1022.

Littel, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parameterizations for Variance-Covariance Matrices", Statistics and Computing, 6, 289-296.

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

See Also

`lmeControl`, `lme.lmList`, `lme.groupedData`, `lmeObject`, `lmList`, `reStruct`, `reStruct`, `varFunc`, `pdClasses`, `corClasses`, `varClasses`

Examples

```
data(Orthodont)
fm1 <- lme(distance ~ age, data = Orthodont) # random is ~ age
fm2 <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)
summary(fm1)
summary(fm2)
```