Optimization Algorithms for MLMs

Originally by Marcus Waldman, with edits by S043 Team

Updated November 2018

Convergence and optimization algorithms

Unlike OLS, which has a simple closed-form solution for parameter estimates, multi-level models are complex and often do not have closed-form solutions.[1] As a result, programming languages use optimization algorithms to fit models. These optimization algorithms are typically iterative processes that repeatedly test potential values and eventually converge to the model estimates.

Typically, optimization algorithms involve approximating the log-likelihood function as a multivariate quadratic function. Sometimes this approximation is easy to find and closely matches the true log-likelihood; in these cases, convergence occurs quickly. However, we've seen that convergence is trickier when the log-likelihood function is flat near the maximum; it's also trickier with more complex and fragile likelihoods, like those created by the link functions from Generalized Least Squares (GLS) models.

What to do when your model won't converge

If your error won't converge, you might get a warning message like this:

Warning message: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, : Model failed to converge with max|grad| = 0.0463355 (tol = 0.001, component 1)

This warning message tells us two things. First, remember that we are trying to find the maximum of the likelihood function, or the place where the `slope = 0`. In the warning, the `tol = 0.001` tells us that R will be happy if it finds estimates where the `slope` $\leq$ `0.001`. It's also saying that our slope when R stopped converging was `0.0463355`.

Steps that you can take to resolve:

1. Try rescaling variables and refitting your model
2. Try changing your optimizer settings

To address items #2 and #3, you add a `Control` option into your `lme`, `lmer`, or `glmer` function. Each of those functions has its own option, but they all take the same arguments:

1. lme: lmeControl()
2. lmer: lmerControl()
3. glmer: glmerControl()

Below are some other optimizer options that you can try. For simplicitly, we're specifying them all as "glmer" options, but you could easily adjust them to match whichever model you are trying (but failing) to fit:

```
# Use a Nelder-Mead optimizer
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
                   (gender + frl_new + f3|sch),
                 data = wide_dat, family = binomial(),
                 control = glmerControl(optimizer = 'Nelder_Mead'))
```

[1] "Closed form" means that there is a formula you can use to simply and directly calculate your estimates. For example, in OLS your matrix equation for $\hat{\beta} = (X'X)^{-1}X'Y$

```r
# Use a BFGS optimizer
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
                   (gender + frl_new + f3|sch),
                 data = wide_dat, family = binomial(),
                 control = glmerControl(optimizer="optim", optimMethod = "BFGS"))

#If these aren't working, you can downlaod a special package to use the optimx optimizer
#install.packages('optimx')
library(optimx)
log_mod <- glmer(pass ~ (gender + frl_new + f3) +
                   (gender + frl_new + f3|sch),
                 data = wide_dat, family = binomial(),
                 glmerControl(optimizer = 'optimx', calc.derivs = FALSE,
                              optCtrl = list(method = "L-BFGS-B",
                                             starttests = FALSE,
                                             kkt = FALSE)))
```

Aside from these examples, there are many other ways to adjust your optimization commands, which can be found here: https://rdrr.io/cran/lme4/man/lmerControl.html

Technical Appendix: Understanding the Types of Optimization Algorithms

There are generally four "types" of algorithms employed to find MLE/REML solutions:

1. Newton methods
2. Quasi-Newton methods
3. EM algorithm
4. Other

Newton Methods

Newton's method is the most "pure" of these approaches; essentially Newton's method uses a Taylor series approximation to approximate a quadratic function and find its maxima. It involves finding the Hessian (a matrix containing all the second and partial derivatives from your likelihood). An advantage of this approach is that it is theoretically the best of the three named approaches because it will often require fewer iterations to converge. However, there are two drawbacks:

1. When there are a large number of parameters, it is time-consuming to analytically calculate or numerically approximate all second order and mixed derivatives needed for the Hessian matrix.
2. In regions where the log-likelihood function is not sufficiently concave down, there is a tendency to dramatically overshoot because the step size to the next point is proportional to the inverse of the second derivative, resulting in pathological oscillations that would amplify if allowed to continue. Thus, where the log-likelihood function is not well approximated by a second order Taylor expansion, the method tends to fail miserably. This would be the case, for example, if the log-likelihood function was a standard normal density and you started out 2 SD from the mean.

Quasi-Newton Methods

Quasi-Newton methods start with a "guess" for the Hessian, apply the quadratic formula to attain a new point, update the guess of the Hessian, and repeat until convergence is attained. Importantly, the approximated Hessian will converge to the Hessian so long as the Wolfe conditions (a set of conditions on the likelihood) are satisfied. The easiest guess for the initial Hessian is the identity matrix, making the first step simply a gradient descent. When the identity matrix is used as an initial guess, the quasi-Newton methods converge "super-linearly"–that is it displays linear convergece initially, but approach quadratic convergence as the approximated Hessian updates itself. There are many quasi-Newton methods, but the most common is the "BFGS" updating method.

In terms of time to convergence, quasi-Newton is typically much faster than pure Newton methods. This addresses the first drawback listed for Newton's method, but it is still susceptible to the second issue. The other potential challenge with Quasi-Newton methods occurs when the Wolfe conditions are not satisfied - the method will typically not converge to the Hessian within a reasonable number of iterations, and can often exceed the maximum iterations set by a program.

EM (Expectation-Maximiation) Algorithm

The EM algorithm is another way of approximating the likelihood function and maximizing that approximation. It does this in a repeating series of stseps: the E (Expectation) step and the M (Maximization) step. In random effect models, where normality is assumed, the E-step results in an a quadratic function to be maximized in the M-step. Importantly, each iteration of the EM algorithm is guaranteed to increase the likelihood function, a feature that that may be too difficult to attain with the Newton methods when a quadratic function is not yet a good approximation. Thus, even if the likelihood function not well approximated by a quadratic function, we are assured to be getting closer to a maximum with the EM algorithm. Thus the EM algorithm fixes the second issue from Newton's method. However, it only displays linear convergence (as opposed to "super linear" or "quadratic") and can therefore take a very long time to converge.

Implementation in Different Programs

Stata/MPlus/HLM

Stata, Mplus, and HLM, each use a combination of the EM and the quasi-Newton methods when estimating models with random effects. The algorithms start with the EM algorithm and proceed until there is sufficient concavity to switch a quasi-Newton method. Using a combination of the EM and quasi-Newton methods minimizes computational time while maximizing the opportunity that the algorithm will converge to a maximum. Mplus and HLM will even switch back to the EM algorithm if the Wolfe conditions are not attained in a set amount of time; thus, my experience has been that Mplus and HLM tend to converge the fastest and tend to minimize convergence issues.

Disclaimer: sometimes you may need to manually increase the number of EM iterations allowed to acheive convergence.

R

If I am interpreting the lmerControls documentation correctly, this method starts with the EM algorithm and then applies "unconstrained and box-constrained optimization using PORT routines" from the nlminb function. I'll classify this algorithm as "other", as opposed to the three named approaches above.

In my opinion, lme's optimization algorithm is less than ideal for two reasons. First, the number of initial EM steps is fixed and who's to say that the default number of EM iterations will bring us to a region where the log-likelihood function is sufficiently concave?

Second, HLM and Mplus have been estimating random effect models for a long time, and developers from both have come to the conclusion that the quasi-Newton method as the second method in a combination is the best for these models. I'll assume this is a very informed decision on the end of these developers. Yet, it does not appear that this is what is occuring in R. Instead, R uses "unconstrained and box-constrained optimization using PORT routines," whatever that is.

Even the according to the "See Also" section in the nlminb help file, the optim function is listed as preferred over the nlminb function. As it turns out, the optim function applies the "BFGS" quasi-Newton method as the default, which is consistent with Stata's approach.