Design Document: Monte Carlo Maximum Likelihood for Generalized Linear Mixed Models

Christina Knudson

November 1, 2014

Abstract

This design document describes the process of performing Monte Carlo maximum likelihood (MCML) for generalized linear mixed models. First, penalized quasi-likelihood (PQL) estimates are calculated, which help generate simulated random effects. Then, the Monte Carlo likelihood approximation (MCLA) is calculated using the simulated random effects. Next, the MCLA is maximized to find Monte Carlo maximum likelihood estimates, the corresponding Fisher Information, and other statistics. Additional inference is then possible, including confidence intervals for the parameters and likelihood ratio tests for comparing nested models.

1 Theory

Let $y = (y_1, \ldots, y_n)^T$ be a vector of observed data. Let $u = (u_1, \ldots, u_q)'$ be a vector of unobserved random effects. Let β be a vector of p fixed effect parameters and let ν be a vector of T variance components for the random effects. Let θ be a vector of length $p + T$ containing all unknown parameters. Then the data y are distributed conditionally on the random effects according to $f_\theta(y|u)$ and the random effects are distributed according to $f_\theta(u)$. Although $f_\theta(u)$ does not actually depend on β and $f_\theta(y|u)$ does not depend on ν, we write the density like this to keep notation simple in future equations.

Since u is unobservable, the log likelihood must be expressed by integrating out the random effects:

$$l(\theta) = \log \int f_\theta(y|u) f_\theta(u)\, du \tag{1}$$

For most datasets, this integral is intractible. In these cases, performing even basic inference on the likelihood is not possible. Instead of evaluating the integral, Geyer and Thompson (1992) suggest using a Monte Carlo approximation to the likelihood. Monte Carlo likelihood approximation

(MCLA) uses an importance sampling distribution $\tilde{f}(u)$ to generate random effects $u_k, k = 1, \ldots, m$ where m is the Monte Carlo sample size. Then the Monte Carlo log likelihood approximation is

$$l_m(\theta) = \log \frac{1}{m} \sum_{k=1}^{m} f_\theta(y|u_k) \frac{f_\theta(u_k)}{\tilde{f}(u_k)} \tag{2}$$

$$= \log \frac{1}{m} \sum_{k=1}^{m} \frac{f_\theta(y, u_k)}{\tilde{f}(u_k)} \tag{3}$$

the gradient vector of the MCLA is

$$\nabla l_m(\theta) = \frac{\sum_{k=1}^{m} \left(\nabla \log f_\theta(u_k, y) \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}{\sum_{k=1}^{m} \left(\frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}, \tag{4}$$

and the Hessian matrix of the MCLA is

$$\nabla^2 l_m(\theta) = \frac{\sum_{k=1}^{m} \left(\nabla^2 \log f_\theta(u_k, y) \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}{\sum_{k=1}^{m} \left(\frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)} - \nabla l_m(\theta)(\nabla l_m(\theta))' \tag{5}$$

$$+ \frac{\sum_{k=1}^{m} \left(\nabla \log f_\theta(u_k, y)(\nabla \log f_\theta(u_k, y))' \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}{\sum_{k=1}^{m} \left(\frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}. \tag{6}$$

Now any inference, such as maximum likelihood, can be performed on $l_m(\theta)$. MCLA theoretically works for any $\tilde{f}(u)$, but the $\tilde{f}(u)$ chosen for this package is a mixture distribution specified in section 8.2.

EDITED TO HERE.

Let X be an $n \times p$ design matrix for the fixed effects. Let U be an unobservable normal random vector with length q, mean 0 and variance matrix D. Let Z be a $n \times q$ model matrix for the random effects.

There are T distinct variance components ν_t, $t = 1, \ldots, T$. For the first version of the package, D is assumed diagonal. Let E_t be a diagonal matrix with indicators on the diagonal so that $\sum_{t=1}^{T} E_t = I$. (Learn more about E_t in section 6.) Then $D = \sum_{t=1}^{T} \nu_t E_t$. Later versions of this package will allow more general covariance structures. With other structures, D will be constructed by $D = \sum_{t=1}^{T} h_t(\nu_t)\ E_t$, but E_t will no longer be diagonal and h_t will be some function. We will have a D for distance (such as in the car theft data) and for autoregressive 1.

Denote the density of U by $f_\theta(u)$ with $\theta = (\beta, \nu_1, \ldots, \nu_T)$. Although $f_\theta(u)$ does not actually depend on β, we write the density like this to keep notation simple. Since U is normally distributed,

we can write its log density as

$$\log f_\theta(u) = -\frac{1}{2}\log|D| - \frac{1}{2}U'D^{-1}U \tag{7}$$

where $|D|$ denotes the determinant of D. More details on the distribution of the random effects can be found in section 8.1.

Let g be the canonical link function and μ be a vector of length n such that

$$g(\mu) = X\beta + ZU \tag{8}$$

The choice of the link function is related to the distribution of the data, $\log f_\theta(y|u)$. If the data have a Bernoulli distribution, the link is $\text{logit}(\mu)$. If the data have a Poisson distribution, the link is $\log(\mu)$. More on these two families can be found in section 3. For simplicity of future notation, let $\eta = g(\mu) = X\beta + ZU$. Let $c(\eta)$ denote the cumulant function such that the log of the data density can be written as

$$Y'\eta - c(\eta) = \sum_i [Y_i\eta_i - c(\eta_i)] \tag{9}$$

Since U is unobservable, the likelihood must be expressed by integrating out the random effects:

$$l(\theta) = \int f_\theta(y|u)f_\theta(u)\,du \tag{10}$$

For most datasets, this integral is intractible. In these cases, performing even basic inference on the likelihood is not possible. Instead of evaluating the integral, Geyer and Thompson (1992) suggest using a Monte Carlo approximation to the likelihood. Monte Carlo likelihood approximation (MCLA) uses an importance sampling distribution $\tilde{f}(u)$ to generate random effects $u_k, k = 1, \ldots, m$ where m is the Monte Carlo sample size. Then the MCLA of $l(\theta)$ is:

$$l_m(\theta) = \sum_{k=1}^{m} \frac{f_\theta(y|u_k)f_\theta(u_k)}{\tilde{f}(u_k)} \tag{11}$$

Now any inference, such as maximum likelihood, can be performed on $l_m(\theta)$. MCLA theoretically works for any $\tilde{f}(u)$, but the $\tilde{f}(u)$ chosen for this package is a mixture distribution specified in section 8.2.

2 Model fitting function

This will be the main function the user will use. The users will need to specify the response and the predictors using the R formula mini-language as interpreted by model.matrix. They'll need

to specify the family (either binomial or Poisson, though really any exponential family would work and I could add more later). The user will specify the random effects in the same way as for the R function `reaster` in the R package `aster` (Geyer, 2014). That is, random effects will be expressed using the R formula mini-language. Thus, a sample command with fixed predictors x_1 and x_2 and with random effects *school* and *classroom* (in data set as categorical variables) would look like

```
glmm(y ~ x1+ x2, list(~0+school,~0+classroom),  family.glmm="binomial.glmm",
data=schooldat,varcomps.names=c("school","classroom"),varcomps.equal=c(1,2),
debug=FALSE )
```

Section 3 contains more information on the family.

It's possible that the user could want some variance components to be set equal. For example, in the Coull and Agresti flu problem, there are 4 years and random effects for each year. The authors want the within-year variance components to be equal. There are also variance components for subject-specific intercepts and for the decreased susceptibility to illness in year 4 (since year 4's virus was a repeat). Suppose year is a categorical variable with four levels, and year1 through year4 are dummy variables. Thus, the call could contain these arguments

```
glmm(y~year,list(~0+subject,~0+year1,~0+year2,~0+repeet,~0+year3,~0+year4),
 varcomps.equal=c(1,2,2,3,2,2), varcomps.names=c("subject","year","repeet"),
data=flu,family.glmm=bernoulli.glmm)
```

Thus model.matrix will make Z into a list of 6 design matrices. Since we have 3 distinct variance components, we want 3 design matrices. What we do is we take the design matrices that share a variance component and then cbind them together. Thus we'll have 3 model matrices in our Z list, one for each random effect. We will want to put them in order (1,2,3 according to varcomps.equal) so that the names for each matrix line up ("subject","year","repeet"). Cbinding the design matrices that share a variance component will not effect the way that we compute η. Also, the variance estimates should come out in this same order as well.

After interpreting the model the user has specified, the next step is to find penalized quasi-likelihood (PQL) estimates. The process of finding these estimates is detailed in section 4. The PQL estimates will help determine the means and variances for the normal distributions from which random effects will be generated. After sampling random effects from a standard normal, they will be scaled and centered to have the desired mean and variance. More information on generated the random effects is in section 7. Next, I'll implement trust on the objective function (details on the objective function are in section 9. Finally, the function will return parameter estimates, the log likelihood evaluated at those estimates, the gradient vector of the log likelihood, and the Hessian

of the log likelihood at those estimates.

3 Families

This function will be hidden from the user. These functions (along with the distribution of random effects) are necessary to approximate the log likelihood.

I'm going to have an S3 class called "glmm.family" with a few options (for now, binomial and poisson). Each family function will output a list including the family name (a character string such as "binomial"), a function that calculates the value of the cumulant function $c(\eta)$, a function that calculates the cumulant's first derivative $c'(\eta)$ with the derivative taken with respect to η, and a function that calculates the cumulant's second derivative $c''(\eta)$.

The user will provide the family in the model-fitting function. They can either enter the character string ("bernoulli.glmm"), the function (bernoulli.glmm()), or the result of invoking it. The following code for using the input to determine the family is adapted from glm.

```
logDensity<-function(family.glmm)
{
if(is.character(family.glmm))
family.glmm<-get(family.glmm,mode="function",envir=parent.frame())
if(is.function(family.glmm))
family.glmm<-family.glmm()
if(!inherits(family.glmm,"glmm.family"))
stop(" 'family.glmm' not recognized")
return(family.glmm)
}
```

We interpret this as follows. If the user has entered the family as a string, go get the R object with that family name, either from the immediate environment or the parent environment. If this has happened, `family.glmm` is now a function. If `family.glmm` is a function (either because the user entered it as a function or because of the preceding step), invoke that function. At this point, `family.glmm` should have class "`glmm.family`." If this is not the case (maybe because of a typo or maybe because they entered "`poisson`" rather than "`poisson.glmm`"), then stop and return an error.

For example, one of the family functions will look like this:

```
poisson.glmm<-function()
{
```

```
family.glmm<- "poisson.glmm"
c <- function(eta) exp(eta)
cp <- function(eta) exp(eta)
cpp<-function(eta) exp(eta)
out<-list(family.glmm=family.glmm, c=c,cp=cp,cpp=cpp)
class(out)<-"glmm.family"
return(out)
}
```

Then, to use these functions in order to calculate $c(\eta_i), c'(\eta_i)$, and $c''(\eta_i)$, I can just call

```
family.glmm$c(args)
family.glmm$cp(args)
family.glmm$cpp(args)
```

For Bernoulli, we calculate these values (c, cp, and cpp) with careful computer arithmetic as follows. We also use the `log1p` function in R for $c(\eta_i)$.

$$c(\eta_i) = \log(1 + e^{\eta_i}) = \begin{cases} \log(1 + e^{\eta_i}) & \text{if } \eta_i \leq 0, \\ \eta_i + \log(e^{-\eta_i} + 1) & \text{if } \eta_i > 0, \end{cases} \tag{12}$$

$$c'(\eta_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}} = \frac{1}{1 + e^{-\eta_i}} \tag{13}$$

$$c''(\eta_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}} - \frac{e^{2\eta_i}}{(1 + e^{\eta_i})^2} = \frac{1}{1 + e^{-\eta_i}} \cdot \frac{1}{1 + e^{\eta_i}} \tag{14}$$

We use the `log1p` function in R for $c(\eta_i)$ for the Bernoulli family. For poisson, these values (c,cp, and cpp) are

$$c(\eta_i) = e^{\eta_i} \tag{15}$$

$$c'(\eta_i) = e^{\eta_i} \tag{16}$$

$$c''(\eta_i) = e^{\eta_i}. \tag{17}$$

Then we use these pieces to create the scalar $c(\eta)$, the vector $c'(\eta)$ and the matrix $c''(\eta)$. We calculate

$$c(\eta) = \sum_i c(\eta_i). \tag{18}$$

The vector $c'(\eta)$ has components $c'(\eta_i)$. The matrix $c''(\eta)$ is diagonal with diagonal elements $c''(\eta_i)$.

In the R function `glm` binomial, the user can choose the link. The canonical link must be used in the `glmm` package so that we have an exponential family. I'm going to include the link in the value of `family.glmm` just in case the user doesn't know it already.

Also, I'll have a function to check that the data are valid given the family type. So I'll make sure that if `family.glmm` is `bernoulli.glmm`, the data should be 0's or 1's. If `family.glmm` is `poisson.glmm`, then the data should be nonnegative integers. If that's not true, the check returns an error message.

3.1 Redone in C

When I redo this in C, I will have a separate function for `cum` ,`cp`, and `cpp`. The inputs will be `eta` (an array of doubles), `neta` (the length of `eta`), the type (to denote the family), and an array of doubles to contain the result. These will be passed in as pointers. The functions will calculate the cumulant function or one of its first two derivatives. Each function will contain a switch statement for the glmm family. The calculations for each of these functions have been shown earlier in this section. This function will be type void: rather than returning the cumulant or its derivatives, the pointers will be changed to contain the results. This function will be invoked by `el`, descriped in section 5.

4 PQL

Before we get started on describing PQL, we need to change notation to avoid constrained optimization. Recall that $D = \mathrm{Var}(u)$ and is (for now) assumed to be diagonal. Let $A = D^{1/2}$ so that A has diagonal components that are positive or negative. Using A rather than D enables unconstrained optimization. If σ is a vector of the distinct standard deviations with components σ_t, we can write A as a function of σ by

$$A = \sum_t E_t \sigma_t.$$

Recall that E_t has a diagonal of indicators to show which random effects have the same variance components, and $\sum_{t=1}^{T} E_t$ is the identity matrix. PQL will estimate the components contained on the diagonal of A. Taking the absolute value of those components will provide the standard deviations (the square root of the variance components).

In addition to using A rather than D, the other change is that we'll be using s where $u = As$. The purpose of this is to avoid $D^{-1/2}$ in the functon we're trying to optimize.

There are two ways to do PQL, both of which are described in the vignette `re.pdf` in the R

package `aster` (Geyer, 2014). In either case, there is an inner optimization and an outer optimization. The inner optimization is well behaved while the other optimization is a little tougher. I will be using the method that is not quite PQL but is pretty close and is better behaved. In this version, the inner optimization finds $\tilde{\beta}$ and $\tilde{s}$ given X, Z and A. Then, given $\tilde{\beta}$ and $\tilde{s}$, the outer optimization finds A.

The **inner** optimization will be done with the trust function in R. We elect to use trust because it requires two derivatives, which will make the optimization more precise. We would like more accuracy in the inner maximization because any sloppiness will carry into the outer optimization.

The inner optimization maximizes the penalized log likelihood. After defining

$$\eta = X\beta + ZAs \tag{19}$$

we calculate the log likelihood as

$$l(\eta) = Y'\eta - c(\eta) \tag{20}$$

and the penalized likelihood as:

$$l(\eta) - \frac{1}{2}s's. \tag{21}$$

This function will be maximized using the `trust` package. We need to give `trust` derivatives with respect to s and β. We're going to express these via the multivariate chain rule, taking advantage of η_i.

Create vector μ from the components $\mu_i = c'(\eta_i)$. Since

$$l(\eta) = \sum_i Y_i\eta_i - c(\eta_i) \tag{22}$$

then

$$\frac{\partial l(\eta)}{\partial \eta_i} = Y_i - c'(\eta_i) = Y_i - \mu_i. \tag{23}$$

Now we can write the following expression:

$$\begin{aligned}
\frac{\partial}{\partial \beta_k}\left[l(\eta) - \frac{1}{2}s's\right] &= \frac{\partial l(\eta)}{\partial \beta_k} && (24)\\
&= \frac{\partial l(\eta)}{\partial \eta_i}\frac{\partial \eta_i}{\partial \beta_k} && (25)\\
&= \sum_i (Y_i - \mu_i)\frac{\partial \eta_i}{\partial \beta_k} && (26)\\
&= \sum_i (Y_i - \mu_i)X_{ik} && (27)
\end{aligned}$$

We find the function's derivative with respect to s as follows:

$$\frac{\partial}{\partial s}\left[l(\eta) - \frac{1}{2}s's\right] = \frac{\partial l(\eta)}{\partial s} - \frac{1}{2}\frac{\partial s's}{\partial s} \tag{28}$$

$$= \frac{\partial l(\eta)}{\partial \eta}\frac{\partial \eta}{\partial s} - s \tag{29}$$

$$= (Y-\mu)'\left[\frac{\partial}{\partial s}ZAs\right] - s \tag{30}$$

$$= (Y-\mu)'ZA - s \tag{31}$$

This gives us the following derivatives of the penalized log likelihood:

$$\frac{\partial}{\partial \beta}\left[l(\eta) - (1/2)s's\right] = X'(Y-\mu) \tag{32}$$

$$\frac{\partial}{\partial s}\left[l(\eta) - (1/2)s's\right] = AZ'(Y-\mu) - s \tag{33}$$

Lastly, we need the Hessian of the penalized likelihood. This matrix can be broken down into four pieces:

1. $\dfrac{\partial^2}{\partial s^2}$
2. $\dfrac{\partial^2}{\partial \beta^2}$
3. $\dfrac{\partial^2}{\partial s\,\partial \beta}$
4. $\left(\dfrac{\partial^2}{\partial s\,\partial \beta}\right)' = \dfrac{\partial^2}{\partial \beta\,\partial s}$

We'll start at the top with $\dfrac{\partial^2}{\partial s^2}$, which is a $q \times q$ matrix.

$$\frac{\partial^2}{\partial s^2}\left[l(\eta) - (1/2)s's\right] = \frac{\partial}{\partial s}\left[(Y - c'(\eta))'ZA - s\right] \tag{34}$$

$$= \left[-\frac{\partial}{\partial s}c'(\eta)\right]'ZA - I_q \tag{35}$$

$$= \left[-\frac{\partial c'(\eta)}{\partial \eta}\frac{\partial \eta}{\partial s}\right]'ZA - I_q \tag{36}$$

$$= \left[-c''(\eta)ZA\right]'ZA - I_q \tag{37}$$

$$= -AZ'\left[c''(\eta)\right]ZA - I_q \tag{38}$$

Note that $c''(\eta)$ is a $q \times q$ diagonal matrix with diagonal elements $c''(\eta_i)$. I_q is the identity matrix of dimension q. This makes $\dfrac{\partial^2}{\partial s^2}$ a $q \times q$ matrix.

Next up is $\dfrac{\partial^2}{\partial \beta^2}$, which is a $p \times p$ matrix.

$$\frac{\partial^2}{\partial \beta^2}\left[l(\eta) - (1/2)s's\right] = \frac{\partial}{\partial \beta}\left[X'(Y - c'(\eta))\right] \tag{39}$$

$$= \frac{\partial}{\partial \beta}\left[X'Y - X'(c'(\eta))\right] \tag{40}$$

$$= \frac{\partial}{\partial \beta}\left[-X'(c'(\eta))\right] \tag{41}$$

$$= -X\left[{}'\frac{\partial}{\partial \beta}c'(\eta)\right] \tag{42}$$

$$= -X'\left[\frac{\partial c'(\eta)}{\partial \eta}\frac{\partial \eta}{\partial \beta}\right] \tag{43}$$

$$= -X'\left[c''(\eta)\right]X \tag{44}$$

Next is the $p \times q$ mixed partial $\dfrac{\partial^2}{\partial \beta\, \partial s}$.

$$\frac{\partial^2}{\partial \beta\, \partial s}\left[l(\eta) - (1/2)s's\right] = \frac{\partial}{\partial \beta}\left\{\left[Y - c'(\eta)\right]' ZA\right\} \tag{45}$$

$$= \frac{\partial}{\partial \beta}\left\{Y'ZA - \left[c'(\eta)\right]' ZA\right\} \tag{46}$$

$$= -\frac{\partial}{\partial \beta}\left\{\left[c'(\eta)\right]' ZA\right\} \tag{47}$$

$$= -\left[\frac{\partial c'(\eta)}{\partial \beta}\right]' ZA \tag{48}$$

$$= -\left[\frac{\partial c'(\eta)}{\partial \eta}\frac{\partial \eta}{\partial \beta}\right]' ZA \tag{49}$$

$$= -\left[c''(\eta)X\right]' ZA \tag{50}$$

$$= -X'\left[c''(\eta)\right]ZA \tag{51}$$

And last we have the $q \times p$ mixed partial $\dfrac{\partial^2}{\partial \beta\, \partial s} = -AZ'[c''(\eta)]X$. These four pieces specify the hessian matrix for the penalized likelihood. Trust can now perform the inner optimization to find $\tilde{\beta}$ and $\tilde{s}$. Trust will maximize the penalized likelihood.

The **outer** optimization is done using `optim` with the default method of "`Nelder-Mead.`" This requires just the function value and no derivatives. This optimization method was chosen because the optimization function already contains second derivatives of the cumulant function; requir-

ing derivatives of the optimization function would in turn require higher-order derivatives of the cumulant.

The default of `optim` is to minimize, but we'd like to do maximization. Reversing the sign of the optimization function will turn the maximization into minimization.

If $\tilde{\beta}$ and $\tilde{s}$ are available from previous calls to the inner optimization function, then they are used here. Otherwise, the values are taken to be 0 and 1. The outer optimization's function is evaluated by first defining

$$\tilde{\eta} = X\tilde{\beta} + ZA\tilde{s} \tag{52}$$

$$l(\tilde{\eta}) = Y'\tilde{\eta} - c(\tilde{\eta}) \tag{53}$$

$$A = \sum_k E_k \sigma_k. \tag{54}$$

Let W be a diagonal matrix with elements $c''(\eta_i)$ on the diagonal. Then the quantity we'd like to maximize is the penalized quasi-likelihood:

$$l(\tilde{\eta}) - \frac{1}{2}\tilde{s}'\tilde{s} - \frac{1}{2}\log|AZ'WZA + I| \tag{55}$$

$$\tag{56}$$

Again, `optim` minimizes, so we have to minimize the negative value of the penalized quasi-likelihood in order to maximize the value of it.

We do need to be careful about the determinant. Let $AZ'WZA + I = LL'$ where L is the lower triangular matrix resulting from a Cholesky decomposition. Then

$$\frac{1}{2}\log|AZ'WZA + I| = \frac{1}{2}\log|LL'| \tag{57}$$

$$= \frac{1}{2}\log(|L|)^2 \tag{58}$$

$$= \log|L| \tag{59}$$

Since L is triangular, the determinant is just the product of the diagonal elements. Let l_i be the diagonal elements of L. Then

$$\frac{1}{2}\log|AZ'WZA + I| = \log\prod l_i \tag{60}$$

$$= \sum \log l_i \tag{61}$$

If I am worried about all variance components being zero, I could implement an eigendecomposition using the R function `eigen`. This would be more numerically stable, but is slower. Let O be the matrix containing the eigenvectors and let Λ be a diagonal matrix with the eigenvalues λ_i on

the diagonal. Then

$$AZ'WZA = O\Lambda O' \tag{62}$$

Then we can rewrite the argument of the determinant as follows:

$$AZ'WZA + I = O\Lambda O' + I = O\Lambda O + OO' = O(I + \Lambda)O' \tag{63}$$

This leads to the careful calculation of our determinant as follows:

$$|AZ'WZA + I| = 1 * \prod_{i=1}^{n}(1 + \lambda_i) * 1 \tag{64}$$

$$\Rightarrow \log|AZ'WZA + I| = \sum \log(1 + \lambda_i) \tag{65}$$

The last quantity can be accurately calculated using the `log1p` function in R.

The outer optimization uses $\tilde{\beta}$ and $\tilde{s}$ provided by the inner optimization, but it does not return them. To keep track of the most recent $\tilde{s}$, so store them in an environment that I call "cache." The purpose of $\tilde{\beta}$ and $\tilde{s}$ is two-fold. First, if they are available from a previous iteration of the inner optimization, then they are used in the outer optimization of PQL. Second, after PQL is finished, $\tilde{s}$ is used to help center the generated random effects.

The point of doing PQL is to construct a decent importance sampling distribution. Thus, the estimates don't have to be perfect. It is possible that one of the σ_k will be 0 according to PQL. If this happens, then I'll just use $\sigma_k = .01$ or something like that for the importance sampling distribution.

5 Log density of the data (`el`)

This section provides details for the log density of the data and two of its derivatives.

5.1 Equations

Recall the log of the data density is

$$\log f_\theta(y|u) = Y'\eta + c(\eta) \tag{66}$$

$$= \sum_i Y_i\eta_i - c(\eta_i) \tag{67}$$

where

$$\eta = X\beta + ZU. \tag{68}$$

The derivative of this with respect to one component, η_j, is

$$\frac{\partial}{\partial \eta_j} \log f_\theta(y|u) = Y_j - c'(\eta_j). \tag{69}$$

The derivative of the component η_j with respect to one of the fixed effect predictors, β_l, is

$$\frac{\partial \eta_j}{\partial \beta_l} = X_{jl} \tag{70}$$

We'd like the derivative of the log of the data density with respect to β. This can be written using the chain rule as follows:

$$\frac{\partial}{\partial \beta_l} \log f_\theta(y|u) = \frac{\partial \eta_j}{\partial \beta_l} \frac{\partial}{\partial \eta_j} \log f_\theta(y|u) \tag{71}$$

$$= \left[Y_j - c'(\eta_j)\right] X_{jl} \tag{72}$$

The mixed partial derivative (with respect to β_{l_1} and β_{l_2}) of the log data density can be written similarly:

$$\frac{\partial^2}{\partial \beta_{l_1} \partial \beta_{l_2}} \log f_\theta(y|u) = \frac{\partial}{\partial \beta_{l_2}} \left(\left[Y_j - c'(\eta_j)\right] X_{jl_1}\right) \tag{73}$$

$$= \frac{\partial \eta_j}{\partial \beta_{l_2}} \frac{\partial}{\partial \eta_j} \left(\left[Y_j - c'(\eta_j)\right] X_{jl_1}\right) \tag{74}$$

$$= -X_{jl_1} X_{jl_2} c''(\eta_j) \tag{75}$$

Letting $c'(\eta)$ be a vector with components $c'(\eta_j)$, the first derivative of the log data density can be written in matrix form as:

$$\frac{\partial}{\partial \beta} \log f_\theta(y|u) = X' \left[Y - c'(\eta)\right]. \tag{76}$$

Letting $c(\eta)$ be a diagonal matrix with components $c''(\eta_j)$, the second derivative of the log data density can be written in matrix form as:

$$\frac{\partial^2}{\partial \beta^2} \log f_\theta(y|u) = X'[-c''(\eta)]X \tag{77}$$

5.2 Redone in C

The C function to calculate the value of the log data density and its two derivatives will be called by reference. The following pointers will be passed in: double `Y`, double `X`, int `nrowX`, int `ncolX`, double `eta`, int `family`, double `elval`, double `elgradient`, double `elhessian`. The pointers `elval`, `elgradient` and `elhessian` will be zeros before `el.C` is invoked. Invoking `el.C` will then place the

calculatedvalue of the log data density and two derivatives into `elval`, `elgradient` and `elhessian`.

The function `el.C` calls the following C functions: `cum3.C` to calculate the cumulant given a value of η, `cp3.C` to calculate the derivative of the cumulant given a value of η, textttcpp3.C to calculate the hessian of the cumulant given a value of η, and functions to perform matrix multiplication.

6 Constructing D (`getEk`)

Since D is a function of E_t, we need to find a way to calculate E_t. This depends on whether D is diagonal or not.

6.1 D diagonal

When D is diagonal, recall E_t are diagonal with either 1 or 0 on the diagonal (1 if that random effect has ν_t as a variance component and 0 otherwise). Let q_t be the number of nonzero entries in E_t and $q = \sum_{t=1}^{T} q_t$. When D is diagonal, then $\sum_{t=1}^{T} E_t = I_q$, the identity matrix with q rows. Finally, recall that $D = \sum_{t=1}^{T} \nu_t E_t$.

At this point, `mod.mcml$z` is a list with T design matrices (one for each distinct variance component). We need to go through and count the number of columns (q_t) for each design matrix t in the list. Then $q = \sum_{t=1}^{T} q_t$ is the total number of random effects in the model, and D will have that many rows (and columns). Thus, each E_t will have q rows and columns as well. Then we know that E_1 has q_1 ones on the diagonal, followed by zeros to fill the rest of the diagonal. E_2 has q_1 zeros, then q_2 ones, then zeros for the rest of the diagonal.

6.2 D in general (not necessarily diagonal)

In progress.

7 Generating random effects (genRand)

This will be hidden from the user. It will be called by the model fitting function.

To approximate the log likelihood, we need simulated random effects. PQL will give us estimates for β, σ and s. Let β^*, s^* and σ^* be the vectors of PQL estimates. Then

$$A^* = \sum_{t=1}^{T} E_t \sigma_t^* \tag{78}$$

and

$$D^* = A^* A^*. \tag{79}$$

Also, we can "unstandardize" our random effects this way:

$$u^* = A^* s^* \tag{80}$$

I'll be generating random effects u from these distributions:

1. $N(0, I)$
2. $N(u^*, D^*)$
3. $N(u^*, (Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1})$

I'll generate them in these proportions: p_1, p_2, p_3 where $p_1 + p_2 + p_3 = 1$.

The first distribution is centered at 0, just where we expect the random effects to truly be centered. The random effects will be generated with variance 1, but will be scaled in the trust objective function so that they have variance ν, where ν is given by the iteration of trust. The second distribution is centered at the PQL best guess of the random effect values and has the PQL guess of the variance. The last distribution is centered at the PQL guess u^* and has a variance based on the PQL penalized likelihood hessian. The idea of this last distribution is to generate random effects from a distribution whose first two derivatives are equal to those of the target distribution $f_\theta(u, y)$.

Let u_k be a vector with length q (the number of random effects in the model). Suppose we want to generate random effects $u_k, k = 1, ..., m$ from a generic distribution $N(\mu, \Sigma)$. We can use eigendecomposition for Σ to get orthogonal matrix O (containing the eigenvectors) and diagonal matrix Λ with diagonal entries being Σ's eigenvalues λ. Eigendecompositions take a little bit more time than Cholesky decompositions, but are more stable. Then

$$\Sigma^{1/2} = O\Lambda^{1/2}O' \tag{81}$$

Finding $\Lambda^{1/2}$ is as easy as taking the root of the diagonal entries. We know that $\Sigma^{1/2}$ is correct because

$$\begin{aligned}
\Sigma^{1/2}\Sigma^{1/2} &= O\Lambda^{1/2}O'O\Lambda^{1/2}O' && (82)\\
&= O\Lambda^{1/2}\Lambda^{1/2}O' && (83)\\
&= O\Lambda O' && (84)\\
&= \Sigma. && (85)
\end{aligned}$$

For each k, draw a sample of size q from the standard normal distribution. Call this sample $\breve{u}_k$. Then we shift and scale it:

$$u_k = \breve{u}_k \Sigma^{1/2} + \mu. \tag{86}$$

This is the process of drawing our sample from $N(\mu, \Sigma^{1/2}\Sigma^{1/2})$ for any mean vector μ and variance matrix Σ.

8 Distribution of the random effects

This will be hidden from the user. This section provides equations for evaluating the log of a normal distribution with a specified mean and variance. When necessary to emphasize and clarify the choice of mean μ and variance Σ , I use the notation $f(u|\mu, \Sigma)$ rather than $f_\theta(u)$.

This section is useful for calculating $\tilde{f}(u_k)$. This section also provides two derivatives in some cases, which is useful for evaluating $f_\theta(u_k)$ and two derivatives. These are needed for the calculation in (147).

Section 8.3 contains details that are not currently implemented, but could be. In this version, all random effects are assumed to be normally distributed and all simulated random effects are generated from a normal distribution.

8.1 Distribution of random effects is normal (`distRand`)

In this subsection, I discuss both the assumed distribution of the unobserved random effects ($N(0, D)$) and the distribution used to generate the simulated random effects. The equations in this section will provide $\tilde{f}(u)$, $\log f_\theta(u)$, $\nabla \log f_\theta(u)$, and $\nabla^2 \log f_\theta(u)$ for equation 147.

We assume the random effects are $N(\mu, \Sigma)$. For $f_\theta(u)$, $\mu = 0$. This is written generally with μ not necessarily 0 so that the function can be used for $\tilde{f}(u)$ as well. Let Σ be the variance matrix. We can then write the distribution of the random effects as:

$$\log f(u|\mu, \Sigma) = (-1/2) \log |\Sigma| - (1/2)(U - \mu)' \Sigma^{-1} (U - \mu) \tag{87}$$

For now, I assume that the true variance of the random effects is $\Sigma = D$ where D is diagonal. Formulas for this are in section 8.1.1. I generate random effects both from a normal distribution with $\Sigma = D$ diagonal (details in 8.1.1) and from a normal distribution with Σ general (details in 8.1.3).

8.1.1 $\Sigma = D$ Diagonal

These equations will be used for both $\log f_\theta(u)$ and for pieces of $\tilde{f}(u)$. This section provides the equations for finding the value of a normal density with D diagonal, the gradient vector, and the hessian matrix.

In the simplest case, $\Sigma = D$ is diagonal with variance components ν_t on the diagonal and 0 off diagonal. This is what I assume to be the true form of D for now. Remember that for every ν_t, we can construct a matrix E_t (with dimensions the same as matrix D) that has 1s on the diagonal elements corresponding to the elements of D that contain ν_t and 0s everywhere else.

We can partition the random effects according to their variance components: $U = (U_1', ..., U_T')'$. Let D_t be the variance matrix for U_t . D_t has q_t rows (and also columns). Thus D is diagonal:

$$D = \begin{bmatrix} D_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_T \end{bmatrix} \tag{88}$$

Since D is diagonal, it follows that D^{-1} is also diagonal with diagonal entries $\frac{1}{\nu_1}$,...,$\frac{1}{\nu_T}$. Also, the assumption that D is diagonal makes calculating the determinant of D easy:

$$|D| = \nu_1^{q_1} ... \nu_T^{q_T} \tag{89}$$

Taking these two pieces of information into account allows us to write the log density for U as follows:

$$\log f_\theta(u) = (-1/2) \log |D| - (1/2)(U - \mu)' D^{-1} (U - \mu) \tag{90}$$

$$= -\frac{1}{2} \left[\sum_{t=1}^{T} q_t \log \nu_t \right] - \frac{1}{2} \sum_{t=1}^{T} \left[\frac{1}{\nu_t} (U_t - \mu_t)'(U_t - \mu_t) \right] \tag{91}$$

The first and second derivatives of each summand with respect to its associated ν_t are:

$$\frac{\partial}{\partial \nu_t} \log f_\theta(u_t) = -\frac{q_t}{2\nu_t} + \frac{1}{2\nu_t^2} (U_t - \mu_t)'(U_t - \mu_t) \tag{92}$$

and

$$\frac{\partial^2}{\partial \nu_t^2} \log f_\theta(u_t) = \frac{q_t}{2\nu_t^2} - \frac{1}{\nu_t^3} (U_t - \mu_t)'(U_t - \mu_t). \tag{93}$$

Any other derivative is equal to 0. That is, for all $t_1 \neq t_2$,

$$\frac{\partial}{\partial \nu_{t_1}} \log f_\theta(u_{t_2}) = 0. \tag{94}$$

Also,

$$\frac{\partial}{\partial \beta} \log f_\theta(u_{t_2}) = 0. \tag{95}$$

Thus, if $\nu = (\nu_1, ..., \nu_T)$, the gradient of the random effects distribution is the following vector of length T:

$$\frac{\partial}{\partial \nu} \log f_\theta(u) = \left[-\frac{q_1}{2\nu_1} + \frac{1}{2\nu_1^2}(U_1 - \mu_1)'(U_1 - \mu_1) \quad ... \quad -\frac{q_T}{2\nu_T} + \frac{1}{2\nu_T^2}(U_T - \mu_T)'(U_T - \mu_T) \right] \tag{96}$$

To calculate this vector, I will create a function to take q_t, U_t, and ν_t and output $\frac{-q_t}{2\nu_t} + \frac{1}{2\nu_t^2} U_t' U_t$. I can then do a loop for t in 1 through T to calculate each entry in the vector.

The Hessian matrix is the following diagonal matrix:

$$\frac{\partial^2}{\partial \nu^2} \log f_\theta(u) = \begin{bmatrix} \frac{q_1}{2\nu_1^2} - \frac{1}{\nu_1^3} U_1' U_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{q_T}{2\nu_T^2} - \frac{1}{\nu_T^3 U_T' U_T} \end{bmatrix} \tag{97}$$

I will create another function to take q_t, U_t, μ_t and ν_t and spit out $\frac{q_t}{2\nu_t^2} - \frac{1}{\nu_t^3}(U_t - \mu_t)'(U_t - \mu_t)$. Again, during the above loop of length T, I will calculate the entries for the diagonal of this Hessian matrix.

The inputs of the function are the vector ν of variance components, U (the vector of random effects), and the list `z` (from `mod.mcml`). The list `z` has T matrices, each with the number of columns equal to q_t. We need $q_t, t = 1, \ldots, T$ to calculate the log density and its derivatives.

The last thing we need to discuss is how to split U into $U_1, ..., U_T$ and μ into $\mu_1, ..., \mu_T$. The same process should work for both, so let's just look at how we'd do it for U. We know that the first q_1 items of U are U_1, the next q_2 items are U_2, etc. In other words, entries 1 through q_1 are U_1. Items $q_1 + 1$ through $q_1 + q_2$ are U_2, etc. In R pseudo code, this means

```
nrand<-lapply(z,ncol) #get the number of columns from each matrix in list z
unlist(nrand) #chance the list into a vector, since each entry in the list is just a scalar

U1<- U[1 through nrand[1]] #first rule is use 1:nrand[1]
Ut<-U[sum(nrand[1:t-1]+1 through sum(nrand[1:t])]
```

Note that we don't need to actually calculate either A or D. We work only through ν.

8.1.2 Newer version of $\Sigma = D$ Diagonal

The newer version of `distRand` (called `distRand3`) uses the same equations as those listed in 8.1.1. The only difference is that the newer version is more efficient. Rather than having each call to `distRand3` recalculate which elements of U belong to each t, this is done ahead of time in the objective function. This information is contained in vector `meow`. This is then an argument in `distRand3`.

Rewritten in C, this function will take as pointers: the double array `nu` that contains the variance components, the int `T` to specify the length of `nu`, the double array `mu` that contains the means of the random effects, the int array `nrandom` of length `T` that contains the number random effects from that variance component, the double array `Uvec` that contains one vector of generated random effects, the int array `meow` that specifies how to split `U` up based on the variance components, the double array `drgradient` that will contain the resulting gradient, and the double array `drhessian` that will contain the resulting hessian.

The result of invoking the C function will be the calculation of the gradient and hessian for the distribution of the random effects. The value will be evaluated by the C function in 8.1.3.

8.1.3 Σ in general (`distRandGeneral`)

For now, this is only used for $\tilde{f}(u)$. (When Σ is general, calculating the derivatives of $\log f_\theta(u)$ is not easy.) Recall the general form for a normal distribution with mean μ and variance Σ is:

$$\log f(u \,|\, \mu, \Sigma) = (1/2) \log |\Sigma^{-1}| - (1/2)(U - \mu)' \Sigma^{-1} (U - \mu) \tag{98}$$

The only part of this to discuss is $|\Sigma^{-1}|$. We can use eigendecomposition to make $\Sigma^{-1} = O \Lambda O'$ where O is orthogonal and Λ is the diagonal matrix with eigenvalues. Since orthogonal matrices have determinant ± 1, then $|O||O'| = 1$. Thus

$$|\Sigma^{-1}| = |O'| \, |\Lambda| \, |O| \tag{99}$$

$$= |O'| \, |O| \, |\Lambda| \tag{100}$$

$$= |\Lambda|, \tag{101}$$

which is just the product of the eigenvalues. The log of the determinant is calculated beforehand to save time, since this function is called $3m$ times each optimization iteration, where m is again the Monte Carlo sample size.

This function is also rewritten in C with the following passed in as pointers: double `Sigma.inv` Σ^{-1}, double `logdet` $\log |\Sigma^{-1}|$, int `nrow`, double `uvec` a vector of random effects, double mu μ, and double `distRandGenVal`.

8.2 $\tilde{f}(u_k)$

I need to evaluate $\log \tilde{f}(u_k)$ in the objective function of trust. I can make a function using the general form for a normal distribution's density as described in the previous subsection, then use that function for the three distributions that I sampled from. Since I generated the random effects from the three distributions in proportions p_1, p_2, p_3, I can write

$$\log \tilde{f}(u_k) = p_1 \log f(u_k \,|\, 0, D^*) + p_2 \log f(u_k \,|\, u^*, D^*) + p_3 \log f(u_k \,|\, u^*, (Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1}) \tag{102}$$

8.3 Distribution of random effects is multivariate t

If we want, we can draw random effects from a t distribution with location parameter u^* from PQL and scale matrix D^* (using the variance components estimated by PQL). These can be drawn using the package mvtnorm. This package can also give the density evaluated at certain points. I'm not sure if I'll use this or just stick to normals.

8.4 Central Limit Theorem for MCLA

Define

$$\gamma_1 = \int f_\theta(u, y) du \tag{103}$$

$$\gamma_2 = \int \tilde{f}(u) du. \tag{104}$$

Recall the calculation for the MCLA gradient in (3). We can rewrite it as

$$\nabla l_m(\theta) = \frac{\frac{1}{m}\sum_{k=1}^m \left(\nabla \log f_\theta(u_k, y) \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)}{\frac{1}{m}\sum_{k=1}^m \left(\frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \right)} \tag{105}$$

to more obviously show that both the numerator and denominator are sample means. The law of large numbers says that the numerator and denominator each converge to their true means. That

is,

$$\frac{1}{m}\sum_{k=1}^{m}\nabla \log f_\theta(u_k,y)\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)} \to E_{\tilde{f}}\left[\nabla \log f_\theta(u,y)\frac{f_\theta(u,y)}{\tilde{f}(u)}\right] \tag{106}$$

$$= \int \nabla \log f_\theta(u,y)\frac{f_\theta(u,y)}{\tilde{f}(u)}\frac{\tilde{f}(u)}{\gamma_2}du \tag{107}$$

$$= \frac{\gamma_1}{\gamma_2}\int \nabla \log f_\theta(u,y)\frac{f_\theta(u,y)}{\gamma_1}du \tag{108}$$

$$= \frac{\gamma_1}{\gamma_2}E_f\left[\nabla \log f_\theta(u,y)\right] \tag{109}$$

and

$$\frac{1}{m}\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)} \to E_{\tilde{f}}\left[\frac{f_\theta(u,y)}{\tilde{f}(u)}\right] \tag{110}$$

$$= \int \frac{f_\theta(u,y)}{\tilde{f}(u)}\frac{\tilde{f}(u)}{\gamma_2}du \tag{111}$$

$$= \frac{\gamma_1}{\gamma_2}\int \frac{f_\theta(u,y)}{\gamma_2}du \tag{112}$$

$$= \frac{\gamma_1}{\gamma_2} \tag{113}$$

Then, by Slutsky's,

$$\nabla l_m(\theta) \to \frac{\frac{\gamma_1}{\gamma_2}E_f\left[\nabla \log f_\theta(u,y)\right]}{\frac{\gamma_1}{\gamma_2}} \tag{114}$$

$$= E_f\left[\nabla \log f_\theta(u,y)\right] \tag{115}$$

In addition to a law of large numbers, we would like a Central Limit Theorem for $\nabla l_m(\theta)$. In other words, the quantity

$$\sqrt{m}\left[\frac{\frac{1}{m}\sum_{k=1}^{m}\left(\nabla \log f_\theta(u_k,y)\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right)}{\frac{1}{m}\sum_{k=1}^{m}\left(\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right)} - E_f\left[\nabla \log f_\theta(u,y)\right]\right] \tag{116}$$

$$= \frac{\frac{1}{\sqrt{m}}\sum_{k=1}^{m}\left(\nabla \log f_\theta(u_k,y)\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right)}{\frac{1}{m}\sum_{k=1}^{m}\left(\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right)} - \sqrt{m}E_f\left[\nabla \log f_\theta(u,y)\right] \tag{117}$$

$$= \frac{\frac{1}{\sqrt{m}}\sum_{k=1}^{m}\left(\nabla \log f_\theta(u_k,y)\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right) - E_f\left[\nabla \log f_\theta(u,y)\right]\frac{1}{\sqrt{m}}\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}}{\frac{1}{m}\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}} \tag{118}$$

will have a normal distribution if and only if the variances of

$$\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \tag{119}$$

and

$$\sum_{k=1}^{m} \nabla \log f_\theta(u_k, y) \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} \tag{120}$$

are each finite. First, we want to show that

$$\sum_{k=1}^{m} \frac{f_\theta(u_k)}{\tilde{f}(u_k)} \tag{121}$$

has finite variance. This is true if and only if

$$\int \frac{f_\theta(u)^2}{\tilde{f}(u)} du < \infty. \tag{122}$$

We see:

$$\int \frac{f_\theta(u)^2}{\tilde{f}(u)} du = \int \frac{N(u|0,D)^2}{p_1 N(u|0,D) + p_2 N(u|u^*, D^*) + p_3 N(u|u^*, ((Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1})} du \tag{123}$$

$$\leq \int \frac{N(u|0,D)^2}{p_1 N(u|0,D)} du \tag{124}$$

$$= \frac{1}{p_1} \int N(u|0,D) du \tag{125}$$

$$= \frac{1}{p_1}. \tag{126}$$

Therefore $\sum_{k=1}^{m} f_\theta(u_k)$ has finite variance as long as $p_1 \neq 0$.

Second, we want to show that the gradient of the MCLA has finite variance. That is, we want to show that

$$\int \frac{N(u|0,D)^2 \left[\nabla \left(\log f_\theta(y|u) + \log f_\theta(u)\right)\right]^2}{p_1 N(u|0,D) + p_2 N(u|u^*, D^*) + p_3 N(u|u^*, ((Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1})} du < \infty. \tag{127}$$

That is, we want to show

$$\int \frac{N(u|0,D)^2 \left[\nabla \left(\log f_\theta(y|u) + \log f_\theta(u)\right)\right]^2}{p_1 N(u|0,D) + p_2 N(u|u^*,D^*) + p_3 N(u|u^*,((Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1})} du \tag{128}$$

$$\leq \int \frac{N(u|0,D)^2 \left[\nabla \left(\log f_\theta(y|u) + \log f_\theta(u)\right)\right]^2}{p_1 N(u|0,D)} du \tag{129}$$

$$\leq \int \frac{N(u|0,D)^2 \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u) \ \frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2}{p_1 N(u|0,D)} du \tag{130}$$

$$\leq \frac{1}{p_1} \int N(u|0,D) \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u) \ \frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2 du \tag{131}$$

$$< \infty. \tag{132}$$

By the Cauchy-Schwartz inequality,

$$\int N(u|0,D) \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u) \ \frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2 du \tag{133}$$

$$\leq \left[\int N(u|0,D) \left[\frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2 du\right]^{1/2} \left[\int N(u|0,D) \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u)\right]^2 du\right]^{1/2}. \tag{134}$$

Therefore, I need to show that each of these two integrals is finite. I will start with the first of the two integrals:

$$\int N(u|0,D) \left[\frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2 du. \tag{135}$$

Recall the notation from section 8.1.1. If

$$\int N(u_t|0,D_t) \left[\frac{\partial}{\partial \nu_t} \log f_\theta(u)\right]^2 du_t < \infty \tag{136}$$

for every $t = 1, ..., T$, then

$$\int N(u|0,D) \left[\frac{\partial}{\partial \nu} \log f_\theta(u)\right]^2 du < \infty \tag{137}$$

by Cauchy Schwartz. Therefore, I will prove the integral is finite for the general case of u_t and ν_t. First, we recognize that our integral is actually an expectation of a function of a normal random variable:

$$\int N(u_t|0,D_t) \left[\frac{\partial}{\partial \nu_t} \log f_\theta(u)\right]^2 du_t = \int \frac{1}{\nu_t^{q_t/2}} e^{-u_t' u_t/(2\nu_t)} \left[-\frac{q_t}{2\nu_t} + \frac{u_t' u_t}{2\nu_t^2}\right]^2 du_t. \tag{138}$$

Normal distributions have finite moments of all orders, so we know that

$$\int N(u_t|0,D_t) \left[\frac{\partial}{\partial \nu_t} \log f_\theta(u)\right]^2 du_t < \infty. \tag{139}$$

Next, we want to show

$$\int N(u|0,D)\left[\frac{\partial}{\partial\beta}\log f_\theta(y|u)\right]^2 du < \infty. \tag{140}$$

Recall that $\eta = X\beta + Zu$. Letting $\frac{\partial}{\partial\eta}c(\eta) = \mu$, we see

$$\int N(u|0,D)\left[\frac{\partial}{\partial\beta}\log f_\theta(y|u)\right]^2 du = \int N(u|0,D)\left[(Y'-\mu')XX'(Y-\mu)\right] du \tag{141}$$

We will consider this integral for the cases of $Y|u$ being a Bernoulli and a Poisson random variable. When $Y|u$ is Bernoulli, then $0 \leq \mu \leq 1$. Therefore,

$$\int N(u|0,D)\left[(Y'-\mu')XX'(Y-\mu)\right] du \tag{142}$$

is finite. When $Y|u$ is a Poisson random variable, then for any η,

$$\frac{\partial}{\partial\eta}c(\eta) = e^\eta. \tag{143}$$

Then

$$\int N(u|0,D)\left[(Y'-\mu')XX'(Y-\mu)\right] du \propto \int N(u|0,D)e^\eta e^\eta du \tag{144}$$

$$\propto \int N(u|0,D)e^{2Zu} du \tag{145}$$

is finite.

Therefore, it is proven that

$$\int \frac{N(u|0,D)^2\left[\nabla\left(\log f_\theta(y|u) + \log f_\theta(u)\right)\right]^2}{p_1 N(u|0,D) + p_2 N(u|u^*,D^*) + p_3 N(u|u^*,((Z'c''(X\beta^* + Zu^*)Z + (D^*)^{-1})^{-1})} du \tag{146}$$

exists and is finite. Therefore, since the gradient of the MCLA has finite variance, we can conclude the gradient of the MCLA has a Central Limit Theorem.

9 Objective function: approximated log likelihood

This will be hidden from the user. It will be called by the model fitting function.

In order to maximize the approximated log likelihood using trust, I will need an objective function that returns the value, the gradient vector, and the Hessian matrix of the log likelihood. This objective function will get $\log f_\theta(u_k)$, $c(\eta_i)$, each of their gradient vectors, and each of their Hessian

matrices to plug into these expressions. Denote

$$b_k = \log f_\theta(u_k, y) - \log \tilde{f}(u_k) \tag{147}$$
$$= \log f_\theta(u_k) + \log f_\theta(y|u_k) - \log \tilde{f}(u_k) \tag{148}$$

For computational stability, we'll set $a = max(b_k)$. Then the value is expressed as:

$$\log f_{\theta,m} = a + \log \left[\frac{1}{m} \sum_{k=1}^{m} e^{b_k - a} \right] \tag{149}$$

Define the weights as:

$$v_\theta(u_k, y) = \frac{e^{b_k - a}}{\sum_{k=1}^{m} e^{b_k - a}} \tag{150}$$

The gradient vector is:

$$G = \sum_{k=1}^{m} \nabla \log f_\theta(u_k, y) v_\theta(u_k, y) \tag{151}$$
$$= \sum_{k=1}^{m} \nabla \left[\log f_\theta(y|u_k) + \log f_\theta(u_k) \right] v_\theta(u_k, y) \tag{152}$$
$$= \sum_{k=1}^{m} \left[\nabla \log f_\theta(y|u_k) + \nabla \log f_\theta(u_k) \right] v_\theta(u_k, y) \tag{153}$$
$$= \sum_{k=1}^{m} \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u_k) \quad \frac{\partial}{\partial \nu} \log f_\theta(y|u_k) \right] v_\theta(u_k, y) + \left[\frac{\partial}{\partial \beta} \log f_\theta(u_k) \quad \frac{\partial}{\partial \nu} \log f_\theta(u_k) \right] v_\theta(u_k, y) \tag{154}$$
$$= \sum_{k=1}^{m} \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u_k) \quad 0 \right] v_\theta(u_k, y) + \left[0 \quad \frac{\partial}{\partial \nu} \log f_\theta(u_k) \right] v_\theta(u_k, y) \tag{155}$$
$$= \sum_{k=1}^{m} \left[\frac{\partial}{\partial \beta} \log f_\theta(y|u_k) \quad \frac{\partial}{\partial \nu} \log f_\theta(u_k) \right] v_\theta(u_k, y) \tag{156}$$

Then the Hessian matrix is:

$$H = \sum_{k=1}^{m} \nabla^2 \log f_\theta(u_k, y) v_\theta(u_k, y) + \sum_{k=1}^{m} \left[\nabla \log f_\theta(u_k, y) \right] \left[\nabla \log f_\theta(u_k, y) \right]^T v_\theta(u_k, y) - GG^T \tag{157}$$

Everything in this has already been defined except:

$$\nabla^2 \log f_\theta(u_k, y) = \begin{bmatrix} \frac{\partial^2}{\partial \beta^2} \log f_\theta(u_k, y) & \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(u_k, y) \\ \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(u_k, y) & \frac{\partial^2}{\partial \nu^2} \log f_\theta(u_k, y) \end{bmatrix} \tag{158}$$

$$= \begin{bmatrix} \frac{\partial^2}{\partial \beta^2} \log f_\theta(u_k) & \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(u_k) \\ \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(u_k) & \frac{\partial^2}{\partial \nu^2} \log f_\theta(u_k) \end{bmatrix} + \begin{bmatrix} \frac{\partial^2}{\partial \beta^2} \log f_\theta(y|u_k) & \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(y|u_k) \\ \frac{\partial^2}{\partial \beta \, \partial \nu} \log f_\theta(y|u_k) & \frac{\partial^2}{\partial \nu^2} \log f_\theta(y|u_k) \end{bmatrix} \tag{159}$$

$$= \begin{bmatrix} 0 & 0 \\ 0 & \frac{\partial^2}{\partial \nu^2} \log f_\theta(u_k) \end{bmatrix} + \begin{bmatrix} \frac{\partial^2}{\partial \beta^2} \log f_\theta(y|u_k) & 0 \\ 0 & 0 \end{bmatrix} \tag{160}$$

$$= \begin{bmatrix} \frac{\partial^2}{\partial \beta^2} \log f_\theta(y|u_k) & 0 \\ 0 & \frac{\partial^2}{\partial \nu^2} \log f_\theta(u_k) \end{bmatrix} \tag{161}$$

10 Summary of model

Typing `summary(mod)` will give more detailed info of the model. This will be broken into two pieces (as is usually done for summaries): there will be the summary.mcml and print.summary.mcml. We split this up because we might have a lot of extra stuff in the summary that we don't necessarily want printed each time. So, when a user types `summary(mod)`, only the basic information is automatically printed. More information can be found in the summary list.

The summary will do all the calculations and its value will be a list of the following:

- call
- the variance estimate(s)
- a matrix with the predictor in the first column, the estimated coefficient in the second column, the standard error in the third column, the z value in the fourth column, and the two-sided pvalue in the fifth column. All inference is asymptotic, so we use the standard normal distribution to calculate the p-values.
- the evaluated Monte Carlo log likelihood along with its first and second derivative
- maybe other things that I haven't thought of

A note on the standard errors: to calculate the standard errors, we take the MCLA Hessian matrix, invert it, take the diagonal elements, and square root them. It's possible that the Hessian

will be noninvertible if it is close enough to singular that the computer thinks it's singular. Then the standard errors will all be infinite. We also need to warn the user why this is happening.

Then print.summary.glmm will print these things:

- call
- the variance estimate(s)
- a matrix similar to the output of summary.glm, which will be put through printCoefmat in order to get the significance stars that we're used to. We'll have the predictor in the first column, the estimated coefficient in the second column, the standard error in the third column, the z value in the fourth column, and the two-sided pvalue in the fifth column, the significance stars, and the significance legend.
- maybe other things that I haven't thought of. I'll probably realize more once I get coding.

I think for the printCoefmat to work, the fixed effects matrix needs to look like

```
            Estimate Std. Error z value     Pr(>|z|)
(Intercept)        2       0.50       4 6.334248e-05
x1                 2       0.25       8 1.244192e-15
```

(or at least it worked when I had these as the column names and didn't work before that).

Note that the summary and print.summary functions will be S3 generic functions. What this means is the user will type "summary(mod)" and summary is a generic function. R checks the class of the model *mod* and then automatically uses the summary and print.summary functions for that class of objects. This also might affect the way that I export the function and document it in the manual. I think that I'll be ok if I copy parts of his summary.aster and print.summary.aster. I need to read about generics more and will check out the hints library.

11 Checks

11.1 Checking the MCLA finite differences

To check the function that calculates the MCLA $l_m(\theta)$, I use finite differences on the Booth and Hobert (1999) example. To do this, I chose a value of $\theta = (\beta, \sigma)$ and a relatively small value of δ, where δ is a vector of length 2. We can check that the value and first derivative of the MCLA function are calculated correctly by making sure the following approximation holds

$$\nabla l_m(\theta) \cdot \delta \approx l_m(\theta + \delta) - l_m(\theta). \tag{162}$$

Then, we can check the first and second derivatives of the MCLA are calculated correctly by checking for the following approximation:

$$\nabla^2 l_m(\theta)\delta \approx \nabla l_m(\theta+\delta) - \nabla l_m(\theta) \tag{163}$$

11.2 Checking functions using the Booth and Hobert example

I also test the objective function by taking the Booth and Hobert (1999) example and rewriting the functions for this specific example. I then compare the values produced by the check functions and the original functions. Functions checked this way are:

1. log of the data density (`el`)
2. log of the density for the random effects (both `distRand` and `distRandGeneral`)
3. the Monte Carlo likelihood approximation (`objfun`)

11.3 Checking a putative MLE using MCMC

Suppose $\hat{\theta}$ is claimed to be the MLE. For example, $\hat{\theta}$ could be the MCMLE. Ideally, we would like to check whether the true likelihood $l(\theta)$ achieves a local max at $\hat{\theta}$. Due to the intractible integral in the likelihood expression, the best we can do is make sure that $\nabla l_m(\theta)$ evaluated at $\hat{\theta}$ is very close to 0.

Suppose $u_k, k = 1, \ldots, m$ are sampled from importance sampling distribution $\tilde{f}(u_k)$. Recall that the gradient of the MCLA is:

$$\nabla l_m(\theta) = \frac{\sum_{k=1}^m \frac{\nabla f_\theta(y|u_k) f_\theta(u_k)}{\tilde{f}(u_k)}}{\sum_{k=1}^m \frac{f_\theta(u_k)}{\tilde{f}(u_k)}} \tag{164}$$

We can choose $\tilde{f}(u_k) = f_{\hat{\theta}}(u_k, y)$. If the putative MLE is truly the MLE, then $\theta = \hat{theta}$, which in turn implies that $\tilde{f}(u_k) = f_\theta(u_k, y)$. To be clear, in this check we no longer use a mixture of normals for $\tilde{f}(u_k)$. With the selected importance sampling distribution, each of the importance sampling weights is equal to 1 and the sum of the weights is m. Therefore, the gradient of the

MCLA simplifies as follows:

$$\nabla l_m(\theta) = \frac{1}{m} \sum_{k=1}^{m} \frac{\nabla f_\theta(y|u_k) f_\theta(u_k)}{\tilde{f}(u_k, y)} \tag{165}$$

$$= \frac{1}{m} \sum_{k=1}^{m} \frac{\nabla f_\theta(u_k, y)}{\tilde{f}(u_k, y)} \tag{166}$$

$$= \frac{1}{m} \sum_{k=1}^{m} \nabla \log f_\theta(u_k, y) \tag{167}$$

This shows that the gradient of the MCLA is the average of the gradient of the complete data log likelihood, as long as $\hat{\theta}$ is truly an MLE. We can produce $u_k, k = 1, \ldots, m$, using Markov chain Monte Carlo. Using u as the variable, w run the Markov chain (perhaps `metrop` from the R package `mcmc`) on the complete data log likelihood. We then use these samples to calculate the gradient of the complete data log likelihood, which in turn calculates the gradient of the MCLA.

If we split the MCMC runs into batches, we can calculate the batch means of the MCLA gradient, the grand mean of the MCLA gradient, and the corresponding Monte Carlo standard error. If the putative MLE truly maximizes the likelihood, then the MCLA gradient's components should be close to 0. We can check that they are close enough to 0 by comparing them to the Monte Carlo standard error.

12 Likelihood ratio test

This is another S3 generic function that the user can choose to implement if they'd like to do likelihood ratio tests for nested models. Eventually I'll want this to handle an arbitrary number of models, but for now we'll stick to comparing two models: $mod2$ nested in $mod1$. I will assume these models have already been fit.

There are a few ways that the two models could differ. In other words, we could be testing:

1. whether one or more fixed effects are zero.
2. whether one variance component is zero.
3. whether multiple variance components are zero.
4. whether one or more fixed effects is zero and one or more variance components are zero.

The hypothesis testing procedure and method for calculating pvalues are different for each of these cases, so I'll go through each one in the following subsubsections.

12.1 Testing whether one or more fixed effects are zero

Consider the case of testing whether one or more fixed effects are zero and the random effects are the same between the two models. I think we just do a likelihood ratio test as we're familiar with for linear models. Let the number of fixed effect parameters in the larger model be p_1 and the number in the nested model be p_2. I will need the log likelihood values of the models outputted (from the main function at the same time as when I output the MCMLEs and the Fisher information). The log likelihood is simply the "value" from the objective function that trust uses. Let's call the log likelihood from the larger model l_1 and the log likelihood from the nested model l_2.

Then the likelihood ratio test statistic is

$$t_{LRT} = -2l_1 + 2l_2 \tag{168}$$

and this follows a χ^2 distribution with $p_1 - p_2$ degrees of freedom. The calculation

$$P(t_{\chi^2_{p_1-p_2}} > LRT). \tag{169}$$

gives us a two-sided pvalue to fit with the two-sided alternative hypothesis.

12.2 Testing whether one variance component is zero

Hypothesis testing for one variance component is easy but not what most people expect. In this setup, the two models have the exact same fixed effects. The only difference is the larger model has one more random effect ν_{t_1}. This means we want to test whether $\nu_{t_1} = 0$. A variance component must be nonnegative, meaning the alternative hypothesis is that $\nu_{t_1} > 0$. In other words, the hypotheses are:

$$H_0 : \nu_{t_1} = 0 \tag{170}$$

$$H_1 : \nu_{t_1} > 0. \tag{171}$$

Note the alternative hypothesis is one-sided. This means we need to calculate a one-sided pvalue. Let $t_{LRT} = 2l_2 - 2l_1$. If we naively follow the pvalue calculation of

$$P(\chi^2_1 > t_{LRT}) \tag{172}$$

we will end up calculating a two-sided pvalue. To fix this, we cut this pvalue in half. In other words, use the standard normal distribution Z and think of calculating the one-sided pvalue this way:

$$P(Z > \sqrt{|t_{LRT}|}) \tag{173}$$

However, this is not obvious; I never would have thought about having a test statistic of

$$\sqrt{2|l_2 - l_1|}. \tag{174}$$

12.3 Testing whether multiple variance components are zero

In this case, the fixed effects are identical between the larger and nested model. The only difference is that the larger model has more than one additional variance components. This gets a lot more complicated (for example, there is no clear way to count parameters so calculating the degrees of freedom is out the window). Luckily, someone's worked out the details already. Charlie suggested a few papers to look at. I'll read about and add this a bit later. Charlie is hoping that we'll have a mixture of χ^2 distributions, such as

$$\frac{1}{2}P(\chi^2_1 > t_{LRT}) + \frac{1}{4}P(\chi^2_2 > t_{LRT}). \tag{175}$$

The reason this is complicated is because the variance components are restricted to be nonnegative. This means that the likelihood is only defined for nonnegative variance components. Then differentiating the likelihood at 0 becomes tricky because that's the boundary.

12.4 Testing whether one or more fixed effects is zero and one or more variance components are zero

This seems very complicated. I don't know if it's any more complicated than the previous case of testing multiple variance components. I'll have to think about it later when I have time.

12.5 Determining the hypothesis test

To follow convention, this command will be called "anova" and the two arguments will be the two models to be compared. The command would look like

```
anova(mod1,mod2)
```

R will first need to figure out if the fixed effects are different and, if they are, which model is the larger model.

```
if(length(coef(mod1))>length(coef(mod2)))
      {bigmod<-mod1;  smallmod <-mod2}
if(length(coef(mod1))<length(coef(mod2)))
      {bigmod<-mod2; smallmod <-mod1}
```

Next, if there is a difference in the fixed effects, I'm going to make sure the fixed effects of the small model are nested in those of the big model. Otherwise, produce an error. Then, continue with the testing. Note: this check for nesting isn't perfect, but no other anova checks which model is bigger and whether they're nested.

```
if(big mod is defined) {
      pnames1<-names(coef(bigmod))
      pnames2<-names(coef(smallmod))
      if(sum(pnames2 %in% pnames1) != length(pnames2)) {
            stop("The models you provided are not nested.")}

     if(the variance components differ between the two models){
             check big model has more variance components (ow: error)
             check variance components of the small model are nested (ow: error)
             calculate pvalue according to section 3.3.4. return it.
      }
      if(the variance components do not differ between the two models){
            calculate pvalue according to 3.3.1. return it.
      }
}
```

If and only if we've gotten to the end of this chunk of code without returning anything, bigmod has not been defined, meaning the fixed effects are same. Therefore, we now need to figure out whether the variance components differ by one or by more than one. In order to compare the number of random effects, we need to figure out how to get the number of variance components from each model. I don't yet have a solid grasp of how formula works its magic, but I have the impression that I can count the number of model matrices returned in order to figure out the number of variance components because there is a model matrix for each one.

```
T1<- number of variance components for mod1
T2<- number of variance components for mod2
if(T1>T2){call mod1 the bigmod and mod2 the smallmod}
if(T1<T2){call mod2 the bigmod and mod1 the smallmod}
if(bigmod is defined){
     check  small model is nested in the big model (ow: error)
     (maybe do this by looking at the call?)
     if(T1==T2+1 || T2==T1+1)
           {calculate and return pvalue according to sec 3.3.2}
     calculate and return pvalue according to section 3.3.3
     }
if still haven't returned anything, produce error.
```

Why the last error? If we have gotten to this point in the code with nothing returned, it ends up that the number of variance components in each model are equal, meaning the user has made some kind of mistake. Either the user provided two identical models or they provided non-nested models (e.g. they have the same number of variance components, but different components in each model). Either way, we can't help them.

This command would return something that first reminds the user what predictors are in each model, then has a table. Each model would have one row of the table. The columns would contain the model name, the log likelihood for each model, the number of parameters in the model. Then the next columns would have one entry each: the calculated difference between the log likelihoods, the calculated difference between the number of parameters, and the pvalue of the test.

13 Confidence intervals

The user can implement this command to create confidence intervals after fitting a model *mod* using the main function. This is an S3 generic.The command would look like

```
confint(mod,parm,level=.95)
```

The only required argument would be the fitted model (the first argument). The third argument (the confidence level) has a default of .95.

If the second argument is omitted, confidence intervals would be created for all of the parameters. There are two options to calculate confidence intervals for a subset of the parameters. The user can provide either the names of the coefficients or a vector with length equal to the number of parameters (entries being 1 if they would like that intervals for that parameter and 0 otherwise). The code to do this is taken from "confint.lm" and is:

```
function (object, parm, level = 0.95, ...)
{
    cf <- coef(object)
    pnames <- names(cf)
    if (missing(parm))
        parm <- pnames
    else if (is.numeric(parm))
        parm <- pnames[parm]
        stopifnot(parm %in% pnames)

        rest of the code to actually do the confidence intervals goes here
```

```
}
```

What this says: write down the coefficient names of the object we're given. If parm is missing, we're going to assume they want to create intervals for all the parameters. If parm is numeric (a vector of 0s and 1s), use that to select the parameter names we want and call that selection parm. Finally, stop the whole process if parm (the parameter names we want intervals for) don't appear in the model.

The form for a confidence interval is the point estimate plus or minus the standard error of the point estimate times some cutoff. The point estimate and the standard error are from the summary of the model. The reference distribution used for the cutoff is the standard normal. This will produce an asymptotic confidence interval.

The output would either be a vector of length 2 (if creating confidence interval for only one parameter) or a matrix with 2 columns (one for the lower bound of the confidence interval, one for the upper bond). The column names will be "$(100 - level)/2$" and "$50 + level/2$." (in the default case, this is"2.5%" and "97.5%").

The only potential hangup is when creating confidence intervals for the variance components $\nu_t, t = 1, ..., T$. We know that $\nu_t > 0$ for all $t = 1, ..., T$. We'll find this boundary again makes things complicated. I don't yet know how to deal with this.

To illustrate the problem, consider the scenario that the margin of error for ν_t is greater than the estimate of ν_t itself. It wouldn't make sense to produce a confidence interval with a negative lower bound. It could make sense to truncate the interval so that the lower bound starts at 0 and the upper bound remains untouched.

I thought for three or four minutes about a one sided confidence interval, but I don't think that captures what we want. We want the range of all likely values for the variance component, not just an upper bound.

References

Booth, J. G. and Hobert, J. P. (1999). Maximizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm. *Journal of the Royal Statistical Society, Series B*, 61:265–285.

Geyer, C. J. (2014). R package `aster` (aster models), version .8-30. *http://cran.r-project.org/package=aster*.

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.