lme4 Emulation

Ronald D. Flores

Note: At its current state, this mixed-effect modeling project is ideal for repeated measurement studies without missing data. Furthermore, modeling must be limited to one continuous predictor and one continuous outcome at a time, with random intercept and slope.

PHASE 1: Exploration

The main goal is to emulate the lme4 R package by reproducing its results when fitting a mixed-effects model using the lmer() function and built-in sleep study data set. These data consist of 18 increasingly sleep-deprived truck drivers measured for reaction time once per day for 10 days. The target model for these data will be reaction time regressed on days of sleep deprivation, with intercepts and slopes allowed to vary by truck driver (and estimated via full information maximum likelihood):

$$lmer(Reaction \sim Days + (Days|Subject),\ data = SS,\ REML = FALSE)$$

Though several statistics are available upon fitting and summarizing a model via the summary() function, the main statistics sought for reproduction will be the <u>fixed effect estimates</u> and <u>random effect variances</u>. Other incidental stats will be computed afterward, such as the random effects correlation, residual variance, fixed effects t-tests, and fixed effects correlation.

To begin, matrix formulae pertinent to classical regression were drawn from Gelman and Hill's 2007 book and converted to R code since these seemed to constitute an intuitive starting point. These formulae yielded values that exactly matched the results generated by using the lm() function:

- $\hat{\beta} = (X'X)^{-1}X'y$ (X and y are data matrix partitions, plus a column of 1's for X)
- $residuals = y - X\hat{\beta}$
- $\hat{\sigma}_e^2 = \Sigma \frac{residuals^2}{n-k}$
- $R^2 = 1 - \frac{\hat{\sigma}_e^2}{Var(y)}$
- $covariance\ matrix\ of\ \beta = (X'X)^{-1} * \hat{\sigma}_e^2$ (square root of diagonals are SEs)

Next, the following unconditional probability distribution notation for y values will be used as a bridge to mixed-effects modeling:

$$y \sim N(X\beta, ZGZ' + \sigma_e^2 I)$$

This distribution (which averages over random effects) will help determine the likelihood of the observed y values in the utilized data set. Furthermore, new code for the Z, G, and $\sigma_e^2 I$ matrices will be needed. First, since the Z matrix appears to be a wider version of the 180-by-2 X matrix, with grouped observations staggered increasingly further to the right (assuming tall data with grouped observations on adjacent rows), it was created with code that pushed chunks of data from X increasingly further right. This yielded a 180-by-36 matrix containing the exact same data as X, but spread over a bigger space with zeros to fill empty space. Second, specific to the current mixed-effects model with varying intercepts and slopes, the G matrix possessed a repeating 2-by-2 block design with the “block” appearing to be a covariance matrix of intercepts and slopes. These intercepts and slopes were believed to be the parameter values assuming each truck driver was treated as a separate classic regression model. Once this covariance matrix was produced, code was used to repeatedly paste it to create G. This resulted in a 36-by-36 matrix of “blocks” running down the diagonal. Zeros, again, filled any empty space. Third, to create $\sigma_e^2 I$, an identity matrix of dimension 180 x 180 was multiplied by the residual variance σ_e^2 computed using the classic regression formula.

```
#### PACKAGES, DATA AND TARGET lme4 MODEL

library(lme4)
library(mvtnorm)
library(numDeriv)
SS <- sleepstudy  ## 18 truck drivers, 10 obs each
summary(lmer(Reaction ~ Days + (Days | Subject), data = SS, REML=FALSE))

#### GOAL: REPLICATE BIG PIECES FROM summary()

## Random effects variances/SDs, correlations, & residual variance/SD
## Fixed effects: estimates, SEs, t-values, & corrs
```

```
#### COMPARING lm() OUTPUT TO OWN CALCULATION (CLASSICAL REGRESSION TESTER)

## Target lm() model
summary(lm(Reaction ~ Days, data = SS))

## Splitting data into two matrices
y <- as.matrix(SS[,1])
X <- cbind(rep(1, length(SS$Days)), SS[,2])

## Deriving lm() components
BETA <- solve(crossprod(X)) %*% t(X) %*% y    ## lm() beta values
r <- y - X %*% BETA                           ## lm() min-max residuals wn this vector
r.vari <- sum(r^2) / (length(SS$Days) - 2)    ## lm() squared residual stand err (k=2 for # of param)
R.sq <- 1 - r.vari/var(y)                     ## lm() adjusted R.sq
CVM.beta <- solve(crossprod(X)) * r.vari      ## lm() variances for each beta (ie: squared SEs)

#### MAKING Z MATRIX (using tall data with two varying components: intercepts and slopes)

## Chunking X and staggering the pieces across a bigger matrix
Z <- matrix(0, nrow=180, ncol=2*18)
for (i in 1:18) {
  Z[(i*10-9):(i*10),(i*2-1):(i*2)] <- X[(i*10-9):(i*10),]
}

#### MAKING G MATRIX (must be 36x36, or 18x18 of 2x2)

## Getting intercepts and slopes for 18 subsets of data (quick and dirty)
intercepts_and_slopes <- matrix(nrow=18, ncol=2)
for (i in 1:18) {
  BETA.sub <- solve(crossprod(X[(i*10-9):(i*10),])) %*% t(X[(i*10-9):(i*10),]) %*% y[(i*10-9):(i*10)]
  intercepts_and_slopes[i,] <- t(BETA.sub)
}
```

```
## Repeatedly pasting "blocks" of CVM.rand to create G
CVM.rand <- cov(intercepts_and_slopes)
G <- matrix(0, nrow=2*18, ncol=2*18)
for (i in 1:18) {
  G[(i*2-1):(i*2),(i*2-1):(i*2)] <- CVM.rand
}

#### MAKING ZGZt, XB, AND IDENTITY OF RESIDUAL VARIANCE MATRICES

ZGZt <- Z %*% G %*% t(Z)
XB <- X %*% BETA
irv <- diag(180) * r.vari
```

PHASE 2: Using dmvnorm() and troubleshooting

After installing the mvtnorm R package, a test run was conducted using a 180-dimension multivariate standard normal distribution. First, a single observation was simulated using this 180-dimension distribution via the rmvnorm() function. Second, the density of this 180-dimension point was calculated using the dmvnorm() function. Repetition of these steps consistently yielded very tiny probabilities with over one-hundred decimal places before observing a non-zero digit (eg: 2.48×10^{-116}). These results appeared consistent with expectations.

The mvtnorm package was then used to simulate data using the unconditional probability distribution function from PHASE 1. The results, however, were consistently "0" which may be a problem of exceedingly small values, which the software can no longer conveniently track.

Despite the apparent impasse, the rmvnorm() function was used to generate random effects, b, to find estimated mean reaction times for each truck driver using the following formula:

$$\mu = X\beta + Zb$$

Since these μ values could be used to simulate observations conditional on b by using $y|b \sim N(\mu, \sigma_e^2 I)$, new data were created and the residual variance for these data was then computed for comparison with the original data. The target at this point became a matter of minimizing residual variance by looping the preceding steps until a set of parameters yielded what appeared to be the lowest possible value of residual variance. However, preliminary results showed inflated β values and raised suspicions as to whether group-level variances were being accounted in these values.

```
#### UTILIZING dmvnorm() TO GET LIKELIHOOD OF DATA

## Testing with multivariate standard normal (180-dimensional)
data <- rmvnorm(1e0, mean = t(rep(0, 180)), sigma = diag(180), method="chol")
dmvnorm(data, mean = t(rep(0, 180)), sigma = diag(180), log=FALSE)

## Testing with simulated data using mean = XB, sigma = ZGZt + irv (180-dimensional)
y <- rmvnorm(1e0, mean = XB, sigma = ZGZt + irv, method="chol")
dmvnorm(y, mean = XB, sigma = ZGZt + irv, log=TRUE)  ## values too small, need log-transform

#### UTILIZING rmvnorm() TO GET y.condb

## Random draws to get Zb, then mu
b <- rmvnorm(1e0, mean = rep(0, 36), sigma = G)
Zb <- Z %*% t(b)
mu <- XB + Zb    ## means for each truck driver
```

```r
## Getting simulated observations and creating new data set based on a single random draw
y.condb <- rmvnorm(1e0, mean=mu, sigma = irv)
SS.simu <- SS; SS.simu[,1] <- t(y.condb) ## new data
```

PHASE 3: Corrections and usage of nlminb()

A few misconceptions from the previous phases were corrected. I will briefly outline each one next before detailing the steps that allowed the emulation of the targeted lme4 components.

- The 2-by-2 "block" within the G matrix does not contain variances and covariances dependent on classical regression computation of intercepts and slopes for each of the 18 data grouping subsets.
- Similarly, the $\sigma_e^2 I$ matrix is also not based on the classical regression computation of residual variance.
- Plugging the unconditional probability distribution into the dmvnorm() function, along with a vector of the observed y values, yielded a likelihood of zero. This was indeed a problem with the software attempting to track exceedingly small values but corrected by using a log-transform.
- New data did not need to be simulated. And a loop was not needed to minimize residual variance using these simulated data. Instead, the nlminb() function was used, which will be detailed next.

The nlminb() function optimizes parameter values for whatever function is fed into it, which was key in estimating correct parameter values for the target lmer() model. Specifically, nlminb() requires a function that generates a single numerical value then seeks to minimize this value in an iterative manner. Once this minimum has been achieved, nlminb() prints the most up-to-date associated parameter values for the function that was fed into it.

The current function that will be fed into nlminb() was built using code from the previous phases. It was designed with six parameter values as input: (i) intercept fixed effect, (ii) slope fixed effect, (iii) residual variance, (iv) intercept random effect variance, (v) slope random effect variance, and (vi) intercept-slope covariance. These inputs were then used to create the Z, G, and identity residual variance matrices with the goal of calculating a singular numerical output based on these inputs: the log-likelihood. Since nlminb() seeks to minimize the output of whatever function is fed into it, and since the utilized log-transform produces negative values, the log-likelihood had to be negated. Now, by minimizing the negative log-likelihood, nlminb() can find the set of parameters with the highest likelihood. In other words, for a vector of observed y values, nlminb() will find the optimal parameter estimates.

The nlminb() function also requires initial, "best guess" parameter values to start. These values were borrowed from the classic regression computations above. After 81 iterations, nlminb() converged on an optimal solution for the targeted six parameter values based on their negative log-likelihood. Five of these optimized parameter values were immediately ready for comparison against the summary() output of the target lmer() model, but the rest required further algebraic manipulation. First, to convert the optimized random effects covariance generated by nlminb() into a correlation, it was divided by the square root of the random intercept and random slope variances. Second, to compute fixed effect SEs, t-values, and the correlation between these fixed effects, the Hessian matrix was needed based on the six optimized parameter values. Briefly, the Hessian matrix is a squared matrix of second-order partial derivatives and it describes the local curvature for a function of multiple variables. Here, after computing the Hessian matrix using hessian(), the SEs for the fixed effects were found by inverting the Hessian matrix and computing the square root of the first two diagonals. The t-values could then be found by dividing each fixed effect by its respective SE. Lastly, using the inverted Hessian matrix, the correlation between the fixed effects was found by taking the Hessian element analogous to their "covariance" and dividing this value by the product of the fixed effect SEs.

```r
#### CREATING FUNCTION FOR USE IN nlminb()

## Must be of form function(parameters), with a single numerical output
## SIX parameters to optimize:
  ## 1. intercept fixed effect
  ## 2. slope fixed effect
  ## 3. residual variance
  ## 4. intercept random effect variance
  ## 5. slope random effect variance
  ## 6. intercept & slope covariance
## To find maximum log-likelihoods using nlminb() minimization, must ALSO negate log-likelihoods

model_negloglikelihood <- function(param){

  ## Getting y & X, and more...
  y <- as.matrix(data[,1])
  X <- cbind(rep(1, length(data$Days)), data[,2])
  BETA <- param[1:2]
  r.vari <- param[3]

  ## Getting Z, covariance matrix of random effects, & G
  Z <- matrix(0, nrow=180, ncol=2*18)
  for (i in 1:18) {
    Z[(i*10-9):(i*10),(i*2-1):(i*2)] <- X[(i*10-9):(i*10),]}
  CVM.rand <- cbind(c(param[4], param[6]), c(param[6], param[5]))
  G <- matrix(0, nrow=2*18, ncol=2*18)
  for (i in 1:18) {
    G[(i*2-1):(i*2),(i*2-1):(i*2)] <- CVM.rand}

  ## Getting negative log-likelihood using BETA, Z, G, and r.vari
  ZGZt <- Z %*% G %*% t(Z)
  XB <- X %*% BETA
  irv <- diag(180) * r.vari
  LL <- dmvnorm(t(y), mean = XB, sigma = ZGZt + irv, log=TRUE)
  (negLL <- -LL)

}

#### TESTING FUNCTION THEN OPTIMIZING VIA nlminb()

## Best guesses from previous section
BETA <- c(251.4, 10.5)
r.vari <- 2276
var_inter <- 838.3
var_slope <- 43.0
covari <- -26.12

## Getting data and creating "param" to test function
data <- SS
param <- c(BETA, r.vari, var_inter, var_slope, covari)
model_negloglikelihood(param)

## Using "param" for nlminb() start values (function evals and iterations increased by about 5x)
```

```
OPTIM <- nlminb(start = param,
                objective = model_negloglikelihood,
                control = list(eval.max=1e3, iter.max=1e3))

#### COMPONENTS FOR COMPARISON WITH lme4()

## Random effects: variances, SDs, and correlation
rev <- OPTIM$par[4:5]
resd <- sqrt(OPTIM$par[4:5])
rec <- OPTIM$par[6] / sqrt(OPTIM$par[4]*OPTIM$par[5])

## Residual variance and SD
rv <- OPTIM$par[3]
rsd <- sqrt(OPTIM$par[3])

## Fixed effects: estimates, SEs, and t-values
fee <- OPTIM$par[1:2]
H <- hessian(func=model_negloglikelihood, x=OPTIM$par)
fese <- sqrt(diag(solve(H)))[1:2]
fet <- OPTIM$par[1:2] / sqrt(diag(solve(H)))[1:2]

## Correlation of fixed effects
fec <- solve(H)[2,1] / sqrt(solve(H)[1,1]*solve(H)[2,2])
```

Though nlminb() seeks to optimize parameter values, results were not exact in comparison to the summary() output for the target lmer() model. Overall, though, results were very satisfactory, as shown in the following tables. The lme4 results come first, via summary() of the target model, followed by a table of the emulated results.

```
############################### SUMMARY() TABLE ###############################

summary(lmer(Reaction ~ Days + (Days | Subject), data = SS, REML=FALSE))
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: Reaction ~ Days + (Days | Subject)
##    Data: SS
##
##      AIC      BIC   logLik deviance df.resid
##   1763.9   1783.1   -876.0   1751.9      174
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9416 -0.4656  0.0289  0.4636  5.1793
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 565.48   23.780
##           Days         32.68    5.717   0.08
##  Residual             654.95   25.592
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  251.405      6.632  37.907
## Days          10.467      1.502   6.968
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138
```

```
################################ EMULATED TABLE ################################

re <- rbind(c(rev[1], resd[1], NA), c(rev[2], resd[2], rec), c(rv, rsd, NA))
rownames(re) <- c("(Intercept)", "Predictor", "Residual")
colnames(re) <- c("Variance", "SD", "Correlation")

fe <- rbind(c(fee[1], fese[1], fet[1], NA), c(fee[2], fese[2], fet[2], fec))
rownames(fe) <- c("(Intercept)", "Predictor")
colnames(fe) <- c("Estimate", "SE", "t-value", "Correlation")

list("Random effects", re, "Fixed effects", fe)
```

```
## [[1]]
## [1] "Random effects"
##
## [[2]]
##              Variance        SD Correlation
## (Intercept) 565.51544 23.780569          NA
## Predictor    32.68221  5.716836  0.08131983
## Residual    654.94098 25.591815          NA
##
## [[3]]
## [1] "Fixed effects"
##
## [[4]]
##              Estimate       SE   t-value Correlation
## (Intercept) 251.40510 6.632277 37.906303          NA
## Predictor    10.46729 1.502237  6.967799  -0.1375536
```

PHASE 4: Generalizing to other data sets

This project was based on a single data set, using a single model. Because of this, the code created up to this point has limitations particular to the structure of the data and the utilized mixed-effects model. Nonetheless, some adjustments were made to extend the usefulness of this code to other data sets, though limited to the same kind of mixed-effects model.

The following is a preliminary attempt at expanding the utility of the current code to other data sets. First, the function fed into nlminb() was modified so that it no longer referenced anything limited to the sleep study data set. This entailed some minor coding adjustments, such as counting the number of groups and observations per group in a new data set. Second, a new function was created, wrapping the nlminb() function, and producing a table of emulated results equivalent to the one above. Again, since the same mixed-effects model will be used, this new function will only need three input values, plus data, to output a table:

$$emu.lmer(outcome,\ predictor,\ group,\ data)$$

Results showed that the expanded code and new function could successfully replicate the lmer() results above (not shown), as well as the lmer() results for other data sets. Below is an example using the well-known R iris data. First, the summary() output is shown for a mixed-effects model based on these data. Second, the emulated results table is shown using the new function. Future steps in further generalizing the code for more diverse models would require the ability of adding more predictors and their random slopes, including more complex forms of grouping, and imbalanced group sizes.

```
################################ SUMMARY() TABLE ################################

summary(lmer(Sepal.Width ~ Petal.Width + (Petal.Width | Species), data=iris, REML=FALSE))
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: Sepal.Width ~ Petal.Width + (Petal.Width | Species)
##    Data: iris
##
##      AIC      BIC   logLik deviance df.resid
##     93.7    111.8    -40.9     81.7      144
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.8856 -0.6178  0.0305  0.6205  2.8753
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Species  (Intercept) 0.59727  0.77283
##           Petal.Width 0.00221  0.04701  -0.11
##  Residual             0.08983  0.29971
## Number of obs: 150, groups:  Species, 3
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   2.1677     0.4689   4.623
## Petal.Width   0.7421     0.1225   6.060
##
## Correlation of Fixed Effects:
##             (Intr)
## Petal.Width -0.317
```

```
################################ EMULATED TABLE ################################

source("emu.lmer.R")
emu.lmer(outcome=Sepal.Width, predictor=Petal.Width, group=Species, data=iris)
```

```
## [[1]]
## [1] "Random effects"
##
## [[2]]
##                Variance         SD Correlation
## (Intercept) 0.597170609 0.77276815          NA
## Predictor   0.002207465 0.04698366  -0.1144535
## Residual    0.089826210 0.29971021          NA
##
## [[3]]
## [1] "Fixed effects"
##
```

```
## [[4]]
##              Estimate        SE  t-value Correlation
## (Intercept) 2.1676697 0.4713244 4.599104          NA
## Predictor   0.7420987 0.1735572 4.275817  -0.2611256
```