# Simple Logit-Normal GLMM Examples

Yun Ju Sung Charles J. Geyer

January 2, 2006

## 1 Logit-Normal GLMM

In a Logit-Normal generalized linear mixed model (GLMM), the observed data is a vector $y$ whose components are conditionally independent Bernoulli random variables given the missing data vector $b$, which is unconditionally jointly mean-zero multivariate normal. The model specification is completed by the specification of the *linear predictor*

$$\eta = X\beta + Zb \tag{1}$$

and the link function. In (1) $X$ and $Z$ are known matrices (the "design" or "model" matrices for fixed and random effects, respectively), $\beta$ is a vector of unknown parameters ("fixed effects"), $b$ is the vector of missing data ("random effects"), and the conditional expectation of $y$ given $\eta$ is $\text{logit}^{-1}(\eta)$, where this notation indicates coordinatewise application of the inverse logit function.

The unknown parameters to be estimated are $\beta$ and any unknown parameters determining the variance matrix of $b$. Usually this variance matrix has simple structure and involves only a few unknown parameters. We have written an R package `bernor` that implements the methods of Sung and Geyer (submitted) for a class of Logit-Normal GLMM. The class of models our package handles is more easily described in R than in mathematical notation. The linear predictor has the form

```
eta = X %*% beta + Z %*% (sigma[i] * b)
```
(2)

where `X` and `Z` are the matrices $X$ and $Z$ in (1) and `X %*% beta` is the matrix multiplication $X\beta$ so the only way in which (1) differs from (2) other than notationally is that $b$ in (1) is replaced by `(sigma[i] * b)` in (2), which, for readers not familiar with R, has the following interpretation: `sigma` is a vector of unknown parameters, `i` is a vector of the same length as `b` and having values that are possible indices for `sigma`, so `sigma[i]` is the vector $(\sigma_{i_1}, \ldots, \sigma_{i_m})$ in ordinary mathematical notation and `*` in (2) denotes coordinatewise multiplication, so if $z_{jk}$ are the components of the matrix $Z$ the second term on the right hand side of (2) has $j$-th component

$$\sum_{k=1}^{m} z_{jk}\sigma_{i_k}b_k$$

written in conventional mathematical notation.

We also change assumptions; in (1) $b$ is general multivariate normal, but in (2) `b` is standard multivariate normal (mean vector zero, variance matrix the identity). Thus the only unknown parameters in our model are the vectors `beta` and `sigma`. Thus our package only deals with the simple situation in which the random effects are (unconditionally) independent. Of course, the components of the linear predictor are not unconditionally independent because of the multiplication by the model matrix.

We also allow for independent and identically distributed (IID) data, in which case the data $y$ is a matrix with IID columns, each column of $y$ modeled as described above.

# 2 McCulloch's Toy Data

In this document we analyze only a simple toy model taken from McCulloch (1997) and also used by Booth and Hobert (1999) in which the log likelihood can be calculated exactly by numerical integration.

These data have the form

$$y_{ij} = \beta x_i + \sigma b_j$$

where $x_i = i/d$, where $d$ is the number of rows of $y$. A simulated data set of this form was given by Booth and Hobert (1999, Table 2). This is the data set `booth` in our `bernor` package (note that our $y$ is the transpose of their $y$ to agree with our convention that columns of $y$ are independent).

## 2.1 Monte Carlo Maximum Likelihood

Our package provides no optimization capabilities, only evaluation of the log likelihood, its derivatives, and related quantities. Use either `nlm` or `optim` for optimization. Here we demonstrate `optim`.

First we attach the data and an optimization package

```
> library(bernor)
> library(trust)
> data(booth)
> attach(booth)
```

Then we create a function that calculates the objective function (the Monte Carlo log likelihood approximation) and its gradient and hessian.

```
> moo <- model("gaussian", length(i), 1)
> nparm <- length(theta0)
> nfix <- length(mu0)
> objfun <- function(theta) {
+     if (!is.numeric(theta))
+         stop("objfun: theta not numeric")
```

```
+     if (length(theta) != nparm)
+         stop("objfun: theta wrong length")
+     beta <- theta[seq(1, nfix)]
+     sigma <- theta[-seq(1, nfix)]
+     .Random.seed <<- .save.Random.seed
+     bnlogl(y, beta, sigma, nmiss, x, z, i, moo, deriv = 2)
+ }
```

Our functions always use the same random seed (the seed is always restored to `.save.Random.seed` just before any function evaluation). This follows the principle of "common random numbers" and assures that the function we are evaluating remains the same throughout the optimization. (We can later try different random seeds if we chose.)

Then we are ready to try it out.

```
> set.seed(42)
> .save.Random.seed <- .Random.seed
> nmiss <- 100
> theta.start <- theta0
> out <- trust(objfun, theta.start, 0.1, 1, minimize = FALSE)
> print(out)

$value
[1] -44.4799

$gradient
[1] 4.118927e-14 2.248202e-14

$hessian
           [,1]       [,2]
[1,] -0.6950719  0.3792456
[2,]  0.3792456 -1.4395068

$argument
[1] 5.899843 1.244416

$converged
[1] TRUE

$iterations
[1] 7
```

The result does not agree closely with the exact maximum likelihood estimate (MLE), which is

```
> print(theta.hat.exact)

[1] 6.132472 1.329156
```

from the `booth` dataset (which we attached above) and agrees with the exact MLE $(6.132, 1.766)$ reported by Booth and Hobert (1999, p. 278) when one takes into consideration that that their second parameter is $\sigma^2$ and ours is $\sigma$.

It is hard to know what lessons one is supposed to draw from a toy problem. In real life we would not, in general, have an exact MLE for comparison. We would have for guidance Monte Carlo standard errors (Section 2.2 below), but rather than calculate them for such a small Monte Carlo sample size `nmiss`, let us increase `nmiss` and redo

```
> nmiss <- 10000
> theta.start <- out$argument
> out <- trust(objfun, theta.start, 0.1, 1, minimize = FALSE)
> print(out)

$value
[1] -44.04912

$gradient
[1] 5.994566e-12 6.786294e-12

$hessian
           [,1]       [,2]
[1,] -0.7659793  0.9184935
[2,]  0.9184935 -4.0043360

$argument
[1] 6.149948 1.308710

$converged
[1] TRUE

$iterations
[1] 4
```

And we are now much closer

```
> theta.hat <- out$argument
> theta.hat - theta.hat.exact

[1]  0.01747619 -0.02044520
```

## 2.2 Monte Carlo Standard Errors

Standard errors for our method involve the matrices $J$, $V$, and $W$ that are estimated as follows.

```
> .Random.seed <<- .save.Random.seed
> out <- bnlogl(y, theta.hat[1], theta.hat[2], nmiss,
```

```
+     x, z, i, moo, deriv = 3)
> print(out)

$value
[1] -44.04912

$gradient
[1] 5.994566e-12 6.786294e-12

$hessian
           [,1]       [,2]
[1,] -0.7659793  0.9184935
[2,]  0.9184935 -4.0043360

$bigv
            [,1]        [,2]
[1,]  0.04293642 -0.01618872
[2,] -0.01618872  0.17062789

> wout <- bnbigw(y, theta.hat[1], theta.hat[2], nmiss,
+     x, z, i, moo)
> print(wout)

            [,1]        [,2]
[1,]  0.03901216 -0.09747693
[2,] -0.09747693  0.32719879

> nobs <- ncol(y)
> bigJ <- (-out$hessian/nobs)
> eigen(bigJ, symmetric = TRUE, only.values = TRUE)$values

[1] 0.42467078 0.05236075

> bigV <- out$bigv
> bigW <- wout
> bigS <- solve(bigJ) %*% (bigV/nobs + bigW/nmiss) %*%
+     solve(bigJ)
> print(bigS)

          [,1]      [,2]
[1,] 1.4430804 0.4341115
[2,] 0.4341115 0.2298396
```

If we write a function to draw ellipses,

```
> doellipse <- function(m, v, Rsq = qchisq(0.95, 2),
+     npoint = 250, plot = TRUE, add = FALSE, ...) {
+     if (!is.numeric(m))
```

```
+         stop("m not numeric")
+     if (!is.numeric(v))
+         stop("v not numeric")
+     if (!is.matrix(v))
+         stop("v not matrix")
+     if (length(m) != 2)
+         stop("m not 2-vector")
+     if (any(dim(v) != 2))
+         stop("v not 2x2-matrix")
+     phi <- seq(0, 2 * pi, length = npoint)
+     foo <- rbind(cos(phi), sin(phi))
+     rsq <- Rsq/diag(t(foo) %*% solve(v) %*% foo)
+     bar1 <- sqrt(rsq) * foo[1, ] + m[1]
+     bar2 <- sqrt(rsq) * foo[2, ] + m[2]
+     if (plot) {
+         if (!add)
+             plot(bar1, bar2, type = "l", ...)
+         else lines(bar1, bar2, ...)
+     }
+     return(invisible(list(x = bar1, y = bar2)))
+ }
```

we can use it to produce confidence regions. Figure 1 shows a nominal 95% confidence ellipse.

```
> bigS.hat.exact <- solve(info.hat.exact) %*% (info.hat.exact/nobs +
+     bigw.hat.exact/nmiss) %*% solve(info.hat.exact)
> bigS0 <- solve(info0) %*% (info0/nobs + bigw0/nmiss) %*%
+     solve(info0)
> fred1 <- doellipse(theta.hat, bigS, plot = FALSE)
> fred2 <- doellipse(theta.hat.exact, bigS.hat.exact,
+     plot = FALSE)
> fred3 <- doellipse(theta0, bigS0, plot = FALSE)
> xlim <- range(fred1$x, fred2$x, fred3$x)
> ylim <- range(fred1$y, fred2$y, fred3$y)
> doellipse(theta.hat, bigS, xlab = expression(beta),
+     ylab = expression(sigma), xlim = xlim, ylim = ylim)
> points(theta.hat[1], theta.hat[2], pch = 19)
> doellipse(theta.hat.exact, bigS.hat.exact, add = TRUE,
+     lty = 2)
> points(theta.hat.exact[1], theta.hat.exact[2], pch = 21)
> doellipse(theta0, bigS0, add = TRUE, lty = 3)
> points(theta0[1], theta0[2], pch = 22)
```

Note that a nominal 95% confidence ellipse is very large. The "simulation truth" parameter value reported by Booth and Hobert (1999, p. 275) is $(5, \sqrt{0.5})$. It is found in the `booth` dataset. So the simulation truth is in a nominal 95%

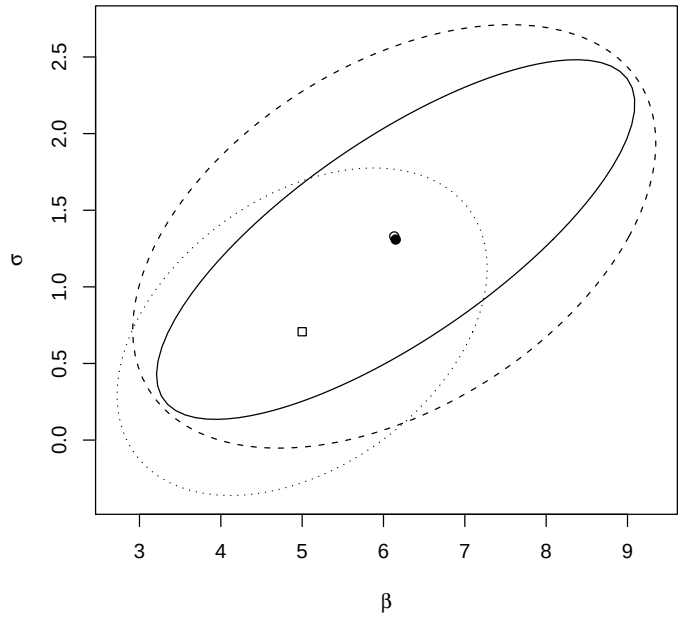


Figure 1: Nominal 95% confidence ellipses for our analysis of the Booth and Hobert data using `nmiss` $= 10^4$. Solid dot and solid line, point estimate and confidence ellipse based on our Monte Carlo point estimate and plug-in estimators of $J$, $V$, and $W$. Hollow dot and dashed line, exact MLE and confidence ellipse based on expected Fisher information and exact $W$ evaluated at the MLE (assumes $V = J^{-1}$). Square and dotted line, simulation "truth" parameter value and confidence ellipse based on on expected Fisher information and exact $W$ evaluated at the simulation truth.

confidence ellipse based on the assumption that `nobs` and `nmiss` are both "large" (which `nmiss` is and `nobs` isn't).

We compare our estimated "big $J$," "big $V$," and "big $W$" matrices with their theoretical counterparts. Both "big $J$" and "big $V$" estimate expected Fisher information (since this model is correctly specified). The exact Fisher information at the exact MLE parameter value is found in the `booth` data as `info.hat.exact`

```
> info.hat.exact

            [,1]       [,2]
[1,]  0.07194203 -0.0739439
[2,] -0.07394389  0.3900270

> bigJ

            [,1]        [,2]
[1,]  0.07659793 -0.09184935
[2,] -0.09184935  0.40043360

> bigV

            [,1]        [,2]
[1,]  0.04293642 -0.01618872
[2,] -0.01618872  0.17062789
```

The exact "big $W$" is found in the `booth` data as `bigw.hat.exact`

```
> bigw.hat.exact

            [,1]        [,2]
[1,]  0.02310320 -0.05240906
[2,] -0.05240906  0.20402222

> bigW

            [,1]        [,2]
[1,]  0.03901216 -0.09747693
[2,] -0.09747693  0.32719879
```

Our estimates are not close, but then `nobs` $= 10$ is hardly "large" so this is no surprise. Another indication that `nobs` is quite small is that the nominal 95% confidence ellipse shown in Figure 1 is so large that we have zero significant figure accuracy, so, pretending for a moment that this is not just toy data, our estimates are scientifically worthless.

# 3 A Simulation Study

We would like to have some idea how well our method works, but the analysis above gives not a hint because the toy data is "scientifically worthless" and `nobs` is far to small to apply asymptotics.

Hence we do a simulation study with the same model but larger `nobs`. Let us try

```
> nobs <- 50
```

We want the two contributions to the error `info0 / nobs + bigw0 / nmiss` to be roughly the same size so we can see both sampling and Monte Carlo variability. Thus we should set

```
> foo <- eigen(info0, symmetric = TRUE, only.values = TRUE)$values
> bar <- eigen(bigw0, symmetric = TRUE, only.values = TRUE)$values
> nmiss <- bar/(foo/nobs)
> print(nmiss)

[1] 12.28846  6.93307
```

Looks like we want about `nmiss` = 10, but that is far too small. Let us try

```
> nobs <- 500
> nmiss <- 100

> nboot <- 100
> nparm <- length(theta0)
> theta.star <- array(NA, c(nboot, nparm))
> for (iboot in 1:nboot) {
+     y <- matrix(NA, nrow(y), nobs)
+     for (k in 1:nobs) {
+         b <- rnorm(length(i))
+         eta <- x %*% mu0 + z %*% (sigma0[i] * b)
+         p <- 1/(1 + exp(-eta))
+         y[, k] <- as.numeric(runif(length(p)) < p)
+     }
+     .save.Random.seed <- .Random.seed
+     nout <- trust(objfun, theta.start, 0.1, 1, minimize = FALSE)
+     if (!nout$converged)
+         stop("convergence failure")
+     theta.star[iboot, ] <- nout$argument
+ }
```

Figure 2 gives the scatter plot of Monte Carlo MLE with these sample sizes (`nobs` = 500 and `nmiss` = 100). The solid ellipse in the figure is an asymptotic 95% coverage ellipse using the theoretical expected Fisher information and "big W" (`info0` and `bigw0`). The dashed ellipse is what we would have if we had very large Monte Carlo sample size `nmiss`, leaving `nobs` the same.

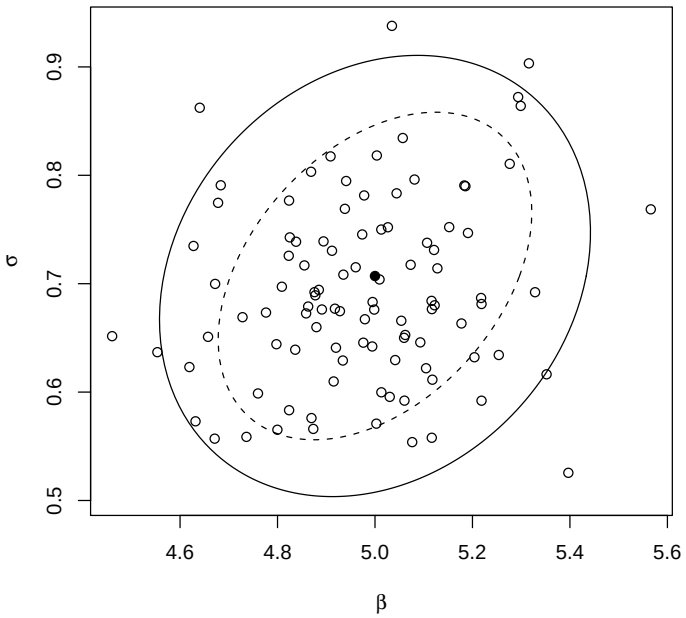


Figure 2: Simulated MLE with asymptotic 95% coverage ellipse (solid curve). The solid dot is the "simulation truth" parameter value (see text). Hollow dots are the Monte Carlo MLE's for `nboot` = 100 simulated data sets. The observed and missing data sample sizes are `nobs` = 500 and `nmiss` = 100. The dashed curve is what the 95% coverage ellipse would be if `nmiss` were infinity.

```
> bigS0 <- solve(info0) %*% (info0/nobs + bigw0/nmiss) %*%
+     solve(info0)
> bigS0part <- solve(nobs * info0)
> foo <- doellipse(theta0, bigS0, plot = FALSE)
> plot(theta.star[, 1], theta.star[, 2], xlab = expression(beta),
+     ylab = expression(sigma), xlim = range(theta.star[,
+         1], foo$x), ylim = range(theta.star[, 2],
+         foo$y))
> doellipse(theta0, bigS0, add = TRUE)
> doellipse(theta0, bigS0part, add = TRUE, lty = 2)
> points(theta0[1], theta0[2], pch = 19)
```

As can be seen, the asymptotics appear to work well at these sample sizes. However, as the dashed curve shows, even if we use a Monte Carlo sample size `nmiss` so large that the Monte Carlo error is negligible, the (non–Monte Carlo)

sampling variability of the estimator is still large, even at `nobs` = 500. The estimator of the fixed effect $\beta$ is fairly precise (about one and a half significant figure accuracy), but the estimator of the random effect scale parameter $\sigma$ is sloppy with zero significant figure accuracy. This analysis casts some doubt on the scientific usefulness of GLMM. It appears that very large sample sizes are necessary for scientifically useful inference.

# 4 Wind-Up

## 4.1 Machine and Timing Info

```
> foo <- proc.time()[1]
> fooh <- floor(foo/60^2)
> foo <- foo - fooh * 60^2
> foom <- floor(foo/60)
> foo <- foo - foom * 60
> cat("total elapsed time:", fooh, "hours,", foom,
+     "minutes, and", foo, "seconds\n")

total elapsed time: 0 hours, 13 minutes, and 14.06 seconds

> foo <- try(system("hostname -f", intern = TRUE))
> if (!inherits(foo, "try-error")) cat("machine:",
+     foo, "\n")

machine: oak.stat.umn.edu
```

## 4.2 Save Data

```
> save(.save.Random.seed, theta.hat, theta.star, bigJ,
+     bigV, bigW, doellipse, file = "bah.RData")
```

# 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 (Statistical Methodology)* 61, 265–285.

R Development Core Team (2005) R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. `http://www.R-project.org`.

Karim, M. R. and Zeger, S. L. (1992) Generalized Linear Models with Random Effects: Salamander Mating Revisited. *Biometrics*, 48, 631–644.

McCullagh, P. and Nelder, J. A. (1989) *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

McCulloch, C. E. (1997) Maximum likelihood algorithms for generalized linear mixed models. *Journal of the American Statistical Association* 92, 162–170.

Nocedal, J. and Wright, S. J. (1999) *Numerical Optimization*. New York: Springer-Verlag.

Sung, Y. J. and Geyer, C. J. (submitted). Monte Carlo likelihood inference for missing data models. `http://www.stat.umn.edu/geyer/bernor/ms.pdf`.