Hamiltonian Monte Carlo

Hedibert F. Lopes
Insper Institute of Education and Research
June 1st 2025

Contents

1 MCMC via Hamiltonian Monte Carlo

Hamiltonian equations originate from classical mechanics and describe the evolution of a physical system in terms of its position and momentum variables over time. Formulated by William R. Hamilton in the 19th century, these equations characterize the system's dynamics through a Hamiltonian function, which typically represents the total energy (kinetic plus potential) of the system. The system evolves along trajectories that conserve this Hamiltonian, making the equations symplectic and volume-preserving in phase space. This framework inspired the development of Hamiltonian Monte Carlo (HMC), a powerful MCMC method used in Bayesian statistics for efficient sampling from complex posterior distributions. HMC leverages the Hamiltonian dynamics to propose distant states with high acceptance rates, overcoming random-walk behavior common in traditional MCMC. The seminal reference

connecting these ideas is Neal's 2011 paper, "MCMC Using Hamiltonian Dynamics" (Neal, 2011), which popularized HMC in statistical computing. The foundation in classical mechanics traces back to Hamilton's original work (Hamilton, 1834), laying the theoretical groundwork that bridges physics and modern computational Bayesian methods.

Thomas and Tu (2021) start the Section 3 of their *The American Statistician* paper "Learning Hamiltonian Monte Carlo in R", by say that

> *HMC improves the efficiency of MH by employing a guided proposal generation scheme. More specifically, HMC uses the gradient of the log posterior to direct the Markov chain towards regions of higher posterior density, where most samples are taken. As a result, a well-tuned HMC chain will accept proposals at a much higher rate than the traditional MH algorithm.*

They continue by saying that

> *To generate θ in a region of high posterior density, one needs to sample θ in the region corresponding to the lower values of $-\log \pi(\theta)$; the region can be reached with the guidance of the gradient of $-\log \pi(\theta)$. In a sense, the approach is analogous to the movement of a hypothetical object on a frictionless curve, where the object traverses and lingers at the bottom of the valley while occasionally visiting the higher grounds on both sides. In classical mechanics, such movements are described by the Hamiltonian equations, where the exchanges of kinetic and potential energy dictate the object's location at any given moment.*

1.1 Hamiltonian MC

Hamiltonian Monte Carlo (HMC) is arguably a highly efficient MCMC method, which can be corroborated, at least empirically, by the vast use of stan/rstan in an increasing number of models. HMC borrows ideas from classical physics to propose distant moves in parameter space with higher acceptance rates.

In classical mechanics, a system's state is described by its position, θ (parameters to sample) and its momentum, p (auxiliary variable). Then, the total energy of the system is given by the Hamiltonian function

$$H(\theta, p) = U(\theta) + K(p),$$

where p, θ are in $\mathbb{R}^k$. Typically, in MCMC problems, the $U(\theta) = -\log \pi(\theta)$, while for momentum $p \sim N_k(0, M)$, such that

$$H(\theta, p) = -\log \pi(\theta) + 0.5 p' M^{-1} p,$$

with the HMC traveling on trajectories that are governed by the following first-order differential equations, known as the Hamiltonian equations:

$$\begin{aligned} \frac{dp}{dt} &= -\frac{\partial H(\theta, p)}{\partial \theta} = -\frac{\partial U(\theta)}{\partial \theta} = \nabla_\theta \log \pi(\theta), \\ \frac{d\theta}{dt} &= \frac{\partial H(\theta, p)}{\partial p} = \frac{\partial K(p)}{\partial p} = M^{-1} p, \end{aligned}$$

where $\nabla_\theta \log \pi(\theta)$ is the gradient of the log posterior density. Therefore, θ could be sampled from the path for (θ, p) defined by the solution to the Hamiltonian equations.

1.2 Solving Hamiltonian equations: the leapfrog algorithm

Solving the Hamiltonian equations becomes a central problem, which is usually tackled by Euler's discretization method. In HMC, the leapfrog algorithm is the alternative to Euler's method that uses a discrete step size ϵ individually for p and θ, with a full step ϵ in θ between two half-steps $\epsilon/2$ for p. More precisely, starting at $(\theta(t), p(t))$,

$$\begin{aligned} p(t+\epsilon/2) &= p(t) + (\epsilon/2)\nabla_\theta \log \pi(\theta(t)), \\ \theta(t+\epsilon) &= \theta(t) + \epsilon M^{-1} p(t+\epsilon/2), \\ p(t+\epsilon) &= p(t+\epsilon/2) + (\epsilon/2)\nabla_\theta \log \pi(\theta(t+\epsilon)). \end{aligned}$$

For HMC, multiple leapfrog steps are typically required to move a sufficient distance to the next proposal. As with other valid MCMC algorithms, HMC's transition probability is designed to meet the theoretical requirements for detailed balance and reversibility. These conditions ensure that HMC samples provide a valid representation of the posterior distribution.

1.2.1 HMC algorithm

- Start at $\theta^{(0)}$
- For $i = 1, \ldots, M$
  - Draw $p^* \sim N(0, M)$
  - Set $\theta^* = \theta^{(i-1)}$
  - For $j = 1, \ldots, L$

$$(p^*, \theta^*) \leftarrow \text{leapfrog}(\theta^*, p^*, \epsilon, M)$$

  - Set $\theta^{(i)} = \theta^*$ with probability

$$\alpha = \min\left\{ \frac{\pi(\theta^*)}{\pi(\theta^{(i-1)})} \frac{\exp\{-0.5 p^{*T} M^{-1} p^*\}}{\exp\{-0.5 p^T M^{-1} p\}} \right\}$$

The function leapfrog(θ, p, ϵ, M) is a three-step routine that (i) takes (θ, p, ϵ, M) as input, (ii) runs the following three steps,

$$\begin{aligned} p &\leftarrow p + \frac{\epsilon}{2}\nabla_\theta \log \pi(\theta) \\ \theta &\leftarrow \theta + \epsilon M^{-1} p \\ p &\leftarrow p + \frac{\epsilon}{2}\nabla_\theta \log \pi(\theta) \end{aligned}$$

and (iii) produces (p, θ) as output.

For a more in-depth and comprehensive understanding of Hamiltonian Monte Carlo (HMC) schemes, the reader is referred to Neal (2011), Girolami and Calderhead (2011), Hoffman and Gelman (2014), and Betancourt (2017). Neal (2011) provides a foundational

and accessible introduction to HMC in a chapter of the Handbook of Markov Chain Monte Carlo, covering the theoretical background and practical implementation details. Girolami and Calderhead (2011) extend HMC by introducing Riemann manifold Hamiltonian Monte Carlo (RMHMC), which adapts the sampler to the local geometry of the parameter space to improve efficiency in complex models. Hoffman and Gelman (2014) propose the No-U-Turn Sampler (NUTS), a popular adaptive HMC variant that automatically selects the trajectory length (number of leapfrog steps) for each iteration, alleviating the need for manual tuning of this critical parameter. Betancourt (2017) offers a thorough theoretical analysis of HMC, discussing its geometric foundations, diagnostic techniques, and practical considerations for robust and efficient sampling.

1.3 Illustration

Let us assume a simple, but didactical context, where the posterior density $\pi(\theta)$ of interest, i.e. which one wish to be able to sample from efficiently is the following:

$$\pi(\theta) = \kappa \exp\left\{-0.5(\theta_1^2\theta_2^2 + \theta_1^2 + \theta_2^2 - 8\theta_1 - 8\theta_2\right\},$$

where κ is the proportionality constant.

```
post = function(theta){
  exp(-0.5*(prod(theta)^2+sum(theta^2)-8*sum(theta)))
}
theta1 = seq(-2,8,length=200)
theta2 = seq(-2,8,length=200)
posterior = matrix(0,200,200)
for (i in 1:200)
  for (j in 1:200)
    posterior[i,j] = post(c(theta1[i],theta2[j]))
contour(theta1,theta2,posterior,drawlabels=FALSE,xlab=expression(theta[1]),
            ylab=expression(theta[2]))
```

1.3.1 Random walk Metropolis

We could sample from π via rejection sampling or sampling importance resampling, as we extensively discussed in Sections xx and yy. However, for now let us assume that we are interested in a iterative scheme such as the random-walk Metropolis (RWM) algorithm.

More precisely, a generic RWM algorithm will start by sampling a proposal draw θ^* from a random walk proposal, $\theta^* \sim N(\theta^{(i)}, \Sigma)$, where Σ is the (local) variance of the proposal density. Then, the new draw, in the array of iterative draws, is accepted, i.e. $\theta^{(i+1)} = \theta^*$, with probability $\alpha = \min\{1, \pi(\theta^*)/\pi(\theta^{(i)})\}$, such that the previous draw, $\theta^{(i)}$ is repeated, i.e. $\theta^{(i+1)} = \theta^{(i)}$, with probability $1 - \alpha$.

In the above example, we will use $\Sigma = \sigma^2 I_2$, such that θ_1 and θ_2 are independently draws from Gaussian proposals.

```
logpost = function(theta){
  -0.5*(prod(theta)^2+sum(theta^2)-8*sum(theta))
}
set.seed(32425)
burn   = 10000
N      = 2000
lag    = 50
niter  = burn+lag*N
sigma  = 0.1
draws.rwm = matrix(0,niter,2)
theta = c(8,8)
for (iter in 1:niter){
  theta.new = rnorm(2,theta,sigma)
  logaccept = min(0,logpost(theta.new)-logpost(theta))
  if (log(runif(1))<logaccept){
    theta = theta.new
  }
  draws.rwm[iter,] = theta
}
draws.rwm = draws.rwm[seq(burn+1,niter,by=lag),]
```

1.4 HMC: log posterior and gradient

For the bivariate target density presented above, it can be easily shown that

$$\log \pi(\theta) = \log \kappa - 0.5\theta_1^2\theta_2^2 + 0.5\theta_1^2 + 0.5\theta_2^2 - 4\theta_1 - 4\theta_2,$$

and that the gradient of $\log \pi(\theta)$ is

$$\nabla_\theta \log \pi(\theta) = \begin{pmatrix} \frac{\partial \log \pi(\theta)}{\partial \theta_1} \\ \frac{\partial \log \pi(\theta)}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} -\theta_1\theta_2^2 + \theta_1 - 4 \\ -\theta_2\theta_1^2 + \theta_2 - 4 \end{pmatrix}.$$

1.4.1 R code

```
gradient = function(theta){
  c(-theta[1]*theta[2]^2+theta[1]-4,-theta[2]*theta[1]^2+theta[2]-4)
}

leapfrog = function(theta,p,eps,iM){
  p     = p     + (eps/2)*gradient(theta)
  theta = theta + eps*iM%*%p
  p     = p     +  (eps/2)*gradient(theta)
  return(list(theta=theta,p=p))
}

set.seed(123456)
```

```
M      = diag(2,2)
iM     = solve(M)
tcM    = t(chol(M))
L      = 5
eps    = 0.05
theta  = c(8,8)
draws.hmc = matrix(0,niter,2)
for (iter in 1:niter){
  p          = tcM%*%rnorm(2)
  p.star     = p
  theta.star = theta
  for (i in 1:L){
    run        = leapfrog(theta.star,p.star,eps,iM)
    p.star     = run$p
    theta.star = run$theta
  }
  term1 = logpost(theta.star)-logpost(theta)
  term2 = -0.5*t(p.star)%*%iM%*%p.star + 0.5*t(p)%*%iM%*%p
  logaccept = min(0,term1+term2)
  if (log(runif(1))<logaccept){
    theta = theta.star
  }
  draws.hmc[iter,]  = theta
}
draws.hmc = draws.hmc[seq(burn+1,niter,by=lag),]
```

1.4.2 Comparing RWM and HMC

```
pdf(file="ex1-rwm-hmc.pdf",width=7,height=7)
par(mfrow=c(2,2))
ts.plot(draws.rwm[,1],xlab="Iterations",ylab=expression(theta),main="RWM")
lines(draws.rwm[,2],col=2)
plot(draws.rwm,xlim=c(-2,8),ylim=c(-2,8),xlab=expression(theta[1]),
       ylab=expression(theta[2]))
contour(theta1,theta2,posterior,drawlabels=FALSE,add=TRUE,col=2)

ts.plot(draws.hmc[,1],xlab="Iterations",ylab=expression(theta),main="HMC")
lines(draws.hmc[,2],col=2)
plot(draws.hmc,xlim=c(-2,8),ylim=c(-2,8),xlab=expression(theta[1]),
       ylab=expression(theta[2]))
contour(theta1,theta2,posterior,drawlabels=FALSE,add=TRUE,col=2)
dev.off()
```

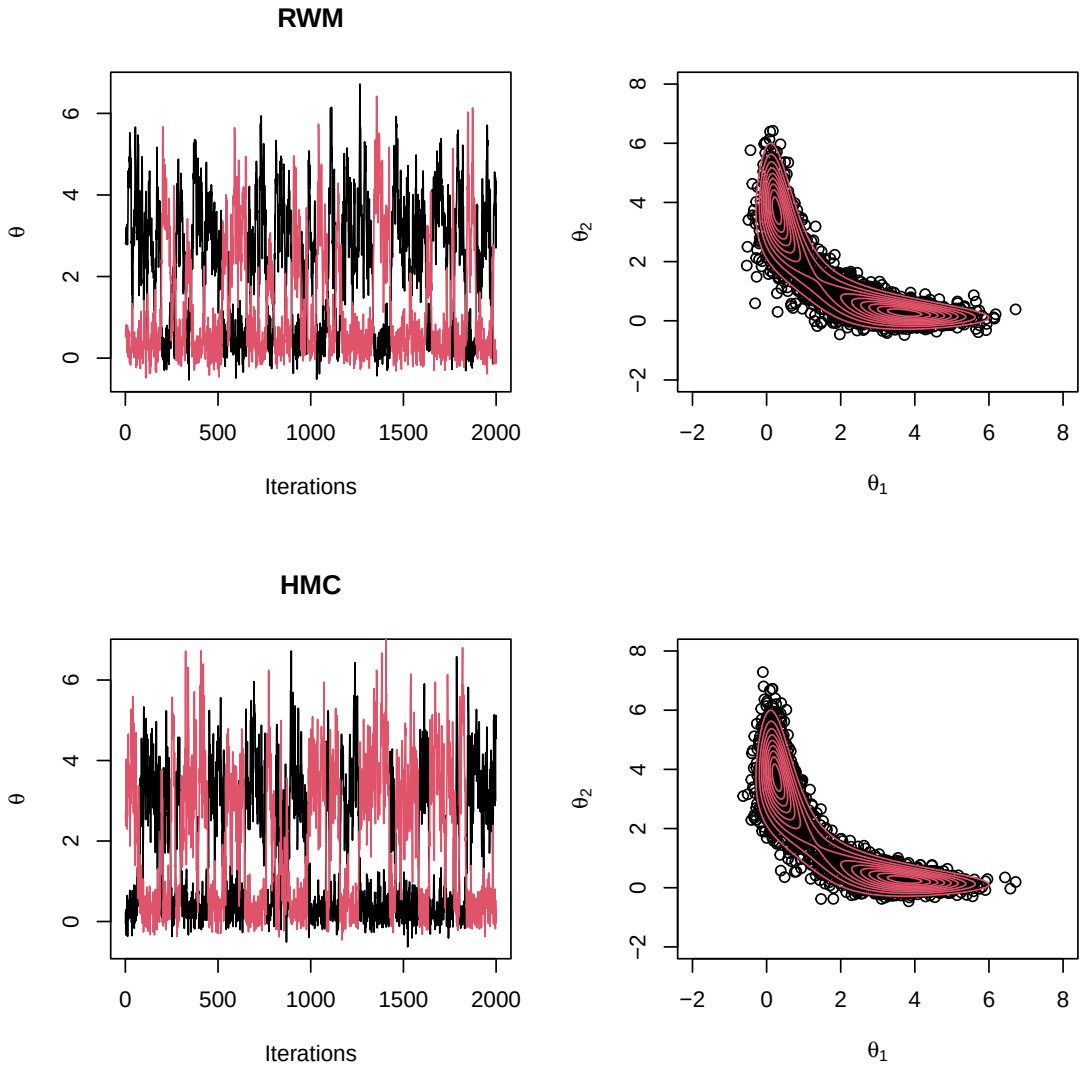

Figure 1: Comparing random-walk Metropolis and Hamiltonian Monte Carlo for the simple example.

2 Gaussian linear regression via HMC

To illustrate the derivation of the gradient functions and the implementation of HMC in a more realistic situation, let us consider the well known Bayesian normal linear regression, $y|X, \beta, \sigma^2 \sim N(X\beta, \sigma^2 I_n)$, where y is the n-dimensional vector of responses, X is the $(n \times p)$ matrix with regressors (the design matrix), and σ^2 is the residual variance of the model. The specification is completed with standard (independent) priors for β and σ^2, i.e. $\beta \sim N(0, \tau^2 I_p)$ and $\sigma^2 \sim IG(a, b)$, with known hyperparameters τ^2, a and b, which will be omitted from the conditioning from now on. In addition, we will consider $\gamma = \log \sigma^2$ for computational purpose. It is straightforward to show that the log posterior density (up to a

constant) is given by

$$\log p(\beta, \gamma | y, X) = -(n/2 + a)\gamma - 0.5e^{-\gamma}(y - X\beta)'(y - X\beta) - 0.5\beta'\beta/\tau^2 - be^{-\gamma},$$

while the gradient functions are given by

$$\begin{aligned} \nabla_\beta \log p(\beta, \gamma | y, X) &\propto e^{-\gamma} X'(y - X\beta) - \beta/\tau^2 \\ \nabla_\gamma \log p(\beta, \gamma | y, X) &\propto -(n/2 + a) + 0.5e^{-\gamma}(y - X\beta)'(y - X\beta) + be^{-\gamma}. \end{aligned}$$

We will also implement a standard Gibbs sampler, whose full conditionals are easily derived as $\beta | \sigma^2, y, X \sim N(b_1, B_1)$ and $\sigma^2 | \beta, y, X \sim IG(a_1, b_1)$, where $B_1^{-1} = X'X/\sigma^2 + I_p/\tau^2$, $b_1 = B_1 X'y/\sigma^2$, $a_1 = a + n/2$ and $b_1 = b + (y - X\beta)'(y - X\beta)/2$.

2.1 R code

```
logpost = function(theta){
  k = length(theta)-1
  beta = theta[1:k]
  gamma = theta[k+1]
  egamma = exp(-gamma)
  return(-(n2+a)*gamma-0.5*egamma*sum((y-X%*%beta)^2)-
        0.5*sum(beta^2)/tau2-b*egamma)
}
gradient = function(theta){
  k      = length(theta)-1
  beta   = theta[1:k]
  gamma  = theta[k+1]
  egamma = exp(-gamma)
  return(c(egamma*t(X)%*%(y-X%*%beta)-beta/tau2,
        -(n2+a)+0.5*egamma*sum((y-X%*%beta)^2)+b*egamma))
}
leapfrog = function(theta,p,eps,iM){
  p     = p     + (eps/2)*gradient(theta)
  theta = theta + eps*iM%*%p
  p     = p     +  (eps/2)*gradient(theta)
  return(list(theta=theta,p=p))
}
```

2.2 Data and ML estimation

We implement both Gibbs sampler and HMC for the data from Tomas and Tu (2020), where the response is the number of warp breaks per loom and the explanatory variables are the yarn's type of wool and levels of tension. The variable `wool` is a categorical variable with levels A and B, while `tension` is a categorical variable with levels L, M and H. The regression includes interaction between type of wool and levels of tension as well, so there are $k = 5$ regressors. We do not include an intercept as we standardized both response and explanatory variables.

```
names = c("woolB","tensionM","tensionH","woolB*tensionM","woolB*tensionH")
data(warpbreaks)
attach(warpbreaks)

n        = nrow(warpbreaks)
woolB    = rep(0,n)
tensionM = rep(0,n)
tensionH = rep(0,n)
woolB[wool=="B"]       = 1
tensionM[tension=="M"] = 1
tensionH[tension=="H"] = 1
X = cbind(woolB,tensionM,tensionH,woolB*tensionM,woolB*tensionH)
k = ncol(X)
X = scale(X)
y = scale(breaks)

# MLE
beta.hat = solve(t(X)%*%X)%*%t(X)%*%y
sig.hat  = sqrt(mean((y-X%*%beta.hat)^2))
```

2.3 Gibbs sampler

```
# Bayesian
a   = 1
b   = 1
tau = 1
iD  = diag(1/tau^2,k)
Xty = t(X)%*%y
XtX = t(X)%*%X

# Gibbs sampler
niter = 10000
draws = matrix(0,niter,k+1)
beta  = rep(0,k)
for (i in 1:niter){
  sig2 = 1/rgamma(1,a+n/2,b+sum((y-X%*%beta)^2)/2)
  B1   = solve(XtX/sig2+iD)
  b1   = B1%*%Xty/sig2
  beta = b1 + t(chol(B1))%*%rnorm(k)
  draws[i,] = c(beta,sqrt(sig2))
}

par(mfrow=c(2,6))
for (i in 1:k){
  ts.plot(draws[,i],main=names[i],xlab="Iterations",ylab="")
```

```
  abline(h=beta.hat[i],col=2,lwd=2)
  abline(h=0,lwd=2,col=3)
}
ts.plot(draws[,k+1],main="sigma",xlab="Iterations",ylab="")
abline(h=sig.hat,col=2,lwd=2)

for (i in 1:k){
  hist(draws[,i],main=names[i],xlab="",prob=TRUE)
  abline(v=beta.hat[i],lwd=2,col=2)
  points(0,0,pch=16,cex=1.5,col=3)
}
hist(draws[,k+1],main="sigma",xlab="",prob=TRUE)
abline(v=sig.hat,col=2,lwd=2)
```

2.4 HMC

```
set.seed(123456)
tau2   = tau^2
n2     = n/2
q      = k + 1
M      = diag(2,q)
iM     = solve(M)
tcM    = t(chol(M))
L      = 5
eps    = 0.05
theta  = c(beta.hat,2*log(sig.hat))
draws.hmc = matrix(0,niter,q)
for (iter in 1:niter){
  p          = tcM%*%rnorm(q)
  p.star     = p
  theta.star = theta
  for (i in 1:L){
    run        = leapfrog(theta.star,p.star,eps,iM)
    p.star     = run$p
    theta.star = run$theta
  }
  term1 = logpost(theta.star)-logpost(theta)
  term2 = -0.5*t(p.star)%*%iM%*%p.star + 0.5*t(p)%*%iM%*%p
  logaccept = min(0,term1+term2)
  if (log(runif(1))<logaccept){
    theta = theta.star
  }
  draws.hmc[iter,]  = theta
}
```

```
par(mfrow=c(2,6))
for (i in 1:k){
  ts.plot(draws.hmc[,i],main=names[i],xlab="Iterations",ylab="")
  abline(h=beta.hat[i],col=2,lwd=2)
  abline(h=0,lwd=2,col=3)
}
ts.plot(draws.hmc[,k+1],main="sigma",xlab="Iterations",ylab="")
abline(h=sig.hat,col=2,lwd=2)

for (i in 1:k){
  hist(draws.hmc[,i],main=names[i],xlab="",prob=TRUE)
  abline(v=beta.hat[i],lwd=2,col=2)
  points(0,0,pch=16,cex=2)
}
hist(draws.hmc[,k+1],main="sigma",xlab="",prob=TRUE)
abline(v=sig.hat,col=2,lwd=2)
```

2.5 Comparing Gibbs sampler and HMC

```
pdf(file="ex2-gibbs-hmc-1.pdf",width=8,height=10)
par(mfrow=c(3,2))
ts.plot(draws[5001:niter,2],ylim=c(-1.5,0.1),xlab="Iterations",ylab="",main="Gibbs sampl
ts.plot(draws.hmc[5001:niter,2],ylim=c(-1.5,0.1),xlab="Iterations",ylab="",main="Hamilto
acf(draws[5001:niter,2],main="")
acf(draws.hmc[5001:niter,2],main="")
hist(draws[5001:niter,2],main="",prob=TRUE,xlab="",xlim=c(-1.5,0.1))
hist(draws.hmc[5001:niter,2],main="",prob=TRUE,xlab="",xlim=c(-1.5,0.1))
dev.off()

pdf(file="ex2-gibbs-hmc.pdf",width=8,height=6)
par(mfrow=c(2,3))
for (i in 1:k){
  plot(density(draws[,i]),type="l",xlab="",main=names[i],lwd=2)
  lines(density(draws.hmc[,i]),col=2,lwd=2)
}
plot(density(draws[,k+1]),type="l",xlab="",main="sigma",lwd=2)
lines(density(exp(draws.hmc[,k+1]/2)),col=2,lwd=2)
legend("topright",legend=c("Gibbs","HMC"),col=1:2,lwd=2,bty="n")
dev.off()
```

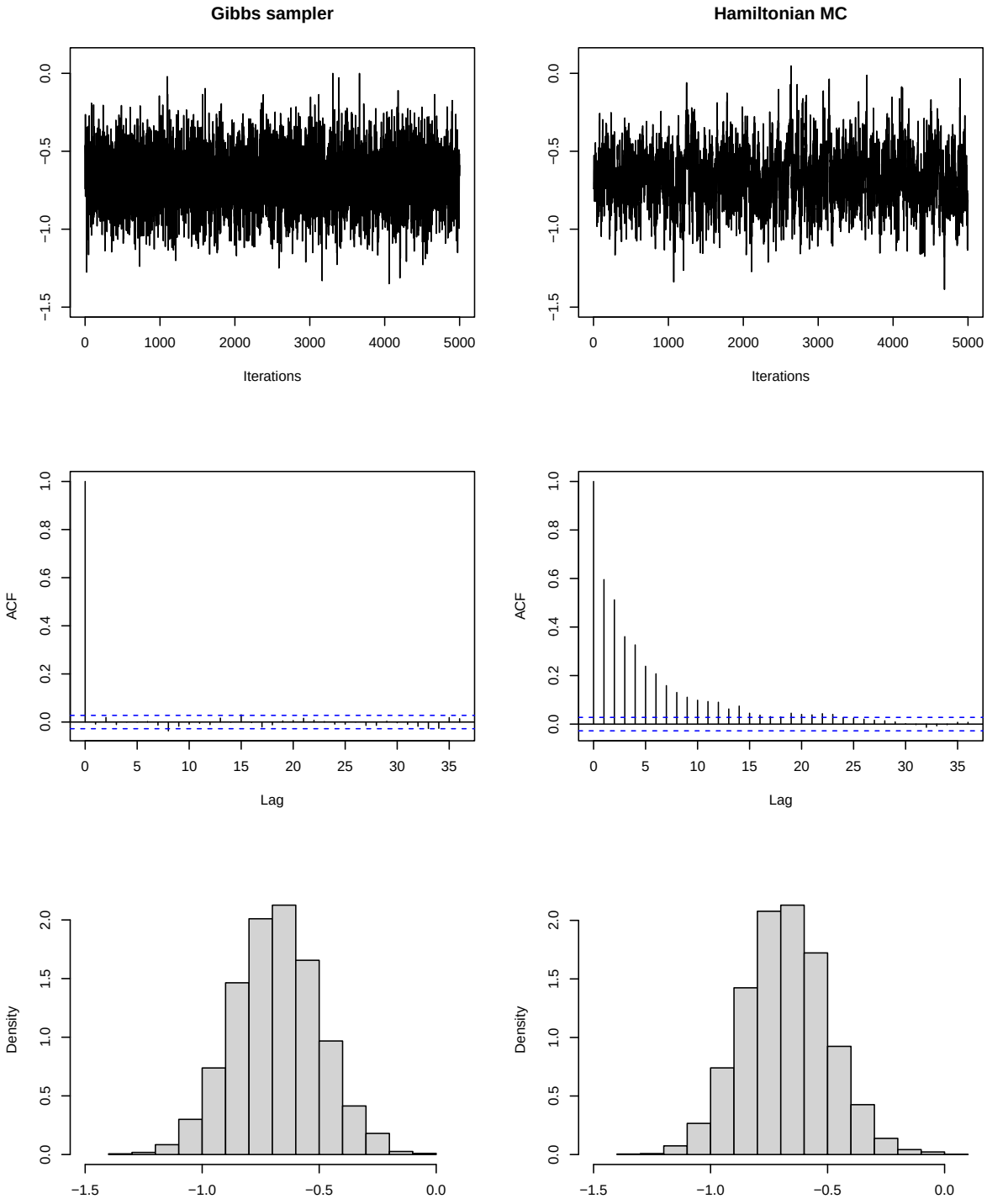

Figure 2: Draws for the coefficient β_1 from both Gibbs sampler and HMC.

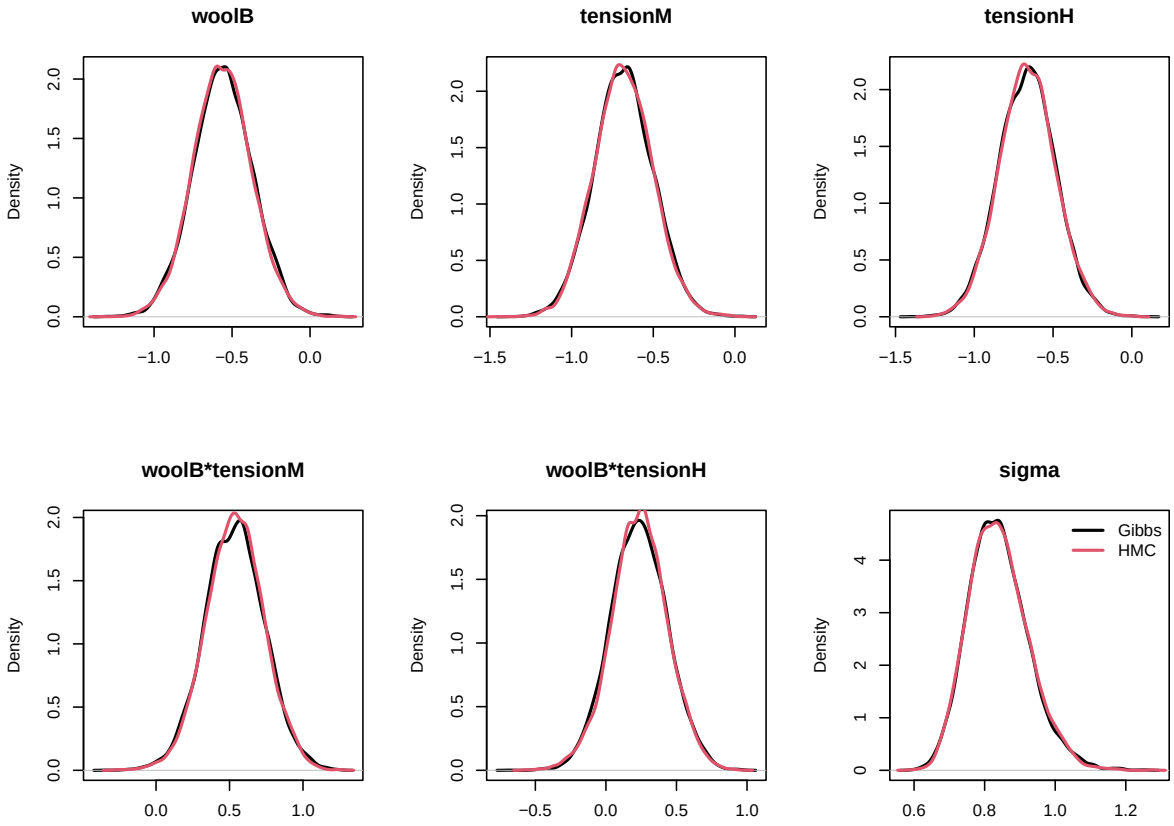

Figure 3: Comparing Gibbs sampler and Hamiltonian Monte Carlo for the Gaussian linear regression example.

3 Reference

Betancourt, M. (2017) A Conceptual Introduction to HMC. `https://arxiv.org/abs/1701.02434`

Girolami, M. and Calderhead, B. (2011) Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods, *Journal of the Royal Statistical Society Series B: Statistical Methodolog*, Volume 73, Issue 2, Pages 123-214. `https://doi.org/10.1111/j.1467-9868.2010.00765.x`

Hoffman, M. D. and Gelman, A. (2014) The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo, *Journal of Machine Learning Research*, 15(47), pages 1593-1623. `https://jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf`

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. L. Jones, & X.-L. Meng (Eds.), *Handbook of Markov Chain Monte Carlo*, 113-162. Chapman

& Hall/CRC Press.

Thomas, S. and Tu, Wanzhu (2021) Learning Hamiltonian Monte Carlo in R *The American Statistician*, Volume 75, Issue 4, Pages 403-413. https://doi.org/10.1080/00031305.2020.1865198