lme4 cheat sheet

Clay Ford

Fall 2015

A cheat sheet for fitting and assessing Linear Mixed-Effect Models using `lme4`.

Limitations of `lme4` (or what it cannot do that `nlme` can do)

- `lme4` does not allow the modeling of heteroscedastic within-group errors. It only fits models with independent residual errors.
- `lme4` only models two types of covariance matrices for random effects: general and diagonal. A general covariance matrix has separate variances for each random effect and covariances between the random effects. A diagonal covariance matrix has separate variances for each random effect and no covariance between random effects.

Features of `lme4` (or what it can do that `nlme` cannot do)

- `lme4` provides facilites for modeling generalized linear mixed-effects models. (see `glmer`)
- `lme4` can fit models with *crossed* random effects.
- `lme4` uses efficient computational algorithms based on sparse-matrix representations that make it suitable for large data sets.

Format of Data

Data should be a data frame in long format. That is, one row per observation of the response variable. For example:

```
head(lme4::cake)
```

```
##   replicate recipe temperature angle temp
## 1         1      A         175    42  175
## 2         1      A         185    46  185
## 3         1      A         195    47  195
## 4         1      A         205    39  205
## 5         1      A         215    53  215
## 6         1      A         225    42  225
```

Fitting Models

To fit linear mixed-effects model, use the `lmer()` function. The formular for `lmer` allows you to express both fixed and random effects. Random effects are defined in parentheses. Random effects are conditioned on groups, typically groups with uninteresting or `random` levels. The conditioning is defined with a pipe: `|`. Two pipes, `||`, specify fitting a model with a diagonal covariance structure for the random effects. (i.e., assume multiple random effects are not correlated.)

Below, `dv` is the dependent variable, `x1` is a predictor, and `g` is a grouping indicator (for example, subject ID)

Random intercept for each level of `g`:
```
lme01 <- lmer(dv ~ x1 + (1 | g), data=df)
```

Random slope for each level of `g`:
```
lme02 <- lmer(dv ~ x1 + (0 + x1 | g), data=df)
```

Correlated random slope and intercept for each level of `g`:
```
lme03 <- lmer(dv ~ x1 + (x1 | g), data=df)
```

Uncorrelated random slope and intercept for each level of `g`:
```
lme04 <- lmer(dv ~ x1 + (x1 || g), data=df)
```

Multilevel models

For models with nested grouping factors (aka multilevel models), use `/` to indicate nesting. Below `tch` is nested in `sch`. For example, teachers nested within schools.

Random intercept for each level of `sch` and for each level of `tch` in `sch` :
```
lme01 <- lmer(dv ~ x1 + (1 | sch/tch), data=df)
```

Random slope for each level of `sch` and for each level of `tch` in `sch`:
```
lme02 <- lmer(dv ~ x1 + (0 + x1 | sch/tch), data=df)
```

Correlated random slope and intercept for each level of `sch` and for each level of `tch` in `sch`:
```
lme03 <- lmer(dv ~ x1 + (x1 | sch/tch), data=df)
```

Uncorrelated random slope and intercept for each level of `sch` and for each level of `tch` in `sch`:
```
lme03 <- lmer(dv ~ x1 + (x1 || sch/tch), data=df)
```

Extracting and viewing model information

Say your model is saved as object `lme01`

- `summary(lme01)` - View summary of `lme01`
- `fixef(lme01)` - View estimated fixed effect coefficients
- `ranef(lme01)` - View predicted random effects
- `coef(lme01)` - View coefficients for LMM for *each group*
- `VarCorr(lme01)` - View estimated variance parameters
- `confint(lme01)` - Compute confidence intervals on the parameters (cutoffs based on the likelihood ratio test)
- `confint(lme1, method="boot")` = Compute confidence intervales on the parameters (computed from the bootstrap distribution)
- `anova(lme1)` - Assess significance of fixed-effect factors
- `predict(lme1)` - View within-group fitted values for `lme01`
- `predict(lme1, re.form=NA)` - View population fitted values for `lme01`

Diagnostic plots

Say we fit the folllowing LMM:
`lme1 <- lmer(y ~ x + (x | id), data=df)`

Within-group errors

Plots for examining the assumption that within-group errors are normally distributed, centered at 0, and have constant variance.

standarized residuals vs fitted values (is the scatter uniform?):
`plot(lme1)`

standardized residuals versus fitted values by x (is the scatter uniform within groups?):
`plot(lme1, form = resid(.) ~ fitted(.) | x)`

box-plots of residuals by id (are they centered at 0?):
`plot(lme1, form = id ~ resid(.))`

To assess normality of residuals (does plot seem to lie on straight line?):
`qqnorm(resid(lme1))`

Random effects

Plots for examining the assumption that random effects are normally distributed, centered at 0, and have constant variance.

To check constant variance of random effects (is the scatter uniform?):
`plot(ranef(lme1))`

To assess normality of random effects (does plot seem to lie on straight line?):
`lattice::qqmath(ranef(lmeEng2))`

Model fit

observed versus fitted values by id (check fit of model):
`plot(lme1, y ~ fitted(.) | id)`

Comparing models

When it comes to hypothesis testing, the choice of ML vs REML estimation is important:

- To compare two models fit by **REML**, each must have the same fixed effects.
- To compare two models fit by **ML**, one must be nested within the other.

In `lmer`, REML is the default. Set `REML=FALSE` to use ML estimation.

Comparing nested models

Say we fit two models:
`lme1 <- lmer(y ~ x + z + x:z + (1 | g), data=df)`
`lme2 <- lmer(y ~ x + z + (1 | g), data=df)`

The following refits the models with ML and compares them via hypothesis test:
`anova(lme1, lme2)`

To supress refitting with ML (only do this if both models have same fixed effects): `anova(lme1, lme2, refit = FALSE)`

Dropping terms from a model

The `drop1` function will drop all possible single fixed-effect terms. We begin with a model and ask, "what would happen if we dropped a term"?

Let's say your model is `y ~ x + r + z + (1|g)`, fitted as object `lme1`.

`drop1(lme1, test="Chisq")` performs a series of Likelihood Ratio hypothesis tests to see how model is affected by individually dropping `x`, dropping `r`, and dropping `z`.

Without specifying `test="Chisq"`, we get a print out of how AIC changes when terms are dropped.

Comparing models with different random effects

The typical test is whether or not a random effect is necessary. This means testing if the variance of a random effect is 0. But variances are positive and 0 is at the boundary of the range of possible values. The result is that the standard hypothesis test (ie, a likelihood ratio test) is *conservative*. The p-value is too high. If you have a small p-value (say < 0.001), that's not a problem. If you have a p-value close to significance, (say about 0.10) you may want to consider calculating a corrected p-value using a *mixture of chi-square distributions*.

A likelihood ratio test (LRT) statistic has a chi-square distribution with degrees of freedom equal to the difference in parameters between two models. Let's say our LRT is on 2 degrees of freedom. The null distribution of this test statistic is NOT a chi-square with 2 degrees of freedom since our null value (variance $= 0$) is on the boundary of the parameter space. It's been suggested that a 50:50 mixture, $0.5\chi^2_{df} + 0.5\chi^2_{df-1}$, can serve as a reference null distribution for computing the p-value. But this is still only an approximation.

Example:
Say we fit two models:

```
lmm1 <- lmer(wt ~ weeks + treat + (1 | subject), data=ratdrink)
lmm2 <- lmer(wt ~ treat + weeks + (weeks | subject), data=ratdrink)
```

Is the `weeks` random effect necessary?

```
aout <- na.omit(anova(lmm1, lmm2, refit = FALSE)) # drop NAs
```

```
# function for 50:50 mixture
pvalMix <- function(stat,df){
  0.5*pchisq(stat, df, lower.tail = FALSE) +
     0.5*pchisq(stat, df-1, lower.tail = FALSE)
}
```