Multilevel Models

8. Bayesian Inference via Metropolis-Hastings

Germán Rodríguez

Princeton University

April 18, 2018

Markov Chain Monte Carlo

The Gibbs sampler is very popular but by no means the only MCMC method. An alternative is the Metropolis-Hastings algorithm, which can sample from a multivariate distribution in one step. Robert and Casella (2010) have a nice introduction to Monte Carlo Methods with R.

The basic idea given a target density f is to build a Markov Chain that has stationary distribution f. You'd think this is hard, but in fact there are methods that work in principle for any density. One such method is Metropolis-Hastings. (Another is Gibbs sampling.)

Stata now has Bayesian methods using Metropolis-Hastings, and R has an interface to Stan, which implements a variant of the algorithm using Hamiltonian dynamics, and includes a package that can fit many standard models by calling Stan to do the work.

We describe some basic features of the algorithm before turning to the implementations.

Metropolis Hastings

For a target f we need a density $q(y|x)$ that is easy to sample from (for example multivariate normal) and such that the ratio $f(y)/q(y|x)$ is known up to a constant independent of x.

If $q(.|x)$ has enough variation to cover the support of f we can build a chain that has stationary distribution f using a surprisingly simple algorithm:

Given $x^{(t)}$,

1. Generate $Y_t \sim q(y|x^{(t)})$, and
2. Take $x^{(t+1)} = Y_t$ with probability $\rho(x^{(t)}, Y_t)$ and $x^{(t)}$ otherwise, where

$$\rho(x,y) = \ min\{\frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}, 1\}$$

is the *acceptance probability*. Note that sometimes we keep the old value! The kernel q is called the *proposal* and affects the acceptance rate and efficiency of the chain.

Independent and Random Walk Variants

The basic algorithm allows the draw to depend on the current state of the chain, but this is not necessary and the proposal can be $q(y|x) = q(y)$. This leads to a simplified algorithm called *independent MH*, which is simple but hard to tune well.

One way to take into account the previous value is to simulate $Y_t = X^{(t)} + \epsilon_t$ where ϵ_t is a random perturbation with distribution g independent of $X^{(t)}$, so the proposal density $q(y|x)$ has the form $g(y - x)$, leading to the *random walk MH*

Given $x^{(t)}$,

1. Generate $Y_t \sim g(y - x^{(t)})$, and
2. Take $x^{(t+1)} = Y_t$ with probability $\min\{f(Y_t)/f(x^{(t)}), 1\}$ and $x^{(t)}$ otherwise

In fact this was the original version of the algorithm. Sometimes, howevever, random walks are slow to converge, and efficiency is highly dependent on the choice of g.

Hybrid or Hamiltonian Monte Carlo (HMC)

The latest development in MCMC is a hybrid algorithm that uses Hamiltonian dynamics borrowed from physics to improve on traditional Metropolis-Hastings by producing proposals far from the current values yet with high probability of acceptance.

The Hamiltonian of a system describes the movement of a particle given its position and momentum in space and leads to differential equations for its trajectory over time.

In statistical MCMC we treat minus the log of the posterior density as the position, and sample the momentum along each dimension from independent Gaussian distributions.

The trajectory is simulated in discrete time using L steps of size ϵ using a method known as *leapfrog* to reach a proposed state, which is then accepted or rejected using H-M with appropriate acceptance probability. See Neal (2011) for an excellent discussion.

The No-U-Turn Sampler (NUTS)

One difficulty with HMC is that it needs the gradient of the log posterior in order to compute momentum. But this can be handled using automatic differentiation.

Another difficulty is the need to fine tune the two HMC parameters L and ϵ, which is essential to obtain an efficient algorithm.

Hoffman and Gelman (2014) proposed an HMC variant known as NUTS that avoids the need to specify the number of steps L while ensuring that the trajectory is followed long enough, and can auto-tune ϵ using a clever scheme to achieve the same efficiency as HMC, and sometimes even better.

The end result is an algorithm that seems very well suited for automatic Bayesian inference without the need for costly tuning steps or substantial expertise.

Stan

The NUTS variant of the HMC algorithm has been implemented in the program Stan, a “probabilistic programming language” from Gelman’s group, named after Stanislaw Ulam, inventor of Monte Carlo. The language has a website at `http://mc-stan.org`.

Stan is a high-level language not unlike BUGS that can be used to specify a model, but then generates a C++ program that is compiled and run to generate the samples efficiently.

There are interfaces to run Stan from R and Stata (as well as Python, Julia, Matlab, Mathematica and Scala) which help a bit, but still require learning the modeling language.

There is also an R package called `RStanArm` that makes using Stan extremely easy for standard models because you can specify them using R syntax!

The Hospital Data

My first experience with Stan was running the Lillard and Panis hospital delivery data. Here's the code, saved in R as a string:

```
data {
    int N; // number of obs (pregnancies)
    int M; // number of groups (women)
    int K; // number of predictors

    int y[N]; // outcome
    row_vector[K] x[N]; // predictors
    int g[N];    // map obs to groups (pregnancies to women)
}
parameters {
    real alpha;
    real a[M];
    vector[K] beta;
    real<lower=0,upper=10> sigma;
}
model {
  alpha ~ normal(0,100);
  a ~ normal(0,sigma);
  beta ~ normal(0,100);
  for(n in 1:N) {
    y[n] ~ bernoulli(inv_logit( alpha + a[g[n]] + x[n]*beta));
  }
}
```

The variable names are not very descriptive because I wanted to write code I could use for other random-intercept logit models.

Running from R

To run the model I first copied the data from hosp to a list with the same names as the Stan code

```
hosp_data <- list(N=nrow(hosp),M=501,K=4,y=hosp[,1],x=hosp[,2:5],g=hosp[,6])
```

I then ran the model specifying 2 chains of 2000 samples each

```
hfit <- stan(model_code=hosp_code, model_name="hospitals", data=hosp_data, iter=2000, chains=2)

TRANSLATING MODEL 'hospitals' FROM Stan CODE TO C++ CODE NOW.
COMPILING THE C++ CODE FOR MODEL 'hospitals' NOW.
...
SAMPLING FOR MODEL 'hospitals' NOW (CHAIN 1).
Iteration: 2000 / 2000 [100%]  (Sampling)
Elapsed Time: 58.065 seconds (Warm-up)
              24.373 seconds (Sampling)
              82.438 seconds (Total)

SAMPLING FOR MODEL 'hospitals' NOW (CHAIN 2).
Iteration: 2000 / 2000 [100%]  (Sampling)
Elapsed Time: 58.074 seconds (Warm-up)
              23.186 seconds (Sampling)
              81.26 seconds (Total)
```

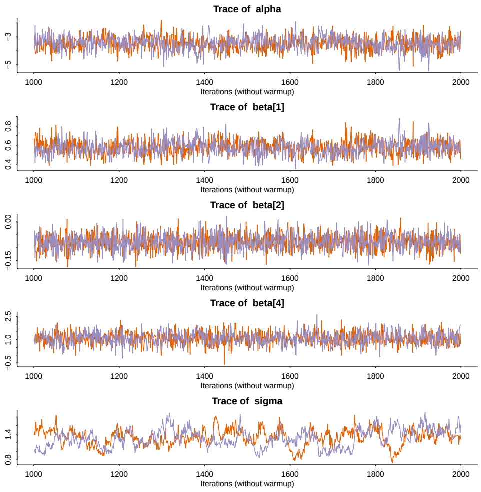
The computing log at hospStan.html has more details about this run.

Stan and hospital deliveries

Here are the Stan trace plots for the parameters, showing two chains.

For generality I used a vector of coefficients β. The actual names are `loginc`, `distance`, `dropout` and `college`.

As you can see, we get mostly fuzzy caterpillars, but the standard deviation of the random effects at the woman level exhibits slow mixing.



Stan Meets Applied Regression Modeling

The R package RStanArm makes it very easy to run the types of models in Gelman and Hill's ARM book by providing an R interface almost identical to glm and glmer to specify the model, which is then run in Stan using pre-compiled code.

Here's the R call for maximum likelihood:

```
glmer(hosp ~ loginc + distance + dropout + college + (1 | mother),
    data = hosp, family = binomial, nAGQ = 12)
```

And here's the equivalent R call for Bayesian estimation:

```
stan_glmer(hosp ~ loginc + distance + dropout + college + (1 | mother),
    data = hosp, family = binomial)
```

This will run four chains with burn-ins and samples of 1,000 observations each.

I recommend using this interface for standard models and then learning the more powerful Stan language to fit a much wider variety of realistically complex models.

Metropolis-Hastings in Stata

Stata now has a bayeshm command that can fit a variety of models using a random walk Metropolis-Hastings algorithm. The developers note that the algorithm is not optimal for Bayesian multilevel models, but can be used in models that do not have too many random effects. Here's a command that will run a random-intercept model with the hospital data

```
bayesmh hospital loginc distance dropout college ibn.group ///
  , likelihood(logit) ///
  prior({hospital:i.group}, normal(0,{var})) ///
  prior({hospital:loginc distance dropout college _cons}, normal(0,1000)) ///
  prior({var}, igamma(0.001,0.001)) ///
  block({hospital:i.group}, reffects) ///
  block({hospital:loginc distance dropout college _cons}) ///
  block({var})
```

The syntax is similar to other Stata commands, treating the grouping variable as a factor without a reference cell. Sampling all parameters together is inefficient and we work in blocks, separating the fixed, random and variance parameters. See the computing log at hospStata.html for details.