Fitting Mixed-effects Models in *R* and *Julia*

Douglas Bates, U. of Wisconsin-Madison

July 19, 2013

Some background on the theory

Effects - fixed and random

- Mixed-effects models, like many statistical models, describe the relationship between a *response* variable and one or more *covariates* recorded with it.
- Coefficients associated with the levels of a categorical covariate are sometimes called the *effects* of the levels.
- When the levels of a covariate are fixed and reproducible (e.g. a covariate `sex` that has levels `male` and `female`) we incorporate them as *fixed-effects* parameters.
- When the levels of a covariate correspond to the particular observational or experimental units in the experiment we incorporate them as *random effects*.

An example - student evaluations of instructors

- The `InstEval` data set from the `lme4` package for *R* provides evaluation scores of 1128 instructors (`d`) in 14 departments by 2972 students (`s`). The `service` column indicates a service course.

```
> str(InstEval)
```

```
'data.frame':   73421 obs. of  7 variables:
 $ s      : Factor w/ 2972 levels "1","2","3","4",..: 1 1 1 1 2 2 3 3
 $ d      : Factor w/ 1128 levels "1","6","7","8",..: 525 560 832 1
 $ studage: Ord.factor w/ 4 levels "2"<"4"<"6"<"8": 1 1 1 1 1 1 1 1
 $ lectage: Ord.factor w/ 6 levels "1"<"2"<"3"<"4"<..: 2 1 2 2 1 1
 $ service: Factor w/ 2 levels "0","1": 1 2 1 2 1 1 2 1 1 1 ...
 $ dept   : Factor w/ 14 levels "15","5","10",..: 14 5 14 12 2 2 13
 $ y      : int  5 2 5 3 2 4 4 5 5 4 ...
```

- Our purpose is not to compare individual students so much as to account for the variability amongst students.

Mixed-effects models

- A model that incorporates both fixed-effects parameters and random effects is called a *mixed-effects* model.
- Mixed-effects models can be very large and complex. Models with over a million random-effects parameters fit to several million observations are not uncommon in some areas.
- Typically factors modeled with fixed-effects have a small number of levels, those modeled with random effects have a large number of levels.
- Random effects are modeled as a sample from a population, usually multivariate Gaussian with mean $\mathbf{0}$ and a parameterized covariance matrix, $\mathbf{\Sigma}_\theta$.

Formulation of a mixed-effects model

- There are many ways to write mixed-effects models – most of them misleading. I will define them in terms of two vector-valued random variables, $\mathcal{Y}$ and $\mathcal{B}$.
- The unconditional distribution of $\mathcal{B}$ is $\mathcal{B} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_\theta)$
- The conditional distribution, $\mathcal{Y}|\mathcal{B} = \boldsymbol{b}$, depends on the *linear predictor*, $\boldsymbol{\eta}$, evaluated from model matrices, $\boldsymbol{X}$ and $\boldsymbol{Z}$, and the fixed-effects parameter vector, $\boldsymbol{\beta}$.

$$\boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{b}$$

- In the case of a linear mixed-effects model the conditional distribution is

$$\mathcal{Y}|\mathcal{B} = \boldsymbol{b} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{b}, \sigma^2 \boldsymbol{I})$$

Specifying mixed-effects models

Mixed-effects model formulas in R

- One of the underrated aspects of the *R* language is the formula language for specifying models and, in the case of the `lattice` package, plots.
- A formula such as `y ~ x + f + x:f` where `y` is the response variable, `x` is a continuous covariate and `f` is a categorical covariate is more compact and, with experience, more understandable than most other specifications.
- In the `lme4` package for *R* the formula language is extended to allow for random effects terms of the form `(l|f)` where `l` is a linear model expression and `f` is the factor whose levels determine the random effects.
- Until you need models with random effects for slopes, it is best to consider these terms as being of the form `(1|f)` where `f` is the *grouping factor* for the random effects.
- A term like `(1|f)` is called a *simple, scalar, random-effects term*.

A model for the InstEval data

- To specify fixed effects parameters for the department and service course indicator and their interaction plus random effects for student and instructor, we use

```
> fm1 <- lmer(y ~ service * dept + (1 | s) + (1 | d), InstEval
> coef(summary(fm1))
```

```
              Estimate Std. Error t value
(Intercept)    3.22953    0.06439 50.1593
service1       0.25205    0.06869  3.6692
dept5          0.12966    0.10188  1.2727
dept10        -0.17659    0.08862 -1.9926
dept12         0.05174    0.08220  0.6294
dept6          0.03496    0.08608  0.4061
dept7          0.14599    0.10031  1.4554
dept4          0.15183    0.08212  1.8489
dept8          0.10431    0.11919  0.8751
```

Estimates of the variance components

```
> VarCorr(fm1)
```

```
Groups   Name        Variance Std.Dev.
s        (Intercept) 0.106    0.325
d        (Intercept) 0.262    0.512
Residual             1.385    1.177
```

- We see that the residual variability is the dominant term (often the case in studies involving human subjects). The instructor-to-instructor variability is somewhat larger than the student-to-student variability.
- Note the the column labelled *Std.Dev.* is simply the square root of the estimate of the variance component. It is (intentionally) **not** a standard error of the estimate.

Size of the model

- There are two random-effects terms in this model, generating a total of 4100 random-effects coefficients from the 73421. The model includes 28 fixed-effects coefficients.
- Determining estimates involves solving *Henderson's mixed-model equations* (MME) for each potential value of $\boldsymbol{\theta}$.

$$\begin{bmatrix} \boldsymbol{X}'\boldsymbol{X} & \boldsymbol{X}'\boldsymbol{Z} \\ \boldsymbol{Z}'\boldsymbol{X} & \boldsymbol{Z}'\boldsymbol{Z} + \sigma^2\Sigma_\theta^{-1} \end{bmatrix} \begin{bmatrix} \widehat{\beta} \\ \tilde{\boldsymbol{b}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}'\boldsymbol{y} \\ \boldsymbol{Z}'\boldsymbol{y} \end{bmatrix}$$

- Storing the matrix $\boldsymbol{Z}$ of size 73421×4100 would be incredibly expensive (over 2 Gb for each copy) and any attempt to solve the MME would be infeasible were it not for the fact the $\boldsymbol{Z}$ is very *sparse*.
- This huge $\boldsymbol{Z}$ is all zeros except for two elements in each row.

Modifying the MME for sparse matrix methods

- An obvious modification is to work with the *relative covariance matrix*, $\mathbf{\Sigma}_\theta/\sigma^2$ which allows us to *profile out* the estimation of σ.
- The next step is to work with the "square root" of this relative covariance. The *relative covariance factor*, $\mathbf{\Lambda}_\theta$, is a sparse lower-triangular matrix such that

$$\mathbf{\Sigma}_\theta = \sigma^2 \mathbf{\Lambda}_\theta \mathbf{\Lambda}'_\theta$$

- For this model, $\boldsymbol{\theta}$ is two-dimensional and $\mathbf{\Lambda}_\theta$ is diagonal with 2972 repetitions of θ_1 followed by 1128 copies of θ_2 on the diagonal.

```
> getME(fm1, "theta")
```

```
s.(Intercept) d.(Intercept)
       0.2762        0.4352
```

Modifying the MME

- With this machinery we can re-express the mixed-model equations as

$$\begin{bmatrix} \Lambda_\theta' \boldsymbol{Z}' \boldsymbol{Z} \Lambda_\theta + \boldsymbol{I} & \Lambda_\theta' \boldsymbol{Z}' \boldsymbol{X} \\ \boldsymbol{X}' \boldsymbol{Z} \Lambda_\theta & \boldsymbol{X}' \boldsymbol{X} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{u}} \\ \widehat{\boldsymbol{\beta}} \end{bmatrix} = \begin{bmatrix} \Lambda_\theta' \boldsymbol{Z}' \boldsymbol{y} \\ \boldsymbol{X}' \boldsymbol{y} \end{bmatrix}$$

- The solution $\tilde{\boldsymbol{b}}$ is calculated as $\tilde{\boldsymbol{b}} = \Lambda_\theta \tilde{\boldsymbol{u}}$
- Although the matrix $\Lambda_\theta' \boldsymbol{Z}' \boldsymbol{Z} \Lambda_\theta + \boldsymbol{I}$ is sparse, it is not diagonal like Λ_θ. In fact its structure can be complicated because the factors `s` and `d` are not nested. (Instructors teach more than one student, students are taught by more than one instructor.)
- The matrix $\Lambda_\theta' \boldsymbol{Z}' \boldsymbol{Z} \Lambda_\theta + \boldsymbol{I}$ is symmetric and is *positive definite* (because of the $+\boldsymbol{I}$) which means that it will have a *Cholesky factor*.

Factoring a large, sparse symmetric matrix

- The problem of producing the sparse *Cholesky factor*, which is a lower-triangular matrix, $\boldsymbol{L}_\theta$ such that

$$\boldsymbol{L}_\theta \boldsymbol{L}_\theta' = \Lambda_\theta' \boldsymbol{Z}' \boldsymbol{Z} \Lambda_\theta + \boldsymbol{I}$$

  has been studied intensively and high-quality software exists for doing this.
- In *R* the `Matrix` package provides such methods using Tim Davis's `CHOLMOD` library. The same code is used in the *Julia* base system.
- Like most methods for sparse matrices the Cholesky factor is created in two phases, a *symbolic* phase, in which the positions of the non-zeros are determined, and a *numeric* phase, in which these values are calculated. In determining the MLE's or the REML estimates of the parameters in a linear mixed model the symbolic phase need only be done once.

Crossed and partially crossed grouping factors

- The factors `s` and `d` are not completely crossed (each student is not taught by all 1128 instructors, each instructor has not taught all 2972 students). Instead they are *partially crossed*.
- Such data are becoming more common, e.g. student scores on the "No Child Left Behind"-mandated exams.
- In some disciplines like psychometrics or item-response theory it is common to have crossed or nearly crossed factors like "Subject" and "Stimulus" which should be modeled with random effects.

Profiling the deviance

- The truly remarkable result is that the *profile likelihood* for a linear mixed-model can be calculated as a function of $\boldsymbol{\theta}$ only. On the deviance scale it is

$$-2\tilde{\ell}(\boldsymbol{\theta}) = \log(|\boldsymbol{L}_\theta|^2) + n\left[1 + \log\left(\frac{2\pi r_\theta^2}{n}\right)\right]$$

  For model `fm1` determining the mle's is just a 2-dimensional optimization.

- In this expression, n is the number of observations and r_θ^2 is the penalized residual sum of squares that is minimized by the solution to the MMEs.

$$r_\theta^2 = \min_{\boldsymbol{u},\boldsymbol{\beta}}\left(||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\boldsymbol{\Lambda}_\theta\boldsymbol{u}||^2 + ||\boldsymbol{u}||^2\right)$$

- The determinant, $|\boldsymbol{L}_\theta|^2$ is easy to evaluate because $\boldsymbol{L}_\theta$ is triangular.

The Julia packages relative to R packages

The *RDatasets* package for *Julia*

- Many of the data sets from R packages are available through the `RDatasets` package for *Julia*

```
julia> using MixedModels, RDatasets
julia> inst = data("lme4", "InstEval")
julia> head(inst)
6x7 DataFrame:
          s      d studage lectage service dept y
[1,]    "1" "1002"     "2"     "2"     "0"  "2" 5
[2,]    "1" "1050"     "2"     "1"     "1"  "6" 2
[3,]    "1" "1582"     "2"     "2"     "0"  "2" 5
[4,]    "1" "2050"     "2"     "2"     "1"  "3" 3
[5,]    "2"  "115"     "2"     "1"     "0"  "5" 2
[6,]    "2"  "756"     "2"     "1"     "0"  "5" 4
```

The *MixedModels* package for *Julia*

- A package called `MixedModels` (that name may change) provides an `lmer` function

```
julia> fm1 = lmer(:(y ~ service*dept + (1|s) + (1|d)), inst);
julia> @time fit(fm1, true)
f_1: 241920.83782176476, [1.0,1.0]
f_2: 244850.35312911012, [1.75,1.0]
f_3: 242983.2665867725, [1.0,1.75]
f_4: 238454.23551272837, [0.25,1.0]
f_5: 241716.05373818352, [1.0,0.25]
f_6: 240026.0596421508, [0.0,0.4459187720601605]
f_7: 241378.58264793456, [0.0,1.2795084971874737]
...
f_46: 237585.55341517925, [0.2759146679975875,0.43199306783163
f_47: 237585.55341516965, [0.2759148830633722,0.43199403927991
XTOL_REACHED
elapsed time: 8.354907641 seconds
```

Comparing the *Julia* and *R* packages

- There are some superficial differences between the two `lmer` functions
  - the *R* version defaults to REML, the *Julia* version to *ML*
  - the *Julia* version requires the formula to be wrapped in `:()`
  - the *Julia* version requires a call to `fit`, which is implicitly called by `show`. (This will probably change)
- The big difference is that the *Julia* version is implemented in *Julia*. The *R* version uses many awkward constructions (*C++*, *Rcpp*, *RcppEigen*) to be able to fit models in a reasonable length of time.
- Even with all the tricks, the *R* version can still use all your memory.

```
> system.time(lmer(y ~ service * dept + (1 | s) + (1 | d), Ins
```

```
  user  system elapsed
 19.52   10.27   15.27
```

Comparing the *Julia* and *R* packages (cont'd)

- The *Julia* version at present uses a relatively simple algorithm and does not cache much information. Even so, it is faster than the *R* version we have been working on for more than 10 years.
- A more carefully optimized *Julia* version for LMMs could be remarkably fast.
- Support for the *R* version is, naturally, more fully developed. Graphics, inference, support for more complex models, etc. are not yet available in *Julia*.
- Until `lmer` for *R* was developed, it would not have been possible to fit model `fm1` - at least I don't know of a way of doing that other than in `lmer`. Neither *SAS PROC MIXED* nor *MLwiN* nor *HLM 6* could do so. (I don't know about *HLM 7*.)

Challenges when writing *Big Data* applications in *R*

- My experience with writing code for such models in *R* brought some of the limitations of *R* for fitting models to "Big Data" to my attention (emphatically).
  - *R* doesn't have scalars, per se. All numerical objects are stored as vectors.
  - Looping in *R* (`for` and `while` loops) is slow.
  - Most of *R* is written in *C*. Most of *Julia* is written in *Julia*.
  - Calling *C* code (or *Fortran* or *C++*) is possible but awkward to write and even more awkward to debug.
  - *R* functions should not modify their arguments. That's fine until you want to do so.
  - Surprisingly, profiling the execution of *Julia* functions shows that allocation and freeing of memory can often be a bottleneck. In *Julia* you can get around this problem to some extent. It is much more difficult to do so in *R*.