13 Notes on Markov Chain Monte Carlo

Markov Chain Monte Carlo is a big, and currently very rapidly developing, subject in statistical computation. Many complex and multivariate types of random data, useful for maximizing likelihood in missing-data settings or for calculating expectations and conditional expectations in Bayesian analysis, can be simulated in this way (generally, via the **Metropolis-Hastings Algorithm** or the **Gibbs Sampler**, the two main simulation algorithms understood under the heading of MCMC). We give a brief sketch of each of these, and give a computational example of somewhat realistic complexity connected with the simulation of conditional distribution for random effects given the observed data in a *random intercept logistic regression* setting.

A good general reference for Markov Chain Monte Carlo topics is the book

Monte Carlo Statistical Methods, by C. Robert and G. Casella, Springer-Verlag 1999.

There is a good deal of background on continuous-state discrete-time Markov Chains to understand, if you want a solid theoretical understanding of the methods. In general, the idea and usefulness of the methods is that if you want to estimate expectations via averages of simulated vector random values X_t from a density $f(\cdot)$, it is enough to simulate values of a Markov Chain which asymptotically becomes stationary and has unique equilibrium distribution with density f. In general, one deals with continuous-state discrete-time chains. The main idea of the subject is given in the

Metropolis-Hastings Algorithm: fix an initial data-vector X_0, and a conditional density (or *proposal distribution* or *transition kernel* $q(x|y)$, which can be arbitrary except that it should (in both x, y arguments) have the same support as f and for good properties of the algorithm should have $f(x)/q(x|y)$ uniformly bounded in x, y. Then simulate values X_{t+1}, $t \geq 0$, according to the Markovian inductive step,

$$Y_t \sim q(\cdot \,|\, X_t) \qquad \textit{and} \qquad \xi_t \sim \textit{Unif}[0,1] \qquad \text{independent}$$

and

$$X_{t+1} = Y_t \qquad \text{if} \qquad \xi_t \leq \frac{f(Y_t)\, q(X_t \,|\, Y_t)}{f(X_t)\, q(Y_t \,|\, X_t)} \quad , \quad \text{otherwise} \quad = \; X_t$$

The key idea of this algorithm is that the distribution of X_{t+1} is a mixture of that of X_t and Y_t, so that if $\rho(x,y) = \min(1,\, f(y)q(x|y)/(f(x)q(y|x))$, then

$$f_{X_{t+1}}(x) \;=\; f_{X_t}(x) \int (1-\rho(x,y))q(y|x)dy \;+\; \int\int \rho(x,y)\, q(y|z)\, f_{X_t}(z)\, dzdy$$

$$\equiv \int K(x,z)\, f_{X_t}(z)dz$$

Thus one can verify the so-called 'detailed balance equation' $K(y,x)\, f(x) = K(x,y)\, f(y)$, from which the reversibility of the Markov Chain X_t and unique invariant measure (mutually absolutely continuous to the measure $f(x)\, dx$) follow. The consequence is that f is an essentially unique invariant density with given support.

The general idea of the **Gibbs Sampler** is apparently different but turns out to be closely related. Suppose that one wants to simulate from a complicated joint density $h(X^{(1)}, \ldots, X^{(p)})$ for which all of the conditional densities of $X^{(j)}$ given the other $(X^{(i)} :\; 1 \le i \le p,\; i \ne j)$ are not too hard to simulate from. Then it turns out that by alternately simulating successively for rotating values i the pieces $X_t^{(i)}$ conditionally from these densities, a Markov-chain equlibrium can again under general conditions guarantee the asymptotic stationarity of the simulated concatenated vectors $(X_t^{(j)},\; 1 \le j \le p)$ with the desired density as essentially unqiue invariant density. This approach is particularly fruitful for carefully designed *hierarchical models*.

13.1 Extended Example: Random-Intercept Logistic Regression

Consider the problem of ML estimation based on independent trivariate data-vectors $((R_i, W_i, n_i),\; i = 1, \ldots, m)$ for parameters $\vartheta = (a, b, \sigma^2)$ in the following model: sample sizes n_i and predictors W_i are either fixed design constants (such that their histogram settles down to some reasonably stable shape for large m) or are *iid* pairs with distribution not depending on the parameters ϑ, and

$$U_i \sim \mathcal{N}(0, \sigma^2) \quad \textit{iid} \text{ independent of } \quad \{W_j\}_{j=1}^m$$

$$R_i \sim Binom(n_i,\ \frac{e^{a+bW_i+U_i}}{1+e^{a+bW_i+U_i}}) \qquad \text{conditionally given} \quad \{(W_j,\ U_j)\}_{j=1}^m$$

Our objective is therefore to maximize over (a, b, σ^2)

$$\sum_{i=1}^m \log\Big\{ \int \frac{e^{(a+bW_i+u)R_i}}{(1+e^{a+bW_i+u})^{n_i}} \cdot \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-u^2/(2\sigma^2)}\, du \Big\} \tag{1}$$

Clearly, this kind of random-effect model is a 'missing-data' model in the same sense as the models which are studied using the EM algorithm. The model would be relatively simple to analyze if the intercept-effects U_i were observable. Without observability of the U_i, one must either numerically integrate the log-likelihood terms in (1) or simulate to estimate them. For documentation of a numerical integration approach to accurate calculation and maximization of log-likelihood for models including this one, see Technical Report #2 of Slud (2000) under

http://www.census.gov/hhes/www/saipe/tecrep.html

For an extended discussion of estimation strategies involving MCMC in problems like the one discussed here, see

McCulloch, C. (1997) Maximum likelihood algorithms for generalized linear mixed models. *Jour. Amer. Statist. Assoc.* **92**, 162-70.

A Direct Monte Carlo Method. In the ML problem posed here, the log-likelihood terms could all be directly estimated via Monte Carlo by choosing a large M, simulating an array $(V_{ik},\ 1 \le i \le m,\ 1 \le k \le M)$ of *iid* $\mathcal{N}(0,1)$ random variables, and maximizing instead

$$\sum_{i=1}^m \log\Big\{ M^{-1} \sum_{k=1}^M \frac{e^{(a+bW_i+\sigma V_{ik})R_i}}{(1+e^{a+bW_i+\sigma V_{ik}})^{n_i}} \Big\} \tag{2}$$

This Monte Carlo evaluation of integrals could work here, but would not be preferred to a numerical integration, e.g. by (adaptive) Gaussian quadratures. The maximization, e.g. by Newton-Raphson, is feasible by either method of evaluation of integrals, but quite costly computationally by this Monte Carlo method, and not especially accurate if some of the n_i are large.

EM Algorithm Method. Another approach would be to use the EM algorithm, if one could simulate $\{U_{it}\}_{t=1}^{T}$ from the conditional distribution of U_i given X_i, W_i. The E-step would then, for a fixed parameter-triple $\vartheta_1 = (a_1, b_1, \sigma_1^2)$, replace the i'th term of (1) by

$$T^{-1} \sum_{t=1}^{T} \log \left(\frac{e^{(a+bW_i+\sigma U_{it})R_i}}{(1+e^{a+bW_i+\sigma U_{it}})^{n_i}} \right) \tag{3}$$

The M-step would maximize over $\vartheta = (a, b, \sigma^2)$ the resulting (E-step estimated conditional expected) log-likelihood summation, the summation over $i = 1, \ldots, m$ of (3), to find the next parameter-iterates $\vartheta_2 = (a_2, b_2, \sigma_2^2)$.

So we continue this example by exploring how to simulate (without numerical integration) approximately stationary random sequences from the conditional distribution of U_i given (R_i, W_i). To keep the discussion simple, we restrict to the case $n_i = 1$ and strip the subscript i. (In the more complicated case of varying n_i, one would use a Metropolis-Hastings idea similar to the following one, but with g replaced by a normal distribution designed to approximate the integrand in the (i'th term of) expression (1).

We adopt an 'independent' Metropolis-Hastings Algorithm, with $q(x|y) \equiv g(x) \equiv e^{-x^2/(2\sigma)^2}/\sqrt{2\pi\sigma^2}$. The algorithm takes the following form: $\xi_t \sim Unif[0,1]$, $Y_t \sim \mathcal{N}(0, \sigma^2)$ are simulated independently of each other and of variables indexed smaller than t, and

$$X_{t+1} = Y_t \quad \text{if} \quad \xi_t \leq \frac{e^{(Y_t - X_t)R}(1+e^{a+bW+X_t})^n}{(1+e^{a+bW+Y_t})^n} \quad , \quad \text{otherwise} \quad = X_t$$

Our next task is to implement and test this algorithm in Splus. Here is a general function to do it, using input-vectors `Uv, Rv, nv, etav` (respectively corresponding to U, R, n, $\eta = a + bW$) of the same length m, and also using $m \times T$ matrices `Um, Ym` respectively of *iid* standard Uniform and Normal$(0, \sigma^2)$ deviates.

```
> MHblk
function(Uv,Rv,etav, nv, sig, nblk=1) {
  LR <- length(Rv)
  Ym <- sig*matrix(rnorm(LR*nblk), ncol=nblk)
  auxm <- matrix(runif(LR*nblk), ncol=nblk)
```

```
  Um <- array(0, dim=c(LR, nblk))
  for(ctr in 1:nblk) {
       Yv <- Ym[,ctr]
       probs <- pmin(exp(Rv*(Yv-Uv))*((1+exp(
         etav+Uv))/(1+exp(etav+Yv)))^nv,1)
       Um[,ctr] <- ifelse(auxm[,ctr] < probs,
         Yv, Uv)
       if(ctr < nblk) Uv <- Um[,ctr]
       }
  Um
  }
```

We test this function on a simple case, by finding the analytical (conditional) expectation of u_i and $\log(\exp(\eta + u_i)R/(1 + exp(\eta + u_i))^n)$ and then reproducing the values by MCMC simulation. Here we use the values $n = 5, R = 2, a = -1, b = 1, \sigma = .3, W = .65$:

```
> integrate(function(x) x*exp((-.35+x)*2)*dnorm(x,sd=.3)/
       (1+exp(-.35+x))^5, -15,15)$value/
   integrate(function(x) exp((-.35+x)*2)*dnorm(x,sd=.3)/
       (1+exp(-.35+x))^5, -15,15)$value
[1]  -0.006108636  ### conditional expected u
> integrate(function(x) log(exp((-.35+x)*2)/(1+
       exp(-.35+x))^5)*exp((-.35+x)*2)*dnorm(x,sd=.3)/
       (1+exp(-.35+x))^5, -15,15)$value/
   integrate(function(x) exp((-.35+x)*2)*dnorm(x,sd=.3)/
       (1+exp(-.35+x))^5, -15,15)$value
[1]  -3.41531   ## desired logLik expectation

## Now generate the U trajectory, starting with U=0.
> Useq <- c(MHblk(0,2,-.35,5,.3, nblk=100))
> c(mean(Useq), var(Useq))
[1] -0.01676858  0.07532460
> mean(log(exp((-.35+Useq)*2)/(1+exp(-.35+Useq))^5))
[1] -3.41071

> Useq <- c(Useq[51:100],MHblk(Useq[100],2,-.35,5,.3,
      nblk=200))  ## uses last 50 of the previously
```

```
### generated values, plus 250 new ones
> c(mean(Useq), var(Useq))
[1] -0.02240471  0.07836955
> mean(log(exp((-.35+Useq)*2)/(1+exp(-.35+Useq))^5))
[1] -3.412525

### Now 1000 more:
> Useq <- c(MHblk(Useq[250],2,-.35,5,.3,nblk=1000))
> c(mean(Useq), var(Useq))
[1] -0.01181766  0.08160613
> mean(log(exp((-.35+Useq)*2)/(1+exp(-.35+Useq))^5))
[1] -3.415085
###  The answers are very stable.
```

These calculations were all very quick (1 second or less for each simulation-block).

13.2 Example Continued: MCMC in EM

Finally ,we provide a summary and **R** log of the use of this MCMC step in an EM iteration for maximizing logLik. We begin by simulating a random-intercept logistic regression dataset of size 100 with vector `nv` of sample sizes generated as $Poisson(3)$ random variables plus 1, with vector W of $Unif[0,1]$ predictors, and parameters $a = -1$, $b = 1$, $\sigma = .3$.

```
> nv <- rpois(100,3) + 1
  etav <- -1 + runif(100)
  Rv <- rbinom(100, nv, plogis(etav+0.3*rnorm(100)))
> summary(Rv/nv)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.1500  0.3333  0.3709  0.5000  1.0000
```

We generate starting values for a, b by ordinary logistic regression (without regard to the random intercept).

```
> glm(cbind(Rv, nv-Rv) ~ I(etav+1), family=binomial)$coef
```

```
(Intercept) I(etav + 1)
  -1.089849    1.083622 ### Use these with sig=.5
> etav1 <- -1.08985 + 1.08362*(1+etav)
```

Now, our inductive EM step involves generating a block of 100 imputed U values for each data-item. Note that the imputed values fill up a 100×100 matrix.

```
> unix.time(Um <- MHblk(rep(0,100),Rv,etav1,nv,0.5, nblk=100))
### very quick !!
> summary(c(Um))
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-1.850000 -0.341900 -0.007248 -0.008859  0.318800  1.716000
```

The next step is to average over the imputed columns to obtain the function to maximize in a, b at the next M-step.

```
> .01*sum(Rv*(etav1+Um)-nv*log(1+exp(etav1+Um)))
[1]  -245.0789   ### estimated log of conditional expected
### complete-data Likelihood.
### Next an M-step maximization over a,b:
> tmpmin <- nlm(function(thet) {
     etapu <- thet[1]+thet[2]*(1+etav)+Um
     .01*sum(nv*log(1+exp(etapu))-Rv*etapu)
     }, c(-1.090, 1.084))
$minimum
[1] 245.0337
$estimate
[1] -1.155570  1.160729
$gradient
[1] -1.606080e-05 -2.791413e-06 ### OK, approx converged
### On this iteration we have improved (conditional-
###  expected complete-data) logLik to -245.034

### For comparison, calculate logLik at true values:
> Um0 <- MHblk(Um[,100],Rv,etav,nv,0.3, nblk=100)
### estimated H0 cond'l expected logLik:
```

```
>  .01*sum(Rv*(etav+Um0)-nv*log(1+exp(etav+Um0)))
[1] -249.5472
> Um0 <- MHblk(Um0[,100],Rv,etav,nv,0.3, nblk=100)
>  .01*sum(Rv*(etav+Um0)-nv*log(1+exp(etav+Um0)))
[1] -249.1056
  ### Recall: this is not the actual logLik
### but the estimate of the conditional expectation
### of the full-data logLik

### Now how would we update the estimate of sigma ?
### The simplest way would seem to be, use the formula
###    for sig^2 = unconditional variance of u =
###    expected conditional variance of u plus
###    variance of conditional expectation of u :

> mean(apply(Um,1,var)) + var(apply(Um,1,mean))
[1] 0.2441128  ### Not bad: two pieces are .204, .040
### and
> sig.old <- sqrt(.244) ###  = 0.4940
```

Next we try a proper 3-parameter M-step in our quasi-EM algorithm. ('Quasi' because we are doing MCMC in place of numerical quadratures to obtain conditional expectations in the E-step.) In this step, we treat `Um/sig` as the imputed standard-normal deviates. We do a series of quasi-EM iterations in this way.

```
> th.old <- c(-1.089849, 1.083622, 0.5)
  Uma <- Um
  eta.old <- th.old[1] + th.old[2]*(etav+1)
  Uma <- MHblk(Uma[,100],Rv,eta.old,nv,th.old[3],
      nblk=100)/th.old[3]
  tmpmin3 <- nlm(function(thet) {
     etapu <- thet[1]+thet[2]*(1+etav)+thet[3]*Uma
     .01*sum(nv*log(1+exp(etapu))-Rv*etapu)
     }, th.old)   ###  converged, 9 iterations
$minimum
[1] 243.9371
$estimate
```

```
[1] -1.1382352  1.1268970  0.4618309
### successive additional iterates of exactly the same type
 -1.1433864  1.1292849  0.4418853
 -1.1291498  1.1123130  0.4070816
 -1.1319344  1.1338868  0.3786870
 -1.1252628  1.1193708  0.3348035
 -1.1333165  1.1340783  0.3227082
 -1.1144844  1.1136013  0.3161008
 -1.1143326  1.1020129  0.3052926
 -1.1186356  1.1016660  0.3010774
 -1.0945004  1.0725082  0.2747137
 -1.1063528  1.0811459  0.2855079
 -1.1148067  1.1088207  0.2687424
 -1.0987998  1.0840951  0.2569771
 -1.1065303  1.0925731  0.2513654
 -1.1077716  1.1034591  0.2502139
 -1.1044592  1.0967187  0.2226806
 -1.1061799  1.0973809  0.2266805
 -1.1033295  1.0832668  0.2158493
 -1.1074619  1.0941746  0.2242768
```

At this point, the EM iterations seem to have essentially converged. We conclude with a simple estimate of actual log-likelihood obtained by averaging complete-data likelihoods over imputed conditional u-values.

```
## (Recall Um0 is imputed array at true values:)
> sum(log(apply(exp((etav+Um0)*Rv)*dnorm(Um0,sd=0.3)/
    (1+exp(etav+Um0))^outer(nv,rep(1,100),"*"),1, mean)))
[1] -255.0079  ### actual logLik est at true params

>  etab <- tmpmin3$est[1] + tmpmin3$est[2]*(etav+1)
 Umb <- MHblk(Uma[,100],Rv, etab, nv,tmpmin3$est[3], nblk=100)
 sum(log(apply(exp((etab+Umb)*Rv)*dnorm(Umb,sd=tmpmin3$est[3])/
    (1+exp(etab+Umb))^outer(nv,rep(1,100),"*"),1, mean)))
[1] -227.4631  ### logLik est at converged param ests
```

It looks as though we should have cause for concern here: why is the (estimated) log-likelihood so different at the true and final estimated parameter

values ? In fact, careful numerically integrated log-likelihoods (not shown here) do show that the quasi-EM steps do work as they are supposed to, increasing or at worst slightly decreasing the true logLik at each EM step: it is the Monte-Carlo estimated log-likelihood values which are at fault here !