A Vignette for the R package `glmm`

Christina Knudson

February 15, 2015

1 Introduction

The R package `glmm` approximates the entire likelihood function for generalized linear mixed models (GLMMs) with a canonical link. After calculating the Monte Carlo likelihood approximation (MCLA), `glmm` maximizes it 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 is returned at the MCMLEs. The Hessian of the MCLA is used to calculate the standard errors for the MCMLEs.

The basis of `glmm` is Monte Carlo likelihood approximation, which was first proposed by Geyer and Thompson (1992). The greatest strength of MCLA is its solid theoretical basis established by Geyer (1994). In particular, the only approximation in this package is the likelihood is approximated by Monte Carlo. This approximation can be made as accurate as you wish by increasing the Monte Carlo sample size (m). As the Monte Carlo sample size increases, the MCLA converges to the likelihood. As the MCLA converges, all likelihood-based inference converges as well. For example, the MCMLEs converge to the MLEs.

This vignette demonstrates how to format the data in section 2, fit the model 3, access the model summary 4, calculate confidence intervals 5, extract fixed effect estimates and variance estimates 6, and access additional output 7.

2 Formatting the Data

In order to use the `glmm` package, your data frame must have the following:

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.

2. at least one vector that will be used for defining the fixed effects. This vector can be a factor or numeric.

3. at least one vector that will be used for defining the random effects. For this version of `glmm`, the vector(s) should be class "factor."

We use the `salamander` dataset as an example in this vignette. For your convenience, it is already included in the `glmm` package. The data are from an experiment conducted at the University of Chicago in 1986 and first presented by McCullagh and Nelder in 1989. 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

We would like to fit a model using `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 (glmm) with a logit link (because the data are 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 tendences 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 $N(0, \nu_F)$. Similarly, we believe the male salamanders' random effects are i.i.d. draws from $N(0, \nu_M)$. 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 defines the random effects. We want two random effects for each cross: one from the female salamander and one from the male salamander. Note that we type `~ 0 + Female` and `~ 0 + Male`. We do this because we want our random effects to be centered at 0. Almost always, you will want your random effects to have mean 0. Next, the argument `varcomps.names` allows us to name the list of variance components. In the salamander model, we have placed the females first in the list defining the random effects. 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.

Next, 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. To see whether your MCMLEs are stable, you can fit the model several times starting from different seeds. More research needs to be done on this subject before I can give you more concrete suggestions on satisfactory Monte Carlo sample sizes. For this vignette, I 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,I suggest a larger Monte Carlo sample size.

Finally, we can decide whether we would like additional output (see details in section 7. If we would like to see the additional output, we type `debug = TRUE`. The default is `debug = FALSE`.

We put this all together in the follow 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.

```
set.seed(1234)
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)
```

3.1 Adding Optional Arguments

Additional arguments may be added for more control over the model fit.

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.

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 distribution. 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 MCMLEs have been identified.

4 Reading the Model Summary

The `summary` command displays

- the function call (to remind you of the model you fit)
- 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 p-values, and the R-standard significance stars.
- 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 p-values, and the R-standard significance stars.

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.glmm,
##     m = 10^4, debug = TRUE)
##
## 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.

5 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% to 90% or 99%, 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% confidence interval for the variance component for the female salamanders, we can again either use indexing or list the name. 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. As the Monte Carlo sample size increases, the Monte Carlo likelihood approximation converges to the likelihood.

6 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
```

7 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 more output. We now look at the `debug` output.

```
out<-sal$debug
names(out)
```

```
##  [1] "beta.pql"       "nu.pql"         "trust.argpath"  "u.star"
##  [5] "umat"           "weights"        "wtsnumer"       "wtsdenom"
##  [9] "m1"             "m2"             "m3"             "trust.argtry"
## [13] "trust.steptype" "trust.accept"   "trust.r"        "trust.rho"
## [17] "trust.valpath"  "trust.valtry"   "trust.preddif"  "trust.stepnorm"
```

First, we have the PQL estimates `beta.pql`, `nu.pql`, and `u.star`, which were used to create the importance sampling distribution.

Next, we have output from `trust`. The `trust` output can be understood by reading the `trust` documentation. All `trust` output begins with `trust.` and ends with the name given by `trust`. For example, `trust.argpath` is the `trust` output named `argpath`.

The next output is `umat`, a matrix of the random effects generated from the importance sampling distribution. The matrix has m rows. The number of columns is equal to the total number of random effects in the model.

The next three arguments relate to the importance sampling weights. `weights` are the importance sampling weights, `wtsnumer` is the numerator of the weights, and `wtsdenom` is the denominator of the weights.

Finally, the next three arguments are `m1`, `m2`, and `m3`. The importance sampling distribution is a mixture of three distributions. `m1`, `m2`, and `m3` represent the number of draws taken from each of the components of the importance sampling distribution. More information on the importance sampling distribution can be found in the "Details" section of the `glmm` documentation.