Logit-Normal GLMM Examples

Yun Ju Sung Charles J. Geyer

August 24, 2018

1 Examples

1.1 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), β is a vector of unknown parameters ("fixed effects"), b is the vector of missing data ("random effects"), and the conditional expectation of y given η is $\text{logit}^{-1}(\eta)$.

The unknown parameters to be estimated are β and any unknown parameters determining the variance matrix of b. Usually this variance matrix has simple structure and involves only a few unknown parameters. For this paper we have written an R package `bernor` that implements the methods of this paper 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.

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.

1.2 McCulloch's Toy Data

We start with 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).

1.2.1 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.

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

Then we create functions that calculate the objective function (the Monte Carlo log likelihood approximation) and its gradient (the gradient is optional, `optim` can use numerical differentiation instead, but supplying the gradient makes for more efficient optimization).

```
> moo <- model("gaussian", length(i), 1.0)
> 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")
+     mu <- theta[seq(1, nfix)]
+     sigma <- theta[- seq(1, nfix)]
+     .Random.seed <<- .save.Random.seed
+     bnlogl(y, mu, sigma, nmiss, x, z, i, moo)$value
+  }
> objgrd <- function (theta) {
+     if (!is.numeric(theta)) stop("objfun: theta not numeric")
+     if (length(theta) != nparm) stop("objfun: theta wrong length")
+     mu <- theta[seq(1, nfix)]
+     sigma <- theta[- seq(1, nfix)]
+     .Random.seed <<- .save.Random.seed
+     bnlogl(y, mu, sigma, nmiss, x, z, i, moo, deriv = 1)$gradient
+  }
```

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 <- 1e2
> totalelapsed <- 0
> theta.start <- theta0
> lower <- c(-Inf, 0)
> control <- list(fnscale = -10)
> tout <- system.time(
+ out <- optim(theta.start, objfun, objgrd, method = "L-BFGS-B",
+          lower = lower, control = control)
+ )
> cat("elapsed time", tout[1], "seconds\n")

elapsed time 0.054 seconds

> totalelapsed <- totalelapsed + tout[1]
> print(out)

$par
[1] 5.899792 1.244591

$value
[1] -44.4799

$counts
function gradient
       8        8
```

```
$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

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 σ^2 and ours is σ.

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 1.2.2 below), but rather than calculate them for such a small Monte Carlo sample size nmiss, let us increase nmiss and redo

```
> nmiss <- 1e4 ##### for real
> ##### nmiss <- 1e3 ##### DEBUG
>
> theta.start <- out$par
> lower <- c(-Inf, 0)
> control <- list(fnscale = out$value)
> tout <- system.time(
+ out <- optim(theta.start, objfun, objgrd, method = "L-BFGS-B",
+          lower = lower, control = control)
+ )
> cat("elapsed time", tout[1], "seconds\n")

elapsed time 3.132 seconds

> totalelapsed <- totalelapsed + tout[1]
> print(out)

$par
[1] 6.149948 1.308710

$value
[1] -44.04912

$counts
function gradient
       8        8
```

```
$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

And we are now much closer

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

```
[1]  0.01747602 -0.02044572
```

1.2.2 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
> tout <- system.time(
+ out <- bnlogl(y, theta.hat[1], theta.hat[2], nmiss, x, z, i, moo, deriv = 3)
+ )
> cat("elapsed time", tout[1], "seconds\n")
```

```
elapsed time 0.262 seconds
```

```
> totalelapsed <- totalelapsed + tout[1]
> print(out)
```

```
$value
[1] -44.04912

$gradient
[1] -3.533342e-07  1.955566e-06

$hessian
           [,1]      [,2]
[1,] -0.7659795  0.9184943
[2,]  0.9184943 -4.0043407

$bigv
            [,1]        [,2]
[1,]  0.04293644 -0.01618877
[2,] -0.01618877  0.17062809
```

```
> tout <- system.time(
+ wout <- bnbigw(y, theta.hat[1], theta.hat[2], nmiss, x, z, i, moo)
+ )
> cat("elapsed time", tout[1], "seconds\n")
```

```
elapsed time 20.804 seconds
```

```
> totalelapsed <- totalelapsed + tout[1]
> print(wout)
```

```
            [,1]        [,2]
[1,]  0.03901216 -0.09747692
[2,] -0.09747692  0.32719879
```

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

```
[1] 0.42467125 0.05236076
```

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

```
          [,1]      [,2]
[1,] 1.4430803 0.4341112
[2,] 0.4341112 0.2298393
```

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.

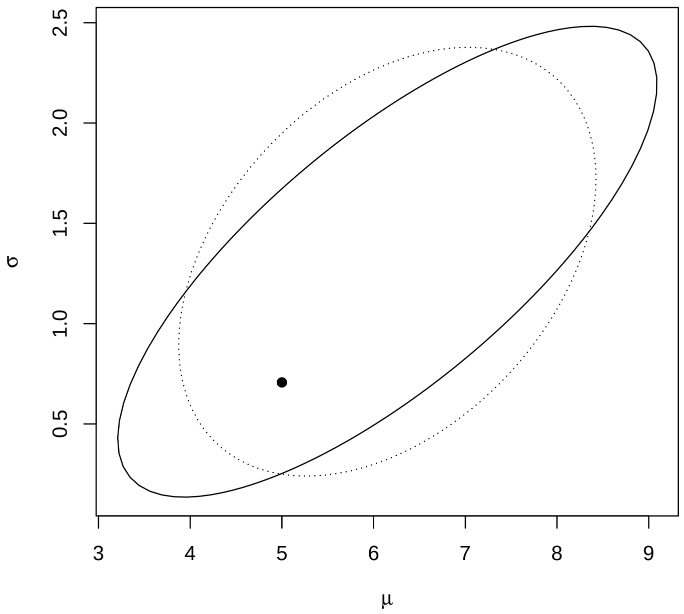

Figure 1: Nominal 95% confidence ellipse for our analysis of the Booth and Hobert data using `nmiss` $= 10^4$ (solid line). The solid dot is the "simulation truth" parameter value (see text). Dotted ellipse uses "true" Fisher information and "big W."

```
> doellipse(theta.hat, bigS, xlab = expression(mu), ylab = expression(sigma))
> points(theta0[1], theta0[2], pch = 19)
> bigS0 <- solve(info0) %*% (info0 / nobs + bigw0 / nmiss) %*% solve(info0)
> doellipse(theta.hat, bigS0, add = TRUE, lty = 3)
```

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% 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 "simulation truth" parameter value is found in the `booth` data as `info0`

```
> info0

           [,1]       [,2]
[1,]  0.1363007 -0.1126909
[2,] -0.1126909  0.6180557

> bigJ

            [,1]        [,2]
[1,]  0.07659795 -0.09184943
[2,] -0.09184943  0.40043407

> bigV

            [,1]        [,2]
[1,]  0.04293644 -0.01618877
[2,] -0.01618877  0.17062809
```

The exact “big W” is found in the `booth` data as `bigw0`

```
> bigw0

            [,1]        [,2]
[1,]  0.03301172 -0.04689475
[2,] -0.04689475  0.14047082

> bigW

            [,1]        [,2]
[1,]  0.03901216 -0.09747692
[2,] -0.09747692  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.

1.3 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
```

Now we want the two contributions to the error `info0 / nobs + bigw / 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 ##### for real
> ##### nboot <- 5 ##### DEBUG
> nparm <- length(theta0)
> theta.hat <- array(NA, c(nboot, nparm))
> tstart <- proc.time()[1]
> for (iboot in 1:nboot) {
+     ##### simulate data #####
+     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)
+     }
+     ##### calculate estimator #####
+     .save.Random.seed <- .Random.seed
+     nout <- optim(theta.start, objfun, objgrd, method = "L-BFGS-B",
+         lower = lower, control = control)
+     if (nout$convergence != 0)
+         stop("convergence failure")
+     theta.hat[iboot, ] <- nout$par
+ }
> tstop <- proc.time()[1]
> cat("elapsed time", tstop - tstart, "seconds\n")
```

```
elapsed time 176.914 seconds
```

```
> totalelapsed <- totalelapsed + (tstop - tstart)
```

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.

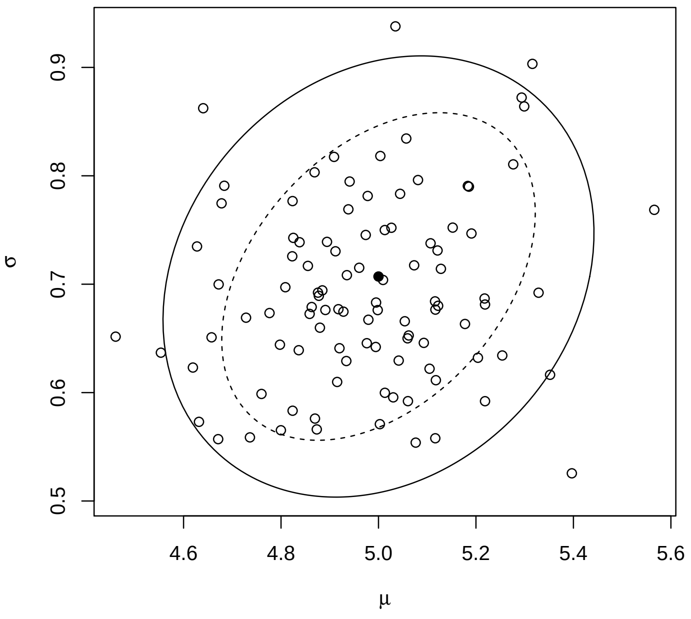

Figure 2: Simulated MLE with asymptotic 95% coverage ellipse (solid curve). The solid dot is the "simulation truth" parameter value (see text). Hollow dots are the Monte Carlo MLE's for `nboot` = 100 simulated data sets. The observed and missing data sample sizes are `nobs` = 500 and `nmiss` = 100. The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

```
> bigS0 <- solve(info0) %*% (info0 / nobs + bigw0 / nmiss) %*% solve(info0)
> bigS0part <- solve(nobs * info0)
> foo <- doellipse(theta0, bigS0, plot = FALSE)
> plot(theta.hat[ , 1], theta.hat[ , 2], xlab = expression(mu),
+     ylab = expression(sigma), xlim = range(theta.hat[ , 1], foo$x),
+     ylim = range(theta.hat[ , 2], foo$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 μ is fairly precise (about one and a half significant figure accuracy), but the estimator of the random effect scale parameter σ 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.

1.4 The Salamander Data

McCullagh and Nelder (1989, Section 14.5) discuss a “salamander mating experiment” whose data has been used by several groups of statisticians as an example of data appropriately analyzed by Logistic-Normal GLMM (see Booth and Hobert, 1999, for one analysis and citations of others), although McCullagh and Nelder (1989) did not use a GLMM and it is not clear that GLMM analyses address any of the questions of scientific interest for which the data were collected. Thus in the GLMM context these data are also “toy data” albeit not especially constructed to be such.

These data are the dataset `salam` in our `bernor` package. We are using what Booth and Hobert (1999) call “Model A” of Karim and Zeger (1992).

```
> detach(booth)
> rmexcept <- function(l, all.names = FALSE) {
+     foo <- ls(.GlobalEnv, all.names = all.names)
+     bar <- match(foo, l)
+     rm(list = foo[is.na(bar)], envir = .GlobalEnv)
+ }
> rmexcept(c("rmexcept", "objfun", "objgrd", "totalelapsed"))
> ls(all.names = TRUE)
```

```
[1] ".Random.seed"      ".save.Random.seed" "objfun"
[4] "objgrd"            "rmexcept"          "totalelapsed"
```

```
> data(salam)
> attach(salam)
> nparm <- ncol(x) + length(unique(i))
> nfix <- ncol(x)
> moo <- model("gaussian", length(i), 1.0)
> .save.Random.seed <- .Random.seed
> nobs <- ncol(y)
> nmiss <- 1e2
> theta.start <- rep(0, nparm)
> names(theta.start) <- c(dimnames(x)[[2]],
+     paste("sigma", c("f", "m"), sep = "_"))
> lower <- rep(0, nparm)
> lower[1:ncol(x)] <- (- Inf)
> trust <- 1
> lowert <- pmax(lower, theta.start - trust)
> uppert <- theta.start + trust
> control <- list(fnscale = -10)
> tout <- system.time(
+ out <- optim(theta.start, objfun, objgrd, method = "L-BFGS-B",
+          lower = lowert, upper = uppert, control = control)
+ )
> cat("elapsed time", tout[1], "seconds\n")

elapsed time 0.143 seconds

> totalelapsed <- totalelapsed + tout[1]
> print(out)

$par
       R/R        R/W        W/R        W/W    sigma_f    sigma_m
 0.8966498  0.1585898 -1.0000000  0.7685329  0.3863437  0.5483293

$value
[1] -217.4654

$counts
function gradient
      14       14

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

For this dataset we have introduced a new trick: trust regions. This is a well known procedure in optimization (Nocedal and Wright, 1999, Chapter 4), although one little known to statisticians. Although the approximation of the log likelihood provided by the `bnlogl` function is global in the sense that the function provides a result for any valid parameter values, the approximation is by no means uniformly accurate, and expecially when the sample sizes `nobs` and `nmiss` are small, can have spurious local maxima "at infinity." Thus one constrains the optimization algorithm to stay within a bounded region, called the *trust region*. Although, not the most commonly used shape, we use "box" trust regions because they are the only shape easily implemented in R. Our trust region is the box centered at `theta.start` and having L^∞ radius `trust`.

As we can see, the trust region has constrained the value of `theta.hat[4]`, the W/R fixed effect. Since, the computing time was so small, we repeat with the same trust radius but larger `nmiss`. Of course we use the current best estimate as the starting point and center the trust region there.

```
> nmiss <- 1e4 ##### for real
> ##### nmiss <- 1e3 ##### DEBUG
>
```

```
> theta.start <- out$par
> lowert <- pmax(lower, theta.start - trust)
> uppert <- theta.start + trust
> control <- list(fnscale = signif(out$value, 1))
> tout <- system.time(
+ out <- optim(theta.start, objfun, objgrd, method = "L-BFGS-B",
+          lower = lowert, upper = uppert, control = control)
+ )
> cat("elapsed time", tout[1], "seconds\n")
```

```
elapsed time 12.596 seconds
```

```
> totalelapsed <- totalelapsed + tout[1]
> print(out, digits = 4)
```

```
$par
    R/R     R/W     W/R     W/W sigma_f sigma_m 
 0.9817  0.1888 -1.9035  0.4869  0.8374  0.8618 

$value
[1] -208.5

$counts
function gradient 
      13       13 

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Now the result is unconstrained by the trust region and presumably it would be safe to dispense with it, although it does no harm and provides some safety if retained. Our results agree qualitatively but not quantatitively with those of Booth and Hobert (1999, Table 5). We fear our `nmiss` is still too small. In real life with non-toy data we would have no other analyses to compare with—Booth and Hobert (1999) compared with Karim and Zeger (1992) and perhaps others—we would have to use our estimates of J, V, and W as guides. Unfortunately, these data have little replication. The "model A" we are using does have some replication with `nobs` $= 3$, but this "replication" is questionable. The other models considered by Karim and Zeger (1992) have *no* replication (no parts of the observed data are IID). In any event, `nobs` $= 3$ is too small to get non-singular estimates of V. So we must avoid "sandwich estimators" and assume $J = V$, as occurs, with a correctly specified model.

```
> theta.hat <- out$par
> mu.hat <- theta.hat[1:nfix]
> sigma.hat <- theta.hat[- (1:nfix)]
> .Random.seed <<- .save.Random.seed
> tout <- system.time(
+     lout <- bnlogl(y, mu.hat, sigma.hat, nmiss, x, z, i, moo, deriv = 2)
+ )
> cat("elapsed time", tout[1], "seconds\n")

elapsed time 10.206 seconds

> totalelapsed <- totalelapsed + tout[1]
> print(lout, digits = 4)

$value
[1] -208.5

$gradient
[1]  0.0001628  0.0005018 -0.0002971  0.0005181 -0.0011572  0.0004140

$hessian
         [,1]    [,2]     [,3]     [,4]    [,5]    [,6]
[1,] -14.0303   1.850  -0.1283   0.4900   5.952   1.420
[2,]   1.8504 -12.122  -1.1270   2.3749   3.497  -4.329
[3,]  -0.1283  -1.127 -11.4603  -0.2394  -7.224  -5.261
[4,]   0.4900   2.375  -0.2394 -13.8738  -1.525  -1.615
[5,]   5.9523   3.497  -7.2241  -1.5247 -59.742   3.864
[6,]   1.4196  -4.329  -5.2606  -1.6150   3.864 -40.015

> tout <- system.time(
+     wout <- bnbigw(y, mu.hat, sigma.hat, nmiss, x, z, i, moo)
+ )
> cat("elapsed time", tout[1], "seconds\n")

elapsed time 61.909 seconds

> totalelapsed <- totalelapsed + tout[1]
> print(wout)

          [,1]       [,2]       [,3]        [,4]       [,5]         [,6]
[1,] 1384.0356   292.9156  256.29510 -905.426864  1370.2050  -588.288631
[2,]  292.9156   604.2178 -106.11965   89.213999   480.1464 -1237.969602
[3,]  256.2951  -106.1196  298.46574  -66.803358   665.0191   149.097686
[4,] -905.4269    89.2140  -66.80336  980.971962  -607.6077     2.235224
[5,] 1370.2050   480.1464  665.01915 -607.607740  4523.1707 -3156.327582
[6,] -588.2886 -1237.9696  149.09769    2.235224 -3156.3276  4912.381178

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

```
[1] 20.744145 13.825826  5.229588  4.627625  3.329065  2.658096

> bigW <- wout
> bigS <- solve(bigJ) %*% (bigV / nobs + bigW / nmiss) %*% solve(bigJ)
> print(bigS, digits = 4)

           [,1]      [,2]     [,3]       [,4]      [,5]      [,6]
[1,]  0.0874797  0.020222 -0.01045  0.0004722  0.013375  0.001712
[2,]  0.0202221  0.103778 -0.01148  0.0196109  0.009759 -0.013144
[3,] -0.0104511 -0.011485  0.10587 -0.0002100 -0.015018 -0.014216
[4,]  0.0004722  0.019611 -0.00021  0.0810990 -0.001589 -0.006500
[5,]  0.0133749  0.009759 -0.01502 -0.0015886  0.021965  0.002142
[6,]  0.0017124 -0.013144 -0.01422 -0.0065000  0.002142  0.031838

> foo <- eigen(bigS, symmetric = TRUE, only.values = TRUE)$values
> print(foo)

[1] 0.13551035 0.10641407 0.08240492 0.06470890 0.02594531 0.01705063

> max(foo) / min(foo)

[1] 7.947529

> foo <- rbind(theta.hat, sqrt(diag(bigS)))
> dimnames(foo) <- list(c("estimate", "std. err."), names(theta.hat))
> print(foo, digits = 4)

             R/R    R/W     W/R    W/W sigma_f sigma_m
estimate  0.9817 0.1888 -1.9035 0.4869  0.8374  0.8618
std. err. 0.2958 0.3221  0.3254 0.2848  0.1482  0.1784
```

The “standard errors” here are to be taken with a grain of salt. Neither nobs nor nmiss is large enough for the asymptotics to be believed. We produce them only because they are the best we have to offer except for a simulation study like that of the preceding section, which would be very time consuming and presumably only show that we have very little accuracy.

Increasing nmiss is easily done. It is just a matter of patience. Our code does not store all the missing data (at some cost in computer time regenerating it when needed) so that arbitrarily large nmiss can be used, if one is willing to wait for an answer.

```
> load("sally/doit.RData")
> load("sally/doit2.RData")
```

We ran, off-line because it took so long, a calculation with nmiss $= 10^7$. The results were

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

```
      R/R        R/W        W/R        W/W    sigma_f    sigma_m 
 1.0044002  0.5336607 -1.7829325  1.2675500  1.0987277  1.1668347 
```

```
> print(lout, digits = 4)
```

```
$value
[1] -207.6

$gradient
[1] -8.974e-04  7.709e-04  6.443e-05 -3.278e-04  1.892e-03 -8.830e-04

$hessian
        [,1]     [,2]    [,3]   [,4]     [,5]     [,6]
[1,] -9.3975   1.0967  0.4885 -2.155   3.2250  -0.3368
[2,]  1.0967 -11.8051 -0.5604  2.012  -0.2736   3.6332
[3,]  0.4885  -0.5604 -8.3937  1.579  -2.8468  -3.0186
[4,] -2.1553   2.0120  1.5788 -7.780   0.3730   8.8591
[5,]  3.2250  -0.2736 -2.8468  0.373 -27.2373   3.5403
[6,] -0.3368   3.6332 -3.0186  8.859   3.5403 -27.0202
```

```
> print(wout, digits = 4)
```

```
        [,1]    [,2]    [,3]    [,4]    [,5]    [,6]
[1,]  419107  -82372    3407 -394115  144641 -594245
[2,]  -82372  258374  -16787  385075 -236443  203519
[3,]    3407  -16787  114685  112801 -102996  209484
[4,] -394115  385075  112801  999341 -731241  991409
[5,]  144641 -236443 -102996 -731241 1151232 -837833
[6,] -594245  203519  209484  991409 -837833 1850253
```

```
> bigV <- bigJ <- (- lout$hessian / nobs)
> eigen(bigJ, symmetric = TRUE, only.values = TRUE)$values
```

```
[1] 10.9682334  8.9913024  4.4604707  2.8738201  2.3546466  0.8962727
```

```
> bigW <- wout
> bigS <- solve(bigJ) %*% (bigV / nobs + bigW / nmiss) %*% solve(bigJ)
> print(bigS, digits = 4)
```

```
           [,1]      [,2]       [,3]      [,4]      [,5]      [,6]
[1,]  0.1441873 -0.023665 -0.0002369 -0.114015  0.012686 -0.046520
[2,] -0.0236650  0.132901 -0.0098016  0.137257  0.004344  0.070456
[3,] -0.0002369 -0.009802  0.1338423  0.007242 -0.016461 -0.015652
[4,] -0.1140148  0.137257  0.0072424  0.461826  0.004899  0.188363
[5,]  0.0126863  0.004344 -0.0164612  0.004899  0.042218  0.008949
[6,] -0.0465202  0.070456 -0.0156520  0.188363  0.008949  0.120399
```

```
> foo <- eigen(bigS, symmetric = TRUE, only.values = TRUE)$values
> print(foo)
```

```
[1] 0.62850038 0.14370194 0.11882461 0.07508027 0.03792843 0.03133870
```

```
> max(foo) / min(foo)
```

```
[1] 20.05509
```

```
> foo <- rbind(theta.hat, sqrt(diag(bigS)))
> dimnames(foo) <- list(c("estimate", "std. err."), names(theta.hat))
> print(foo, digits = 4)
```

```
             R/R    R/W     W/R    W/W sigma_f sigma_m
estimate  1.0044 0.5337 -1.7829 1.2675  1.0987   1.167
std. err. 0.3797 0.3646  0.3658 0.6796  0.2055   0.347
```

For comparison, Booth and Hobert (1999) give the following MLE

```
> mu <- c(1.03, 0.32, -1.95, 0.99)
> sigmasq <- c(1.40, 1.25)
> theta.hat.booth <- c(mu, sqrt(sigmasq))
> names(theta.hat.booth) <- names(theta.hat)
> print(theta.hat, digits = 4)
```

```
    R/R     R/W     W/R     W/W sigma_f sigma_m
 1.0044  0.5337 -1.7829  1.2675  1.0987  1.1668
```

```
> print(theta.hat.booth, digits = 4)
```

```
    R/R     R/W     W/R     W/W sigma_f sigma_m
  1.030   0.320  -1.950   0.990   1.183   1.118
```

(We have independently verified using MCMC that the latter appear to be correct to three significant figures).

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.

Ihaka R and Gentleman R. (1996) R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5, 299–314.

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.