

MCMC algorithms for fitting Bayesian models

Sudipto Banerjee

`sudiptob@biostat.umn.edu`

University of Minnesota

The need for MCMC

- Bayesian analysis proceeds from the posterior distribution:

$$P(\boldsymbol{\theta}|Y) \propto P(\boldsymbol{\theta}) \times P(Y|\boldsymbol{\theta})$$

- The proportional constant is the marginal distribution:

$$\int P(\boldsymbol{\theta}) \times P(Y|\boldsymbol{\theta})d\boldsymbol{\theta},$$

which can be a high-dimensional integral.

- Usually a careful choice of priors is needed to avoid computation of this marginal distribution. This limits applicability of Bayesian models. Markov Chain Monte Carlo simulation methods (MCMC) methods enhance this applicability.

The Gibbs Sampler

- Let $\theta = (\theta_1, \dots, \theta_p)$ be the parameters in our model. The Gibbs sampler starts a *Markov Chain* with a set of initial values $\theta_0 = (\theta_{01}, \dots, \theta_{0p})$ and then performs the i^{th} iteration, say for $i = 1, \dots, M$, by updating successively from the *full conditional* distributions:

$$\theta_{i1} \sim P(\theta_1 | \theta_{i-1,2}, \dots, \theta_{i-1,p}, Y)$$

$$\theta_{i2} \sim P(\theta_2 | \theta_{i,1}, \theta_{i-1,3}, \dots, \theta_{i-1,p}, Y)$$

...

$$\text{(the generic } k^{th} \text{ element)} \theta_{ik} \sim P(\theta_k | \theta_{i,1}, \dots, \theta_{i,k-1}, \theta_{i-1,k+1}, \dots, \theta_{i-1,p}, Y)$$

...

$$\theta_{ip} \sim P(\theta_p | \theta_{i,1}, \dots, \theta_{i,p-1}, Y)$$

- The completion of the above loop results in a *single iterate* of the Gibbs sampler with an update of $\theta_1 = (\theta_{11}, \dots, \theta_{1p})$. This is repeated M times to obtain a Gibbs sample of vectors $\theta_1, \dots, \theta_M$. From Markov chain theory we know that such a chain will eventually converge to a *stationary* or *equilibrium* distribution which is **precisely** the posterior distribution $P(\theta | Y)$. What this means from a practical standpoint is that if we sample long enough, in the above scheme, we will eventually be sampling from the posterior distribution itself. So, after we discard an initial set of samples (called *burn-in*) we retain the remaining samples as our posterior sample and carry out all inference on them.

The Linear Model: again!

- Let us consider the linear model example. We have the equation:

$$Y = X\beta + \epsilon$$

where Y is a $n \times 1$ vector of responses, X is a $n \times p$ vector of covariates, β is the $p \times 1$ vector of regression parameters and ϵ is a vector of i.i.d. errors, distributed as $N(0, \sigma^2)$. All our inference will be implicitly conditioned on X . Consider again a flat prior on β , but an Inverted-Gamma prior, $IG(a, b)$ (so $1/\sigma^2$ has a Gamma distribution with mean = a/b , variance = a/b^2) on σ^2 .

- Therefore, $\theta = (\beta, \sigma^2)$ and we need to update these. The full conditional distributions are easily obtained as:

$$P(\beta|Y, \sigma^2) = N((X^T X)^{-1} X^T Y, \sigma^2 (X^T X)^{-1})$$

$$P(\sigma^2|Y, \beta) = IG(a + n/2, b + \frac{1}{2}(Y - X\beta)^T (Y - X\beta)).$$

The latter is much easier to identify than the *marginal* distribution of σ^2 . This is the primary benefit of the Gibbs sampler – it helps avoid computing unfriendly marginal distributions.

contd.

- Note that the decomposition of $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma^2)$ involves a vector $\boldsymbol{\beta}$ and a scalar σ^2 . The vector $\boldsymbol{\beta}$ is *block-updated* within the Gibbs sampler using a multivariate normal distribution. Indeed, more generally, we can partition a parameter vector $\boldsymbol{\theta}$ in any manner we like, say into sub-vectors or single scalar components. This choice is often based upon ease of coding and sampler properties. While the convergence properties of the MCMC might be affected depending upon how you update, eventually they will still converge.
- It is fundamentally important to note that the full conditional distributions are also needed only up to a proportionality constant - and are simply proportional to: $P(\boldsymbol{\theta}) \times P(Y | \boldsymbol{\theta})$. Why? Note that:

$$P(\theta_i | \boldsymbol{\theta}_{-i}, Y) = \frac{P(\boldsymbol{\theta} | Y)}{P(\boldsymbol{\theta}_{-i} | Y)} \propto P(\boldsymbol{\theta} | Y) \propto P(\boldsymbol{\theta}) \times P(Y | \boldsymbol{\theta}).$$

Gibbs Sampler - Homework

- Write your own `R` code to fit a linear model with all the covariates to the Chicago real estate linear model data in Chapter 2, Exercise 11 by designing a Gibbs sampler as above. Use an Inverse Gamma $IG(2, b)$ prior with $b = 0.01$ and 100 for σ^2 and a flat prior for β . Run one chain for 2100 iterations. Discard the first 100 samples as “burn-in” and present your posterior estimates from the remaining 2000 samples. Compare your results with those obtained by running `WinBUGS` on the same data set (You have already done this in an earlier hw).

The Metropolis-Hastings algorithm

- In principle, the Gibbs sampler will work for extremely complex hierarchical models. The only issue is sampling from the full conditionals. They may not be amenable to easy sampling – when these are not in closed form. A more general and extremely powerful - and often easier to code - algorithm is the Metropolis-Hastings (MH) algorithm.
- This algorithm also constructs a Markov Chain, but does not necessarily care about full conditionals.
- This algorithm uses a *candidate* or *proposal* distribution, say $q(\cdot, \nu)$, where ν is/are parameters that is/are *fixed* by the user - called tuning parameters. As we outline in the next page, the MH algorithm constructs a Markov Chain by proposing a value for θ from this candidate distribution, and then either accepting or rejecting this value (with a certain probability). Theoretically the proposal distribution can be any distribution, but in practice you choose something really simple: a Normal distribution if your parameter can be any real number, (e.g. β) or a log-normal if it has positive support (e.g. σ^2).

contd.

- The MH algorithm: Select a *candidate* or *proposal* distribution $q(\cdot, \nu)$. Fix ν .
Start with a initial value for $\theta = \theta_0$. At the i^{th} iteration for $i = 1, \dots, M$ do the following:
Draw $\theta^* \sim q(\cdot | \theta_{i-1}, \nu)$ and compute

$$r = \frac{P(\theta^* | Y)q(\theta_{i-1} | \theta^*, \nu)}{P(\theta_{i-1} | Y)q(\theta^* | \theta_{i-1}, \nu)}$$

If $r \geq 1$ then set $\theta_i = \theta^*$

If $r \leq 1$ then draw $U \sim (0, 1)$. If $U \leq r$ then $\theta_i = \theta^*$. Otherwise, $\theta_i = \theta_{i-1}$.

- This generates samples $\theta_1, \dots, \theta_M$, which after a burn-in period will sample from the true posterior distribution.
- It is important to monitor the acceptance ratio r of the sampler through the iterations. When updating vectors as a whole, it is recommended that this ratio be approx. around 20%. If updating a single scalar, it is recommended that r be around 40%. Setting lower values of ν will increase r , while setting higher values of ν will tend to lower r .

MH fitting with linear model

- A symmetric $q(\cdot|\boldsymbol{\theta})$, say $q(\boldsymbol{\theta}^*|\boldsymbol{\theta}_{i-1}, \nu) = MVN(\boldsymbol{\theta}_{i-1}, \nu)$, will cancel out in the computation of r . Then $\log(r) = \log(P(\boldsymbol{\theta}^*|Y)) - \log(P(\boldsymbol{\theta}_{i-1}|Y))$. In practice it is better to compute $\log(r)$. For the proposal, $MVN(\boldsymbol{\theta}_{i-1}, \nu)$, ν is a $d \times d$ variance-covariance matrix, and $d = \dim(\boldsymbol{\theta})$.
- If $\log r \geq 0$ then set $\boldsymbol{\theta}_i = \boldsymbol{\theta}^*$. If $\log r \leq 0$ then draw $U \sim (0, 1)$. If $U \leq r$ (or $\log U \leq \log r$) then $\boldsymbol{\theta}_i = \boldsymbol{\theta}^*$. Otherwise, $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1}$.
- Example: For the linear model, our parameters are $(\boldsymbol{\beta}, \sigma^2)$. To use a multivariate normal distribution for $q(\cdot|\boldsymbol{\theta})$ (by far the most popular choice), we need to write $\boldsymbol{\theta} = (\boldsymbol{\beta}, \log(\sigma^2))$. The log transformation on σ^2 ensures that all components of $\boldsymbol{\theta}$ have support on the entire real line – hence can lead to meaningful proposed values from the multivariate normal proposal. But we need to transform our prior to $p(\boldsymbol{\beta}, \log(\sigma^2))$ now!
- Let $z = \log(\sigma^2)$ and assume $p(\boldsymbol{\beta}, z) = p(\boldsymbol{\beta})p(z)$. Let us derive $p(z)$. **REMEMBER:** we need to adjust for the jacobian. Then $p(z) = p(\sigma^2)|d\sigma^2/dz| = p(e^z)e^z$. The jacobian here is $e^z = \sigma^2$.
- Let $p(\boldsymbol{\beta}) = 1$ and an $p(\sigma^2) = IG(a, b)$. Then log-posterior is:

$$-(a + n/2 + 1)z + z - \frac{1}{e^z} \left\{ b + \frac{1}{2} (Y - X\boldsymbol{\beta})^T (Y - X\boldsymbol{\beta}) \right\}.$$

MH - Homework

- Write your own R code to fit a linear model from the same regression dataset (Ch 2, Ex 11), but now designing a MH sampler. First fit a classical linear model (OLS) using R. Next adopt a Bayesian approach using an Inverse Gamma $IG(2, b)$ prior for σ^2 with a value of b that gives a *reasonable* prior estimate for σ^2 (from your OLS) and a flat prior for β . Run 4000 samples with a burn-in of 2000. Set tuning parameters for your metropolis to yield an acceptance rate between 15% to 30%. Compare from your WinBUGS and Gibbs sampler results.

The component-wise MH algorithm

- In practical data analysis, it is sometimes difficult to maintain a healthy acceptance rate for the MH algorithm. Then, a component-wise MH algorithm that updates each component using a univariate MH step is preferred.
- If $\theta = (\theta_1, \dots, \theta_p)$, then we now update each parameter using its own MH proposal $q_j(\theta_j | \cdot, \nu_j)$ for $j = 1, \dots, p$. Note that having their own tuning parameter ν_j greatly increases our control over the Markov Chain.

Start with a initial value: $\theta_0 = (\theta_{01}, \dots, \theta_{0p})$. At the i^{th} iteration for $i = 1, \dots, M$ perform the following cycle of j sub-iterations for $j = 1, \dots, p$:

Draw $\theta_j^* \sim q(\cdot | \theta_{i-1,j}, \nu_j)$ and compute

$$r = \frac{P(\theta_j^* | Y) q(\theta_{i-1,j} | \theta_j^*, \nu)}{P(\theta_{i-1,j} | Y) q(\theta_j^* | \theta_{i-1,j}, \nu_j)}$$

If $r \geq 1$ then set $\theta_{i,j} = \theta_j^*$

If $r \leq 1$ then draw $U \sim (0, 1)$. If $U \leq r$ then $\theta_{i,j} = \theta_j^*$. Otherwise, $\theta_{i,j} = \theta_{i-1,j}$.

- Note that the proposals $q_j(\cdot)$ can depend upon the other components θ_{-j} . In fact, quite like magic, if we take these proposals as the full-conditional distributions, we have a Gibbs sampler.

Finer points

- The R package `mcmc` (written by Charlie Geyer) has a function called `metrop` that implements MCMC using a random-walk metropolis proposal. The proposal used is a multivariate normal proposal.
- Users can build their target function (log of the joint posterior) and provide this as an input to the `mcmc` package.
- In addition, users will need to supply a vector of initial values and the Cholesky-square-root of a variance-covariance matrix of the multivariate normal proposal as the tuning matrix. The `metrop` function outputs a matrix with the MCMC chains. These can be analyzed using `CODA`.
- Recall: MCMC outputs *correlated samples*, as opposed to *i.i.d.* samples. Inference from correlated samples is still valid by virtue of the “ergodic theorem” for Markov Chains: given samples $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T\}$, and any function, $g(\boldsymbol{\theta})$, of the parameters.

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T g(\boldsymbol{\theta}_t) = E_{\boldsymbol{\theta}|\mathbf{y}}[g(\boldsymbol{\theta})],$$

- Even chains that have converged can be autocorrelated. Your inference will still be Monte-Carlo unbiased but you may need a large Monte Carlo sample size to increase accuracy. (See Carlin and Louis, 2008, p150–160).

Finer points – contd.

- Users **CANNOT** build their own proposal distributions. They must reparametrize their model to make it compatible with normal proposals.
- This approach will be demonstrated with a few examples.
- Suppose, you want to model $\sigma \sim U(a, b)$, as an alternative to the inverse-Gamma prior. Proposing metropolis updates for σ from a normal proposal may be problematic as they have different supports; i.e. σ can only lie in the interval $[a, b]$, while the proposed values could lie anywhere on the real line.
- So, we need to transform σ to some variable z that will have support over the whole real line. **Remember** we will need to adjust for the jacobian of the transformation.
- Example:

$$\sigma = b - (b - a)/(1 + \exp(z)); \quad d\sigma/dz = \exp\{z - 2 \log(1 + \exp(z))\}$$

- Another example:

$$\sigma = a + \frac{b - a}{\pi} \left(\arctan(z) + \frac{\pi}{2} \right); \quad d\sigma/dz = \frac{b - a}{\pi(1 + z^2)}$$

Investigating mle's and posterior modes

- Often it is useful to find maximum likelihood estimates and their variances, even for Bayesian computing. They may reveal weirdness of the likelihood surface, and sometimes help greatly in getting nice starting points for your chains.
- R has an excellent function called `optim` that uses several Newton-Raphson type methods to optimize user-defined functions.
- This same algorithm will also help in obtaining posterior modes – instead of the log-likelihood simply provide the joint-posterior.
- `optim` also computes the Hessian matrix of the parameters: $I(\boldsymbol{\theta})$.
- Note: this immediately yields approximate posterior inference via the Bayesian Central Limit Theorem:

$$p(\boldsymbol{\theta}|\mathbf{y}) \approx N(\hat{\boldsymbol{\theta}}, I^{-1}(\hat{\boldsymbol{\theta}}))$$

- You can now sample from this asymptotic distribution. This method is still widely used in the machine-learning and engineering communities, where the requirements for measures of uncertainty are not as stringent.
- We will now illustrate with the `optim` function.

Some practical tips

- It is sometimes useful to reparametrize the parameters in a manner that will ease optimization.
- As an example, in OLS regression, we might want to optimize with respect to (β, z) , where $z = \log(\sigma^2)$. Then all the parameters have range over the entire real line and we can use *unconstrained optimization*.

Once the parameter is estimated, and we have obtained the variance-covariance matrix $I(\theta)$, using the “delta-method”. we can then obtain $Var(\mathbf{g}(\theta))$

- More generally, get $\mathbf{g}(\theta)$ be a transformation of the parameters θ . Then,

$$I^{-1}(\mathbf{g}(\hat{\theta})) \approx [\nabla \mathbf{g}(\hat{\theta})]^T I^{-1}(\hat{\theta}) [\nabla \mathbf{g}(\hat{\theta})].$$

- Example with OLS: let $\theta = (\beta, z)$ and let $\mathbf{g}(\theta) = (\beta, e^z)$. Then,

$$I^{-1}(\mathbf{g}(\hat{\theta})) \approx \begin{pmatrix} I_p & 0 \\ 0 & e^{\hat{z}} \end{pmatrix} I^{-1}(\hat{\beta}, \hat{z}) \begin{pmatrix} I_p & 0 \\ 0 & e^{\hat{z}} \end{pmatrix}$$

- Alternative: use *constrained optimization* on σ^2 , which often does not work as well.