(Last adjustments: December 6, 2017)

Workshop on statistical challenges in astronomy –Hierarchical models in Stan

Presenter

Dr. John T. Ormerod

School of Mathematics & Statistics F07

University of Sydney

(w) 02 9351 5883

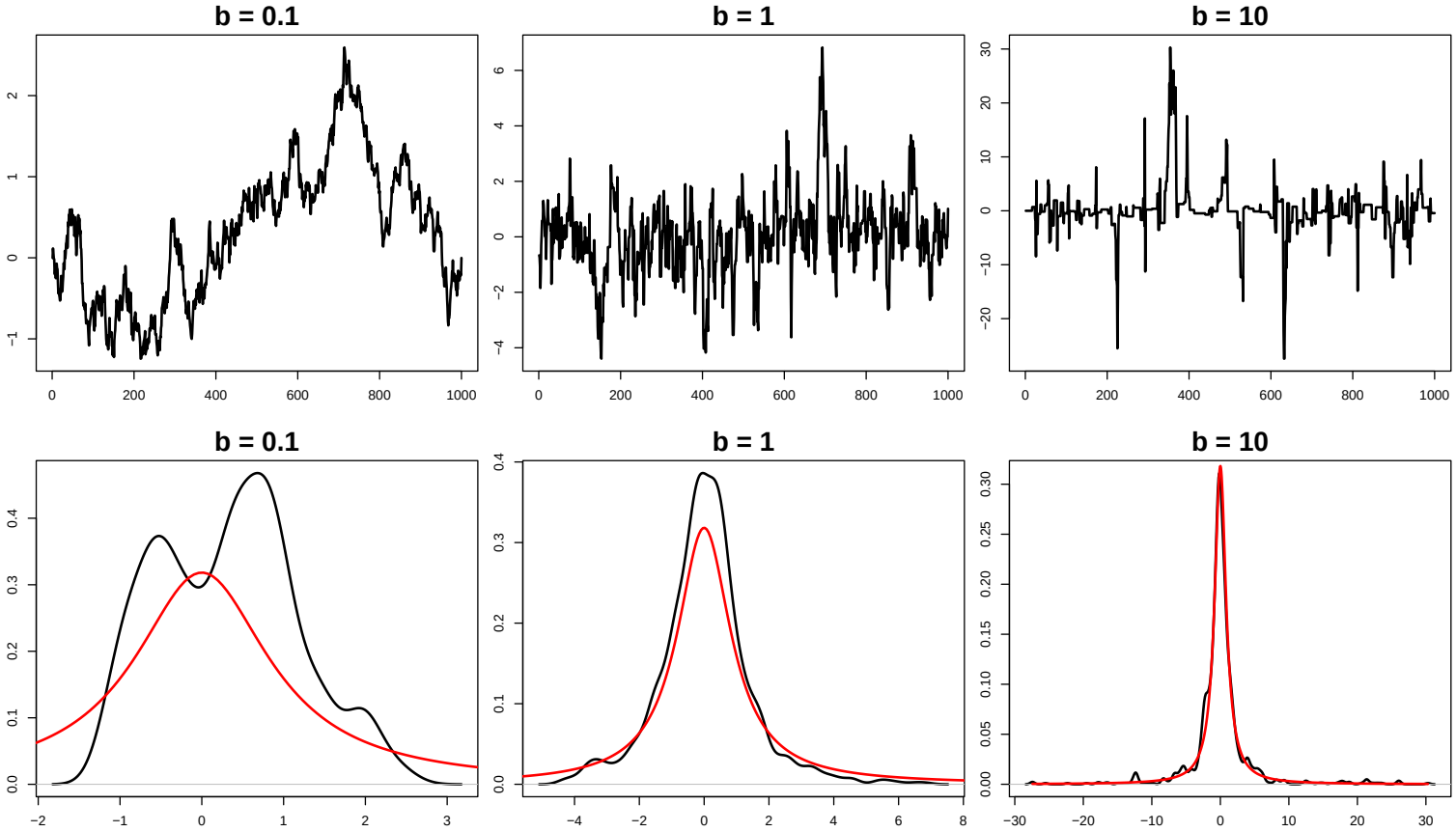
(e) john.ormerod (at) sydney.edu.au

Hamiltonian Monte Carlo

Hamiltonian Monte Carlo – Motivation

- One of the inherent problems with Gibbs sampling and Metropolis-Hastings MCMC methods is that, for particular problems, the Markov Chains samples can exhibit high amounts of autocorrelation leading to random walk behaviour.
- Zigging and zagging whole moving thought the target distribution. Reparameterization and other tricks can improve the situation, but for complex models it can be a time consuming problem to tune all of the proposals in the model.

Hamiltonian Monte Carlo – Motivation



Hamiltonian Monte Carlo

Hamiltonian Monte Carlo or Hybrid Monte Calro (HMC) borrows an idea from physics to suppress the coal random walk behaviour in the Metropolis-Hastings algorithm, thus allowing it to move much more rapidly through the target distribution.

- For each parameter θ_j in the parameter space we introduce a "momentum" variable ϕ_j.
- Both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are updated together in a new Metropolis-Hastings algorithm in which the jumping distribution for $\boldsymbol{\theta}$ is largely determined by $\boldsymbol{\phi}$.
- Each iteration of HMC consists of L steps during which the position and momentum evolve based on rules imitating the behaviour of position.
- The steps can move rapidly where possible though the space of $\boldsymbol{\theta}$ and can even turn corners in parameter space to preserve the "total energy" of the trajectory.

Hamiltonian Monte Carlo

- As usual we only need to know the posterior $p(\boldsymbol{\theta}|\mathbf{y}) \propto p(\mathbf{y}, \boldsymbol{\theta})$ up to proportionality.
- This is augmented by an independent distribution $p(\boldsymbol{\phi})$ defining an new joint distribution

$$p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathbf{y}) \propto p(\boldsymbol{\phi})p(\mathbf{y}, \boldsymbol{\theta}).$$

- We then simulate from the join distribution, but we are only interested in the simulations for $\boldsymbol{\theta}$.
- The simulations from $\boldsymbol{\phi}$ are auxiliary and can be discarded.
- The role $\boldsymbol{\phi}$ allows us to move faster through parameter space.

Hamiltonian Monte Carlo

- One of the requirements of the algorithm is the calculation of the log-posterior density.

$$\frac{\partial \log p(\boldsymbol{\theta}|\mathbf{y})}{\partial \boldsymbol{\theta}^T} = \frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} = \left(\frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \theta_d} \right)$$

- While the derivatives can be calculated numerically. This is not recommended because of the number of function evaluations involved.

Hamiltonian Monte Carlo – Momemntum distrubiton $p(\boldsymbol{\phi})$

- It is usual to give $p(\boldsymbol{\phi})$ a multivariate normal distribution with mean $\mathbf{0}$ and covariance $\mathbf{M}$ which is sometimes called the "mass matrix" (so called by analogy to the physical model of Hamiltonian dynamics).
- To further simplify matters we will assume that the mass matrix is diagonal so

$$\phi_j \sim N(0, M_{jj})$$

- It can be useful for $\mathbf{M}$ to roughly scale with $\text{Cov}(\boldsymbol{\theta}|\mathbf{y})$, but the algorithm will work in any case. This choice only makes the algorithm more efficient.

Hamiltonian Monte Carlo – Steps

HMC proceeds via iterations similar to any Metropolis-Hastings algorithm with a proposal which takes L steps. Each iteration consists of four parts.

1. Draw $\boldsymbol{\phi} \sim N(\mathbf{0}, \mathbf{M})$.
2. Perform the following "Leapfrog" steps L times
   (a) Use the gradient of the log-posterior to make a half-step of $\boldsymbol{\phi}$ via
   $$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \frac{1}{2} \cdot \epsilon \cdot \frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T}$$
   (b) Use the "momentum" vector $\boldsymbol{\phi}$ to update the "position" vector $\boldsymbol{\theta}$ via
   $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \cdot \mathbf{M}^{-1} \boldsymbol{\phi}$$
   (c) Again use the gradient of $\boldsymbol{\theta}$ to half update $\boldsymbol{\phi}$.
   $$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \frac{1}{2} \cdot \epsilon \cdot \frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T}$$
   (As a rule of thumb the inventors of HMC use $L\epsilon \approx 1$.)

3. Label $(\boldsymbol{\theta}^{(t-1)}, \boldsymbol{\phi}^{(t-1)})$ as the value of the parameter and momentum vectors at the previous iteration and $(\boldsymbol{\theta}^*, \boldsymbol{\phi}^*)$ as the value after L leapfrog steps. Use the accept reject step

$$r = \frac{p(\mathbf{y}, \boldsymbol{\theta}^*)p(\boldsymbol{\phi}^*)}{p(\mathbf{y}, \boldsymbol{\theta}^{(t-1)})p(\boldsymbol{\phi}^{(t-1)})}$$

4. Set

$$(\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}^{(t)}) = \begin{cases} (\boldsymbol{\theta}^*, \boldsymbol{\phi}^*) & \text{with probability } \min(r, 1) \\ (\boldsymbol{\theta}^{(t-1)}, \boldsymbol{\phi}^{(t-1)}) & \text{otherwise.} \end{cases}$$

As with any MCMC we repeat these iterations until approximate convergences and the effective sample sizes for each change is large enough for inference of quantities of interest.

Hamiltonian Monte Carlo – Leapfrog steps

What is happening in the leapfrog steps?

- The stepping starts with a half-step for ϕ, $L-1$ full steps of the parameter vector $\boldsymbol{\theta}$ and the momentum vector $\boldsymbol{\phi}$ and then concludes with a half step of $\boldsymbol{\phi}$.
- It is called a "leapfrog" algorithm because of the splitting of the momentum updates into half steps and is a discrete approximation to physical Hamiltonian dynamics i which both the position and momentum evolve in continuous time.

- Suppose that $p(\mathbf{y}, \boldsymbol{\theta})p(\boldsymbol{\phi})$ is in a flat area of of the posterior then

$$\frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T} \approx \mathbf{0}.$$

  and the momentum update

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \frac{1}{2}\epsilon \frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T}$$

  will remain approximately constant. In this case the the leapfrog steps will skate along $\boldsymbol{\theta}$ space with constant velocity.

- Now suppose that $\boldsymbol{\theta}$ moves toward an area of low posterior density. Then

$$\frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T}$$

  will be negative in this direction, thus in step 2 inducing a decrease in the momentum in the direction of movement.

- As the leapfrog steps continue to move into an area of low density in $\boldsymbol{\theta}$ space the momentum will continue to decrease.

- If iterations continue to move in this direction the iterations will slow to zero and then black down or curve around the dip.
- If the algorithm heads in a direction where $p(\mathbf{y}, \boldsymbol{\theta})$ us increasing then

$$\frac{\partial \log p(\mathbf{y}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^T}$$

  will be positive leading to an increase of momentum in that direction. As $p(\mathbf{y}, \boldsymbol{\theta})$ increase $p(\boldsymbol{\phi})$ increases correspondingly until the trajectory eventually moves past or around the mode and ten starts to slow again.
- Note that the theoretically optimal acceptance rate for HMC is around 65%. So L, ϵ and $\mathbf{M}$ can be tuned if necessary to achieve this.

Combining HMC with Gibbs sampling

There are two ways in which te ideas of the Gibbs sampler fit into HMC.

- First it can bake sense to partition variables into blocks, either to simplify computation or to speed convergence.
- Consider a hierarchical model with J groups with parameter vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J)$ where $\boldsymbol{\theta}_j$ is itself a vector of parameters each corresponding to a different group so that the likelihood can be factored into

$$p(\mathbf{y}, \boldsymbol{\theta}) = \prod_{j=1}^{j} p(\mathbf{y}_j|\boldsymbol{\theta}_j)p(\boldsymbol{\theta}_j)$$

  (or something similar) then the log-likelihood derivatives themselves decompose.
- The second, perhaps more important use of Gibbs sampling is though the update of discrete parameter/latent variables (since HMC is only defined for continuous distributions).

Hamiltonian Monte Carlo – Code

First we need to define the log-joint and gradient functions

```
f.fun <- function(vbeta,vy,mX,sigma2.beta)
{
    veta <- mX%*%vbeta
    log.p <- sum(vy*veta - log(1 + exp(veta))) - 0.5*sum(vbeta^2)/sigma2.beta
    return(log.p)
}

vg.fun <- function(vbeta,vy,mX,sigma2.beta)
{
    veta <- mX%*%vbeta
    vmu <- 1/(1+exp(-veta))
    vg <- t(mX)%*%(vy - vmu) - vbeta/sigma2.beta
    return(vg)
}
```

Hamiltonian Monte Carlo – Code

```
hmc_iteration <- function(vbeta,vy,mX,sigma2.beta,epsilon,L,M)
{
    M_inv <- 1/M; d <- length(vbeta); phi <- rnorm(d,0,sqrt(M)); vbeta.old <- vbeta
    log.p.old <- f.fun(vbeta,vy,mX,sigma2.beta) - 0.5*sum(M_inv*phi*phi)
    phi <- phi + 0.5*epsilon*vg.fun(vbeta,vy,mX,sigma2.beta)
    for (l in 1:L) {
        vbeta <- vbeta + epsilon*M_inv*phi
        phi <- phi + (if (l==L) 0.5 else 1)*epsilon*vg.fun(vbeta,vy,mX,sigma2.beta)
    }
    phi <- -phi
    log.p.star <- f.fun(vbeta,vy,mX,sigma2.beta) - 0.5*sum(M_inv*phi*phi)
    r <- exp(log.p.star - log.p.old)
    if (is.nan(r)) { r <- 0; }
    p_jump <- min(r,1)
    vbeta.new <- if (runif(1) < p_jump) vbeta else vbeta.old
    return(list(vbeta=vbeta.new,p_jump=p_jump))
}
```

Hamiltonian Monte Carlo – Code

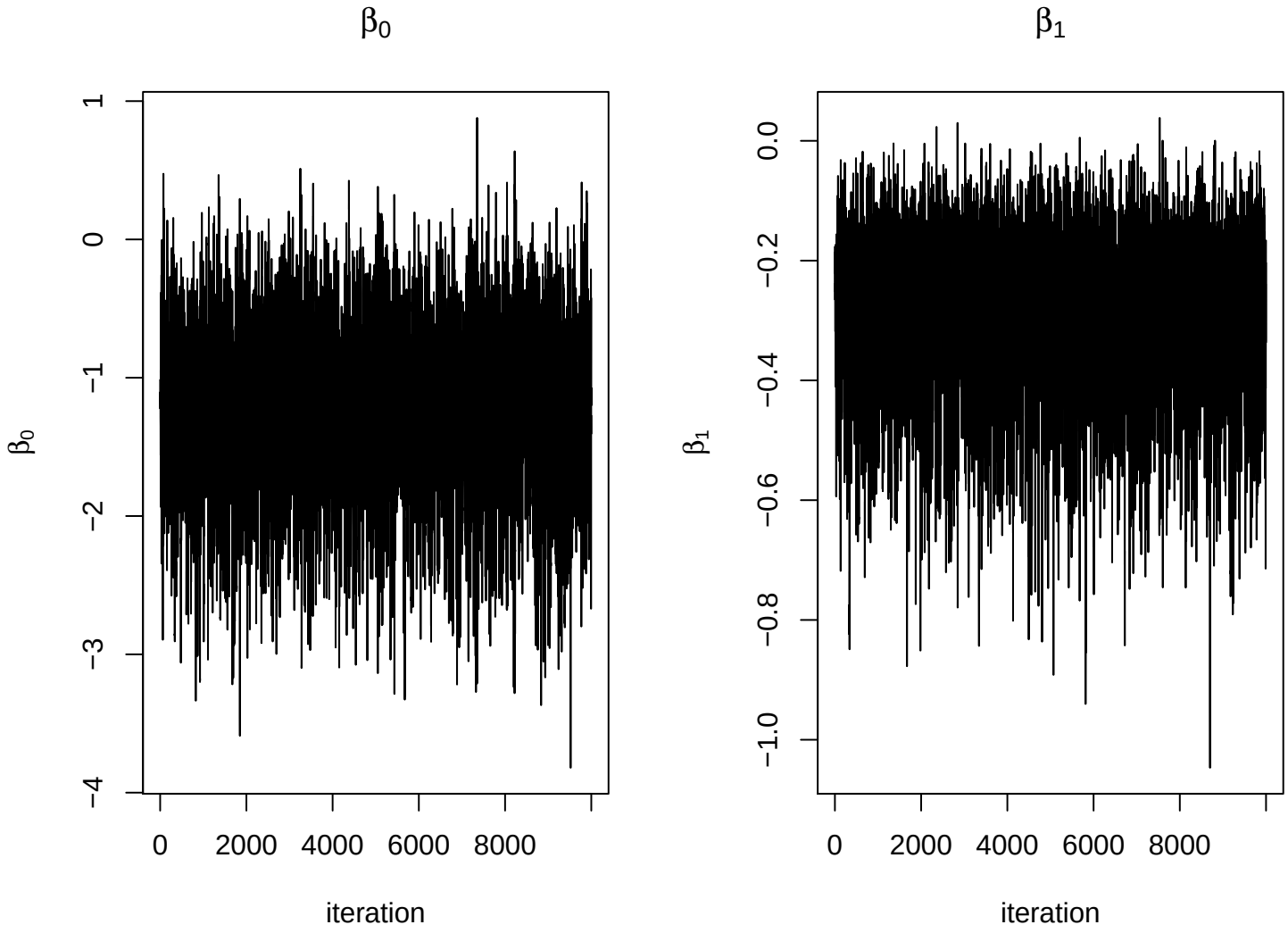
```
hmc_run <- function(vbeta.start,iter,epsilon_0,L_0,M,vy,mX,sigma2.beta)
{
    d <- length(vbeta.start)
    mBeta <- matrix(0,d,iter)
    vbeta <- vbeta.start
    p_jump <- c()
    for (i in 1:iter) {
        epsilon <- runif(1,0,2*epsilon_0)
        L <- ceiling(2*L_0*runif(1))
        temp <- hmc_iteration(vbeta,vy,mX,sigma2.beta,epsilon,L,M)
        p_jump[i] <- temp$p_jump
        mBeta[,i] <- temp$vbeta
        vbeta <- temp$vbeta
    }
    return(list(mBeta=mBeta,p_jump=p_jump))
}
```

Hamiltonian Monte Carlo – Code

```
vbeta.start <- res.glm$coefficients
iter <- 10000
epsilon_0 <- 0.1
L_0 <- 10
M <- 3

res <- hmc_run(vbeta.start,iter,epsilon_0,L_0,M,vy,mX,sigma2.beta=1.0E8)
```

Hamiltonian Monte Carlo – Chains



Hamiltonian Monte Carlo – Effective sample size

Note `iter <- 10000`. The `coda` library is a convenient function for finding the effective number of samples.

```
> ar(res$mBeta[1,],order.max =1)
Call:
ar(x = res$mBeta[1, ], order.max = 1)
Coefficients:
1
0.5026
Order selected 1  sigma^2 estimated as  0.2808
ar(res$mBeta[2,],order.max =1)
Call:
ar(x = res$mBeta[2, ], order.max = 1)
Coefficients:
1
0.2511
Order selected 1  sigma^2 estimated as  0.01598
library(coda)
effectiveSize(res$mBeta[1,])
> 3310.221
effectiveSize(res$mBeta[2,])
> 5000.928
```

Hamiltonian Monte Carlo – Stan

- `Stan` is a Bayesian modelling package which fits models using HMC. It has also implemented a stochastic variational Bayes (SVB) algorithm (which uses a mixture of normal specifications for $q(\boldsymbol{\theta})$.)
- It is available form http://mc-stan.org where you can find code and further examples.

Stan – Code

```
library(rstan)
dat <- list(
    n=n,
    vy = vy,
    vx = vx
)
```

Stan – Code

```
model_logisticRegression <- "
data {
    int<lower=0> n;
    vector[n] vx;
    int<lower=0,upper=1> vy[n];
}
parameters {
    real alpha0;
    real alpha1;
}
model {
    alpha0 ~ normal(0.0,1.0E3);
    alpha1 ~ normal(0.0,1.0E3);
    for (i in 1:n)
        vy[i] ~ bernoulli_logit(alpha0 + alpha1*vx[i]);
}"
```

Stan – Code

```
fit <- stan(model_code = model_logisticRegression , data = dat ,
    iter = 10000 , warmup = 100 , init="random" , chains = 1 )
res.stan <- extract(fit)
beta0 <- res.stan$alpha0
beta1 <- res.stan$alpha1
```

Hamiltonian Monte Carlo – Chains