An Introduction to Model-Fitting with the R package `glmm`

Christina Knudson

April 26, 2015

Contents

1 Introduction

The R package `glmm` approximates the entire likelihood function for generalized linear mixed models (GLMMs) with a canonical link. `glmm` calculates and maximizes the Monte Carlo likelihood approximation (MCLA) to find Monte Carlo maximum likelihood estimates (MCMLEs) for the fixed effects and variance components. Additionally, the value, gradient vector, and Hessian matrix of the MCLA are calculated at the MCMLEs. The Hessian of the MCLA is used to calculate the standard errors for the MCMLEs.

The basis of `glmm` is MCLA, which was first proposed by Geyer (1990) for approximating the likelihood of unnormalized densities. MCLA was used by Geyer and Thompson (1992) to approximate the likelihood for normalized densities for models with random effects. Gelfand and Carlin (1993) proposed applying MCLA to unnormalized densities for models with random effects. The theoretical foundation for MCLA was established by Geyer (1994). Sung and Geyer (2007) prepared an R package `bernor` that, when given model matrices, fits maximum likelihood estimates for the logit-normal model. Their importance sampling distribution is chosen independently of the data.

2 Formatting the Data

The following vectors can be used to fit a generalized linear mixed model using the `glmm` package. These vectors can be contained in a data frame, but they do not need to be.

1. A response vector. If your response is Poisson, then the entries in the response vector must be natural numbers. If your response is Bernoulli, then the entries in the response vector must be 0 and 1. (For this version of `glmm`, these are the only two response types possible. If you need to fit a model with a different response, contact me.)

2. At least one vector that will be used for defining the random effects' design matrix. For this version of `glmm`, the vector(s) should be class `factor`.

3. Vector(s) that will be used for defining the fixed effects' design matrix. The vector(s) can be of class `factor` or `numeric`.

The first two types of vectors described in the list are required. The last type is optional. That is, the minimum requirement to fit a `glmm` model is

the response vector and one vector for defining the random effects' design matrix.

We use the `salamander` dataset as an example in this vignette. For your convenience, it is already included in the `glmm` package. The data arose from an experiment conducted at the University of Chicago in 1986 and were first presented by McCullagh and Nelder (1989, section 14.5). Scientists paired female and male salamanders of two types (Rough Butt and White Side) and collected data on whether or not they mated.

The variable `Mate` tells us whether the pair of salamanders mated. The value is 1 if they successfully mated and 0 if they did not. The variable `Cross` describes the type of female and male salamander. For example, `Cross = W/R` indicates a White Side female was crossed with a Rough Butt male. The variable `Female` contains the identification number of the female salamander, and the variable `Male` contains the identification number of the male salamander.

The first R command shown below gives us access to the `glmm` package and and all of its commands. The second line of code gives us access to the `salamander` data frame. The next three commands help us begin to understand the data. We have four variables: `Mate`, `Cross`, `Female`, and `Male`. The summary shows us `Mate` is numeric, `Cross` is a factor with four levels, `Female` is a factor, and `Male` is a factor.

```
library(glmm)
data(salamander)
names(salamander)
```

```
[1] "Mate"   "Cross"  "Female" "Male"
```

```
head(salamander)
```

```
  Mate Cross Female Male
1    1   R/R     10   10
2    1   R/R     11   14
3    1   R/R     12   11
4    1   R/R     13   13
5    1   R/R     14   12
6    1   R/W     15   28
```

```
summary(salamander)
```

```
      Mate       Cross        Female         Male
 Min.   :0.000   R/R:90   10     :  6   10     :  6
 1st Qu.:0.000   R/W:90   11     :  6   11     :  6
 Median :1.000   W/R:90   12     :  6   12     :  6
 Mean   :0.525   W/W:90   13     :  6   13     :  6
 3rd Qu.:1.000            14     :  6   14     :  6
 Max.   :1.000            15     :  6   15     :  6
                          (Other):324   (Other):324
```

3 Fitting the Model

Following Model A from Karim and Zeger (1992), we set `Mate` as the response, `Cross` as the fixed effect variable, and `Female` and `Male` as the random effect variables. That is, we would like to fit a generalized linear mixed model with a logit link (because the response is Bernoulli). We will have four fixed effect parameters ($\beta_{R/R}, \beta_{R/W}, \beta_{W/R}, \beta_{W/W}$). There is likely to be variability among the females and variability among the males. That is, some females will be more likely to mate than other females, and we would like the model to reflect the tendencies of the individual salamanders. We incorporate this into the model by including a random effect for each female salamander and a random effect for each male salamander. We believe the female salamanders' random effects are i.i.d. draws from $N(0, \nu_F)$, where ν_F is an unknown parameter to be estimated. Similarly, we believe the male salamanders' random effects are i.i.d. draws from $N(0, \nu_M)$, where ν_M is an unknown parameter to be estimated. Finally, we believe the female and male random effects are independent of one another.

In the following code, we fit the model using the `glmm` command and save the model under the name `sal`. Because `Mate` is our response, it is on the left of the $\sim$. We want to have a fixed effect for each of the four levels of `Cross`, so we type `Mate ~ 0 + Cross`. Because `Cross` is a factor, typing `Mate ~ Cross` would fit an equivalent model.

Next, the `random` list creates the design matrices for the random effects. Since we want two random effects for each cross (one from the female salamander and one from the male salamander), we type `list(~ 0 + Female, ~ 0 + Male)`. We include the `0` because we want our random effects to be centered at 0. Almost always, you will want your random effects to have

mean 0.

Following the `random` list, the argument `varcomps.names` allows us to name the list of variance components. In the `random` list, we have placed the females first. Therefore, the order of the variance components names are first "F" and then "M."

Next, we specify the name of our data set. This is an optional argument. If the data set is not specified, `glmm` looks to the parent environment for the variables you have referenced.

After the name of the data set, we need to specify the type of the response. In the salamander mating example, the response is binary: the salamanders either mated or they did not. Therefore, the family is `bernoulli.glmm`. If your response is a count, then the family is `poisson.glmm`.

Next, we specify our Monte Carlo sample size `m`. The general rule is the larger the Monte Carlo sample size, the more accurate the Monte Carlo likelihood approximation (MCLA) will be, and the more accurate the resulting Monte Carlo maximum likelihood estimates (MCMLEs) will be. Ideally, you want the largest `m` that time allows. For this vignette, we have chosen a Monte Carlo sample size that allows for quick computation. If you are interested in accuracy in the resulting estimates for the salamander model, we suggest a larger Monte Carlo sample size.

We put this all together in the following commands. Note that we set the seed so that we can have reproducible results. In other words, if you set your seed to the same number and type the exact command listed below, your results should be identical to those listed here. Additionally, the `proc.time` commands have been used to give you an idea of how quickly the model can be fit. The times shown here are from fitting a model on an ultrabook that cost 500 USD in 2013.

```
set.seed(1234)
ptm<-proc.time()
sal <- glmm(Mate ~ 0 + Cross, random = list(~ 0 + Female,
~ 0 + Male), varcomps.names = c("F", "M"), data = salamander,
family.glmm = bernoulli.glmm, m = 10^4, debug = TRUE)
proc.time() - ptm
```

```
   user  system elapsed
 44.315   0.024  44.384
```

3.1 Adding Optional Arguments

Additional arguments may be added for more control over the model fit. If you're an introductory user, go ahead and ignore this section.

3.1.1 Setting Variance Components Equal

By default, `glmm` assumes each variance component should be distinct. Suppose we want to set $\nu_F = \nu_M$. Then we would add the argument `varcomps.equal` to indicate the equality. Since the list of random effects has two entries and we want those entries to share a variance component, we would set `varcomps.equal = c(1,1)`. In this scenario, we would only have one variance component, so we only need one entry in `varcomps.names`. Thus, the new command to fit this updated model with one variance component could be the following:

```
sal <- glmm(Mate ~ 0 + Cross, random = list(~ 0 + Female,
~ 0 + Male), varcomps.equal = c( 1, 1), varcomps.names =
c("Only Varcomp"), data = salamander, family.glmm =
bernoulli.glmm, m = 10^4, debug = TRUE)
```

As another example, suppose the list `random` has three entries, indicating three variance components ν_1, ν_2, ν_3. To set $\nu_1 = \nu_3$, we write `varcomps.equal = c(1,2,1)`. Thus, the shared variance component would be listed first in any output, and ν_2 would be listed second. Note that the entries in the `varcomps.equal` vector must start at 1, then continue through the integers. The order of the names of the variance components listed in `varcomps.names` must correspond to the integers in `varcomps.equal`. In this problem, the names could be `varcomps.names = c("shared", "two")`.

3.1.2 Altering the Importance Sampling Distribution

The following default arguments can be adapted to alter the importance sampling distribution: `doPQL`, `p1`, `p2`, `p3`, and `zeta`.

By default, penalized quasi-likelihood estimates are used to form the importance sampling distribution for the generated random effects. To skip PQL, add the argument `doPQL=FALSE`. If PQL is skipped, then the importance

sampling distribution uses arbitrary estimates of 0 for the random effects, 0 for the fixed effects, and 1 for the variance components. Sometimes the examples in the `glmm` documentation skip the PQL step so that the package can load more quickly. Most of the time, the model will fit more accurately and efficiently if PQL estimates are used in the importance sampling distribution.

The importance sampling distribution is a mixture of three distributions. By default, the mixture is evenly weighted, with each component's contribution set at 1/3. If you wish to change the mixture, you can alter `p1`, `p2`, and `p3` from the default of `p1 = 1/3`, `p2 = 1/3`, and `p3 = 1/3`. The only restrictions are that the three probabilities must sum to 1 and `p1` must be positive.

The first component of the importance sampling distribution is a scaled multivariate t-distribution with `zeta` degrees of freedom. Therefore, another way to alter the importance sampling distribution is by changing `zeta` from its default of 5.

3.1.3 Adjusting Optimization Arguments

It may be useful to adjust the `trust` arguments `rmax` and `iterlim`. The argument `rmax` is the maximum allowed trust region radius. By `glmm` default, this is set to the arbitrary, somewhat large number of 1000. If this is set to a small number, then the optimization will move more slowly.

The argument `iterlim` must be a positive integer that limits the length of the optimization. If `iterlim` is too small, then the `trust` optimization will end before the MCMLA has been maximized.

If `iterlim` is reached, then `trust` has not converged to the MCMLE. When the `summary` command is called, a warning will be printed telling the user that the parameter values are not MCMLEs, but `glmm` can be rerun starting at these outputted parameter values. To do this, use the `par.init` argument in section 3.1.4.

3.1.4 Starting at a Specified Parameter Value

Rather than using the PQL estimates, you can provide parameter values to `glmm` using the argument `par.init`. The `glmm` argument `par.init` is a vector that specifies the user-supplied values of the fixed effects and variance

components. The parameters must be inputted in the order that `summary` outputs them, with fixed effects followed by variance components.

If `par.init` is provided, then PQL estimates will not be computed. The `par.init` estimates will be used instead to form the importance sampling distribution. Then, `trust` will use `par.init` as the starting point for the optimization. This argument may be useful for very hard problems that require iteration.

4 Reading the Model Summary

The `summary` command displays

- the function call (to remind you of the model you fit).
- the link function.
- the fixed effect estimates, their standard errors (calculated using observed Fisher information), their `z value` test statistics (testing whether the coefficients are significantly different from zero), the test's p-values, and the R-standard significance stars (optional).
- the variance component estimates, their standard errors (calculated using observed Fisher information), their `z value` test statistics (testing whether the coefficients are significantly different from zero), the test's p-values, and the R-standard significance stars (optional).

Note that the p-value for the fixed effects is calculated using a two-sided alternative hypothesis ($H_A : \beta \neq 0$) while the p-value for the variance components is calculated using a one-sided alternative hypothesis ($H_A : \nu > 0$) because variance components must be nonnegative.

To view the model summary, we use the `summary` command.

```
summary(sal)
```

```
Call:
glmm(fixed = Mate ~ 0 + Cross, random = list(~0 + Female, ~0 +
    Male), varcomps.names = c("F", "M"), data = salamander, family.glmm = bernoulli.g
    m = 10^4, debug = TRUE)
```

```
Link is: "logit (log odds)"

Fixed Effects:
         Estimate Std. Error z value Pr(>|z|)
CrossR/R   0.9560     0.3503   2.729  0.00634 **
CrossR/W   0.2805     0.3660   0.766  0.44347
CrossW/R  -1.8968     0.4223  -4.492 7.05e-06 ***
CrossW/W   0.9723     0.3580   2.716  0.00661 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Variance Components for Random Effects (P-values are one-tailed):
  Estimate Std. Error z value Pr(>|z|)/2
F   1.2878     0.4435   2.904    0.00184 **
M   1.0840     0.4131   2.624    0.00435 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Looking at our output, we can see that the type of cross significantly affects the salamanders' odds of mating. Additionally, both the variance components are significantly different from zero and should be retained in the model.

The summary provides the estimates needed to write our model. First, we establish a little notation. Let π_i represent the probability of successful mating for salamander pair i. Let $I()$ be an indicator function, so that $I(\text{Cross=R/R})$ is 1 when the variable `Cross = R/R` and 0 otherwise. Let u_i^F represent the random effect from the female salamander in the ith pair. Let u_i^M represent the random effect from the male salamander in the ith pair. Since the response is Bernoulli, the canonical link is the log odds of successful mating. Using this notation, we write the model as follows.

$$
\begin{aligned}
\log\left(\frac{\pi_i}{1-\pi_i}\right) &= 0.956 * I(\text{Cross=R/R}) + 0.2805 * I(\text{Cross=R/W}) \\
&\quad + -1.8968 * I(\text{Cross=W/R}) + 0.9723 * I(\text{Cross=W/W}) \\
&\quad + u_i^F + u_i^M \\
u_i^F &\overset{i.i.d.}{\sim} N(0, 1.288) \\
u_i^M &\overset{i.i.d.}{\sim} N(0, 1.084)
\end{aligned}
$$

Recall that `m` in the above model was chosen for convenience to save time. The resulting parameter estimates have a little too much variability. If we increase `m`, the Monte Carlo standard error decreases.

5 Isolating the Parameter Estimates

If we wish to extract the estimates for the fixed effect coefficients or the variance components, we use the commands `coef` and `varcomps`, respectively. These commands isolate the estimates that are shown in the summary (as displayed in section 4).

To extract the fixed effect coefficients, the only argument needed is the model. The commands `coef` and `coefficients` are interchangeable. We can type either of the following:

```
coef(sal)
```

```
  CrossR/R   CrossR/W   CrossW/R   CrossW/W
 0.9560113  0.2804932 -1.8968316  0.9722904
```

```
coefficients(sal)
```

```
  CrossR/R   CrossR/W   CrossW/R   CrossW/W
 0.9560113  0.2804932 -1.8968316  0.9722904
```

To extract the variance components, the only argument needed is the model.

```
varcomps(sal)
```

```
       F        M
1.287848 1.083975
```

To further isolate variance components or fixed effects, use indexing. The following demonstrates how to extract the last two fixed effects and the first variance component.

```
coef(sal)[c(3,4)]
```

```
  CrossW/R   CrossW/W
-1.8968316  0.9722904
```

```
varcomps(sal)[1]
```

```
       F
1.287848
```

6 Calculating Confidence Intervals

We can calculate confidence intervals for parameters using the `confint` command. (Note that prediction is not yet possible in this version of the package). If we wish to calculate 95% confidence intervals for all of our parameters, the only argument is the model name.

```
confint(sal)
```

```
               0.025      0.975
CrossR/R   0.9647676  1.2975071
CrossR/W   0.2896434  0.6373526
CrossW/R -1.8862749 -1.4851238
CrossW/W   0.9812412  1.3213731
F          1.2989347  1.7202402
M          1.0943035  1.4867789
```

The output is a matrix. Each row represents one parameter. The first column is the lower bound of the confidence interval, and the second column is the upper bound of the confidence interval.

If we wish to change the level of confidence from the default of 95%, we use the argument `level` and specify a number between 0 and 1. For example, to 90% confidence intervals and 99% confidence intervals, we type the following:

```
confint(sal,level=.9)
```

```
               0.05       0.95
CrossR/R  0.9735239  1.2887508
CrossR/W  0.2987937  0.6282024
CrossW/R -1.8757183 -1.4956804
CrossW/W  0.9901921  1.3124222
F         1.3100217  1.7091532
M         1.1046318  1.4764506
```

```
confint(sal,level=.99)
```

```
              0.005      0.995
CrossR/R  0.9577625  1.3045121
CrossR/W  0.2823232  0.6446728
CrossW/R -1.8947202 -1.4766785
CrossW/W  0.9740806  1.3285338
F         1.2900651  1.7291098
M         1.0860409  1.4950415
```

We can calculate 90% confidence intervals for the first and third fixed effects through indexing or by listing the names of the fixed effects:

```
confint(sal,level=.9,c(1,3))
```

```
               0.05      0.95
CrossR/R  0.9735239  1.288751
CrossW/R -1.8757183 -1.495680
```

```
confint(sal,level=.9,c("CrossR/R","CrossW/R"))
```

```
               0.05      0.95
CrossR/R  0.9735239  1.288751
CrossW/R -1.8757183 -1.495680
```

To calculate a 93 percent confidence interval for the variance component for the female salamanders, we can again either use indexing or list the name of the variable. Note that there are four fixed effects so ν_F is the fifth parameter in this model. (Similarly, ν_M is the sixth parameter in this model).

```
confint(sal,level=.93,c(5))
```

```
    0.035    0.965
F 1.30337 1.715805
```

```
confint(sal,level=.93,c("F"))
```

```
    0.035    0.965
F 1.30337 1.715805
```

Note that all confidence intervals are calculated using the observed Fisher information from the Monte Carlo likelihood approximation.

7 Estimating the Variance-Covariance Matrix

The variance-covariance matrix for the parameter estimates can be found using the `vcov` function. The only input is the model name.

```
vcov(sal)
```

The variance-covariance matrix can be useful for some hypothesis testing. For example, suppose we want to test the hypotheses:

$$H_0 : \beta_{RR} - \beta_{WW} = 0$$
$$H_0 : \beta_{RR} - \beta_{WW} \neq 0.$$

The test statistic is

$$\frac{\hat{\beta}_{RR} - \hat{\beta}_{WW} - 0}{\sqrt{\text{Var}\left(\hat{\beta}_{RR} - \hat{\beta}_{WW}\right)}} \sim N(0, 1).$$

To calculate is the denominator of the test statistic is

$$\text{Var}\left(\hat{\beta}_{RR} - \hat{\beta}_{WW}\right) = \text{Var}\left(\hat{\beta}_{RR}\right) + \text{Var}\left(\hat{\beta}_{WW}\right) - 2\,\text{Cov}\left(\hat{\beta}_{RR}, \hat{\beta}_{WW}\right)$$

and these variances and covariances are found in the variance-covariance matrix.

8 Accessing Additional Output

The model produced by `glmm` has information that is not displayed by the `summary` command. The `names` command helps us see what we can access.

```
names(sal)
```

```
 [1] "beta"                "nu"                  "likelihood.value"
 [4] "likelihood.gradient" "likelihood.hessian"  "trust.converged"
 [7] "mod.mcml"            "fixedcall"           "randcall"
[10] "x"                   "y"                   "z"
[13] "family.glmm"         "call"                "varcomps.names"
[16] "varcomps.equal"      "debug"
```

The first two items are `beta` and `nu`. These are the MCMLEs for the fixed effects and variance components.

The third item is `likelihood.value`, the value of the MCLA evaluated at the MCMLEs. The fourth item is `likelihood.gradient`, the gradient vector of the MCLA evaluated at the MCMLEs. The fifth item is `likelihood.hessian`, the Hessian matrix of the MCLA evaluated at the MCMLEs.

Next is `trust.converged`, which tell us whether the `trust` function in the `trust` package was able to converge to the optimizer of the MCLA.

Items 7 through 16 relate to the original function call. `mod.mcml` contains the model matrix for the fixed effects, a list of model matrices for the random effects, and the response vector. These are also displayed in `x`, `z`, and `y`, respectively. Then, the call (the original formula representations of the fixed and random effects) are contained in `fixedcall`, `randcall`, and `call`.

The last argument is `debug`. If the model was fit with the default `debug = FALSE`, then this argument is just `FALSE`. If the model was fit with `debug = TRUE`, then `debug` contains a list of output for advanced users and programmers.

References

Gelfand, A. and Carlin, B. (1993). Maximum-likelihood estimation for constrained- or missing-data models. *Canadian Journal of Statistics*,

21:303–311.

Geyer, C. (1990). *Likelihood and Exponential Families*. PhD thesis, University of Washington.

Geyer, C. J. (1994). On the convergence of Monte Carlo maximum likelihood calculations. *Journal of the Royal Statistical Society, Series B*, 61:261–274.

Geyer, C. J. and Thompson, E. (1992). Constrained Monte Carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society, Series B*, 54:657–699.

Karim, M. and Zeger, S. (1992). generalized linear models with random effects; salamander mating revisited. *Biometrics*, 48:631–644.

McCullagh, P. and Nelder, J. (1989). *Generalized Linear Models*. Chapman and Hall/CRC.

Sung, Y. J. and Geyer, C. J. (2007). Monte Carlo likelihood inference for missing data models. *Annals of Statistics*, 35:990–1011.