Stat 8931 Flu Homework Solution, Part 1

Charles J. Geyer

October 26, 2005

1 Setup

First we load the library.

```
> library(mcmc)
```

Then we load the data, copied from Table 1 in Coull and Agresti (2000).

```
> data <- read.table("flu.txt", header = TRUE)
> nobs <- data$nobs
> data$nobs <- NULL
> ypat <- as.matrix(data)
> ypat <- ypat[nobs > 0, ]
> nobs <- nobs[nobs > 0]
```

Then we specify the design matrices

```
> x <- diag(4)
> z <- rbind(c(1, 1, 1, 0, 0, 0), c(1, 1, 0, 1, 0,
+     0), c(1, 1, 0, 0, 1, 0), c(1, -1, 0, 0, 0, 1))
> idx <- c(1, 2, 3, 3, 3, 3)
```

1.1 Putative MLE

Coull and Agresti (2000) give the following MLE for this model.

```
> beta.putative <- c(-4, -4.4, -4.7, -4.5)
> sigmasq.putative <- 4.05^2
> rho1.putative <- 0.43
> rho2.putative <- (-0.25)
> delta3.putative <- sqrt(sigmasq.putative * (1 - rho1.putative))
> delta2.putative <- sqrt(sigmasq.putative * (rho1.putative -
+     rho2.putative)/2)
> delta1.putative <- sqrt(sigmasq.putative * (rho1.putative +
+     rho2.putative)/2)
> delta.putative <- c(delta1.putative, delta2.putative,
+     delta3.putative)
> beta.putative

[1] -4.0 -4.4 -4.7 -4.5

> delta.putative

[1] 1.215000 2.361536 3.057683
```

2 The Assigned Homework

Check their MLE using the `metrop` function in the MCMC package to calculate the score. More precisely if $f_\theta(b, y)$ is the complete data joint density (you have to work this out for yourself, although it is described implicitly in the first section), do the following.

1. Simulate using the `metrop` function the conditional distribution of b given y under the parameter value θ (which denotes a vector of seven parameters, four betas and three deltas).

2. If b_1, b_2, ... are the resulting Markov chain, calculate

$$\frac{1}{n} \sum_{i=1}^{n} \nabla \log f_\theta(b_i, y) \tag{1}$$

   where ∇ denotes differentiation w. r. t. θ (and hence is a seven-dimensional vector of partial derivatives).

3. Calculate MCSE (using the method of batch means or any other valid method) for each component of (1).

4. You should get zero (to within MCSE) for (1), assuming Coull and Agresti (2000) are correct, which all the calculations I have done seem to support. As statisticians we know that you can never "accept" a null hypothesis, you can only "fail to reject" it (in the words of one intro textbook I have used). So we can't be sure from the MCMC that this is the true MLE, but we can check it to arbitrary accuracy. Get your MCSE for the components of (1) to less than 0.001.

3 MCMC

3.1 Log Unnormalized Density of Equilibrium Distribution

Unnormalized log density of b given y, same as complete data log likelihood.

```
> h <- function(b) {
+     stopifnot(length(b) == ncol(z))
+     stopifnot(length(y) == ncol(ypat))
+     eta <- as.numeric(x %*% beta + z %*% (delta[idx] *
+         b))
+     p <- 1/(1 + exp(-eta))
+     q <- 1/(1 + exp(eta))
+     sum(y * log(p) + (1 - y) * log(q)) + sum(dnorm(b,
+         log = TRUE))
+ }
```

3.2 Initial Runs

```
> bstart <- rep(0, ncol(z))
> beta <- beta.putative
> delta <- delta.putative
> y <- ypat[1, ]
> set.seed(42)

> mout <- metrop(h, bstart, nbatch = 1000)
> mout$accept

[1] 0.192
```

Seems we lucked out with the acceptance rate.

Figure 1 is produced by the following code

```
> par(mfrow = c(3, 2))
> for (i in 1:6) acf(mout$batch[, i], main = paste("Series b",
+     i, sep = ""), lag.max = 250)
```

and appears on p. 4. It looks like batches of length 50 will do the job (for this y pattern).

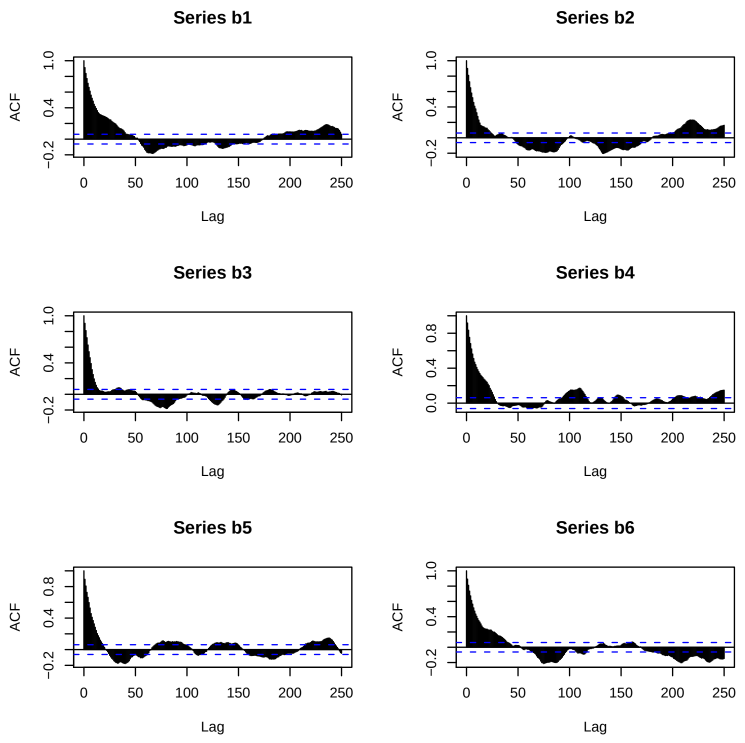

Figure 1: Autocorrelation plots of the coordinates of the conditional distribution of random effects given observed data. Data pattern all zeros.

3.3 Output Function

We need to consider the functional we wish to integrate, which is the derivative of the complete data log likelihood. If μ is the mean value parameter, then the derivative with respect to β is $(y - \mu)^T X$ and the derivative with respect to δ is

$$(y - \mu)^T Z \frac{\partial \Delta}{\partial \delta} b$$

where Δ is a diagonal matrix whose diagonal elements are deltas (the first component is δ_1, the second δ_3, and the other four δ_3).

```
> ofun <- function(b) {
+     stopifnot(length(b) == ncol(z))
+     stopifnot(length(y) == ncol(ypat))
+     eta <- as.numeric(x %*% beta + z %*% (delta[idx] *
+         b))
+     p <- 1/(1 + exp(-eta))
+     foo <- y - p
+     bar <- as.numeric(t(foo) %*% x)
+     baz <- double(length(delta))
+     for (j in 1:length(baz)) {
+         part <- as.numeric(idx == j)
+         baz[j] <- sum(foo * as.numeric(z %*% (part *
+             b)))
+     }
+     c(bar, baz)
+ }
```

The following checks that it does indeed calculate the derivative.

```
> b <- rnorm(ncol(z))
> epsilon <- 1e-08
> logl <- h(b)
> grad <- ofun(b)
> frad <- 0 * grad
> for (i in 1:length(frad)) {
+     if (i <= length(beta)) {
+         beta.save <- beta
+         beta[i] <- beta[i] + epsilon
+         frad[i] <- (h(b) - logl)/epsilon
+         beta <- beta.save
+     }
+     else {
+         j <- i - length(beta)
+         delta.save <- delta
+         delta[j] <- delta[j] + epsilon
+         frad[i] <- (h(b) - logl)/epsilon
+         delta <- delta.save
+     }
+ }
> grad

[1] -0.99942965 -0.04527425 -0.26390958 -0.29694123 -1.76308942
[6] -1.47916099 -2.59610026

> frad

[1] -0.99942952 -0.04527436 -0.26390978 -0.29694149 -1.76308923
[6] -1.47916097 -2.59610076

> abs(grad - frad)

[1] 1.314892e-07 1.093115e-07 2.075047e-07 2.601689e-07
[5] 1.856886e-07 2.333286e-08 4.939575e-07
```

We look at autocorrelations for this `outfun`.

```
> mout <- metrop(mout, outfun = ofun)
> mout$accept

[1] 0.203
```

Figure 2 is produced by the following code

```
> par(mfrow = c(4, 2), mar = c(3, 3, 0, 0) + 0.1)
> for (i in 1:7) acf(mout$batch[, i], main = "", xlab = "",
+     ylab = "", lag.max = 250)
```

and appears on p. 7.

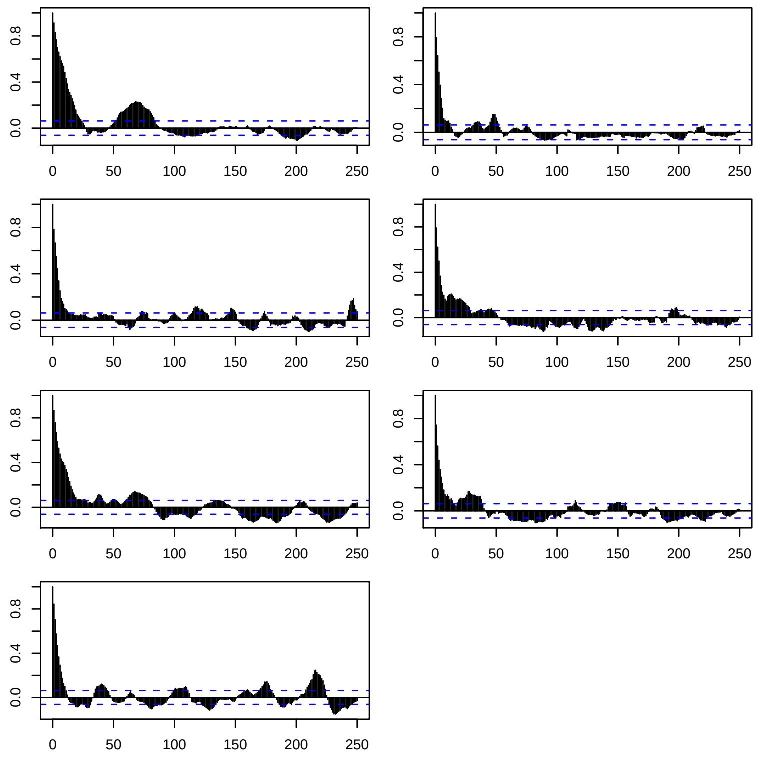

Figure 2: Autocorrelation plots of the coordinates of the complete data score when missing data has the conditional distribution of random effects given observed data. Data pattern all zeros.

3.4 Runs For All Data Patterns

3.4.1 Try I

Still looks like batches of length 100 should be safe. Let's do all data patterns with 100 batches of length 100, and 10^3 burn in.

```
> nbatch <- 100
> blen <- 100
> scale <- 1

> batches <- list()
> accepts <- list()
> for (j in 1:nrow(ypat)) {
+     y <- ypat[j, ]
+     mout <- metrop(h, bstart, nbatch = 1000, scale = scale)
+     mout <- metrop(mout, outfun = ofun, nbatch = nbatch,
+         blen = blen)
+     batches[[j]] <- mout$batch
+     accepts[[j]] <- mout$accept
+ }
> unlist(accepts)

 [1] 0.2045 0.1979 0.1602 0.1450 0.1745 0.1507 0.1550 0.1290
 [9] 0.1682 0.1499 0.1499 0.1517 0.1296 0.1535
```

We should probably adjust the scale to get somewhat higher acceptance rates. But for now, lets just go ahead with the calculation of the Monte Carlo estimate of the score and its MCSE.

```
> score <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score <- score + nobs[j] *
+     apply(batches[[j]], 2, mean)
> score.mcse <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse <- score.mcse +
+     nobs[j]^2 * apply(batches[[j]], 2, var)
> score.mcse <- sqrt(score.mcse/nbatch)

> score

[1] -0.7027893 -0.9693171  0.8567693 -1.6193280 -1.6203882
[6]  0.4341054 -3.0324250
```

```
> score.mcse

[1] 0.8134124 0.7054106 0.5780822 0.9818045 1.9818864 1.3782145
[7] 2.1924758
```

Some of the components of the score are several times the size of their (estimated) MCSE. Perhaps the batch length is too short? Now we have seven components to look at for each of 14 chains. Perhaps we should use time series methods to automate the process. We use the initial positive sequence method of Geyer (1992).

Here is a function to calculate them

```
> initpos <- function(x) {
+     foo <- acf(x, type = "covariance", plot = FALSE,
+         lag.max = 49)
+     even <- foo$lag%%2 == 0
+     bigGamma <- foo$acf[even] + foo$acf[!even]
+     bigGammaPos <- cumprod(as.numeric(bigGamma >
+         0))
+     2 * sum(bigGamma * bigGammaPos) - foo$acf[1]
+ }
```

What if we use that to calculate MCSE?

```
> score.mcse.too <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse.too <- score.mcse.too +
+     nobs[j]^2 * apply(batches[[j]], 2, initpos)
> score.mcse.too <- sqrt(score.mcse.too/nbatch)

> score.mcse

[1] 0.8134124 0.7054106 0.5780822 0.9818045 1.9818864 1.3782145
[7] 2.1924758

> score.mcse.too

[1] 0.8637926 0.8054317 0.6117598 1.0490013 2.0131987 1.2916503
[7] 1.9940297
```

3.4.2 Try II

So let's redo with smaller scale.

```
> nbatch <- 100
> blen <- 100
> scale <- 0.75
> batches <- list()
> accepts <- list()
> for (j in 1:nrow(ypat)) {
+     y <- ypat[j, ]
+     mout <- metrop(h, bstart, nbatch = 1000, scale = scale)
+     mout <- metrop(mout, outfun = ofun, nbatch = nbatch,
+         blen = blen)
+     batches[[j]] <- mout$batch
+     accepts[[j]] <- mout$accept
+ }
> unlist(accepts)
```

```
 [1] 0.3414 0.3069 0.2709 0.2535 0.2786 0.2562 0.2469 0.2200
 [9] 0.2872 0.2500 0.2613 0.2491 0.2270 0.2671
```

```
> score <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score <- score + nobs[j] *
+     apply(batches[[j]], 2, mean)
> score.mcse <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse <- score.mcse +
+     nobs[j]^2 * apply(batches[[j]], 2, var)
> score.mcse <- sqrt(score.mcse/nbatch)
> score.mcse.too <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse.too <- score.mcse.too +
+     nobs[j]^2 * apply(batches[[j]], 2, initpos)
> score.mcse.too <- sqrt(score.mcse.too/nbatch)
> score
```

```
[1]  0.56789651  0.09893464  0.41607577  0.16023674  0.53359443
[6] -1.16730496  2.92230914
```

```
> score.mcse
```

```
[1] 0.6353533 0.6850297 0.5853159 0.6971711 1.7685452 1.3287945
[7] 1.6671390
```

```
> score.mcse.too

[1] 0.6562002 0.7013442 0.5947506 0.6418533 2.5671687 1.3216236
[7] 1.4736705
```

3.4.3 Try III

So let's redo with much longer run, 100 times as long.

```
> nbatch <- 1000
> blen <- 1000
> scale <- 0.75
> batches <- list()
> accepts <- list()
> for (j in 1:nrow(ypat)) {
+     y <- ypat[j, ]
+     mout <- metrop(h, bstart, nbatch = 1000, scale = scale)
+     mout <- metrop(mout, outfun = ofun, nbatch = nbatch,
+         blen = blen)
+     batches[[j]] <- mout$batch
+     accepts[[j]] <- mout$accept
+ }
> unlist(accepts)

 [1] 0.332064 0.310623 0.275514 0.249020 0.276426 0.249975
 [7] 0.254908 0.220910 0.280193 0.252624 0.258470 0.260188
[13] 0.224729 0.267271

> score <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score <- score + nobs[j] *
+     apply(batches[[j]], 2, mean)
> score.mcse <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse <- score.mcse +
+     nobs[j]^2 * apply(batches[[j]], 2, var)
> score.mcse <- sqrt(score.mcse/nbatch)
> score.mcse.too <- double(length(beta) + length(delta))
> for (j in 1:length(batches)) score.mcse.too <- score.mcse.too +
+     nobs[j]^2 * apply(batches[[j]], 2, initpos)
> score.mcse.too <- sqrt(score.mcse.too/nbatch)
> score
```

```
[1] -0.15021176 -0.01377469  0.09862223 -0.30213127 -0.09304470
[6] -0.25053139 -0.25285920

> score.mcse

[1] 0.07314144 0.07189782 0.06549132 0.08199005 0.20369366
[6] 0.15022140 0.18777749

> score.mcse.too

[1] 0.07871421 0.07771390 0.06869062 0.08300130 0.19810998
[6] 0.15388374 0.19034998
```

4 Elapsed Time

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

total elapsed time: 1 hours, 50 minutes, and 2942 seconds

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

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

References

Coull, B. A. and Agresti, A. (2000). Random effects modeling of multiple binomial responses using the multivariate binomial logit-normal distribution. *Biometrics*, **56**, 73–80.

Geyer, C. J. (1992). Practical Markov chain Monte Carlo (with discussion). *Statistical Science*, **7**, 473–511.