MCMC Package Example

Charles J. Geyer

April 4, 2005

This is an example of using the `mcmc` package in R. The problem comes from a take-home question on a (take-home) PhD qualifying exam (School of Statistics, University of Minnesota).

Simulated data for the problem are in the file `logit.txt`. There are five variables in the data set, the response `y` and four predictors, `x1`, `x2`, `x3`, and `x4`.

A frequentist analysis for the problem is done by the following R statements

```
> foo <- read.table(url("http://www.stat.umn.edu/geyer/PhD/F03/logit.txt"),
+     header = TRUE)
> out <- glm(y ~ x1 + x2 + x3 + x4, data = foo,
+     family = binomial())
> summary(out)

Call:
glm(formula = y ~ x1 + x2 + x3 + x4, family = binomial(), data = foo)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7461  -0.6907   0.1540   0.7041   2.1943

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.6328     0.3007   2.104  0.03536 *
x1            0.7390     0.3616   2.043  0.04100 *
x2            1.1137     0.3627   3.071  0.00213 **
x3            0.4781     0.3538   1.351  0.17663
x4            0.6944     0.3989   1.741  0.08172 .
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 137.628  on 99  degrees of freedom
Residual deviance:  87.668  on 95  degrees of freedom
AIC: 97.668
```

```
Number of Fisher Scoring iterations: 6
```

But this problem isn't about that frequentist analysis, we want a Bayesian analysis. For our Bayesian analysis we assume the same data model as the frequentist, and we assume the prior distribution of the five parameters (the regression coefficients) makes them independent and identically normally distributed with mean 0 and standard deviation 2.

The log unnormalized posterior (log likelihood plus log prior) density for this model is calculated by the following R function (given the preceding data definitions)

```
> x <- foo
> x$y <- NULL
> x <- as.matrix(x)
> x <- cbind(1, x)
> dimnames(x) <- NULL
> y <- foo$y
> lupost <- function(beta, x, y) {
+     eta <- x %*% beta
+     p <- 1/(1 + exp(-eta))
+     logl <- sum(log(p[y == 1])) + sum(log(1 -
+         p[y == 0]))
+     return(logl + sum(dnorm(beta, 0, 2, log = TRUE)))
+ }
```

With those definitions in place, the following code runs the Metropolis algorithm to simulate the posterior.

```
> library(mcmc)
> set.seed(42)
> beta.init <- as.numeric(coefficients(out))
> out <- metrop(lupost, beta.init, 1000, x = x,
+     y = y)
> names(out)

 [1] "accept"       "batch"        "initial"
 [4] "final"        "initial.seed" "final.seed"
 [7] "time"         "lud"          "nbatch"
[10] "blen"         "nspac"        "scale"

> out$accept

[1] 0.008
```

The output is in the component `out$batch` returned by the `metrop` function. We'll look at it presently, but first we need to adjust the proposal to get a higher acceptance rate (`out$accept`). It is generally accepted (Gelman, Roberts, and

Gilks, 1996) that an acceptance rate of about 20% is right, although this recommendation is based on the asymptotic analysis of a toy problem (simulating a multivariate normal distribution) for which one would never use MCMC and is very unrepresentative of difficult MCMC applications.

Geyer and Thompson (1995) came to a similar conclusion, that a 20% acceptance rate is about right, in a very different situation. But they also warned that a 20% acceptance rate could be very wrong and produced an example where a 20% acceptance rate was impossible and attempting to reduce the acceptance rate below 70% would keep the sampler from ever visiting part of the state space. So the 20% magic number must be considered like other rules of thumb we teach in intro courses (like $n > 30$ means means normal approximation is valid). We know these rules of thumb can fail. There are examples in the literature where they do fail. We keep repeating them because we want something simple to tell beginners, and they are all right for some problems.

Be that as it may, we try for 20%.

```
> out <- metrop(out, scale = 0.1, x = x, y = y)
> out$accept

[1] 0.739

> out <- metrop(out, scale = 0.3, x = x, y = y)
> out$accept

[1] 0.371

> out <- metrop(out, scale = 0.5, x = x, y = y)
> out$accept

[1] 0.148

> out <- metrop(out, scale = 0.4, x = x, y = y)
> out$accept

[1] 0.209
```

O. K. That does it for the acceptance rate. So let's do a longer run and look at the results.

```
> out <- metrop(out, nbatch = 10000, x = x, y = y)
> out$accept

[1] 0.2345

> out$time

[1] 1.22 0.49 1.70 0.00 0.00
```

Figure 1 (page 4) shows the time series plot made by the R statement

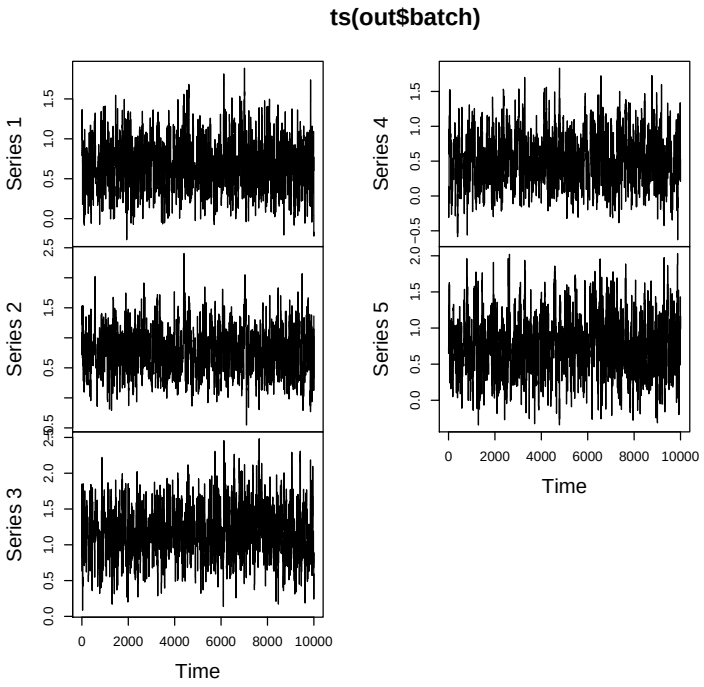

Figure 1: Time series plot of MCMC output.

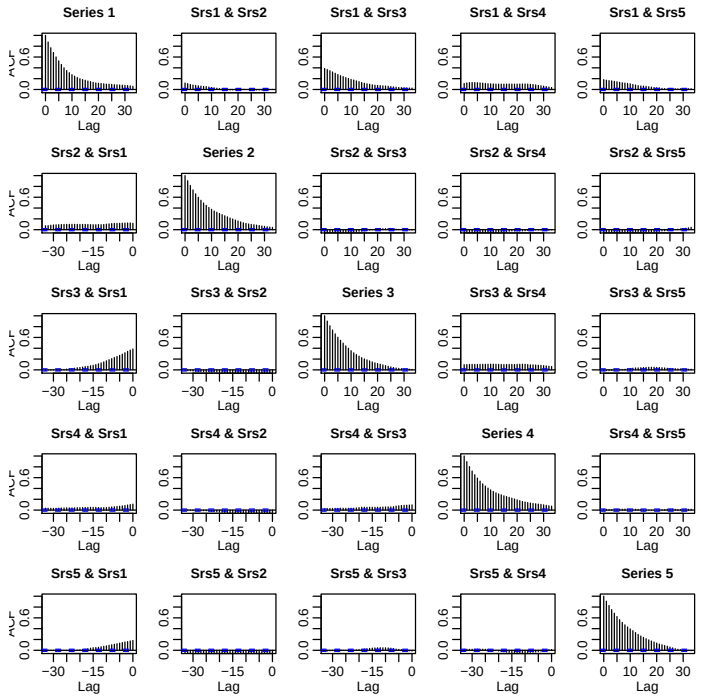

Figure 2: Autocorrelation plot of MCMC output.

```
> plot(ts(out$batch))
```

Another way to look at the output is an autocorrelation plot. Figure 2 (page 5) shows the time series plot made by the R statement

```
> acf(out$batch)
```

As with any multiplot plot, this is a bit hard to read, but all we are trying to see is that the autocorrelations are negligible after about lag 25. Thus batches of length 25 should be sufficient. But let's use 100 to be safe.

```
> out <- metrop(out, nbatch = 100, blen = 100, outfun = function(z,
+     ...) c(z, z^2), x = x, y = y)
> out$accept

[1] 0.2332

> out$time

[1] 1.25 0.48 1.73 0.00 0.00
```

We have added an argument `outfun` that gives the "functional" of the state we want to average. For this problem we are interested in both posterior mean and variance. Mean is easy, just average the variables in question. But variance is a little tricky. We need to use the identity

$$\mathrm{var}(X) = E(X^2) - E(X)^2$$

to write variance as a function of two things that can be estimated by simple averages. Hence we want to average the state itself and the squares of each component. Hence our `outfun` returns `c(z, z^2)` for an argument (the state vector) `z`.

The grand means (means of batch means) are

```
> apply(out$batch, 2, mean)

 [1] 0.6531950 0.7920342 1.1701075 0.5077331 0.7488265
 [6] 0.5145751 0.7560775 1.4973807 0.3913837 0.7244162
```

The first 5 numbers are the Monte Carlo estimates of the posterior means. The second 5 numbers are the Monte Carlo estimates of the posterior absolute second moments. We get the posterior variances by

```
> foo <- apply(out$batch, 2, mean)
> mu <- foo[1:5]
> sigmasq <- foo[6:10] - mu^2
> mu

[1] 0.6531950 0.7920342 1.1701075 0.5077331 0.7488265

> sigmasq

[1] 0.08791134 0.12875924 0.12822924 0.13359081 0.16367507
```

Monte Carlo standard errors (MCSE) are calculated from the batch means. This is simplest for the means.

```
> mu.mcse <- apply(out$batch[, 1:5], 2, sd)/sqrt(out$nbatch)
> mu.mcse

[1] 0.01224260 0.01417916 0.01793129 0.01468594 0.01582040
```

The extra factor `sqrt(out$nbatch)` arises because the batch means have variance σ^2/b where b is the batch length, which is `out$blen`, whereas the overall means `mu` have variance σ^2/n where n is the total number of iterations, which is `out$blen * out$nbatch`.

MCSE for the posterior variances is also simple but a bit tricky. First note that the two calculations

```
> apply(out$batch[, 6:10], 2, mean) - mu^2
```

```
[1] 0.08791134 0.12875924 0.12822924 0.13359081 0.16367507
```

```
> apply(sweep(out$batch[, 6:10], 2, mu^2), 2, mean)
```

```
[1] 0.08791134 0.12875924 0.12822924 0.13359081 0.16367507
```

do the same thing. The latter makes it clear that

```
> sigmasq.mcse <- apply(sweep(out$batch[, 6:10],
+     2, mu^2), 2, sd)/sqrt(out$nbatch)
> sigmasq.mcse
```

```
[1] 0.01726152 0.02482315 0.04655002 0.01798337 0.02562800
```

does the MCSE for the posterior variance.

If we are also interested in the posterior standard deviation (a natural question, although not asked on the exam problem), the delta method gives its standard error in terms of that for the variance

```
> sigma <- sqrt(sigmasq)
> sigma.mcse <- sigmasq.mcse/(2 * sigma)
> sigma
```

```
[1] 0.2964985 0.3588304 0.3580911 0.3655008 0.4045678
```

```
> sigma.mcse
```

```
[1] 0.02910896 0.03458897 0.06499746 0.02460100 0.03167331
```

So that's it. The only thing left to do is a little more precision (the exam problem directed "use a long enough run of your Markov chain sampler so that the MCSE are less than 0.01")

```
> out <- metrop(out, nbatch = 500, blen = 400, x = x,
+     y = y)
> out$accept
```

```
[1] 0.235155
```

```
> out$time
```

```
[1] 24.93  9.63 34.57  0.00  0.00
```

```
> foo <- apply(out$batch, 2, mean)
> mu <- foo[1:5]
> sigmasq <- foo[6:10] - mu^2
> mu
```

```
[1] 0.6624650 0.7941013 1.1712710 0.5066326 0.7261414
```

```
> sigmasq

[1] 0.09189246 0.13323054 0.13230811 0.12871293 0.15978638

> mu.mcse <- apply(out$batch[, 1:5], 2, sd)/sqrt(out$nbatch)
> mu.mcse

[1] 0.002960128 0.003647420 0.003787855 0.003632080
[5] 0.004273624

> sigmasq.mcse <- apply(sweep(out$batch[, 6:10],
+     2, mu^2), 2, sd)/sqrt(out$nbatch)
> sigmasq.mcse

[1] 0.004250030 0.006448611 0.009418470 0.004182393
[5] 0.006657834

> sigma <- sqrt(sigmasq)
> sigma.mcse <- sigmasq.mcse/(2 * sigma)
> sigma

[1] 0.3031377 0.3650076 0.3637418 0.3587658 0.3997329

> sigma.mcse

[1] 0.007010065 0.008833530 0.012946642 0.005828862
[5] 0.008327853
```

and some nicer output, which is presented in three tables constructed from the R variables defined above using the R `xtable` command in the `xtable` library.

First the posterior means, then the posterior variances (table on page 9),

Table 1: Posterior Means

	constant	x_1	x_2	x_3	x_4
estimate	0.6625	0.7941	1.1713	0.5066	0.7261
MCSE	0.0030	0.0036	0.0038	0.0036	0.0043

and finally the posterior standard deviations (table on page 9).

Note for the record that the all the results presented in the tables are from "one long run" where long here took only 24.93 seconds (on whatever computer it was run on).

References

Gelman, A., G. O. Roberts, and W. R. Gilks (1996). Efficient Metropolis jumping rules. In *Bayesian Statistics, 5 (Alicante, 1994)*, pp. 599–607. Oxford University Press.

Table 2: Posterior Variances

	constant	x_1	x_2	x_3	x_4
estimate	0.0919	0.1332	0.1323	0.1287	0.1598
MCSE	0.0043	0.0064	0.0094	0.0042	0.0067

Table 3: Posterior Standard Deviations

	constant	x_1	x_2	x_3	x_4
estimate	0.3031	0.3650	0.3637	0.3588	0.3997
MCSE	0.0070	0.0088	0.0129	0.0058	0.0083

Geyer, C. J. and E. A. Thompson (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90, 909–920.