Fitting Linear Mixed-Effects Models using lme4

Douglas Bates U. of Wisconsin - Madison
Martin Mächler ETH Zurich
Benjamin M. Bolker McMaster University
Steven C. Walker McMaster University

Abstract

Maximum likelihood or restricted maximum likelihood (REML) estimates of the parameters in linear mixed-effects models can be determined using the `lmer` function in the **lme4** package for R. As for most model-fitting functions in R, the model is described in an `lmer` call by a formula, in this case including both fixed- and random-effects terms. The formula and data together determine a numerical representation of the model from which the profiled deviance or the profiled REML criterion can be evaluated as a function of some of the model parameters. The appropriate criterion is optimized, using one of the constrained optimization functions in R, to provide the parameter estimates. We describe the structure of the model, the steps in evaluating the profiled deviance or REML criterion, and the structure of classes or types that represents such a model. Sufficient detail is included to allow specialization of these structures by users who wish to write functions to fit specialized linear mixed models, such as models incorporating pedigrees or smoothing splines, that are not easily expressible in the formula language used by `lmer`.

Keywords: sparse matrix methods, linear mixed models, penalized least squares, Cholesky decomposition.

Submitted to *Journal of Statistical Software*

1. Introduction

The **lme4** package for R provides functions to fit and analyze linear mixed models, generalized linear mixed models and nonlinear mixed models. In each of these names, the term "mixed" or, more fully, "mixed effects", denotes a model that incorporates both fixed- and random-effects terms in a linear predictor expression from which the conditional mean of the response can be evaluated. In this paper we describe the formulation and representation of linear mixed models. The techniques used for generalized linear and nonlinear mixed models will be described separately, in a future paper.

The development of general software for fitting mixed models remains an active area of research with many open problems. Consequently, the **lme4** package has evolved since it was first released, and continues to improve as we learn more about mixed models. However, we recognize the need to maintain stability and backward compatibility of **lme4** so that it continues to be broadly useful. In order to maintain stability while continuing to advance mixed-model computation, we have developed several additional frameworks that draw on the basic ideas of `lme4` but modify its structure or implementation in various ways. These descendants include the **MixedModels** package in Julia, the **lme4pureR** package in R, and the `flexLambda` development branch of **lme4**. The current article is largely restricted to describ-

ing the current stable version of the **lme4** package (1.1-7), with Appendix A describing hooks into the computational machinery that are designed for extension development. The **gamm4** (Wood and Scheipl 2013) and **blme** (Dorie 2013) packages currently make use of these hooks.

Another goal of this article is to contrast the approach used by **lme4** with previous formulations of mixed models. The expressions for the profiled log-likelihood and profiled REML (restricted maximum likelihood) criteria derived in Section 3.4 are similar to those presented in Bates and DebRoy (2004) and, indeed, are closely related to "Henderson's mixed-model equations" (Henderson 1982). Nonetheless there are subtle but important changes in the formulation of the model and in the structure of the resulting penalized least squares (PLS) problem to be solved (Section 3.6). We derive the current version of the PLS problem (Section 3.2) and contrast this result with earlier formulations (Section 3.5).

This article is organized into four main sections (2; 3; 4; 5), each of which corresponds to one of the four largely separate modules that comprise **lme4**. Before describing the details of each module, we describe the general form of the linear mixed model underlying **lme4** (Section 1.1); introduce the `sleepstudy` data that will be used as an example throughout (Section 1.2); and broadly outline **lme4**'s modular structure (Section 1.3).

1.1. Linear mixed models

Just as a linear model is described by the distribution of a vector-valued random response variable, $\mathcal{Y}$, whose observed value is $\boldsymbol{y}_{\text{obs}}$, a linear mixed model is described by the distribution of two vector-valued random variables: $\mathcal{Y}$, the response, and $\mathcal{B}$, the vector of random effects. In a linear model the distribution of $\mathcal{Y}$ is multivariate normal,

$$\mathcal{Y} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{o}, \sigma^2\boldsymbol{W}^{-1}), \tag{1}$$

where n is the dimension of the response vector, $\boldsymbol{W}$ is a diagonal matrix of known prior weights, $\boldsymbol{\beta}$ is a p-dimensional coefficient vector, $\boldsymbol{X}$ is an $n \times p$ model matrix, and $\boldsymbol{o}$ is a vector of known prior offset terms. The parameters of the model are the coefficients $\boldsymbol{\beta}$ and the scale parameter σ.

In a linear mixed model it is the *conditional* distribution of $\mathcal{Y}$ given $\mathcal{B} = \boldsymbol{b}$ that has such a form,

$$(\mathcal{Y}|\mathcal{B} = \boldsymbol{b}) \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{b} + \boldsymbol{o}, \sigma^2\boldsymbol{W}^{-1}), \tag{2}$$

where $\boldsymbol{Z}$ is the $n \times q$ model matrix for the q-dimensional vector-valued random effects variable, $\mathcal{B}$, whose value we are fixing at $\boldsymbol{b}$. The unconditional distribution of $\mathcal{B}$ is also multivariate normal with mean zero and a parameterized $q \times q$ variance-covariance matrix, $\boldsymbol{\Sigma}$,

$$\mathcal{B} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}). \tag{3}$$

As a variance-covariance matrix, $\boldsymbol{\Sigma}$ must be positive semidefinite. It is convenient to express the model in terms of a *relative covariance factor*, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, which is a $q \times q$ matrix, depending on the *variance-component parameter*, $\boldsymbol{\theta}$, and generating the symmetric $q \times q$ variance-covariance matrix, $\boldsymbol{\Sigma}$, according to

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \sigma^2\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top, \tag{4}$$

where σ is the same scale factor as in the conditional distribution (2).

Although equations 2, 3, and 4 fully describe the class of linear mixed models that **lme4** can fit, this terse description hides many important details. Before moving on to these details, we make a few observations:

- This formulation of linear mixed models allows for a relatively compact expression for the profiled log-likelihood of $\boldsymbol{\theta}$ (Section 3.4, Equation 34).
- The matrices associated with random effects, $\boldsymbol{Z}$ and $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, typically have a sparse structure with a sparsity pattern that encodes various model assumptions. Sections 2.3 and 3.7 provide details on these structures, and how to represent them efficiently.
- The interface provided by **lme4**'s `lmer` function is slightly less general than the model described by equations 2, 3, and 4. To take advantage of the entire range of possibilities, one may use the modular functions (Sections 1.3 and Appendix A) or explore the experimental `flexLambda` branch of **lme4** on Github.

1.2. Example

Throughout our discussion of **lme4**, we will work with a data set on the average reaction time per day for subjects in a sleep deprivation study (Belenky *et al.* 2003). 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 response variable, `Reaction`, represents average reaction times in milliseconds (ms) on a series of tests given each `Day` to each `Subject` (Figure 1),

```
R> str(sleepstudy)

'data.frame': 180 obs. of  3 variables:
 $ Reaction: num  250 259 251 321 357 ...
 $ Days    : num  0 1 2 3 4 5 6 7 8 9 ...
 $ Subject : Factor w/ 18 levels "308","309","310",..: 1 1 1 1 1 1 1..
```

Each subject's reaction time increases approximately linearly with the number of sleep-deprived days. However, subjects also appear to vary in the slopes and intercepts of these relationships, which suggests a model with random slopes and intercepts. As we shall see, such a model may be fitted by minimizing the REML criterion (Equation 39) using

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

The estimates of the standard deviations of the random effects for the intercept and the slope are 24.74 ms and 5.92 ms/day. The fixed-effects coefficients, $\boldsymbol{\beta}$, are 251.4 ms and 10.47 ms/day for the intercept and slope. In this model, one interpretation of these fixed effects is that they are the estimated population mean values of the random intercept and slope (Section 2.2).

1.3. High level modular structure

The `lmer` function is composed of four largely independent modules. In the first module, a mixed-model formula is parsed and converted into the inputs required to specify a linear mixed model (Section 2). The second module uses these inputs to construct an R function which takes the covariance parameters, $\boldsymbol{\theta}$, as arguments and returns negative twice the log profiled likelihood or the REML criterion (Section 3). The third module optimizes this objective

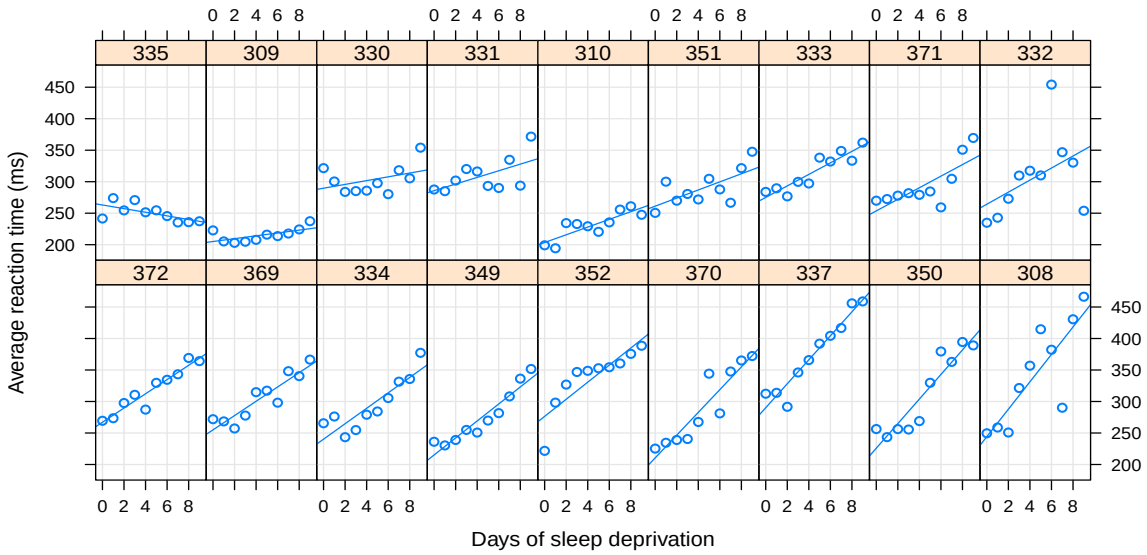


Figure 1: Average reaction time versus days of sleep deprivation by subject. Subjects ordered by increasing slope of subject-specific linear regressions.

function to produce maximum likelihood (ML) or REML estimates of $\boldsymbol{\theta}$ (Section 4). Finally, the fourth module provides utilities for interpreting the optimized model (Section 5).

To illustrate this modularity, we recreate the `fm1` object by a series of four modular steps; the formula module,

```
R> parsedFormula <- lFormula(formula = Reaction ~ Days + (Days | Subject),
+                            data = sleepstudy)
```

the objective function module,

```
R> devianceFunction <- do.call(mkLmerDevfun, parsedFormula)
```

the optimization module,

```
R> optimizerOutput <- optimizeLmer(devianceFunction)
```

and the output module,

```
R> mkMerMod(   rho = environment(devianceFunction),
+              opt = optimizerOutput,
+          reTrms = parsedFormula$reTrms,
+              fr = parsedFormula$fr)
```

Module		R function	Description
Formula module	(Section 2)	`lFormula`	Accepts a mixed-model formula, data, and other user inputs, and returns a list of objects required to fit a linear mixed model.
Objective function module	(Section 3)	`mkLmerDevfun`	Accepts the results of `lFormula` and returns a function to calculate the deviance (or restricted deviance) as a function of the covariance parameters, $\boldsymbol{\theta}$.
Optimization module	(Section 4)	`optimizeLmer`	Accepts a deviance function returned by `mkLmerDevfun` and returns the results of the optimization of that deviance function.
Output module	(Section 5)	`mkMerMod`	Accepts an optimized deviance function and packages the results into a useful object.

Table 1: The high-level modular structure of `lmer`.

2. Formula module

2.1. Mixed-model formulas

Like most model-fitting functions in R, `lmer` takes as its first two arguments a *formula* specifying the model and the *data* with which to evaluate the formula. This second argument, `data`, is optional but recommended and is usually the name of an R data frame. In the R `lm` function for fitting linear models, formulas take the form `resp ~ expr`, where `resp` determines the response variable and `expr` is an expression that specifies the columns of the model matrix. Formulas for the `lmer` function contain special random-effects terms,

```
R> resp ~ FEexpr + (REexpr1 | factor1) + (REexpr2 | factor2) + ...
```

where `FEexpr` is an expression determining the columns of the fixed-effects model matrix, $\boldsymbol{X}$, and the random-effects terms, `(REexpr1 | factor1)` and `(REexpr2 | factor2)`, determine both the random-effects model matrix, $\boldsymbol{Z}$ (Section 2.3.1), and the structure of the relative covariance factor, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ (Section 2.3.2). In principle, a mixed-model formula may contain arbitrarily many random-effects terms, but in practice the number of such terms is typically low.

2.2. Understanding mixed-model formulas

Before describing the details of how **lme4** parses mixed-model formulas (Section 2.3), we provide an informal explanation and then some examples. Our discussion assumes familiarity with the standard R modelling paradigm (Chambers 1993).

Formula	Alternative	Meaning
`(1 | g)`	`1 + (1 | g)`	Random intercept with fixed mean
`0 + offset(o) + (1 | g)`	`-1 + offset(o) + (1 | g)`	Random intercept with *a priori* means
`(1 | g1/g2)`	`(1 | g1)+(1 | g1:g2)`	Intercept varying among `g1` and `g2` within `g1`
`(1 | g1)+(1 | g2)`	`1 + (1 | g1) + (1 | g2)`	Intercept varying among `g1` and `g2`
`x + (x | g)`	`1 + x + (1 + x | g)`	Correlated random intercept and slope
`x + (x || g)`	`1 + x + (1 | g) + (0 + x | g)`	Uncorrelated random intercept and slope

Table 2: Examples of the right-hand sides of mixed-effects model formulas. The names of grouping factors are denoted `g`, `g1`, and `g2`, and covariates and *a priori* known offsets as `x` and `o`.

Each random-effects term is of the form `(expr | factor)`. The expression `expr` is evaluated as a linear model formula, producing a model matrix following the same rules used in standard R modelling functions (e.g., `lm` or `glm`). The expression `factor` is evaluated as an R factor. One way to think about the vertical bar operator is as a special kind of interaction between the model matrix and the grouping factor. This interaction ensures that the columns of the model matrix have different effects for each level of the grouping factor. What makes this a special kind of interaction is that these effects are modelled as unobserved random variables, rather than unknown fixed parameters. Much has been written about important practical and philosophical differences between these two types of interactions (e.g., Henderson (1982); Gelman (2005)). For example, the random-effects implementation of such interactions can be used to obtain shrinkage estimates of regression coefficients (e.g., Efron and Morris (1977)), or account for lack of independence in the residuals due to block structure or repeated measurements (e.g., Laird and Ware (1982)).

Table 2 provides several examples of the right-hand-sides of mixed-model formulas. The first example, `(1 | g)`, is the simplest possible mixed-model formula, where each level of the grouping factor, `g`, has its own random intercept. The mean and standard deviation of these intercepts are parameters to be estimated. Our description of this model incorporates any non-zero mean of the random effects as fixed-effects parameters. If one wishes to specify that a random intercept has *a priori* known means, one may use the `offset` function as in the second model in Table 2. This model contains no fixed effects, or more accurately the fixed-effects model matrix, $\boldsymbol{X}$, has zero columns and $\boldsymbol{\beta}$ has length zero.

We may also construct models with multiple grouping factors. For example, if the observations are grouped by `g2`, which is nested within `g1`, then the third formula in Table 2 can be used to model variation in the intercept. A common objective in mixed modeling is to account for such nested (or hierarchical) structure. However, one of the most useful aspects of **lme4** is that it can be used to fit random-effects associated with non-nested grouping factors. For example, suppose the data are grouped by fully crossing two factors, `g1` and `g2`, then the

fourth formula in Table 2 may be used. Such models are common in item response theory, where `subject` and `item` factors are fully crossed (Doran, Bates, Bliese, and Dowling 2007). In addition to varying intercepts, we may also have varying slopes (e.g., the `sleepstudy` data, Section 1.2). The fifth example in Table 2 gives a model where both the intercept and slope vary among the levels of the grouping factor.

Specifying uncorrelated random effects

By default, **lme4** assumes that all coefficients associated with the same random-effects term are correlated. To specify an uncorrelated slope and intercept (for example), one may either use double-bar notation, `(x || g)`, or equivalently use multiple random effects terms, `x + (1 | g) + (0 + x | g)`, as in the final example of Table 2. For example, from the printout of model `fm1` of the `sleepstudy` data (Section 1.2), we see that the estimated correlation between the slope for `Days` and the intercept is fairly low (0.066) (See Section 5.2.2 below for more on how to extract the random effects covariance matrix.) We may use double-bar notation to fit a model that excludes a correlation parameter:

```
R> fm2 <- lmer(Reaction ~ Days + (Days || Subject), sleepstudy)
```

Although mixed models where the random slopes and intercepts are assumed independent are commonly used to reduce the complexity of random-slopes models, they do have one subtle drawback. Models in which the slopes and intercepts are allowed to have a non-zero correlation (e.g., `fm1`) are invariant to additive shifts of the continuous predictor (`Days` in this case). This invariance breaks down when the correlation is constrained to zero; any shift in the predictor will necessarily lead to a change in the estimated correlation, and in the likelihood and predictions of the model. For example, we can eliminate the correlation in `fm1` simply by shifting `Days` by an amount equal to the ratio of the estimated among-subject standard deviations multiplied by the estimated correlation (i.e., $\sigma_{\text{slope}}/\sigma_{\text{intercept}} \cdot \rho_{\text{slope:intercept}}$). The use of such models should ideally be restricted to cases where the predictor is measured on a ratio scale (i.e., the zero point on the scale is meaningful, not just a location defined by convenience or convention).

2.3. Algebraic and computational account of mixed-model formulas

The fixed-effects terms of a mixed-model formula are parsed to produce the fixed-effects model matrix, $\boldsymbol{X}$, in the same way that the R `lm` function generates model matrices. However, a mixed-model formula incorporates $k \geq 1$ random-effects terms of the form `(r | f)` as well. These k terms are used to produce the random effects model matrix, $\boldsymbol{Z}$ (Equation 2; Section 2.3.1), and the structure of the relative covariance factor, $\boldsymbol{\Lambda_\theta}$ (Equation 4; Section 2.3.2), which are matrices that typically have a sparse structure. We now describe how one might construct these matrices from the random-effects terms, considering first a single term, `(r | f)`, and then generalizing to multiple terms. Tables 3 and 4 summarize the matrices and vectors that determine the structure of $\boldsymbol{Z}$ and $\boldsymbol{\Lambda_\theta}$.

The expression, `r`, is a linear model formula that evaluates to an R model matrix, $\boldsymbol{X}_i$, of size $n \times p_i$, called the *raw random effects model matrix* for term i. A term is said to be a *scalar* random-effects term when $p_i = 1$, otherwise it is *vector-valued*. For a *simple, scalar* random-effects term of the form `(1 | f)`, $\boldsymbol{X}_i$ is the $n \times 1$ matrix of ones, which implies a random intercept model.

Symbol	Size
n	Length of the response vector, $\mathcal{Y}$
p	Number of columns of fixed effects model matrix, $\boldsymbol{X}$
$q = \sum_i^k q_i$	Number of columns of random effects model matrix, $\boldsymbol{Z}$
p_i	Number of columns of the raw model matrix, $\boldsymbol{X}_i$
ℓ_i	Number of levels of the grouping factor indices, $\boldsymbol{i}_i$
$q_i = p_i \ell_i$	Number of columns of the term-wise model matrix, $\boldsymbol{Z}_i$
k	Number of random effects terms
$m_i = \binom{p_i+1}{2}$	Number of covariance parameters for term i
$m = \sum_i^k m_i$	Total number of covariance parameters

Table 3: Dimensions of linear mixed models. The subscript $i = 1, \ldots, k$ denotes a specific random effects term.

Symbol	Size	Description
$\boldsymbol{X}_i$	$n \times p_i$	Raw random effects model matrix
$\boldsymbol{J}_i$	$n \times \ell_i$	Indicator matrix of grouping factor indices
$\boldsymbol{X}_{ij}$	$p_i \times 1$	Column vector containing jth row of $\boldsymbol{X}_i$
$\boldsymbol{J}_{ij}$	$\ell_i \times 1$	Column vector containing jth row of $\boldsymbol{J}_i$
$\boldsymbol{i}_i$	n	Vector of grouping factor indices
$\boldsymbol{Z}_i$	$n \times q_i$	Term-wise random effects model matrix
$\boldsymbol{\theta}$	m	Covariance parameters
$\boldsymbol{T}_i$	$p_i \times p_i$	Lower triangular template matrix
$\boldsymbol{\Lambda}_i$	$q_i \times q_i$	Term-wise relative covariance factor

Table 4: Symbols used to describe the structure of the random effects model matrix and the relative covariance factor. The subscript $i = 1, \ldots, k$ denotes a specific random effects term.

The expression `f` evaluates to an R factor, called the *grouping factor*, for the term. For the ith term, we represent this factor mathematically with a vector $\boldsymbol{i}_i$ of *factor indices*, which is an n-vector of values from $1, \ldots, \ell_i$.[1] Let $\boldsymbol{J}_i$ be the $n \times \ell_i$ matrix of indicator columns for $\boldsymbol{i}_i$. Using the **Matrix** package (Bates and Maechler 2014) in R, we may construct the transpose of $\boldsymbol{J}_i$ from a factor vector, `f`, by coercing `f` to a `sparseMatrix` object. For example,

```
R> (f <- gl(3, 2))

[1] 1 1 2 2 3 3
Levels: 1 2 3

R> (Ji <- t(as(f, Class = "sparseMatrix")))

6 x 3 sparse Matrix of class "dgCMatrix"
     1 2 3
```

[1] In practice, fixed-effects model matrices and random-effects terms are evaluated with respect to a *model frame*, ensuring that any expressions for grouping factors have been coerced to factors and any unused levels of these factors have been dropped. That is, ℓ_i, the number of levels in the grouping factor for the ith random-effects term, is well-defined.

```
[1,] 1 . .
[2,] 1 . .
[3,] . 1 .
[4,] . 1 .
[5,] . . 1
[6,] . . 1
```

When $k > 1$ we order the random-effects terms so that $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_k$; in general, this ordering reduces "fill-in" (i.e., the proportion of elements that are zero in the lower triangle of $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{W} \boldsymbol{Z} \boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}$ but not in the lower triangle of its left Cholesky factor, $\boldsymbol{L}_{\boldsymbol{\theta}}$, described below in Equation 18). This reduction in fill-in provides more efficient matrix operations within the penalized least squares algorithm (Section 3.2).

Constructing the random effects model matrix

The ith random effects term contributes $q_i = \ell_i p_i$ columns to the model matrix $\boldsymbol{Z}$. We group these columns into a matrix, $\boldsymbol{Z}_i$, which we refer to as the *term-wise model matrix* for the ith term. Thus q, the number of columns in $\boldsymbol{Z}$ and the dimension of the random variable, $\mathcal{B}$, is

$$q = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} \ell_i \, p_i. \tag{5}$$

Creating the matrix $\boldsymbol{Z}_i$ from $\boldsymbol{X}_i$ and $\boldsymbol{J}_i$ is a straightforward concept that is, nonetheless, somewhat awkward to describe. Consider $\boldsymbol{Z}_i$ as being further decomposed into ℓ_i blocks of p_i columns. The rows in the first block are the rows of $\boldsymbol{X}_i$ multiplied by the 0/1 values in the first column of $\boldsymbol{J}_i$. Similarly for the subsequent blocks. With these definitions we may define the term-wise random-effects model matrix, $\boldsymbol{Z}_i$, for the ith term as a transposed Khatri-Rao product,

$$\boldsymbol{Z}_i = (\boldsymbol{J}_i^\top * \boldsymbol{X}_i^\top)^\top = \begin{bmatrix} \boldsymbol{J}_{i1}^\top \otimes \boldsymbol{X}_{i1}^\top \\ \boldsymbol{J}_{i2}^\top \otimes \boldsymbol{X}_{i2}^\top \\ \vdots \\ \boldsymbol{J}_{in}^\top \otimes \boldsymbol{X}_{in}^\top \end{bmatrix} \tag{6}$$

where $*$ and $\otimes$ are the Khatri-Rao[2] (Khatri and Rao 1968) and Kronecker products, and $\boldsymbol{J}_{ij}^\top$ and $\boldsymbol{X}_{ij}^\top$ are row vectors of the jth rows of $\boldsymbol{J}_i$ and $\boldsymbol{X}_i$. These rows correspond to the jth sample in the response vector, $\mathcal{Y}$, and thus j runs from $1 \ldots n$. The **Matrix** package for R contains a `KhatriRao` function, which can be used to form $\boldsymbol{Z}_i$. For example, if we begin with a raw model matrix,

```
R> (Xi <- cbind(1, rep.int(c(-1, 1), 3L)))

     [,1] [,2]
[1,]    1   -1
[2,]    1    1
```

[2]Note that the original definition of the Khatri-Rao product is more general than the definition used in **Matrix** package, which is the definition we use here.

```
[3,]    1   -1
[4,]    1    1
[5,]    1   -1
[6,]    1    1
```

then the term-wise random effects model matrix is,

```
R> (Zi <- t(KhatriRao(t(Ji), t(Xi))))

6 x 6 sparse Matrix of class "dgCMatrix"

[1,] 1 -1 .  . .  .
[2,] 1  1 .  . .  .
[3,] .  . 1 -1 .  .
[4,] .  . 1  1 .  .
[5,] .  . .  . 1 -1
[6,] .  . .  . 1  1
```

In particular, for a simple, scalar term, $\boldsymbol{Z}_i$ is exactly $\boldsymbol{J}_i$, the matrix of indicator columns. For other scalar terms, $\boldsymbol{Z}_i$ is formed by element-wise multiplication of the single column of $\boldsymbol{X}_i$ by each of the columns of $\boldsymbol{J}_i$.

Because each $\boldsymbol{Z}_i$ is generated from indicator columns, its cross-product, $\boldsymbol{Z}_i^\top \boldsymbol{Z}_i$ is block-diagonal consisting of ℓ_i diagonal blocks each of size p_i.[3] Note that this means that when $k = 1$ (i.e., there is only one random-effects term, and $\boldsymbol{Z}_i = \boldsymbol{Z}$), $\boldsymbol{Z}^\top \boldsymbol{Z}$ will be block diagonal. These block-diagonal properties allow for more efficient sparse matrix computations (Section 3.7).

The full random effects model matrix, $\boldsymbol{Z}$, is constructed from $k \geq 1$ blocks,

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{Z}_1 & \boldsymbol{Z}_2 & \dots & \boldsymbol{Z}_k \end{bmatrix} \tag{7}$$

By transposing Equation 7 and substituting in Equation 6, we may represent the structure of the transposed random effects model matrix as follows,

$$\boldsymbol{Z}^\top = \begin{array}{c} \begin{array}{cccc} \text{sample 1} & \text{sample 2} & \dots & \text{sample n} \end{array} \\ \begin{bmatrix} \boldsymbol{J}_{11} \otimes \boldsymbol{X}_{11} & \boldsymbol{J}_{12} \otimes \boldsymbol{X}_{12} & \dots & \boldsymbol{J}_{1n} \otimes \boldsymbol{X}_{1n} \\ \boldsymbol{J}_{21} \otimes \boldsymbol{X}_{21} & \boldsymbol{J}_{22} \otimes \boldsymbol{X}_{22} & \dots & \boldsymbol{J}_{2n} \otimes \boldsymbol{X}_{2n} \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \end{array} \begin{array}{l} \text{term 1} \\ \text{term 2} \\ \vdots \end{array} \tag{8}$$

Note that the proportion of elements of $Z^\top$ that are structural zeros is

$$\frac{\sum_{i=1}^{k} p_i(\ell_i - 1)}{\sum_{i=1}^{k} p_i} \quad . \tag{9}$$

[3] To see this, note that by the properties of Kronecker products we may write the cross-product matrix $Z_i^\top Z_i$ as $\sum_{j=1}^{n} \boldsymbol{J}_{ij}\boldsymbol{J}_{ij}^\top \otimes \boldsymbol{X}_{ij}\boldsymbol{X}_{ij}^\top$. Because $\boldsymbol{J}_{ij}$ is a unit vector along a coordinate axis, the cross product $\boldsymbol{J}_{ij}\boldsymbol{J}_{ij}^\top$ is a $p_i \times p_i$ matrix of all zeros except for a single 1 along the diagonal. Therefore, the cross products, $\boldsymbol{X}_{ij}\boldsymbol{X}_{ij}^\top$, will be added to one of the ℓ_i blocks of size $p_i \times p_i$ along the diagonal of $Z_i^\top Z_i$.

Therefore, the sparsity of $\boldsymbol{Z}^\top$ increases with the number of grouping factor levels. As the number of levels is often large in practice, it is essential for speed and efficiency to take account of this sparsity, for example by using sparse matrix methods, when fitting mixed models (Section 3.7).

Constructing the relative covariance factor

The $q \times q$ covariance factor, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, is a block diagonal matrix whose ith diagonal block, $\boldsymbol{\Lambda}_i$, is of size $q_i, i = 1, \ldots, k$. We refer to $\boldsymbol{\Lambda}_i$ as the *term-wise relative covariance factor*. Furthermore, $\boldsymbol{\Lambda}_i$ is a homogeneous block diagonal matrix with each of the ℓ_i lower-triangular blocks on the diagonal being a copy of a $p_i \times p_i$ lower-triangular *template matrix*, $\boldsymbol{T}_i$. The covariance parameter vector, $\boldsymbol{\theta}$, of length $m_i = \binom{p_i+1}{2}$, consists of the elements in the lower triangle of $\boldsymbol{T}_i, i = 1, \ldots, k$. To provide a unique representation we require that the diagonal elements of the $\boldsymbol{T}_i, i = 1, \ldots, k$ be non-negative.

The template, $\boldsymbol{T}_i$, can be constructed from the number p_i alone. In R code we denote p_i as `nc`. For example, if we set `nc <- 3`, we could create the template for term i as,

```
R> (rowIndices <- rep(1:nc, 1:nc))

[1] 1 2 2 3 3 3

R> (colIndices <- sequence(1:nc))

[1] 1 1 2 1 2 3

R> (template <- sparseMatrix(rowIndices, colIndices,
+                            x = 1 * (rowIndices == colIndices)))

3 x 3 sparse Matrix of class "dgCMatrix"

[1,] 1 . .
[2,] 0 1 .
[3,] 0 0 1
```

Note that the `rowIndices` and `colIndices` fill the entire lower triangle, which contains the initial values of the covariance parameter vector, $\boldsymbol{\theta}$,

```
R> (theta <- template@x)

[1] 1 0 0 1 0 1
```

(because the `@x` slot of the sparse matrix `template` is a numeric vector containing the non-zero elements). This template contains three types of elements: structural zeros (denoted by `.`), off-diagonal covariance parameters (initialized at `0`), and diagonal variance parameters (initialized at `1`). The next step in the construction of the relative covariance factor is to repeat the template once for each level of the grouping factor to construct a sparse block

diagonal matrix. For example, if we set the number of levels, ℓ_i, to two, `nl <- 2`, we could create the transposed term-wise relative covariance factor, $\mathbf{\Lambda}_i^\top$, using the `.bdiag` function in the **Matrix** package,

```
R> (Lambdati <- .bdiag(rep(list(t(template)), nl)))

6 x 6 sparse Matrix of class "dgTMatrix"

[1,] 1 0 0 . . .
[2,] . 1 0 . . .
[3,] . . 1 . . .
[4,] . . . 1 0 0
[5,] . . . . 1 0
[6,] . . . . . 1
```

For a model with a single random effects term, $\mathbf{\Lambda}_i^\top$ would be the initial transposed relative covariance factor itself.

The transposed relative covariance factor, $\mathbf{\Lambda}_{\boldsymbol{\theta}}^\top$, that arises from parsing the formula and data is set at the initial value of the covariance parameters, $\boldsymbol{\theta}$. However, during model fitting, it needs to be updated to a new $\boldsymbol{\theta}$ value at each iteration (see Section 3.6.1). This is achieved by constructing a vector of indices, `Lind`, that identifies which elements of `theta` should be placed in which elements of `Lambdat`,

```
R> LindTemplate <- rowIndices + nc * (colIndices - 1) - choose(colIndices, 2)
R> (Lind <- rep(LindTemplate, nl))

 [1] 1 2 4 3 5 6 1 2 4 3 5 6
```

For example, if we randomly generate a new value for `theta`,

```
R> thetanew <- round(runif(length(theta)), 1)
```

we may update `Lambdat` as follows,

```
R> Lambdati@x <- thetanew[Lind]
```

Section 3.6.1 describes the process of updating the relative covariance factor in more detail.

3. Objective function module

3.1. Model reformulation for improved computational stability

In our initial formulation of the linear mixed model (Equations 2, 3, and 4), the covariance parameter vector, $\boldsymbol{\theta}$, appears only in the marginal distribution of the random effects (Equation 3). However, from the perspective of computational stability and efficiency, it is

advantageous to reformulate the model such that $\boldsymbol{\theta}$ appears only in the conditional distribution for the response vector given the random effects. Such a reformulation allows us to work with singular covariance matrices, which regularly arise in practice (e.g., during intermediate steps of the nonlinear optimizer, Section 4).

The reformulation is made by defining a *spherical*[4] *random effects* variable, $\mathcal{U}$, with distribution

$$\mathcal{U} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}_q). \tag{10}$$

If we set,

$$\mathcal{B} = \boldsymbol{\Lambda}_{\boldsymbol{\theta}} \mathcal{U}, \tag{11}$$

then $\mathcal{B}$ will have the desired $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$ distribution (Equation 3). Although it may seem more natural to define $\mathcal{U}$ in terms of $\mathcal{B}$ we must write the relationship as in eqn. 11 to allow for singular $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$. The conditional distribution (Equation 2) of the response vector given the random effects may now be reformulated as,

$$(\mathcal{Y}|\mathcal{U} = \boldsymbol{u}) \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{Y}|\mathcal{U}=\boldsymbol{u}}, \sigma^2 \boldsymbol{W}^{-1}) \tag{12}$$

where

$$\boldsymbol{\mu}_{\mathcal{Y}|\mathcal{U}=\boldsymbol{u}} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\boldsymbol{u} + \boldsymbol{o} \tag{13}$$

is a vector of linear predictors, which can be interpreted as a conditional mean (or mode). Similarly, we also define $\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}$ as the conditional mean (or mode) of the spherical random effects given the observed value of the response vector. Note also that we use the $\boldsymbol{u}$ symbol throughout to represent a specific value of the random variable, $\mathcal{U}$.

3.2. Penalized least squares

Our computational methods for maximum likelihood fitting of the linear mixed model involve repeated applications of the PLS method. In particular, the PLS problem is to minimize the penalized weighted residual sum-of-squares,

$$r^2(\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{u}) = \rho^2(\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{u}) + \|\boldsymbol{u}\|^2, \tag{14}$$

over $\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{\beta}\end{bmatrix}$, where,

$$\rho^2(\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{u}) = \left\| \boldsymbol{W}^{1/2} \left[\boldsymbol{y}_{\text{obs}} - \boldsymbol{\mu}_{\mathcal{Y}|\mathcal{U}=\boldsymbol{u}} \right] \right\|^2, \tag{15}$$

is the weighted residual sum-of-squares. This notation makes explicit the fact that r^2 and ρ^2 both depend on $\boldsymbol{\theta}$, $\boldsymbol{\beta}$, and $\boldsymbol{u}$. The reason for the word 'penalized' is that the term, $\|\boldsymbol{u}\|^2$, in Equation 14 penalizes models with larger magnitude values of $\boldsymbol{u}$.

In the so-called "pseudo-data" approach we write the penalized weighted residual sum-of-squares as the squared length of a block matrix equation,

$$r^2(\boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{u}) = \left\| \begin{bmatrix} \boldsymbol{W}^{1/2}(\boldsymbol{y}_{\text{obs}} - \boldsymbol{o}) \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{W}^{1/2}\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} & \boldsymbol{W}^{1/2}\boldsymbol{X} \\ \boldsymbol{I}_q & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\beta} \end{bmatrix} \right\|^2. \tag{16}$$

[4] $\mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ distributions are called "spherical" because contours of the probability density are spheres.

This pseudo-data approach shows that the PLS problem may also be thought of as a standard least squares problem for an extended response vector, which implies that the minimizing value, $\begin{bmatrix}\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}\\ \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}\end{bmatrix}$, satisfies the normal equations,

$$\begin{bmatrix}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}(\boldsymbol{y}_{\text{obs}}-\boldsymbol{o})\\ \boldsymbol{X}^{\top}\boldsymbol{W}(\boldsymbol{y}_{\text{obs}}-\boldsymbol{o})\end{bmatrix}=\begin{bmatrix}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}+\boldsymbol{I} & \boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\boldsymbol{X}\\ \boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} & \boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X}\end{bmatrix}\begin{bmatrix}\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}\\ \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}\end{bmatrix}, \tag{17}$$

where $\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}$ is the conditional mean of $\mathcal{U}$ given that $\mathcal{Y}=\boldsymbol{y}_{\text{obs}}$. Note that this conditional mean depends on $\boldsymbol{\theta}$, although we do not make this dependency explicit in order to reduce notational clutter.

The cross-product matrix in Equation 17 can be Cholesky decomposed,

$$\begin{bmatrix}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}+\boldsymbol{I} & \boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\boldsymbol{X}\\ \boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} & \boldsymbol{X}^{\top}\boldsymbol{W}\boldsymbol{X}\end{bmatrix}=\begin{bmatrix}\boldsymbol{L}_{\boldsymbol{\theta}} & \boldsymbol{0}\\ \boldsymbol{R}_{ZX}^{\top} & \boldsymbol{R}_{X}^{\top}\end{bmatrix}\begin{bmatrix}\boldsymbol{L}_{\boldsymbol{\theta}}^{\top} & \boldsymbol{R}_{ZX}\\ \boldsymbol{0} & \boldsymbol{R}_{X}\end{bmatrix}. \tag{18}$$

We may use this decomposition to rewrite the penalized weighted residual sum-of-squares as,

$$r^2(\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{u})=r^2(\boldsymbol{\theta})+\left\|\boldsymbol{L}_{\boldsymbol{\theta}}^{\top}(\boldsymbol{u}-\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}})+\boldsymbol{R}_{ZX}(\boldsymbol{\beta}-\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2+\left\|\boldsymbol{R}_{X}(\boldsymbol{\beta}-\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2 \quad , \tag{19}$$

where we have simplified notation by writing $r^2(\boldsymbol{\theta},\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}},\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}})$ as $r^2(\boldsymbol{\theta})$. This is an important expression in the theory underlying **lme4**. It relates the penalized weighted residual sum-of-squares, $r^2(\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{u})$, with its minimum value, $r^2(\boldsymbol{\theta})$. This relationship is useful in the next section where we integrate over the random effects, which is required for maximum likelihood estimation.

3.3. Probability densities

The residual sums-of-squares discussed in the previous section can be used to express various probability densities, which are required for maximum likelihood estimation of the linear mixed model[5],

$$f_{\mathcal{Y}|\mathcal{U}}(\boldsymbol{y}_{\text{obs}}|\boldsymbol{u})=\frac{|\boldsymbol{W}|^{1/2}}{(2\pi\sigma^2)^{n/2}}\exp\left[\frac{-\rho^2(\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{u})}{2\sigma^2}\right], \tag{20}$$

$$f_{\mathcal{U}}(\boldsymbol{u})=\frac{1}{(2\pi\sigma^2)^{q/2}}\exp\left[\frac{-\|\boldsymbol{u}\|^2}{2\sigma^2}\right], \tag{21}$$

$$f_{\mathcal{Y},\mathcal{U}}(\boldsymbol{y}_{\text{obs}},\boldsymbol{u})=\frac{|\boldsymbol{W}|^{1/2}}{(2\pi\sigma^2)^{(n+q)/2}}\exp\left[\frac{-r^2(\boldsymbol{\theta},\boldsymbol{\beta},\boldsymbol{u})}{2\sigma^2}\right], \tag{22}$$

$$f_{\mathcal{U}|\mathcal{Y}}(\boldsymbol{u}|\boldsymbol{y}_{\text{obs}})=\frac{f_{\mathcal{Y},\mathcal{U}}(\boldsymbol{y}_{\text{obs}},\boldsymbol{u})}{f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}})}, \tag{23}$$

where,

$$f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}})=\int f_{\mathcal{Y},\mathcal{U}}(\boldsymbol{y}_{\text{obs}},\boldsymbol{u})d\boldsymbol{u}, \tag{24}$$

[5]These expressions only technically hold at the observed value, $\boldsymbol{y}_{\text{obs}}$, of the response vector, $\mathcal{Y}$.

The log-likelihood to be maximized can therefore be expressed as,

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}, \sigma^2 | \boldsymbol{y}_{\text{obs}}) = \log f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}}) \tag{25}$$

The integral in Equation 24 may be more explicitly written as,

$$\begin{aligned} f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}}) = \frac{|\boldsymbol{W}|^{1/2}}{(2\pi\sigma^2)^{(n+q)/2}} \exp\left[\frac{-r^2(\boldsymbol{\theta}) - \left\|\boldsymbol{R}_X(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{2\sigma^2}\right] \\ \int \exp\left[\frac{-\left\|\boldsymbol{L}_{\boldsymbol{\theta}}^{\top}(\boldsymbol{u} - \boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}) + \boldsymbol{R}_{ZX}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{2\sigma^2}\right] d\boldsymbol{u}, \end{aligned} \tag{26}$$

which can be evaluated with the change of variables,

$$\boldsymbol{v} = \boldsymbol{L}_{\boldsymbol{\theta}}^{\top}(\boldsymbol{u} - \mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}) + \boldsymbol{R}_{ZX}(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}), \tag{27}$$

The Jacobian determinant of the transformation from $\boldsymbol{u}$ to $\boldsymbol{v}$ is $|\boldsymbol{L}_{\boldsymbol{\theta}}|$. Therefore we are able to write the integral as,

$$\begin{aligned} f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}}) = \frac{|\boldsymbol{W}|^{1/2}}{(2\pi\sigma^2)^{(n+q)/2}} \exp\left[\frac{-r^2(\boldsymbol{\theta}) - \left\|\boldsymbol{R}_X(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{2\sigma^2}\right] \\ \int \exp\left[\frac{-\|\boldsymbol{v}\|^2}{2\sigma^2}\right] |\boldsymbol{L}_{\boldsymbol{\theta}}|^{-1} d\boldsymbol{v}, \end{aligned} \tag{28}$$

which by the properties of exponential integrands becomes,

$$\exp \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}, \sigma^2 | \boldsymbol{y}_{\text{obs}}) = f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}}) = \frac{|\boldsymbol{W}|^{1/2}|\boldsymbol{L}_{\boldsymbol{\theta}}|^{-1}}{(2\pi\sigma^2)^{n/2}} \exp\left[\frac{-r^2(\boldsymbol{\theta}) - \left\|\boldsymbol{R}_X(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{2\sigma^2}\right]. \tag{29}$$

3.4. Evaluating and profiling the deviance and REML criterion

We are now in a position to understand why the formulation in equations 2 and 3 is particularly useful. We are able to explicitly profile $\boldsymbol{\beta}$ and σ out of the log-likelihood (Equation 25), to find a compact expression for the profiled deviance (negative twice the profiled log-likelihood) and the profiled REML criterion as a function of the relative covariance parameters, $\boldsymbol{\theta}$, only. Furthermore these criteria can be evaluated quickly and accurately.

To estimate the parameters, $\boldsymbol{\theta}$, $\boldsymbol{\beta}$, and σ^2, we minimize negative twice the log-likelihood, which can be written as,

$$-2\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}, \sigma^2 | \boldsymbol{y}_{\text{obs}}) = \log \frac{|\boldsymbol{L}_{\boldsymbol{\theta}}|^2}{|\boldsymbol{W}|} + n \log(2\pi\sigma^2) + \frac{r^2(\boldsymbol{\theta})}{\sigma^2} + \frac{\left\|\boldsymbol{R}_X(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{\sigma^2}. \tag{30}$$

It is very easy to profile out $\boldsymbol{\beta}$, because it enters into the ML criterion only through the final term, which is zero if $\boldsymbol{\beta} = \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}$, where $\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}$ is found by solving the penalized least-squares problem in Equation 16. Therefore we can write a partially profiled ML criterion as,

$$-2\mathcal{L}(\boldsymbol{\theta}, \sigma^2 | \boldsymbol{y}_{\text{obs}}) = \log \frac{|\boldsymbol{L}_{\boldsymbol{\theta}}|^2}{|\boldsymbol{W}|} + n \log(2\pi\sigma^2) + \frac{r^2(\boldsymbol{\theta})}{\sigma^2}. \tag{31}$$

This criterion is only partially profiled because it still depends on σ^2. Differentiating this criterion with respect to σ^2 and setting the result equal to zero yields,

$$0 = \frac{n}{\widehat{\sigma}^2_{\boldsymbol{\theta}}} - \frac{r^2(\boldsymbol{\theta})}{\widehat{\sigma}^4_{\boldsymbol{\theta}}}, \tag{32}$$

which leads to a maximum profiled likelihood estimate,

$$\widehat{\sigma}^2_{\boldsymbol{\theta}} = \frac{r^2(\boldsymbol{\theta})}{n}. \tag{33}$$

This estimate can be substituted into the partially profiled criterion to yield the fully profiled ML criterion,

$$-2\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}}) = \log\frac{|\boldsymbol{L}_{\boldsymbol{\theta}}|^2}{|\boldsymbol{W}|} + n\left[1 + \log\left(\frac{2\pi r^2(\boldsymbol{\theta})}{n}\right)\right]. \tag{34}$$

This expression for the profiled deviance depends only on $\boldsymbol{\theta}$. Although q, the number of columns in $\boldsymbol{Z}$ and the size of $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$, can be very large indeed, the dimension of $\boldsymbol{\theta}$ is small, frequently less than 10. The **lme4** package uses generic nonlinear optimizers (Section 4) to optimize this expression over $\boldsymbol{\theta}$ to find its maximum likelihood estimate.

The REML criterion

The REML criterion can be obtained by integrating the marginal density for $\mathcal{Y}$ with respect to the fixed effects (Laird and Ware 1982),

$$\int f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}})d\boldsymbol{\beta} = \frac{|\boldsymbol{W}|^{1/2}|\boldsymbol{L}_{\boldsymbol{\theta}}|^{-1}}{(2\pi\sigma^2)^{n/2}}\exp\left[\frac{-r^2(\boldsymbol{\theta})}{2\sigma^2}\right]\int\exp\left[\frac{-\left\|\boldsymbol{R}_X(\boldsymbol{\beta}-\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})\right\|^2}{2\sigma^2}\right]d\boldsymbol{\beta}, \tag{35}$$

which can be evaluated with the change of variables,

$$\boldsymbol{v} = \boldsymbol{R}_X(\boldsymbol{\beta} - \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}). \tag{36}$$

The Jacobian determinant of the transformation from $\boldsymbol{\beta}$ to $\boldsymbol{v}$ is $|\boldsymbol{R}_X|$. Therefore we are able to write the integral as,

$$\int f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}})d\boldsymbol{\beta} = \frac{|\boldsymbol{W}|^{1/2}|\boldsymbol{L}_{\boldsymbol{\theta}}|^{-1}}{(2\pi\sigma^2)^{n/2}}\exp\left[\frac{-r^2(\boldsymbol{\theta})}{2\sigma^2}\right]\int\exp\left[\frac{-\|\boldsymbol{v}\|^2}{2\sigma^2}\right]|\boldsymbol{R}_X|^{-1}d\boldsymbol{v}, \tag{37}$$

which simplifies to,

$$\int f_{\mathcal{Y}}(\boldsymbol{y}_{\text{obs}})d\boldsymbol{\beta} = \frac{|\boldsymbol{W}|^{1/2}|\boldsymbol{L}_{\boldsymbol{\theta}}|^{-1}|\boldsymbol{R}_X|^{-1}}{(2\pi\sigma^2)^{(n-p)/2}}\exp\left[\frac{-r^2(\boldsymbol{\theta})}{2\sigma^2}\right]. \tag{38}$$

Minus twice the log of this integral is the (unprofiled) REML criterion,

$$-2\mathcal{L}_R(\boldsymbol{\theta},\sigma^2|\boldsymbol{y}_{\text{obs}}) = \log\frac{|\boldsymbol{L}_{\boldsymbol{\theta}}|^2|\boldsymbol{R}_X|^2}{|\boldsymbol{W}|} + (n-p)\log(2\pi\sigma^2) + \frac{r^2(\boldsymbol{\theta})}{\sigma^2}. \tag{39}$$

Note that because $\boldsymbol{\beta}$ gets integrated out, the REML criterion cannot be used to find a point estimate of $\boldsymbol{\beta}$. However, we follow others in using the maximum likelihood estimate, $\widehat{\boldsymbol{\beta}}_{\widehat{\boldsymbol{\theta}}}$, at the optimum value of $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$. The REML estimate of σ^2 is,

$$\widehat{\sigma^2_{\boldsymbol{\theta}}} = \frac{r^2(\boldsymbol{\theta})}{n-p}, \tag{40}$$

which leads to the profiled REML criterion,

$$-2\mathcal{L}_R(\boldsymbol{\theta}|\boldsymbol{y}_{\text{obs}}) = \log\frac{|\boldsymbol{L}_{\boldsymbol{\theta}}|^2|\boldsymbol{R}_X|^2}{|\boldsymbol{W}|} + (n-p)\left[1+\log\left(\frac{2\pi r^2(\boldsymbol{\theta})}{n-p}\right)\right]. \tag{41}$$

3.5. Changes relative to previous formulations

We compare the PLS problem as formulated in Section 3.2 with earlier versions and describe why we use this version. What have become known as "Henderson's mixed-model equations" are given as Equation 6 of Henderson (1982) and would be expressed as,

$$\begin{bmatrix}\boldsymbol{X}^\top\boldsymbol{X}/\sigma^2 & \boldsymbol{X}^\top\boldsymbol{Z}/\sigma^2\\ \boldsymbol{Z}^\top\boldsymbol{X}/\sigma^2 & \boldsymbol{Z}^\top\boldsymbol{Z}/\sigma^2+\boldsymbol{\Sigma}^{-1}\end{bmatrix}\begin{bmatrix}\widehat{\boldsymbol{\beta}}_\theta\\ \boldsymbol{\mu}_{\mathcal{B}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{X}^\top\boldsymbol{y}_{\text{obs}}/\sigma^2\\ \boldsymbol{Z}^\top\boldsymbol{y}_{\text{obs}}/\sigma^2\end{bmatrix}, \tag{42}$$

in our notation (ignoring weights and offsets, without loss of generality). The matrix written as $\boldsymbol{R}$ in Henderson (1982) is $\sigma^2\boldsymbol{I}_n$ in our formulation of the model.

Bates and DebRoy (2004) modified the PLS equations to

$$\begin{bmatrix}\boldsymbol{Z}^\top\boldsymbol{Z}+\boldsymbol{\Omega} & \boldsymbol{Z}^\top\boldsymbol{X}\\ \boldsymbol{X}^\top\boldsymbol{Z} & \boldsymbol{X}^\top\boldsymbol{X}\end{bmatrix}\begin{bmatrix}\boldsymbol{\mu}_{\mathcal{B}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}\\ \widehat{\boldsymbol{\beta}}_\theta\end{bmatrix} = \begin{bmatrix}\boldsymbol{X}^\top\boldsymbol{y}_{\text{obs}}\\ \boldsymbol{Z}^\top\boldsymbol{y}_{\text{obs}}\end{bmatrix}. \tag{43}$$

where $\boldsymbol{\Omega}_\theta = \left(\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\right)^{-1} = \sigma^2\boldsymbol{\Sigma}^{-1}$ is the *relative precision matrix* for a given value of $\boldsymbol{\theta}$. They also showed that the profiled log-likelihood can be expressed (on the deviance scale) as

$$-2\mathcal{L}(\boldsymbol{\theta}) = \log\left(\frac{|\boldsymbol{Z}^\top\boldsymbol{Z}+\boldsymbol{\Omega}|}{|\boldsymbol{\Omega}|}\right) + n\left[1+\log\left(\frac{2\pi r^2_\theta}{n}\right)\right]. \tag{44}$$

The primary difference between Equation 42 and Equation 43 is the order of the blocks in the system matrix. The PLS problem can be solved using a Cholesky factor of the system matrix with the blocks in either order. The advantage of using the arrangement in Equation 43 is to allow for evaluation of the profiled log-likelihood. To evaluate $|\boldsymbol{Z}^\top\boldsymbol{Z}+\boldsymbol{\Omega}|$ from the Cholesky factor that block must be in the upper-left corner, not the lower right. Also, $\boldsymbol{Z}$ is sparse whereas $\boldsymbol{X}$ is usually dense. It is straightforward to exploit the sparsity of $\boldsymbol{Z}^\top\boldsymbol{Z}$ in the Cholesky factorization when the block containing this matrix is the first block to be factored. If $\boldsymbol{X}^\top\boldsymbol{X}$ is the first block to be factored it is much more difficult to preserve sparsity.

The main change from the formulation in Bates and DebRoy (2004) to the current formulation is the use of a relative covariance factor, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, instead of a relative precision matrix, $\boldsymbol{\Omega}_\theta$, and solving for the mean of $\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}$ instead of the mean of $\mathcal{B}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}$. This change improves stability, because the solution to the PLS problem in Section 3.2 is well-defined when $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ is

singular whereas the formulation in Equation 43 cannot be used in these cases because $\boldsymbol{\Omega}_\theta$ does not exist.

It is important to allow for $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ to be singular because situations where the parameter estimates, $\widehat{\boldsymbol{\theta}}$, produce a singular $\boldsymbol{\Lambda}_{\widehat{\theta}}$ do occur in practice. And even if the parameter estimates do not correspond to a singular $\boldsymbol{\Lambda}_\theta$, it may be desirable to evaluate the estimation criterion at such values during the course of the numerical optimization of the criterion.

Bates and DebRoy (2004) also provided expressions for the gradient of the profiled log-likelihood expressed as Equation 44. These expressions can be translated into the current formulation. From Equation 34 we can see that (again ignoring weights),

$$\begin{aligned}
\nabla\left(-2\mathcal{L}(\theta)\right) &= \nabla\log(|\boldsymbol{L}_\theta|^2) + \nabla\left(n\log(r^2(\boldsymbol{\theta}))\right) \\
&= \nabla\log(|\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}|) + n\left(\nabla r^2(\boldsymbol{\theta})\right)/r^2(\boldsymbol{\theta}). \\
&= \nabla\log(|\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}|) + \left(\nabla r^2(\boldsymbol{\theta})\right)/(\widehat{\sigma^2})
\end{aligned} \tag{45}$$

The first gradient is easy to express but difficult to evaluate for the general model. The individual elements of this gradient are

$$\begin{aligned}
\frac{\partial\log(|\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}|)}{\partial\theta_i} &= \operatorname{tr}\left[\frac{\partial\left(\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\right)}{\partial\theta_i}\left(\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}\right)^{-1}\right] \\
&= \operatorname{tr}\left[\left(\boldsymbol{L}_\theta\boldsymbol{L}_\theta^\top\right)^{-1}\left(\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top\boldsymbol{Z}^\top\boldsymbol{Z}\frac{\partial\boldsymbol{\Lambda}_{\boldsymbol{\theta}}}{\partial\theta_i} + \frac{\partial\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top}{\partial\theta_i}\boldsymbol{Z}^\top\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\right)\right].
\end{aligned} \tag{46}$$

The second gradient term can be expressed as a linear function of the residual, with individual elements of the form

$$\frac{\partial r^2(\boldsymbol{\theta})}{\partial\theta_i} = -2\boldsymbol{u}^\top\frac{\partial\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top}{\partial\theta_i}\boldsymbol{Z}^\top\left(\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}\boldsymbol{u} - \boldsymbol{X}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}\right), \tag{47}$$

using the results of Golub and Pereyra (1973). Although we do not use these results in **lme4**, they are used for certain model types in the **MixedModels** package for Julia and do provide improved performance.

3.6. Penalized least squares algorithm

For efficiency, in **lme4** itself, PLS is implemented in compiled C++ code using the **Eigen** templated C++ package for numerical linear algebra. Here however, in order to improve readability we describe a version in pure R. Section 3.7 provides greater detail on the techniques and concepts for computational efficiency, which is important in cases where the nonlinear optimizer (Section 4) requires many iterations.

The PLS algorithm takes a vector of covariance parameters, $\boldsymbol{\theta}$, as inputs and returns the profiled deviance (Equation 34) or the REML criterion (Equation 41). This PLS algorithm consists of four main steps:

1. Update the relative covariance factor (Section 3.6.1),
2. Solve the normal equations (Section 3.6.2),

Name/description	Pseudocode	Math	Type
Mapping from covariance parameters to relative covariance factor	`mapping`		function
Response vector	`y`	$\boldsymbol{y}_{\text{obs}}$ (Section 1.1)	double vector
Fixed effects model matrix	`X`	$\boldsymbol{X}$ (Equation 2)	double dense[a] matrix
Transposed random effects model matrix	`Zt`	$\boldsymbol{Z}^\top$ (Equation 2)	double sparse matrix
Square-root weights matrix	`sqrtW`	$\boldsymbol{W}^{1/2}$ (Equation 2)	double diagonal matrix
Offset	`offset`	$\boldsymbol{o}$ (Equation 2)	double vector

[a]In previous versions of **lme4** a sparse X matrix, useful for models with categorical fixed-effect predictors with many levels, could be specified; this feature is not currently available.

Table 5: Inputs into a linear mixed model.

Pseudocode	Math
`ZtW`	$\boldsymbol{Z}^\top\boldsymbol{W}^{1/2}$
`ZtWy`	$\boldsymbol{Z}^\top\boldsymbol{W}\boldsymbol{y}_{\text{obs}}$
`ZtWX`	$\boldsymbol{Z}^\top\boldsymbol{W}\boldsymbol{X}$
`XtWX`	$\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{X}$
`XtWy`	$\boldsymbol{X}^\top\boldsymbol{W}\boldsymbol{y}_{\text{obs}}$

Table 6: Constant symbols involved in penalized least squares.

3. Update the linear predictor and residuals (Section 3.6.3),
4. Compute and return the profiled deviance (Section 3.6.4).

PLS also requires the objects described in Table 5, which define the structure of the model. These objects do not get updated during the PLS iterations, and so it is useful to store various matrix products involving them (Table 6). Table 7 lists the objects that do get updated over the PLS iterations. The symbols in this table correspond to a version of **lme4** that is implemented entirely in R (i.e., no compiled code as in **lme4** itself). This implementation is called **lme4pureR** and is currently available on Github (https://github.com/lme4/lme4pureR/).

PLS step I: update relative covariance factor

The first step of PLS is to update the relative covariance factor, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, from the current value of the covariance parameter vector, $\boldsymbol{\theta}$. The updated $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ is then used to update the random effects Cholesky factor, $\boldsymbol{L}_{\boldsymbol{\theta}}$ (Equation 18). The mapping from the covariance parameters to the relative covariance factor can take many forms, but in general involves a function that takes $\boldsymbol{\theta}$ into the values of the non-zero elements of $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$.

If $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ is stored as an object of class `"dgCMatrix"` from the **Matrix** package for R, then we may update $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ from $\boldsymbol{\theta}$ by,

```
R> Lambdat@x[] <- mapping(theta)
```

where `mapping` is an R function that both accepts and returns a numeric vector. The non-zero elements of sparse matrix classes in **Matrix** are stored in a slot called `x`.

Name/description	Pseudocode	Math	Type
Relative covariance factor	`lambda`	$\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ (Equation 4)	sparse double lower-triangular matrix
Random-effects Cholesky factor	`L`	$\boldsymbol{L}_{\boldsymbol{\theta}}$ (Equation 18)	double sparse triangular matrix
Intermediate vector in the solution of the normal equations	`cu`	$\boldsymbol{c}_u$ (Equation 48)	double vector
Block in the full Cholesky factor	`RZX`	$\boldsymbol{R}_{ZX}$ (Equation 18)	double dense matrix
Cross-product of the fixed-effects Cholesky factor	`RXtRX`	$\boldsymbol{R}_X^\top \boldsymbol{R}_X$ (Equation 50)	double dense matrix
Fixed effects coefficients	`beta`	$\boldsymbol{\beta}$ (Equation 2)	double vector
Spherical conditional modes	`u`	$\boldsymbol{u}$ (Section 3.1)	double vector
Non-spherical conditional modes	`b`	$\boldsymbol{b}$ (Equation 2)	double vector
Linear predictor	`mu`	$\boldsymbol{\mu}_{\mathcal{Y}\mid\mathcal{U}=\boldsymbol{u}}$ (Equation 13)	double vector
Weighted residuals	`wtres`	$\boldsymbol{W}^{1/2}(\boldsymbol{y}_{\text{obs}} - \mu)$	double vector
Penalized weighted residual sum-of-squares	`pwrss`	$r^2(\boldsymbol{\theta})$ (Equation 19)	double
Twice the log determinant random-effects Cholesky factor	`logDet`	$\log \lvert\boldsymbol{L}_{\boldsymbol{\theta}}\rvert^2$	double

Table 7: Quantities updated during an evaluation of the linear mixed model objective function.

In the current version of **lme4** (v. 1.1-7) the mapping from $\boldsymbol{\theta}$ to $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ is represented as an R integer vector, `Lind`, of indices, so that

```
R> mapping <- function(theta) theta[Lind]
```

The index vector `Lind` is computed during the model setup and stored in the function's environment. Continuing the example from Section 2.3.2, consider a new value for `theta`,

```
R> thetanew <- c(1, -0.1, 2, 0.1, -0.2, 3)
```

To put these values in the appropriate elements in `Lambdati`, we use `mapping`,

```
R> Lambdati@x[] <- mapping(thetanew)
R> Lambdati

6 x 6 sparse Matrix of class "dgTMatrix"

[1,] 1 -0.1  2.0 .   .    .
[2,] .  0.1 -0.2 .   .    .
[3,] .   .   3.0 .   .    .
[4,] .   .    .  1 -0.1  2.0
[5,] .   .    .  .  0.1 -0.2
[6,] .   .    .  .   .   3.0
```

This `Lind` approach can be useful for extending the capabilities of **lme4** by using the modular approach to fitting mixed models. For example, Appendix A.2 shows how to use `Lind` to fit a model where a random slope and intercept are uncorrelated and have the same variance.

The mapping from the covariance parameters to the relative covariance factor is treated differently in other implementations of the `lme4` approach to linear mixed models. At the other extreme, the `flexLambda` branch of **lme4** and the **lme4pureR** package allows the capabilities for a completely general mapping. This added flexibility has the advantage of allowing a much wider variety of models (e.g., compound symmetry, auto-regression). However, the disadvantage of this approach is that it becomes possible to fit a much wider variety of ill-posed models. Finally, if one would like such added flexibility with the current stable version of **lme4**, it is always possible to use the modular approach to wrap the `Lind`-based deviance function in a general mapping function taking a parameter to be optimized, say ϕ, into $\boldsymbol{\theta}$. However, this approach is likely to be inefficient in many cases.

The `update` method from the **Matrix** package efficiently updates the random effects Cholesky factor, $\boldsymbol{L}_{\boldsymbol{\theta}}$, from a new value of $\boldsymbol{\theta}$ and the updated $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$.

```
R> L <- update(L, Lambdat %*% ZtW, mult = 1)
```

The `mult = 1` argument corresponds to the addition of the identity matrix to the upper-left block on the left-hand-side of Equation 18.

PLS step II: solve normal equations

With the new covariance parameters installed in $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, the next step is to solve the normal equations (Equation 17) for the current estimate of the fixed effects coefficients, $\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}$, and the conditional mode, $\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}$. We solve these equations using a sparse Cholesky factorization (Equation 18). In a complex model fit to a large data set, the dominant calculation in the evaluation of the profiled deviance (Equation 34) or REML criterion (Equation 41) is this sparse Cholesky factorization (Equation 18). The factorization is performed in two phases; a symbolic phase and a numeric phase. The symbolic phase, in which the fill-reducing permutation $\boldsymbol{P}$ is determined along with the positions of the non-zeros in $\boldsymbol{L}_{\boldsymbol{\theta}}$, does not depend on the value of $\boldsymbol{\theta}$. It only depends on the positions of the nonzeros in $\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$. The numeric phase uses $\boldsymbol{\theta}$ to determine the numeric values of the non-zeros in $\boldsymbol{L}_{\boldsymbol{\theta}}$. Using this factorization, the solution proceeds by the following steps,

$$\boldsymbol{L}_{\boldsymbol{\theta}}\boldsymbol{c}_u = \boldsymbol{P}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{W}\boldsymbol{y} \tag{48}$$

$$\boldsymbol{L}_{\boldsymbol{\theta}}\boldsymbol{R}_{ZX} = \boldsymbol{P}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{W}\boldsymbol{X} \tag{49}$$

$$\boldsymbol{R}_X^\top \boldsymbol{R}_X = \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{X} - \boldsymbol{R}_{ZX}^\top \boldsymbol{R}_{ZX} \tag{50}$$

$$\left(\boldsymbol{R}_X^\top \boldsymbol{R}_X\right)\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}} = \boldsymbol{X}^\top \boldsymbol{W}\boldsymbol{y} - \boldsymbol{R}_{ZX}\boldsymbol{c}_u \tag{51}$$

$$\boldsymbol{L}_{\boldsymbol{\theta}}^\top \boldsymbol{P}\boldsymbol{u} = \boldsymbol{c}_u - \boldsymbol{R}_{ZX}\widehat{\boldsymbol{\beta}} \tag{52}$$

which can be solved using the **Matrix** package with,

```
R> cu[] <- as.vector(solve(L, solve(L, Lambdat %*% ZtWy,
+                                   system = "P"), system = "L"))
```

```
R> RZX[] <- as.vector(solve(L, solve(L, Lambdat %*% ZtWX,
+                                    system = "P"), system = "L"))
R> RXtRX <- as(XtWX - crossprod(RZX), "dpoMatrix")
R> beta[] <- as.vector(solve(RXtRX, XtWy - crossprod(RZX, cu)))
R> u[] <- as.vector(solve(L, solve(L, cu - RZX %*% beta,
+                                  system = "Lt"), system = "Pt"))
R> b[] <- as.vector(crossprod(Lambdat, u))
```

Notice the nested calls to `solve`. The inner calls of the first two assignments determine and apply the permutation matrix (`system = "P"`), whereas the outer calls actually solve the equation (`system = "L"`). In the assignment to `u[]`, the nesting is reversed in order to return to the original permutation.

PLS step III: update linear predictor and residuals

The next step is to compute the linear predictor, $\mu_{\mathcal{Y}|\mathcal{U}}$ (Equation 13), and the weighted residuals with new values for $\widehat{\boldsymbol{\beta}}_\theta$ and $\mu_{\mathcal{B}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}$. In **lme4pureR** these quantities can be computed as,

```
R> mu[] <- as.vector(crossprod(Zt, b) + X %*% beta + offset)
R> wtres <- sqrtW * (y - mu)
```

where `b` represents the current estimate of $\mu_{\mathcal{B}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}$.

PLS step IV: compute profiled deviance

Finally, the updated linear predictor and weighted residuals can be used to compute the profiled deviance (or REML criterion), which in **lme4pureR** proceeds as,

```
R> pwrss <- sum(wtres^2) + sum(u^2)
R> logDet <- 2*determinant(L, logarithm = TRUE)$modulus
R> if (REML) logDet <- logDet + determinant(RXtRX,
+                                           logarithm = TRUE)$modulus
R> attributes(logDet) <- NULL
R> profDev <- logDet + degFree * (1 + log(2 * pi * pwrss) - log(degFree))
```

The profiled deviance consists of three components: (1) log-determinant(s) of Cholesky factorization (`logDet`), (2) the degrees of freedom (`degFree`), and the penalized weighted residual sum-of-squares (`pwrss`).

3.7. Sparse matrix methods

In fitting linear mixed models, an instance of the PLS problem (17) must be solved at each evaluation of the objective function during the optimization (Section 4) with respect to $\boldsymbol{\theta}$. Because this operation must be performed many times it is worthwhile considering how to provide effective evaluation methods for objects and calculations involving the sparse matrices associated with random effects terms (Sections 2.3).

The CHOLMOD library of C functions (Chen, Davis, Hager, and Rajamanickam 2008), on which the sparse matrix capabilities of the **Matrix** package for R and the sparse Cholesky factorization in Julia are based, allows for separation of the symbolic and numeric phases. Thus we perform the symbolic phase as part of establishing the structure representing the model (Section 2). Furthermore, because CHOLMOD functions allow for updating $\boldsymbol{L}_{\boldsymbol{\theta}}$ directly from the matrix $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top$ without actually forming $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{Z} \boldsymbol{\Lambda}_{\boldsymbol{\theta}} + \boldsymbol{I}$ we generate and store $\boldsymbol{Z}^\top$ instead of $\boldsymbol{Z}$ (note that we have ignored the weights matrix, $\boldsymbol{W}$, for simplicity). We can update $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top$ directly from $\boldsymbol{\theta}$ without forming $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ and multiplying two sparse matrices. Although such a direct approach is used in the **MixedModels** package for Julia, in **lme4** we first update $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top$ then form the sparse product $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top$. A third alternative, which we employ in **lme4pureR**, is to compute and save the cross-products of the model matrices, $\boldsymbol{X}$ and $\boldsymbol{Z}$, and the response, $\boldsymbol{y}$, before starting the iterations. To allow for case weights, we save the products $\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$, $\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{y}$, $\boldsymbol{Z}^\top \boldsymbol{W} \boldsymbol{X}$, $\boldsymbol{Z}^\top \boldsymbol{W} \boldsymbol{y}$ and $\boldsymbol{Z}^\top \boldsymbol{W} \boldsymbol{Z}$ (see Table 6).

We wish to use structures and algorithms that allow us to take a new value of $\boldsymbol{\theta}$ and evaluate the $\boldsymbol{L}_\theta$ (eqn. 18) efficiently. The key to doing so is the special structure of $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{W}^{1/2}$. To understand why this matrix, and not its transpose, is of interest we describe the sparse matrix structures used in Julia and in the **Matrix** package for R.

Dense matrices are stored in R and in Julia as a one-dimensional array of contiguous storage locations addressed in *column-major order*. This means that elements in the same column are in adjacent storage locations whereas elements in the same row can be widely separated in memory. For this reason, algorithms that work column-wise are preferred to those that work row-wise.

Although a sparse matrix can be stored in a *triplet* format, where the row position, column position and element value the nonzeros are recorded as triplets, the preferred storage forms for actual computation with sparse matrices are compressed sparse column (CSC) or compressed sparse row (CSR) (Davis 2006, Ch. 2). The CHOLMOD (and, more generally, the SuiteSparse package of C libraries) uses the CSC storage format. In this format the non-zero elements in a column are in adjacent storage locations and access to all the elements in a column is much easier than access to those in a row (similar to dense matrices stored in column-major order).

The matrices $\boldsymbol{Z}$ and $\boldsymbol{Z}\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$ have the property that the number of nonzeros in each row, $\sum_{i=1}^k p_i$, is constant. For CSC matrices we want consistency in the columns rather than the rows, which is why we work with $\boldsymbol{Z}^\top$ and $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top$ rather than their transposes.

An arbitrary $m \times n$ sparse matrix in CSC format is expressed as two vectors of indices, the row indices and the column pointers, and a numeric vector of the non-zero values. The elements of the row indices and the nonzeros are aligned and are ordered first by increasing column number then by increasing row number within column. The column pointers are a vector of size $n + 1$ giving the location of the start of each column in the vectors of row indices and nonzeros.

Because the number of nonzeros in each column of $\boldsymbol{Z}^\top$, and in each column of matrices derived from $\boldsymbol{Z}^\top$ (such as $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}^\top \boldsymbol{Z}^\top \boldsymbol{W}^{1/2}$) is constant, the vector of nonzeros in the CSC format can be viewed as a dense matrix, say $\boldsymbol{N}$, of size $(\sum_{i=1}^n p_i) \times n$. We do not need to store the column pointers because the columns of $\boldsymbol{Z}^\top$ correspond to columns of $\boldsymbol{N}$. All we need is $\boldsymbol{N}$, the dense matrix of nonzeros, and the row indices, which are derived from the grouping factor index vectors $\boldsymbol{i}_i, i = 1, \ldots, k$ and can also be arranged as a dense matrix of size $(\sum_{i=1}^n p_i) \times n$. Matrices like $\boldsymbol{Z}^\top$, with the property that there are the same number of nonzeros in each

column, are sometimes called *regular sparse column-oriented* (RSC) matrices.

4. Nonlinear optimization module

The objective function module produces a function which implements the penalized least squares algorithm for a particular mixed model. The next step is to optimize this function over the covariance parameters, $\boldsymbol{\theta}$, which is a nonlinear optimization problem. The **lme4** package separates the efficient computational linear algebra required to compute profiled likelihoods and deviances for a given value of $\boldsymbol{\theta}$ from the nonlinear optimization algorithms, which use general-purpose nonlinear optimizers.

One benefit of this separation is that it allows for experimentation with different nonlinear optimizers. Throughout the development of **lme4**, the default optimizers and control parameters have changed in response to feedback from users about both efficiency and convergence properties. **lme4** incorporates two built-in optimization choices, an implementation of the Nelder-Mead simplex algorithm adapted from Steven Johnson's `NLopt` C library (Johnson 2014) and a wrapper for Powell's BOBYQA algorithm, implemented in the **minqa** package Bates, Mullen, Nash, and Varadhan (2014) as a wrapper around Powell's original Fortran code (Powell 2009). More generally, **lme4** allows any user-specified optimizer that (1) can work with an objective function (i.e., does not require a gradient function to be specified), (2) allows box constraints on the parameters, and (3) conforms to some simple rules about argument names and structure of the output. An internal wrapper allows the use of the **optimx** package (Nash and Varadhan 2011), although the only optimizers provided via **optimx** that satisfy the constraints above are the `nlminb` and `L-BFGS-B` algorithms that are themselves wrappers around the versions provided in base R. Several other algorithms from Steven Johnson's `NLopt` package are also available via the **nloptr** wrapper package (e.g., alternate implementations of Nelder-Mead and BOBYQA, and Powell's COBYLA algorithm).

This flexibility assists with diagnosing convergence problems — it is easy to switch among several algorithms to determine whether the problem lies in a failure of the nonlinear optimization stage, as opposed to a case of model misspecification or unidentifiability or a problem with the underlying PLS algorithm. To date we have only observed PLS failures, which arise if $\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X} - \boldsymbol{R}_{ZX}^\top \boldsymbol{R}_{ZX}$ becomes singular during an evaluation of the objective function, with badly scaled problems (i.e., problems with continuous predictors that take a very large or very small numerical range of values).

The requirement for optimizers that can handle box constraints stems from our decision to parameterize the variance-covariance matrix in a constrained space, in order to allow for singular fits. In contrast to the approach taken in the **nlme** package (Pinheiro, Bates, DebRoy, Sarkar, and R Core Team 2014), which goes to some lengths to use an unconstrained variance-covariance parameterization (the *log-Cholesky* parameterization: Pinheiro and Bates (1996)), we instead use the Cholesky parameterization but require the elements of $\boldsymbol{\theta}$ corresponding to the diagonal elements of the Cholesky factor to be non-negative. With these constraints, the variance-covariance matrix is singular if and only if any of the diagonal elements is exactly zero. Singular fits are common in practical data-analysis situations, especially with small- to medium-sized data sets and complex variance-covariance models, so being able to fit a singular model is an advantage: when the best fitting model lies on the boundary of a constrained space, approaches that try to remove the constraints by fitting parameters on a transformed

scale will often give rise to convergence warnings as the algorithm tries to find a maximum on an asymptotically flat surface (Bolker *et al.* 2013).

In principle the likelihood surfaces we are trying to optimize over are smooth, but in practice using gradient information in optimization may or may not be worth the effort. In special cases, we have a closed-form solution for the gradients (Equations 45–47), but in general we will have to approximate them by finite differences, which is expensive and has limited accuracy. (We have considered using automatic differentiation approaches to compute the gradients more efficiently, but this strategy requires a great deal of additional machinery, and would have drawbacks in terms of memory requirements for large problems.) This is the primary reason for our switching to derivative-free optimizers such as BOBYQA and Nelder-Mead in the current version of **lme4**, although as discussed above derivative-based optimizers based on finite differencing are available as an alternative.

There is most likely further room for improvement in the nonlinear optimization module; for example, some speed-up could be gained by using parallel implementations of derivative-free optimizers that evaluated several trial points at once (Klein and Neira 2013). In practice users most often have optimization difficulties with poorly scaled or centred data — we are working to implement appropriate diagnostic tests and warnings to detect these situations.

5. Output module

Here we describe some of the methods in **lme4** for exploring fitted linear mixed models (Table 8), which are represented as objects of the `S4` class `lmerMod`. We begin by describing the theory underlying these methods (Section 5.1) and then continue the `sleepstudy` example introduced in Section 1.2 to illustrate these ideas in practice.

5.1. Theory underlying the output module

Covariance matrix of the fixed effect coefficients

In the `lm` function, the variance-covariance matrix of the coefficients is the inverse of the cross product of the model matrix, times the residual variance (Chambers 1993). The inverse cross-product matrix is computed by first inverting the upper triangular matrix resulting from the QR decomposition of the model matrix, and then taking its cross-product.

$$\mathrm{Var}_{\boldsymbol{\theta},\sigma}\left(\begin{bmatrix}\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\mathrm{obs}}}\\ \hat{\boldsymbol{\beta}}\end{bmatrix}\right)=\sigma^2\begin{bmatrix}\boldsymbol{L}_{\boldsymbol{\theta}}^\top & \boldsymbol{R}_{ZX}\\ \mathbf{0} & \boldsymbol{R}_X\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{L}_{\boldsymbol{\theta}} & \mathbf{0}\\ \boldsymbol{R}_{ZX}^\top & \boldsymbol{R}_X^\top\end{bmatrix}^{-1} \tag{53}$$

Because of normality, the marginal covariance matrix of $\hat{\boldsymbol{\beta}}$ is just the lower-right p-by-p block of $\mathrm{Var}_{\boldsymbol{\theta},\sigma}\left(\begin{bmatrix}\boldsymbol{\mu}_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\mathrm{obs}}}\\ \hat{\boldsymbol{\beta}}\end{bmatrix}\right)$. This lower-right block is

$$\mathrm{Var}_{\boldsymbol{\theta},\sigma}(\hat{\boldsymbol{\beta}})=\sigma^2\boldsymbol{R}_X^{-1}(\boldsymbol{R}_X^\top)^{-1} \tag{54}$$

which follows from the Schur complement identity (Theorem 1.2 of Horn and Zhang (2005)). This matrix can be extracted from a fitted `merMod` object as,

```
R> RX <- getME(fm1, "RX")
R> sigma2 <- sigma(fm1)^2
R> sigma2 * chol2inv(RX)

           [,1]      [,2]
[1,] 46.574573 -1.451097
[2,] -1.451097  2.389463
```

which could be computed with **lme4** as `vcov(fm1)`.

The square-root diagonal of this covariance matrix contains the estimates of the standard errors of fixed effects coefficients. These standard errors are used to construct Wald confidence intervals with `confint(., method = "Wald")`. Such confidence intervals are approximate, and depend on the assumption that the likelihood profile of the fixed effects is linear on the ζ scale (Section 5.1.2).

Profiling

The theory of likelihood profiles is straightforward: the deviance (or likelihood) profile, $-2\mathcal{L}_p()$, for a focal model parameter P is the minimum value of the deviance conditioned on a particular value of P. For each parameter of interest, our goal is to evaluate the deviance profile for many points — optimizing over all of the non-focal parameters each time — over a wide enough range and with high enough resolution to evaluate the shape of the profile (in particular, whether it is quadratic, which would allow use of Wald confidence intervals and tests) and to find the values of P such that $-2\mathcal{L}_p(P) = -2\mathcal{L}(\widehat{P}) + \chi^2(1-\alpha)$, which represent the profile confidence intervals. While profile confidence intervals rely on the asymptotic distribution of the minimum deviance, this is a much weaker assumption than the log-quadratic likelihood surface required by Wald tests.

An additional challenge in profiling arises when we want to compute profiles for quantities of interest that are not parameters of our PLS function. We have two problems in using the deviance function defined above to profile linear mixed models. First, a parameterization of the random effects variance-covariance matrix in terms of standard deviations and correlations, or variances and covariances, is much more familiar to users, and much more relevant as output of a statistical model, than the parameters, $\boldsymbol{\theta}$, of the relative covariance factor — users are interested in inferences on variances or standard deviations, not on $\boldsymbol{\theta}$. Second, the fixed-effect coefficients and the residual standard deviation, both of which are also of interest to users, are profiled out (in the sense used above) of the deviance function (Section 3.4), so we have to find a strategy for estimating the deviance for values of $\boldsymbol{\beta}$ and σ^2 other than the profiled values.

To handle the first problem we create a new version of the deviance function that first takes a vector of standard deviations (and correlations), and a value of the residual standard deviation, and maps them to a $\boldsymbol{\theta}$ vector, and then computes the PLS as before; it uses the specified residual standard deviation rather than the PLS estimate of the standard deviation (Equation 33) in the deviance calculation. We compute a profile likelihood for the fixed-effect parameters, which are profiled out of the deviance calculation, by adding an offset to the linear predictor for the focal element of $\boldsymbol{\beta}$. The resulting function is not useful for general nonlinear optimization — one can easily wander into parameter regimes corresponding to in-

feasible (non-positive semidefinite) variance-covariance matrices — but it serves for likelihood profiling, where one focal parameter is varied at a time and the optimization over the other parameters is likely to start close to an optimum.

In practice, the `profile` method systematically varies the parameters in a model, assessing the best possible fit that can be obtained with one parameter fixed at a specific value and comparing this fit to the globally optimal fit, which is the original model fit that allowed all the parameters to vary. The models are compared according to the change in the deviance, which is the likelihood ratio test statistic. We apply a signed square root transformation to this statistic and plot the resulting function, which we call the *profile zeta function* or ζ, versus the parameter value. The signed aspect of this transformation means that ζ is positive where the deviation from the parameter estimate is positive and negative otherwise, leading to a monotonically increasing function which is zero at the global optimum. A ζ value can be compared to the quantiles of the standard normal distribution. For example, a 95% profile deviance confidence interval on the parameter consists of the values for which $-1.96 < \zeta < 1.96$. Because the process of profiling a fitted model can be computationally intensive — it involves refitting the model many times — one should exercise caution with complex models fit to large data sets.

The standard approach to this computational challenge is to compute ζ at a limited number of parameter values, and to fill in the gaps by fitting an interpolation spline. Often one is able to invert the spline to obtain a function from ζ to the focal parameter, which is necessary in order to construct profile confidence intervals. However, even if the points themselves are monotonic, it is possible to obtain non-monotonic interpolation curves. In such a case, **lme4** falls back to linear interpolation, with a warning.

The last part of the technical specification for computing profiles is deciding on a strategy for choosing points to sample. In one way or another one wants to start at the estimated value for each parameter and work outward either until a constraint is reached (i.e., a value of 0 for a standard deviation parameter or a value of ± 1 for a correlation parameter), or until a sufficiently large deviation from the minimum deviance is attained. **lme4**'s profiler chooses a cutoff ϕ based on the $1 - \alpha_{\max}$ critical value of the χ^2 distribution with a number of degrees of freedom equal to the *total* number of parameters in the model, where $\alpha_{\max}$ is set to 0.05 by default. The profile computation initially adjusts the focal parameter p_i by an amount $\epsilon = 1.01\hat{p}_i$ from its ML or REML estimate $\hat{p}_i$ (or by $\epsilon = 0.001$ if $\hat{p}_i$ is zero, as in the case of a singular variance-covariance model). The local slope of the likelihood profile $(\zeta(\hat{p}_i + \epsilon) - \zeta(\hat{p}_i))/\epsilon$ is used to pick the next point to evaluate, extrapolating the local slope to find a new ϵ that would be expected to correspond to the desired step size $\Delta\zeta$ (equal to $\phi/8$ by default, so that 16 points would be used to cover the profile in both directions if the log-likelihood surface were exactly quadratic). Some fail-safe testing is done to ensure that the step chosen is always positive, and less than a maximum; if a deviance is ever detected that is lower than that of the ML deviance, which can occur if the initial fit was wrong due to numerical problems, the profiler issues an error and stops.

Parametric bootstrapping

To avoid the asymptotic assumptions of the likelihood ratio test, at the cost of greatly increased computation time, one can estimate confidence intervals by parametric bootstrapping — that is, by simulating data from the fitted model, refitting the model, and extracting the

new estimated parameters (or any other quantities of interest). This task is quite straightforward, since there is already a `simulate` method, and a `refit` function which re-estimates the (RE)ML parameters for new data, starting from the previous (RE)ML estimates and re-using the previously computed model structures (including the fill-reducing permutation) for efficiency. The `bootMer` function is thus a fairly thin wrapper around a `simulate`/`refit` loop, with a small amount of additional logic for parallel computation and error-catching. (Some of the ideas of `bootMer` are adapted from `merBoot`, a more ambitious framework for bootstrapping **lme4** model fits which unfortunately seems to be unavailable at present (Sánchez-Espigares and Ocaña 2009).)

Conditional variances of random effects

It is useful to clarify that the conditional covariance concept in **lme4** is based on a simplification of the linear mixed model. In particular, we simplify the model by assuming that the quantities, $\boldsymbol{\beta}$, $\boldsymbol{\Lambda}_{\boldsymbol{\theta}}$, and σ, are known (i.e., set at their estimated values). In fact, the only way to define the conditional covariance is at fixed parameter values. Our approximation here is using the estimated parameter values for the unknown "true" parameter values. In this simplified case, $\mathcal{U}$ is the only quantity in the statistical model that is both random and unknown.

Given this simplified linear mixed model, a standard result in Bayesian linear regression modelling (Gelman *et al.* 2013) implies that the conditional distribution of the spherical random effects given the observed response vector is Gaussian,

$$(\mathcal{U}|\mathcal{Y} = \boldsymbol{y}_{\text{obs}}) \sim \mathcal{N}(\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}}, \widehat{\sigma}^2\boldsymbol{V}) \tag{55}$$

where,

$$\boldsymbol{V} = \left(\boldsymbol{\Lambda}_{\widehat{\boldsymbol{\theta}}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\boldsymbol{Z}\boldsymbol{\Lambda}_{\widehat{\boldsymbol{\theta}}} + \boldsymbol{I}_q\right)^{-1} = \left(\boldsymbol{L}_{\widehat{\boldsymbol{\theta}}}^{-1}\right)^{\top}\left(\boldsymbol{L}_{\widehat{\boldsymbol{\theta}}}^{-1}\right) \tag{56}$$

is the unscaled conditional variance and,

$$\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}} = \boldsymbol{V}\boldsymbol{\Lambda}_{\widehat{\boldsymbol{\theta}}}^{\top}\boldsymbol{Z}^{\top}\boldsymbol{W}\left(\boldsymbol{y}_{\text{obs}} - \boldsymbol{o} - X\widehat{\boldsymbol{\beta}}\right) \tag{57}$$

is the conditional mean/mode. Note that in practice the inverse in Equation 56 does not get computed directly, but rather an efficient method is used that exploits the sparse structures.

The random effects coefficient vector, $\mathbf{b}$, is often of greater interest. The conditional covariance matrix of $\mathcal{B}$ may be expressed as,

$$\widehat{\sigma}^2\boldsymbol{\Lambda}_{\widehat{\boldsymbol{\theta}}}\boldsymbol{V}\boldsymbol{\Lambda}_{\widehat{\boldsymbol{\theta}}}^{\top} \tag{58}$$

The hat matrix

The hat matrix, $\boldsymbol{H}$, is a useful object in linear model diagnostics (Cook and Weisberg 1982). In general, $\boldsymbol{H}$ relates the observed and fitted response vectors, but we specifically define it as,

$$\left(\boldsymbol{\mu}_{\mathcal{Y}|\mathcal{U}=\boldsymbol{u}} - \boldsymbol{o}\right) = \boldsymbol{H}\left(\boldsymbol{y}_{\text{obs}} - \boldsymbol{o}\right) \tag{59}$$

To find $\boldsymbol{H}$ we note that,

$$\left(\boldsymbol{\mu}_{\mathcal{Y}|\mathcal{U}=\boldsymbol{u}} - \boldsymbol{o}\right) = \begin{bmatrix}\boldsymbol{Z}\boldsymbol{\Lambda} & \boldsymbol{X}\end{bmatrix}\begin{bmatrix}\mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}} \\ \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}\end{bmatrix} \tag{60}$$

Next we get an expression for $\begin{bmatrix} \mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}} \\ \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}} \end{bmatrix}$ by solving the normal equations (Equation 17),

$$\begin{bmatrix} \mu_{\mathcal{U}|\mathcal{Y}=\boldsymbol{y}_{\text{obs}}} \\ \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}_{\boldsymbol{\theta}}^{\top} & \boldsymbol{R}_{ZX} \\ \boldsymbol{0} & \boldsymbol{R}_X \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{L}_{\boldsymbol{\theta}} & \boldsymbol{0} \\ \boldsymbol{R}_{ZX}^{\top} & \boldsymbol{R}_X^{\top} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top} \boldsymbol{Z}^{\top} \\ \boldsymbol{X}^{\top} \end{bmatrix} \boldsymbol{W}(\boldsymbol{y}_{\text{obs}} - \boldsymbol{o}) \tag{61}$$

By the Schur complement identity (Horn and Zhang 2005),

$$\begin{bmatrix} \boldsymbol{L}_{\boldsymbol{\theta}}^{\top} & \boldsymbol{R}_{ZX} \\ \boldsymbol{0} & \boldsymbol{R}_X \end{bmatrix}^{-1} = \begin{bmatrix} \left(\boldsymbol{L}_{\boldsymbol{\theta}}^{\top}\right)^{-1} & -\left(\boldsymbol{L}_{\boldsymbol{\theta}}^{\top}\right)^{-1} \boldsymbol{R}_{ZX} \boldsymbol{R}_X^{-1} \\ \boldsymbol{0} & \boldsymbol{R}_X^{-1} \end{bmatrix} \tag{62}$$

Therefore, we may write the hat matrix as,

$$\boldsymbol{H} = (\boldsymbol{C}_L^{\top} \boldsymbol{C}_L + \boldsymbol{C}_R^{\top} \boldsymbol{C}_R) \tag{63}$$

where,

$$\boldsymbol{L}_{\boldsymbol{\theta}} \boldsymbol{C}_L = \boldsymbol{\Lambda}_{\boldsymbol{\theta}}^{\top} \boldsymbol{Z}^{\top} \boldsymbol{W}^{1/2} \tag{64}$$

and,

$$\boldsymbol{R}_X^{\top} \boldsymbol{C}_R = \boldsymbol{X}^{\top} \boldsymbol{W}^{1/2} - \boldsymbol{R}_{ZX}^{\top} \boldsymbol{C}_L \tag{65}$$

The trace of the hat matrix is often used as a measure of the effective degrees of freedom (e.g., Vaida and Blanchard (2005)). Using a relationship between the trace and vec operators, the trace of $\boldsymbol{H}$ may be expressed as,

$$\operatorname{tr}(\boldsymbol{H}) = \|\operatorname{vec}(\boldsymbol{C}_L)\|^2 + \|\operatorname{vec}(\boldsymbol{C}_R)\|^2 \tag{66}$$

5.2. Using the output module

The user interface for the output module largely consists of a set of methods (Table 8) for objects of class `merMod`, which is the class of objects returned by `lmer` (In addition to these methods, the `getME` function can be used to extract various objects from a fitted mixed model in **lme4**.) Here we illustrate the use of several of these methods.

Updating fitted mixed models

To illustrate the `update` method for `merMod` objects we construct a random intercept only model. This task could be done in several ways, but we choose to first remove the random effects term `(Days | Subject)` and add a new term with a random intercept,

```
R> fm3 <- update(fm1, . ~ . - (Days | Subject) + (1 | Subject))
R> formula(fm3)

Reaction ~ Days + (1 | Subject)
```

Model summary and associated accessors

Generic (Section)	Brief description of return value
`anova` (5.2.4)	Decomposition of fixed-effects contributions or model comparison.
`as.function`	Function returning profiled deviance or REML criterion.
`coef`	Sum of the random and fixed effects for each level.
`confint` (5.2.6)	Confidence intervals on linear mixed-model parameters.
`deviance` (5.2.2)	Minus twice maximum log-likelihood. (Use `REMLcrit` for the REML criterion.)
`df.residual`	Residual degrees of freedom.
`drop1`	Drop allowable single terms from the model.
`extractAIC`	Generalized Akaike information criterion
`fitted`	Fitted values given conditional modes (Equation 13).
`fixef` (5.2.2)	Estimates of the fixed effects coefficients, $\widehat{\boldsymbol{\beta}}$
`formula` (2.2.1)	Mixed-model formula of fitted model.
`logLik`	Maximum log-likelihood.
`model.frame`	Data required to fit the model.
`model.matrix`	Fixed effects model matrix, $\boldsymbol{X}$.
`ngrps (5.2.2)`	Number of levels in each grouping factor.
`nobs (5.2.2)`	Number of observations.
`plot`	Diagnostic plots for mixed-model fits.
`predict` (5.2.8)	Various types of predicted values.
`print`	Basic printout of mixed-model objects.
`profile` (5.1.2)	Profiled likelihood over various model parameters.
`ranef` (5.2.2)	Conditional modes of the random effects.
`refit` (5.1.3)	A model (re)fitted to a new set of observations of the response variable.
`refitML` (5.2.4)	A model (re)fitted by maximum likelihood.
`residuals` (5.2.2)	Various types of residual values.
`sigma` (5.2.2)	Residual standard deviation.
`simulate` (5.2.8)	Simulated data from a fitted mixed model.
`summary` (5.2.2)	Summary of a mixed model.
`terms`	Terms representation of a mixed model.
`update` (5.2.1)	An updated model using a revised formula or other arguments.
`VarCorr` (5.2.2)	Estimated random-effects variances, standard deviations, and correlations.
`vcov` (5.2.2)	Covariance matrix of the fixed effect estimates.
`weights`	Prior weights used in model fitting.

Table 8: List of currently available methods for objects of the class `merMod`.

The `summary` method for `merMod` objects provides information about the model fit. Here we consider the output of `summary(fm1)` in four steps. The first few lines of output indicate that the model was fitted by REML as well as the value of the REML criterion (Equation 39) at convergence (which may also be extracted using `deviance(fm1)`). The beginning of the summary also reproduces the model formula and the scaled Pearson residuals,

```
Linear mixed model fit by REML ['lmerMod']
Formula: Reaction ~ Days + (Days | Subject)
   Data: sleepstudy

REML criterion at convergence: 1743.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.9536 -0.4634  0.0231  0.4634  5.1793
```

This information may also be obtained using standard accessor functions,

```
R> formula(fm1)
R> REMLcrit(fm1)
R> quantile(residuals(fm1, "pearson", scaled = TRUE))
```

The second piece of `summary` output provides information regarding the random effects and residual variation,

```
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Subject  (Intercept) 612.09   24.740
          Days         35.07    5.922   0.07
 Residual             654.94   25.592
Number of obs: 180, groups:  Subject, 18
```

which can also be accessed using,

```
R> print(vc <- VarCorr(fm1), comp = c("Variance", "Std.Dev."))
R> nobs(fm1)
R> ngrps(fm1)
R> sigma(fm1)
```

The print method for `VarCorr` hides the internal structure of `VarCorr.merMod` objects. The internal structure of an object of this class is a list of matrices, one for each random effects term. The standard deviations and correlation matrices for each 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 programming use, these objects can be summarized differently using their `as.data.frame` method,

```
R> as.data.frame(VarCorr(fm1))

       grp        var1 var2       vcov       sdcor
1  Subject (Intercept) <NA> 612.089963 24.74045195
2  Subject        Days <NA>  35.071661  5.92213312
3  Subject (Intercept) Days   9.604306  0.06555113
4 Residual        <NA> <NA> 654.941028 25.59181564
```

which contains one row for each variance or covariance parameter. The `grp` column gives the grouping factor associated with this parameter. The `var1` and `var2` columns give the names of the variables associated with the parameter (`var2` is `<NA>` unless it is a covariance parameter). The `vcov` column gives the variances and covariances, and the `sdcor` column gives these numbers on the standard deviation and correlation scales.

The next chunk of output gives the fixed effect estimates,

```
Fixed effects:
            Estimate Std. Error t value
(Intercept)  251.405      6.825   36.84
Days          10.467      1.546    6.77
```

Note that there are no p values (see Section 5.2.5). The fixed effect point estimates may be obtained separately via `fixef(fm1)`. Conditional modes of the random effects coefficients can be obtained with `ranef` (see section 5.1.4 for information on the theory). Finally, we have the correlations between the fixed effect estimates

```
Correlation of Fixed Effects:
     (Intr)
Days -0.138
```

The full variance-covariance matrix of the fixed effects estimates can be obtained in the usual R way with the `vcov` method (Section 5.1.1). Alternatively, `coef(summary(fm1))` will return the full fixed-effects parameter table as shown in the summary.

Diagnostic plots

lme4 provides tools for generating most of the standard graphical diagnostic plots (with the exception of those incorporating influence measures, e.g., Cook's distance and leverage plots), in a way modeled on the diagnostic graphics of **nlme** (Pinheiro and Bates 2000). In particular, the familiar `plot` method in base R for linear models (objects of class `lm`) can be used to create the standard fitted vs. residual plots,

```
R> plot(fm1, type = c("p", "smooth"))
```

scale-location plots,

```
R> plot(fm1, sqrt(abs(resid(.))) ~ fitted(.),
+       type = c("p", "smooth"))
```

and quantile-quantile plots, (from **lattice**)

```
R> qqmath(fm1, id = 0.05)
```

In contrast to `plot.lm`, these scale-location and Q-Q plots are based on raw rather than standardized residuals.

In addition to these standard diagnostic plots, which examine the validity of various assumptions (linearity, homoscedasticity, normality) at the level of the residuals, one can also use the `dotplot` and `qqmath` methods for the conditional modes (i.e., `ranef.mer` objects generated by `ranef(fit)`) to check for interesting patterns and normality in the conditional modes. **lme4** does not provide influence diagnostics, but these (and other useful diagnostic procedures) are available in the dependent packages **HLMdiag** and **influence.ME** (Loy and Hofmann 2014; Nieuwenhuis, Te Grotenhuis, and Pelzer 2012).

Finally, *posterior predictive simulation* (Gelman and Hill 2006) is a generally useful diagnostic tool, adapted from Bayesian methods, for exploring model fit. The user picks some summary statistic of interest. They then compute the summary statistic for an ensemble of simulations (Section 5.2.8), and see where the observed data falls within the simulated distribution; if the observed data is extreme, we might conclude that the model is a poor representation of reality. For example, using the sleep study fit and choosing the interquartile range of the reaction times as the summary statistic:

```
R> iqrvec <- sapply(simulate(fm1, 1000), IQR)
R> obsval <- IQR(sleepstudy$Reaction)
R> post.pred.p <- mean(obsval >= c(obsval, iqrvec))
```

The (one-tailed) posterior predictive p value of 0.78 indicates that the model represents the data adequately, at least for this summary statistic. In contrast to the full Bayesian case, the procedure described here does not allow for the uncertainty in the estimated parameters. However, it should be a reasonable approximation when the residual variation is large.

Sequential decomposition and model comparison

Objects of class `merMod` have an `anova` method which returns F statistics corresponding to the sequential decomposition of the contributions of fixed-effects terms. In order to illustrate this sequential ANOVA decomposition, we fit a model with orthogonal linear and quadratic `Days` terms,

```
R> fm4 <- lmer(Reaction ~ polyDays[ , 1] + polyDays[ , 2] +
+              (polyDays[ , 1] + polyDays[ , 2] | Subject),
+              within(sleepstudy, polyDays <- poly(Days, 2)))
R> anova(fm4)

Analysis of Variance Table
```

```
              Df  Sum Sq Mean Sq F value
polyDays[, 1]  1 23874.5 23874.5 46.0756
polyDays[, 2]  1   340.3   340.3  0.6567
```

The relative magnitudes of the two sums of squares indicate that the quadratic term explains much less variation than the linear term. Furthermore, the magnitudes of the two F statistics strongly suggest significance of the linear term, but not the quadratic term. Notice that this is only an informal assessment of significance as there are no p values associated with these F statistics, an issue discussed in more detail in Section 5.2.5.

To understand how these quantities are computed, let $\boldsymbol{R}_i$ contain the rows of $\boldsymbol{R}_X$ (Equation 18) associated with the ith fixed-effects term. Then the sum of squares for term i is,

$$SS_i = \widehat{\boldsymbol{\beta}}^\top \boldsymbol{R}_i^\top \boldsymbol{R}_i \widehat{\boldsymbol{\beta}} \tag{67}$$

If DF_i is the number of columns in $\boldsymbol{R}_i$, then the F statistic for term i is,

$$F_i = \frac{SS_i}{\widehat{\sigma}^2 DF_i} \tag{68}$$

For multiple arguments, the `anova` method returns model comparison statistics,

```
R> anova(fm1, fm2, fm3)

refitting model(s) with ML (instead of REML)

Data: sleepstudy
Models:
fm3: Reaction ~ Days + (1 | Subject)
fm2: Reaction ~ Days + ((1 | Subject) + (0 + Days | Subject))
fm1: Reaction ~ Days + (Days | Subject)
    Df    AIC    BIC  logLik deviance   Chisq Chi Df Pr(>Chisq)
fm3  4 1802.1 1814.8 -897.04   1794.1
fm2  5 1762.0 1778.0 -876.00   1752.0 42.0754      1  8.782e-11
fm1  6 1763.9 1783.1 -875.97   1751.9  0.0639      1     0.8004
```

The output shows χ^2 statistics representing the difference in deviance between successive models, as well as p values based on likelihood ratio test comparisons. In this case, the sequential comparison shows that adding a linear effect of time uncorrelated with the intercept leads to an enormous and significant drop in deviance (Δdeviance ≈ 42, $p \approx 10^{-10}$), while the further addition of correlation between the slope and intercept leads to a trivial and non-significant change in deviance (Δdeviance ≈ 0.06). For objects of class `lmerMod` the default behavior is to refit the models with ML if fitted with `REML = TRUE`, which is necessary in order to get sensible answers when comparing models that differ in their fixed effects; this can be controlled via the `refit` argument.

Computing p values

One of the more controversial design decisions of **lme4** has been to omit the output of p values associated with sequential ANOVA decompositions of fixed effects. The absence of analytical results for null distributions of parameter estimates in complex situations (e.g., unbalanced or partially crossed designs) is a long-standing problem in mixed-model inference. While the null distributions (and the sampling distributions of non-null estimates) are asymptotically normal, these distributions are not t distributed for finite size samples — nor are the corresponding null distributions of differences in scaled deviances F distributed. Thus approximate methods for computing the approximate degrees of freedom for t distributions, or the denominator degrees of freedom for F statistics (Satterthwaite 1946; Kenward and Roger 1997), are at best *ad hoc* solutions.

However, computing finite-size-corrected p values is sometimes necessary. Therefore, although the package does not provide them (except via parametric bootstrapping, Section 5.1.3), we have provided a help page to guide users in finding appropriate methods:

```
R> help("pvalues")
```

This help page provides pointers to other packages that provide machinery for calculating p values associated with `merMod` objects. It also suggests framing the inference problem as a likelihood ratio test (achieved by supplying multiple `merMod` objects to the `anova` method (Section 5.2.4), as well as alternatives to p values such as confidence intervals (Section 5.2.6).

Previous versions of **lme4** provided the `mcmcsamp` function, which generated a Markov chain Monte Carlo sample from the posterior distribution of the parameters (assuming flat 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, `mcmcsamp` has been withdrawn.

Computing confidence intervals

As described above (Sections 5.1.1, 5.1.2, 5.1.3), **lme4** provides confidence intervals (using `confint`) via Wald approximations (for fixed-effect parameters only), likelihood profiling, or parametric bootstrapping (the `boot.type` argument selects the bootstrap confidence interval type).

As is typical for computationally intensive profile confidence intervals in R, the intervals can be computed either directly from fitted `merMod` objects (in which case profiling is done as an interim step), or from a previously computed likelihood profile (of class `thpr`, for "theta profile"). Parametric bootstrapping confidence intervals use a thin wrapper around the `bootMer` function that passes the results to `boot.ci` from the **boot** package (Canty and Ripley 2013; Davison and Hinkley 1997) for confidence interval calculation.

In the running sleep study examples, the 95% confidence intervals estimated by all three methods are quite similar. The largest change is a 26% difference in confidence interval widths between profile and parametric bootstrap methods for the correlation between the intercept and slope random effects ($\{-0.54,0.98\}$ vs. $\{-0.48,0.68\}$).

General profile zeta and related plots

lme4 provides several functions (built on **lattice** graphics) for plotting the profile zeta functions (Section 5.1.2) and other related quantities.

- The *profile zeta* plot (Figure 2; `xyplot`) is simply a plot of the profile zeta function for each model parameter; linearity of this plot for a given parameter implies that the likelihood profile is quadratic (and thus that Wald approximations would be reasonably accurate).
- The *profile density plot* (Figure 3; `densityplot`) displays an approximation of the probability density function of the sampling distribution for each parameter. These densities are derived by setting the cumulative distribution function (c.d.f) to be $\Phi(\zeta)$ where Φ is the c.d.f. of the standard normal distribution. This is not quite the same as evaluating the distribution of the estimator of the parameter, which for mixed models can be very difficult, but it gives a reasonable approximation. If the profile zeta plot is linear, then the profile density plot will be Gaussian.
- The *profile pairs plot* (Figure 4; `splom`) gives an approximation of the two-dimensional profiles of pairs of parameters, interpolated from the univariate profiles as described in Bates and Watts (1988, Chapter 6). The profile pairs plot shows two-dimensional 50%, 80%, 90%, 95% and 99% marginal confidence regions based on the likelihood ratio, as well as the *profile traces*, which indicate the conditional estimates of each parameter for fixed values of the other parameter. While panels above the diagonal show profiles with respect to the original parameters (with random effects parameters on the standard deviation/correlation scale, as for all profile plots), the panels below the diagonal show plots on the (ζ_i, ζ_j) scale. The below-diagonal panels allow us to see distortions from an elliptical shape due to nonlinearity of the traces, separately from the one-dimensional distortions caused by a poor choice of scale for the parameter. The ζ scales provide, in some sense, the best possible set of single-parameter transformations for assessing the contours. On the ζ scales the extent of a contour on the horizontal axis is exactly the same as the extent on the vertical axis and both are centered about zero.

For users who want to build their own graphical displays of the profile, there is an `as.data.frame` method that converts profile (`thpr`) objects to a more convenient format.

Computing fitted and predicted values; simulating

Because mixed models involve random coefficients, one must always clarify whether predictions should be based on the marginal distribution of the response variable or on the distribution that is conditional on the modes of the random effects (Equation 12). The `fitted` method retrieves fitted values that are conditional on all of the modes of the random effects; the `predict` method returns the same values by default, but also allows for predictions to be made conditional on different sets of random effects. For example, if the `re.form` argument is set to `NULL` (the default), then the predictions are conditional on all random effects in the model; if `re.form` is `~0` or `NA`, then the predictions are made at the population level (all random effect values set to zero). In models with multiple random effects, the user can give `re.form` as a formula that specifies which random effects are conditioned on.

lme4 also provides a `simulate` method, which allows similar flexibility in conditioning on random effects; in addition it allows the user to choose (via the `use.u` argument) between conditioning on the fitted conditional modes or choosing a new set of simulated conditional modes (zero-mean Normal deviates chosen according to the estimated random effects variance-covariance matrices). Finally, the `simulate` method has a method for `formula` objects, which

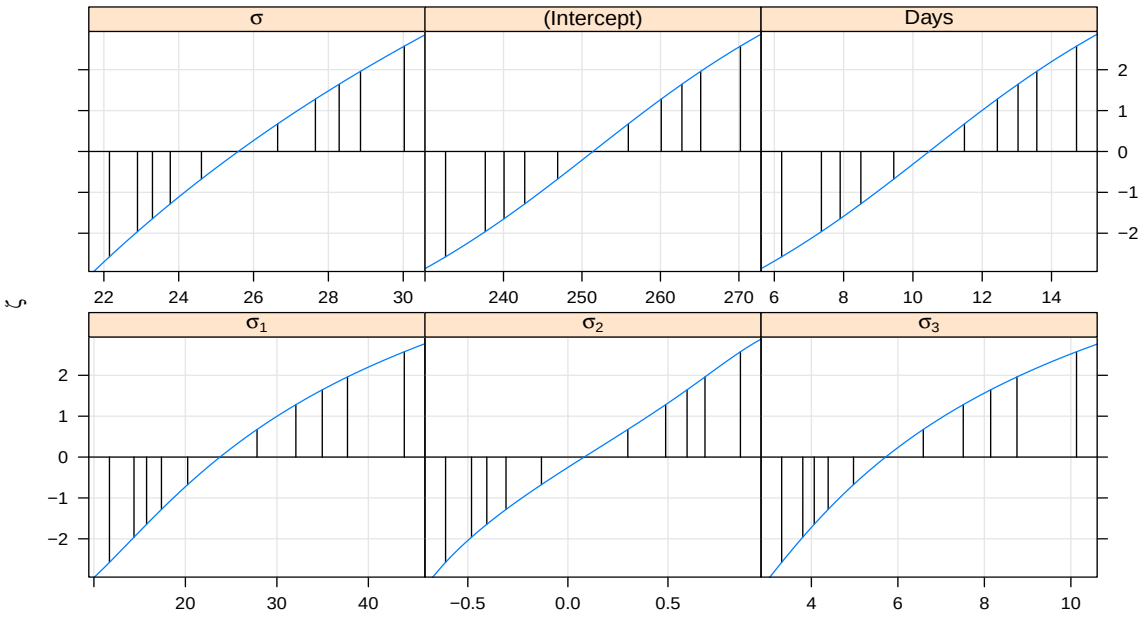


Figure 2: Profile zeta plot: `xyplot(prof.obj)`

allows for *de novo* simulation in the absence of a fitted model. In this case, the user must specify the random effects ($\boldsymbol{\theta}$), fixed effects ($\boldsymbol{\beta}$), and residual standard deviation (σ) parameters via the `newparams` argument. The standard simulation method (based on a `merMod` object) is the basis of parametric bootstrapping (Section 5.1.3) and posterior predictive simulation (Section 5.2.3); *de novo* simulation based on a formula provides a flexible framework for power analysis.

6. Conclusion

Mixed models are an extremely useful but computationally intensive approach. Computational limitations are especially important because mixed models are commonly applied to moderately large data sets (10^4–10^6 observations). By developing an efficient, general, and (now) well-documented platform for fitted mixed models in R, we hope to provide both a practical tool for end users interested in analyzing data and a reusable, modular framework for downstream developers interested in extending the class of models that can be easily and efficiently analyzed in R.

We have learned much about linear mixed models in the process of developing **lme4**, both from our own attempts to develop robust and reliable procedures and from the broad community of **lme4** users; we have attempted to describe many of these lessons here. In moving forward, our main priorities are (1) to maintain the reference implementation of **lme4** on CRAN, developing relatively few new features; (2) to improve the flexibility, efficiency and scalability of mixed-model analysis across multiple compatible implementations, including both the **MixedModels** package for Julia and the experimental `flexLambda` branch of **lme4**.

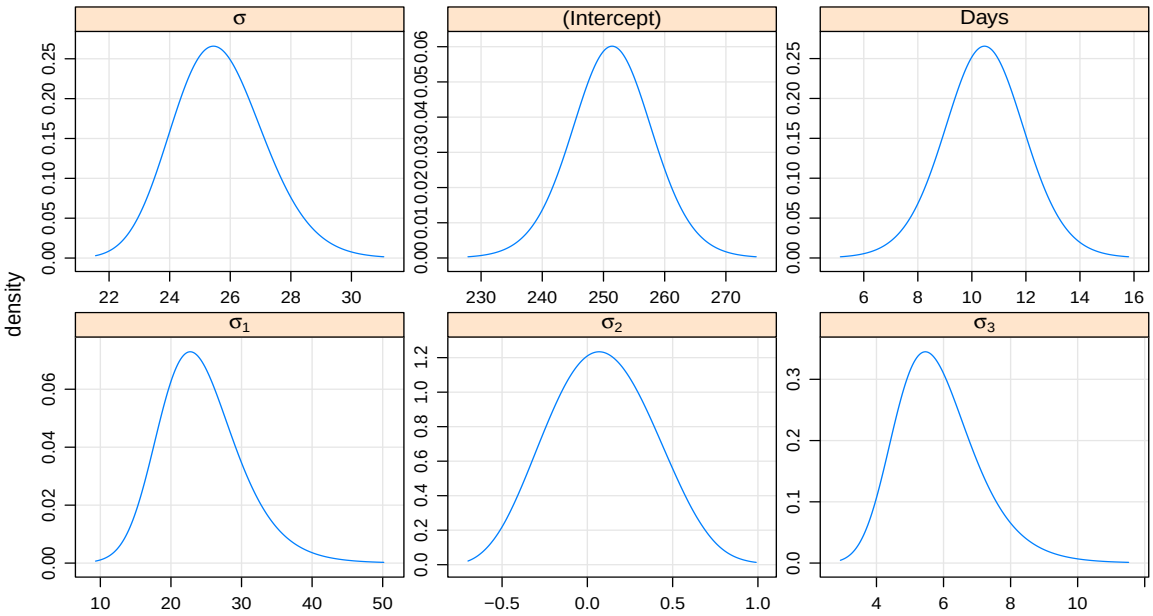


Figure 3: Profile density plot: `densityplot(prof.obj)`

Acknowledgements

Rune Haubo Christensen, Henrik Singmann, Fabian Scheipl, Vincent Dorie, and Bin Dai contributed ideas and code to the current version of **lme4**; the large community of **lme4** users has exercised the software, discovered bugs, and made many useful suggestions. Søren Højsgaard participated in useful discussions and Xi Xia and Christian Zingg exercised the code and reported problems. We would like to thank the Banff International Research Station for hosting a working group on **lme4**, and an NSERC Discovery grant and NSERC postdoctoral fellowship for funding.

References

Bates D, Maechler M (2014). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.1-3, URL http://CRAN.R-project.org/package=Matrix.

Bates D, Mullen KM, Nash JC, Varadhan R (2014). *minqa: Derivative-free optimization algorithms by quadratic approximation*. R package version 1.2.3, URL http://CRAN.R-project.org/package=minqa.

Bates DM, DebRoy S (2004). "Linear mixed models and penalized least squares." *Journal of Multivariate Analysis*, **91**(1), 1–17. doi:10.1016/j.jmva.2004.04.013.

Bates DM, Watts DG (1988). *Nonlinear Regression Analysis and Its Applications*. Wiley, Hoboken, NJ. ISBN 0-471-81643-4.

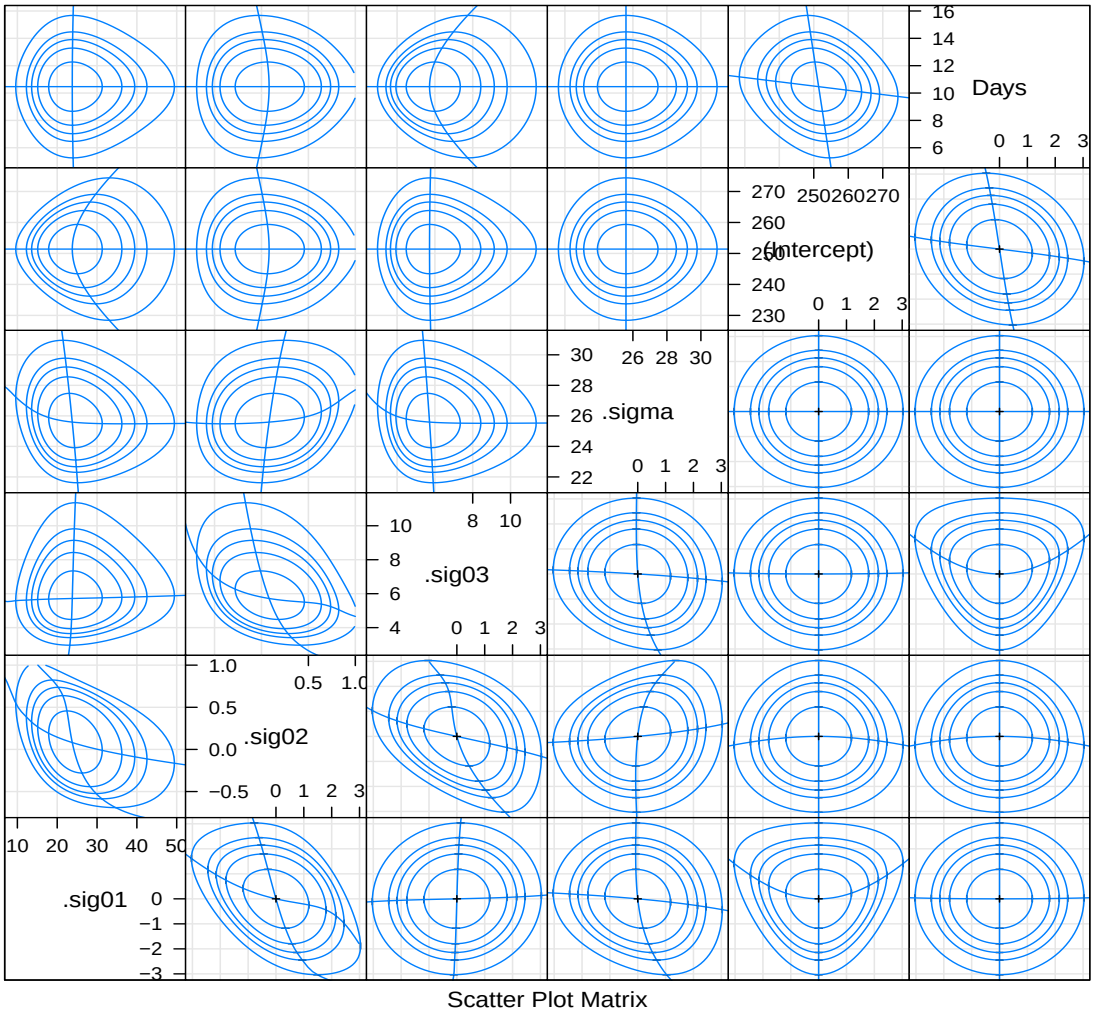


Figure 4: Profile pairs plot: `splom(prof.obj)`

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

Bolker BM, Gardner B, Maunder M, Berg CW, Brooks M, Comita L, Crone E, Cubaynes S, Davies T, de Valpine P, Ford J, Gimenez O, Kéry M, Kim EJ, Lennert-Cody C, Magnusson A, Martell S, Nash J, Nielsen A, Regetz J, Skaug H, Zipkin E (2013). "Strategies for fitting nonlinear ecological models in R, AD Model Builder, and BUGS." *Methods in Ecology and Evolution*, **4**(6), 501–512. ISSN 2041210X. doi:10.1111/2041-210X.12044. URL http://doi.wiley.com/10.1111/2041-210X.12044.

Canty A, Ripley B (2013). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-9, URL http://CRAN.R-project.org/package=boot.

Chambers JM (1993). "Linear Models." In JM Chambers, TJ Hastie (eds.), *Statistical Models in S*, chapter 4, pp. 95–144. Chapman & Hall.

Chen Y, Davis TA, Hager WW, Rajamanickam S (2008). "Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate." *ACM Trans. Math. Softw.*, **35**(3), 22:1–22:14. ISSN 0098-3500. doi:10.1145/1391989.1391995. URL http://doi.acm.org/10.1145/1391989.1391995.

Cook RD, Weisberg S (1982). *Residuals and influence in regression*. New York: Chapman and Hall.

Davis T (2006). *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA.

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge, England. ISBN 0-521-57391-2.

Doran H, Bates D, Bliese P, Dowling M (2007). "Estimating the multilevel Rasch model: With the lme4 package." *Journal of Statistical Software*, **20**(2), 1–18.

Dorie V (2013). *blme: Bayesian Linear Mixed-Effects Models*. R package version 1.0-1, URL http://CRAN.R-project.org/package=blme.

Efron B, Morris C (1977). "Stein's Paradox in Statistics." *Scientific American*, **236**, 119–127. doi:10.1038/scientificamerican0577-119.

Gelman A (2005). "Analysis of variance — why it is more important than ever." *The Annals of Statistics*, **33**(1), 1–53.

Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB (2013). *Bayesian data analysis*. CRC press.

Gelman A, Hill J (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, Cambridge, England. URL http://www.stat.columbia.edu/~gelman/arm/.

Golub GH, Pereyra V (1973). "The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems Whose Variables Separate." *SIAM Journal on Numerical Analysis*, **10**(2), pp. 413–432. ISSN 00361429. URL http://www.jstor.org/stable/2156365.

Henderson Jr CR (1982). "Analysis of Covariance in the Mixed Model: Higher-Level, Nonhomogeneous, and Random Regressions." *Biometrics*, **38**(3), 623–640. ISSN 0006341X. URL http://www.jstor.org/stable/2530044.

Horn R, Zhang F (2005). "Basic Properties of the Schur Complement." In F Zhang (ed.), *The Schur Complement and Its Applications*, volume 4 of *Numerical Methods and Algorithms*, pp. 17–46. Springer US. URL http://dx.doi.org/10.1007/0-387-24273-2_2.

Johnson SG (2014). "The `NLopt` nonlinear-optimization package." URL http://ab-initio.mit.edu/nlopt.

Kenward MG, Roger JH (1997). "Small sample inference for fixed effects from restricted maximum likelihood." *Biometrics*, **53**(3), 983–997.

Khatri C, Rao CR (1968). "Solutions to some functional equations and their applications to characterization of probability distributions." *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 167–180.

Klein K, Neira J (2013). "Nelder-Mead Simplex Optimization Routine for Large-Scale Problems: A Distributed Memory Implementation." *Computational Economics*. doi:10.1007/s10614-013-9377-8.

Laird NM, Ware JH (1982). "Random-Effects Models for Longitudinal Data." *Biometrics*, **38**, 963–974.

Loy A, Hofmann H (2014). "HLMdiag: A Suite of Diagnostics for Hierarchical Linear Models in R." *Journal of Statistical Software*, **56**(5), 1–28. URL http://www.jstatsoft.org/v56/i05/.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: optimx for R." *Journal of Statistical Software*, **43**(9), 1–14. URL http://www.jstatsoft.org/v43/i09/.

Nieuwenhuis R, Te Grotenhuis M, Pelzer B (2012). "Influence.ME: Tools for Detecting Influential Data in Mixed Effects Models." *R Journal*, **4**(2), 38–47.

Pinheiro J, Bates D, DebRoy S, Sarkar D, R Core Team (2014). *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-117, URL http://CRAN.R-project.org/package=nlme.

Pinheiro JC, Bates DM (1996). "Unconstrained parametrizations for variance-covariance matrices." *Statistics and Computing*, **6**(3), 289–296. doi:10.1007/BF00140873.

Pinheiro JC, Bates DM (2000). *Mixed-Effects Models in S and S-Plus*. Springer. ISBN 0-387-98957-0.

Powell MJD (2009). "The BOBYQA algorithm for bound constrained optimization without derivatives." *Technical Report DAMTP 2009/NA06*, Centre for Mathematical Sciences, University of Cambridge, Cambridge, England. URL http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf.

Satterthwaite FE (1946). "An Approximate Distribution of Estimates of Variance Components." *Biometrics Bulletin*, **2**(6), 110–114.

Sánchez-Espigares JA, Ocaña J (2009). "An R implementation of bootstrap procedures for mixed models." Conference presentation, useR! Accessed 25 May 2014, URL http://www.r-project.org/conferences/useR-2009/slides/SanchezEspigares+Ocana.pdf.

Vaida F, Blanchard S (2005). "Conditional Akaike information for mixed-effects models." *Biometrika*, **92**(2), 351–370.

Wood S, Scheipl F (2013). *gamm4: Generalized additive mixed models using mgcv and lme4*. R package version 0.2-2, URL http://CRAN.R-project.org/package=gamm4.

A. Appendix: modularization examples

A.1. Homogeneous covariance among random effects terms

Consider the following mixed-model formula,

```
R> form <- respVar ~ 1 + (explVar1 | groupFac1) + (explVar2 | groupFac2)
```

in which we have a fixed intercept and two random effects terms, each with a random slope and intercept. The `lmer` function would fit this formula to data by estimating two 2×2 covariance matrices—one for each random effects term. It is not possible to use the current version of `lmer` to fit a model in which both terms share the same 2×2 covariance matrix. We illustrate the use of the modular functions in **lme4** to fit such a model.

We simulate data from this homogeneous covariance model using a balanced design that crosses both grouping factors. We first set the seed and number of groups per grouping factor,

```
R> set.seed(1)
R> nGrps <- 50
```

We next simulate two explanatory variables,

```
R> explVar <- data.frame(explVar1 = rnorm(nGrps^2),
+                        explVar2 = rnorm(nGrps^2))
```

two grouping factors,

```
R> groupFac <- expand.grid(groupFac1 = as.factor(1:nGrps),
+                          groupFac2 = as.factor(1:nGrps))
```

random intercepts and slopes that are negatively correlated,

```
R> randomIntercept <- expand.grid(randomIntercept1 = rnorm(nGrps),
+                                 randomIntercept2 = rnorm(nGrps))
R> rnmdSlope <- expand.grid(randomSlope1 = rnorm(nGrps),
+                           randomSlope2 = rnorm(nGrps)) - randomIntercept
```

and finally a linear predictor, residual error, and response variable,

```
R> linearPredictor <- apply(randomIntercept + rnmdSlope * explVar, 1, sum)
R> residError <- rnorm(nGrps^2)
R> respVar <- residError + linearPredictor
```

We then combine these data into a single object,

```
R> dat <- data.frame(respVar, explVar, groupFac)
R> head(dat)

     respVar   explVar1   explVar2 groupFac1 groupFac2
1 -1.9018754 -0.6264538 -1.8054836         1         1
2  2.2711486  0.1836433 -0.6780407         2         1
3 -3.1780501 -0.8356286 -0.4733581         3         1
4 -3.0882014  1.5952808  1.0274171         4         1
5 -0.2487894  0.3295078 -0.5973876         5         1
6 -3.6510836 -0.8204684  1.1598494         6         1
```

To fit `form` to `dat` using a homogeneous covariance model, we construct the `homoLmer` function, which we describe in sections. To simplify the problem, we only allow the specification of three arguments: `formula`, `data`, and `use.mkMerMod`. This last argument specifies how to process the fitted model, which we discuss below. All other relevant parameters are set to `lmer` defaults, unless otherwise noted below. Here is the argument list for `homoLmer`,

```
R> homoLmer <- function(formula, data, use.mkMerMod = FALSE) {}
```

After saving the `call`, the next step is to parse the formula and data using the `lFormula` modular function (setting `REML` to `FALSE` for simplicity),

```
R> mc <- match.call()
R> lfHetero <- lfHomo <- lFormula(formula = formula, data = data, REML = FALSE)
```

We then check to make sure that this `formula` is a candidate for a homogeneous covariance model. In particular, a homogeneous covariance model only makes sense if each random effects term has the same number of random effects coefficients,

```
R> if(length(pRE <- unique(sapply(cnms <- lfHomo$reTrms$cnms, length))) > 1L) {
+      stop("each random effects term must have the same number\n",
+           "of model matrix columns for a homogeneous structure")
+  }
```

We save the number of fixed effects coefficients, `p`, number of covariance parameters per term, `nth`, and number random effects terms, `n_trms`,

```
R> p <- ncol(lfHomo$X)
R> nth <- choose(pRE + 1, 2)
R> n_trms <- length(cnms)
```

We now modify the parsed random effects terms to have homogeneous covariance structure, which requires three lines. First, use only the first `nth` elements of the covariance parameters, `theta`, as these will be repeated to generate common covariance structure across random effects terms. Second, match the lower bounds of the covariance parameters to the new `theta`. Third, adjust the mapping from the new `theta` to the nonzero elements of the transposed relative covariance factor, `Lambdat`.

```
R> lfHomo$reTrms <- within(lfHomo$reTrms, {
+      theta <- theta[1:nth]
+      lower <- lower[1:nth]
+      Lind <- rep(1:nth, length = length(lfHomo$reTrms$Lambdat@x))
+  })
```

Then we construct and minimize a deviance function for the homogeneous model,

```
R> devf <- do.call(mkLmerDevfun, lfHomo)
R> opt <- optimizeLmer(devf)
```

The object `opt` contains the optimized values of the homogeneous covariance parameters. However, in order to interpret and make inferences about this model, we need to do more work. We highlight two general approaches to doing this, which in this example are selected with the `use.mkMerMod` logical argument. If `use.mkMerMod` is set to `FALSE`, then the parsed data and formula object, `lfHomo`, deviance function, `devf`, and optimization results, `opt`, are returned,

```
R> if(!use.mkMerMod) {
+      return(list(lf = lfHomo, devf = devf, opt = opt))
+  }
```

This option means that the convenience functions of the `lme4` output module (Section 5) are not available because the resulting object is not a `merMod` object. However, constructing a `merMod` object that will be treated appropriately by the convenience functions of the output module is rather difficult, and sometimes impossible, and should therefore always be done with caution. In particular, `mkMerMod` should be used to work with specific functions from the output module; users should never assume that because some aspects of the output module give appropriate results with a `merMod` object constructed from a modular fit, all aspects will work. It will sometimes be necessary, as we next illustrate, to extend the `merMod` class and rewrite some methods from the output module that are specialized for this new class.

Next we update the unmodified model by capturing the modified model estimates of the covariance parameters,

```
R> th <- rep(opt$par, n_trms)
```

constructing a deviance function for the unmodified model,

```
R> devf0 <- do.call(mkLmerDevfun, lfHetero)
```

and installing the parameters in the environment of the unmodified deviance function,

```
R> devf0(opt$par <- th)
```

Finally, we need to extend the `merMod` class,

```
R> setClass("homoLmerMod", representation(thetaUnique = "numeric"),
+           contains = "lmerMod")
```

and redefine some methods for this extension,

```
R> logLik.homoLmerMod <- function(object, ...) {
+      ll <- lme4:::logLik.merMod(object, ...)
+      attr(ll, "df") <- length(object@beta) +
+          length(object@thetaUnique) +
+              object@devcomp[["dims"]][["useSc"]]
+      return(ll)
+  }
R> refitML.homoLmerMod <- function(object, newresp, ...) {
+      if(!isREML(object) && missing(newresp)) return(object)
+      stop("can't refit homoLmerMod objects yet")
+  }
```

```
R> mod <- homoLmer(form, dat, use.mkMerMod = TRUE)
R> summary(mod)

Linear mixed model fit by maximum likelihood  ['homoLmerMod']
Formula: form
   Data: dat

     AIC      BIC   logLik deviance df.resid
  7920.4   7949.5  -3955.2   7910.4     2492

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.0940 -0.6401 -0.0084  0.6482  3.4260

Random effects:
 Groups    Name        Variance Std.Dev. Corr
 groupFac1 (Intercept) 0.9546   0.9771
           explVar1    1.8783   1.3705   -0.76
 groupFac2 (Intercept) 0.9546   0.9771
           explVar2    1.8783   1.3705   -0.76
 Residual              1.0257   1.0128
Number of obs: 2500, groups:  groupFac1, 50; groupFac2, 50

Fixed effects:
            Estimate Std. Error t value
(Intercept) -0.04772    0.12907   -0.37
```

A.2. Homogeneous variance over all random effects

We modify the `sleepstudy` example,

```
R> parsedFormula <- lFormula(formula = Reaction ~ Days + (Days | Subject),
+                            data = sleepstudy)
```

such that the random slope and intercept are uncorrelated and have identical variance (because the slope and intercept have different units, this example is not entirely sensible — it is offered for technical illustration only). The `reTrms` component of `parsedFormula` contains the relevant information, and can be modified using the `within` function by (1) altering the mapping (`Lind`) from `theta` to the non-zero elements of `Lambda`, and force a diagonal `Lambda`,

```
R> parsedFormula$reTrms <- within(parsedFormula$reTrms, {
+      q <- nrow(Lambdat)
+      Lind <- rep(1, q)
+      Lambdat <- sparseMatrix(1:q, 1:q, x = Lind)
+      theta <- 1
+  })
```

We fit this modified model using the modularised functions,

```
R> devianceFunction <- do.call(mkLmerDevfun, parsedFormula)
R> optimizerOutput <- optimizeLmer(devianceFunction)
```

Because `lme4` is not designed for post-processing of such models, `mkMerMod` cannot be expected to give sensible results. To illustrate this point, we use it anyway and point out where it makes mistakes,

```
R> mkMerMod(   rho = environment(devianceFunction),
+              opt = optimizerOutput,
+           reTrms = parsedFormula$reTrms,
+               fr = parsedFormula$fr)

Linear mixed model fit by REML ['lmerMod']
REML criterion at convergence: 1759.15

Warning in split.default(theta, rep.int(ncseq, (nc * (nc + 1))/2)):  data length
is not a multiple of split variable

Random effects:
 Groups   Name        Std.Dev. Corr
 Subject  (Intercept)  8.899
          Days        12.585   0.71
 Residual             27.374
Number of obs: 180, groups:  Subject, 18
Fixed Effects:
(Intercept)         Days
     251.41        10.47
```

Note that `mkMerMod` is attempting to estimate a correlation between the slope and intercept, which makes no sense within the current model. When using modular functions, the user is responsible for producing interpretable output. For example, to find the residual and random-effects standard deviation, we may use,

```
R> with(environment(devianceFunction), {
+      n <- length(resp$y)
+      p <- length(pp$beta0)
+      pwrss <- resp$wrss() + pp$sqrL(1)
+      sig <- sqrt(pwrss/(n-p))
+      c(Residual = sig, Subject = sig*pp$theta)
+  })

 Residual   Subject
27.374472  8.898746
```

Note that the random effects variance requires no matrix algebra in this case because the covariance matrix of the random effects is proportional to the identity matrix

A.3. Additive models

It is possible to interpret additive models as a particular class of mixed models (Wood and Scheipl 2013). The main benefit of this approach is that it bypasses the need to select a smoothness parameter using cross-validation or generalized cross-validation. However, it is inconvenient to specify additive models using the **lme4** formula interface. The **gamm4** package wraps the modularized functions of **lme4** within a more convenient interface (Wood and Scheipl 2013). In particular, the strategy involves the following steps,

1. Convert a **gamm4** formula into an **lme4** formula that approximates the intended model.
2. Parse this formula using `lFormula`.
3. Modify the resulting transposed random effects model matrix, $\boldsymbol{Z}^\top$, so that the intended additive model results.
4. Fit the resulting model using the remaining modularized functions (Table 1).

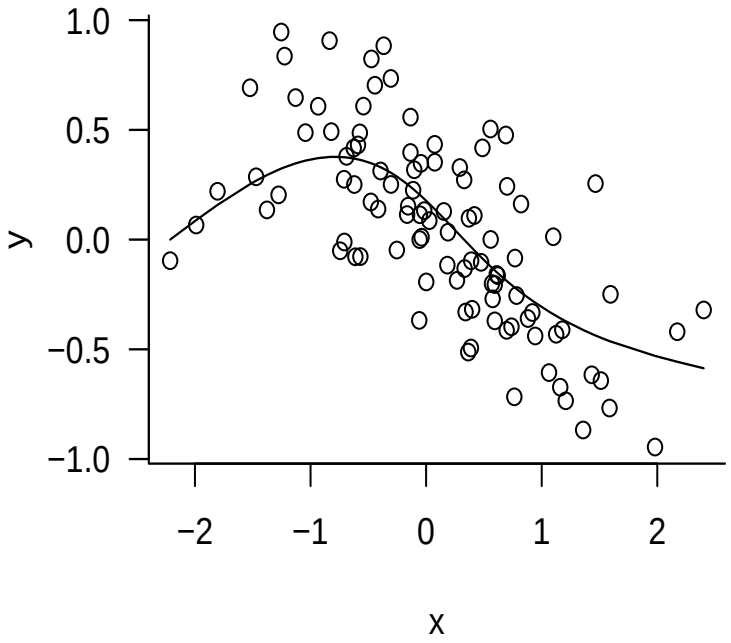
Here we illustrate this general strategy using a simple simulated data set,

```
R> library("splines")
R> set.seed(1)
R> n <- 100
R> pSimulation <- 3
R> pStatistical <- 8
R> x <- rnorm(n)
R> Bsimulation <- ns(x, pSimulation)
R> Bstatistical <- ns(x, pStatistical)
R> beta <- rnorm(pSimulation)
R> y <- as.numeric(Bsimulation %*% beta + rnorm(n, sd = 0.3))
```

where `pSimulation` is the number of columns in the simulation model matrix and `pStatistical` is the number of columns in the statistical model matrix.

We plot the resulting data along with the predictions from the generating model,

```
R> par(mar = c(4, 4, 1, 1), las = 1, bty = "l")
R> plot(x, y, las = 1)
R> lines(x[order(x)], (Bsimulation %*% beta)[order(x)])
```



We now set up an approximate **lme4** model formula, and parse it using `lFormula`,

```
R> pseudoGroups <- as.factor(rep(1:pStatistical, length = n))
R> parsedFormula <- lFormula(y ~ x + (1 | pseudoGroups))
```

We now insert a spline basis into the `parsedFormula` object,

```
R> parsedFormula$reTrms <- within(parsedFormula$reTrms, {
+      Bt <- t(as.matrix(Bstatistical))[]
+      cnms$pseudoGroups <- "spline"
+      Zt <- as(Bt, class(Zt))
+  })
```

Finally we continue with the remaining modular steps,

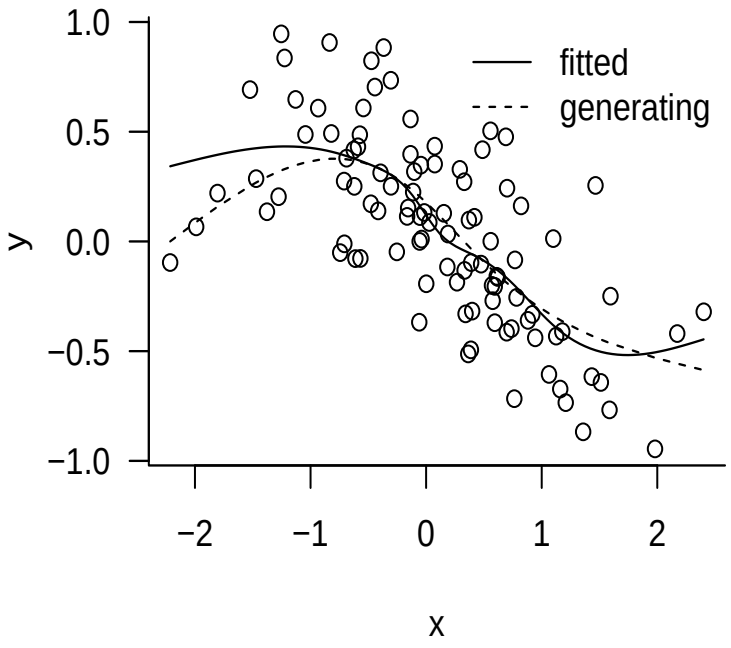
```
R> devianceFunction <- do.call(mkLmerDevfun, parsedFormula)
R> optimizerOutput <- optimizeLmer(devianceFunction)
R> mSpline <- mkMerMod(   rho = environment(devianceFunction),
+                         opt = optimizerOutput,
```

```
+                       reTrms = parsedFormula$reTrms,
+                           fr = parsedFormula$fr)
R> mSpline

Linear mixed model fit by REML ['lmerMod']
REML criterion at convergence: 60.734
Random effects:
 Groups       Name   Std.Dev.
 pseudoGroups spline 0.2930
 Residual            0.3001
Number of obs: 100, groups:  pseudoGroups, 8
Fixed Effects:
(Intercept)            x
   -0.03712     -0.17137
```

Computing the fitted values of this additive model requires some custom code, and illustrates the general principle that methods for `merMod` objects constructed from modular fits should only be used if the user knows what she is doing,

```
R> xNew <- seq(min(x), max(x), length = 100)
R> newBstatistical <- predict(Bstatistical, xNew)
R> yHat <-   cbind(1, xNew) %*% getME(mSpline, "fixef") +
+          newBstatistical %*% getME(mSpline, "u")
R> par(mar = c(4, 4, 1, 1), las = 1, bty = "l")
R> plot(x, y)
R> lines(xNew, yHat)
R> lines(x[order(x)], (Bsimulation %*% beta)[order(x)],lty = 2)
R> legend("topright", bty = "n", c("fitted", "generating"), lty = 1:2,col = 1)
```



Affiliation:

Douglas Bates
Department of Statistics, University of Wisconsin
1300 University Ave.
Madison, WI 53706, U.S.A.
E-mail: bates@stat.wisc.edu

Martin Mächler
Seminar für Statistik, HG G 16
ETH Zurich
8092 Zurich, Switzerland
E-mail: maechler@stat.math.ethz.ch

Benjamin M. Bolker
Departments of Mathematics & Statistics and Biology
McMaster University
1280 Main Street W
Hamilton, ON L8S 4K1, Canada
E-mail: bolker@mcmaster.ca

Steven C. Walker
Department of Mathematics & Statistics
McMaster University
1280 Main Street W

Hamilton, ON L8S 4K1, Canada
E-mail: scwalker@math.mcmaster.ca