HMC in R

```
library(hmclearn)
library(MASS)
```

Introduction

Hamiltonian Monte Carlo (HMC) has emerged as a general purpose tool for Bayesian practitioners. A key advantage of HMC over more traditional Markov Chain Monte Carlo (MCMC) algorithms is its improved computational efficiency in fitting high-dimensional models. While the algorithm itself is not difficult to program, the substantial number of tuning parameters can be daunting to those unfamiliar with the theory behind the method. Until recently, practical access to HMC was limited to individuals with both the mathematical background to understand the algorithm and the programming skill to implement the simulation in a high-performance environment.

Modern Bayesian software such as Stan has made HMC accessible to practitioners who are comfortable with any one of a variety of well-known programming platforms (e.g. R, Python, Matlab). The Stan language is similar in style to WinBUGS, which is familiar to many Bayesian statisticians. The software translates Stan code to a lower-level language to maximize speed and efficiency. In addition, Stan automates the challenging process of tuning the many parameters in HMC. As a result, Stan has succeeded in making HMC accessible to many Bayesian practitioners around the world in both academia and industry.

While Stan and other high-performance software (e.g. PyMC, Edward) provide enormous practical value to analysts, the intuition of how HMC works can be lost in the process of fitting models. HMC can appear to be an opaque, "black-box" algorithm behind the sophisticated automation. This is an unfortunate consequence. While understanding HMC is not necessary to use production software, intuition behind the method can be helpful both in fine-tuning the simulation process and in instilling confidence in the results.

The purpose of this paper is to introduce HMC to analysts using R software only, an open-source statistical environment that is familiar to many. While many excellent introductions to HMC are available on a

conceptual level, this paper will focus on learning HMC by doing. Familiarity with popular MCMC algorithms such as Metropolis-Hastings (MH) is helpful, but not required. A companion R package called hmclearn contains the R code for all of the functions used in this introduction is freely available to download.

MCMC Basic Concepts

We consider n observations from a simple random sample $\boldsymbol{X} = (X_1, ..., X_n)$, where each element is independent and identically distributed (iid). From this sample, we want to fit the distribution of our k-dimensional parameter of interest $\Theta = (\theta_1, ..., \theta_k)$. The posterior distribution $p(\Theta|\boldsymbol{X})$ can be written as a function of the Likelihood $p(\boldsymbol{X}|\Theta)$ and prior $p(\Theta)$ using Bayes formula.

$$\begin{aligned} p(\Theta|\boldsymbol{X}) &= \frac{p(\boldsymbol{X}|\Theta)p(\Theta)}{\int p(\boldsymbol{X}|\Theta)p(\Theta)d\Theta} \\ &\propto p(\boldsymbol{X}|\Theta)p(\Theta) \end{aligned}$$

In many practical data analyses, the integral in the denominator cannot be evaluated directly. Since the denominator is constant with respect to Θ, only the unnormalized posterior $p(\boldsymbol{X}|\Theta)$ is available.

Metropolis-Hastings

The first widely-used MCMC method capable of simulating directly from $p(\boldsymbol{X}|\Theta)$ is called the Metropolis algorithm, originating in the 1950's from an application to statistical physics. Nearly two decades later, Hastings generalized the algorithm, which is now called Metropolis-Hastings (MH). We begin with a brief introduction to MH since HMC builds on many similar concepts.

The objective of MH is to simulate values of Θ that accurately reflect the posterior density $p(\Theta|\boldsymbol{X})$. For brevity, we will shorten our notation of the posterior as $p(\Theta)$. The Markov chain simulates values from this density in sequence from $t = 1, .., N$, provided some starting point $\Theta^{(0)}$ which is typically provided by the analyst or the computer program.

MH defines a transition probability that produces a Markov chain that is ergodic and satisfies detailed balance. Values of $\Theta^{(t)}$ in the chain are defined in part by a proposal density, which we define as $q(\Theta^{\text{PROP}}|\Theta^{t-1})$. Here, Θ^{PROP} is a proposal for the next value in the chain. This proposal density is conditioned on the previously stored value $\Theta^{(t-1)}$. A variety of proposal functions can be used, with random walk proposals being a common choice. We now outline the MH algorithm in full.

Each proposal in MH is accepted at probability

Algorithm 1 Metropolis-Hastings

procedure MH($\theta^{(0)}, \pi^*(\theta), q(\theta^{(x)}|\theta^{(y)}), N$)
  Calculate $\pi^*(\theta^{(0)})$ ▷ Initial value for posterior
  for $t = 1, ..., N$ **do** ▷ Repeat simulation N times
    $\theta^{\text{PROP}} \leftarrow q(\theta^{\text{PROP}}|\theta^{(t-1)})$ ▷ Randomly sample proposal
    $u \leftarrow U(0, 1)$ ▷ Randomly sample from a uniform density, 0 to 1
    $\alpha = \min\left(1, \frac{\pi(\theta^{\text{PROP}})q(\theta^{\text{PROP}}|\theta^{(t-1)})}{\pi(\theta^{(t-1)})q(\theta^{(t-1)}|\theta^{\text{PROP}})}\right)$ ▷ Calculate acceptance proposal probability
    If $\alpha < u$, then $\theta^{(t)} \leftarrow \theta^{\text{PROP}}$. Otherwise, $\theta^{(t)} \leftarrow \theta^{(t-1)}$ ▷ Select proposal or previous value
  end for
  return $\theta^{(1)}...\theta^{(N)}$ ▷ Return simulated values of θ from the unnormalized posterior
end procedure

$$\alpha = \min\left(1, \frac{p(\Theta^{\text{PROP}})q(\Theta^{\text{PROP}}|\Theta^{(t-1)})}{p(\Theta^{(t-1)})q(\Theta^{(t-1)}|\Theta^{\text{PROP}})}\right)$$

which simplifies when $q(\Theta^{PROP}|\Theta^{(t-1)})$ is symmetric (i.e. the Metropolis algorithm)

$$\alpha = \min\left(1, \frac{p(\Theta^{\text{PROP}})}{p(\Theta^{(t-1)})}\right).$$

Recall that the denominator of the posterior is constant with respect to Θ. As such, the ratio of posterior densities at two different points Θ^{PROP} and $\Theta^{(t-1)}$ can be formulated even when the denominator is unknown (i.e. the constants in the denominator cancel).

Intuition into why MH works can be obtained by examining the acceptance ratio α closely. Two different outcomes are possible depending on the value of the posterior at the proposed Θ^{PROP}:

1. If $p(\Theta^{PROP}) \geq p(\Theta^{(t-1)})$, then the posterior has a higher density at the proposed value of Θ than at the previous point in the chain $t-1$. When this occurs, the proposal is always accepted (i.e. at probability 1).

2. If $p(\Theta^{PROP}) < p(\Theta^{(t-1)})$, then the posterior has a lower density at the proposed value of Θ than at the previous point in the chain. When this occurs, we accept the proposal at random based on the ratio $0 < \alpha < 1$. If the proposal is not accepted, then the proposal is discarded and the Markov chain remains in place $\Theta^t := \Theta^{(t-1)}$.

As such, MH tends to sample more points in the region of higher posterior values. However, the tails of the posterior are also sampled based on acceptance ratio. Given enough samples, the MCMC chain samples Θ at the proportion of the true posterior density. The resulting simulated values can then be used for statistical inference. Much more can be said regarding MH. Interested readers can refer to the following references ...

(list references here)

Limitations of Metropolis-Hastings

The theoretical requirements for fitting models using MH are minimal, making MH an attractive choice for Bayesian inference even today. The limits of MH are primarily computational in nature. Since the proposals of Θ are randomly selected, many simulations are required to accurately describe the true posterior. Even efficient MH implementations may accept less than 25% of the proposals (cite Gelman).

The limited efficiency of MH can be overcome by high computational power for certain applications. When the dimensionality of the data is small to moderate, a well-programmed MH algorithm can sample enough points from the posterior density in a reasonable period of time. The challenge of relying on MH occurs when dealing with high-dimensional data or complex statistical models. In these situations, MH is known to be inefficient (reference here) and can be impractical for such applications.

A popular, often efficient alternative to MH is Gibbs Sampling (footnote that Gibbs is a particular case of MH?). Gibbs is widely used in many Bayesian software platforms such as WinBUGS and JAGS. When the conditional posterior densities can be explicitly formulated, Gibbs remains a viable choice for the Bayesian practitioner. Such restrictions limit the application of Gibbs to particular combinations of models and priors. Gibbs therefore lacks the flexibility of MH, in addition to having certain other efficiency limitations of its own (cite Robert).

Given the modern computational demands of large dataset and complex models, a more efficient MCMC algorithm is desirable. Ideally, such an algorithm would retain the theoretical advantages and flexibility of MH, while providing a more computational efficient method of selecting proposals. Here we transition to the modern HMC algorithm, and why HMC has emerged as a standard inferential tool for many Bayesian practitioners.

HMC Background

MH and HMC are equally flexible in their theoretical capabilities to fit a variety of model and prior specifications. The key advantage of HMC over MH is the use of additional information from the posterior to guide proposals. HMC uses the gradient of the log posterior to direct the Markov chain to the region the region of highest posterior density, where most of the samples should occur. In contrast, MH relies entirely on the acceptance ratio to guide the chain. As a result, a well-tuned HMC chain will accept proposals approximately 3 times the frequency of a similarly well-tuned MH algorithm (footnote?: based on MH

theoretical optimal acceptance rate of 0.234 and HMC acceptance of 0.6-0.9).

It should be noted that HMC is a MCMC simulation method, not an optimization method. While the HMC algorithm guides proposals to regions of high density (sometimes called the typical set Betancourt), the tails of the density are properly sampled as well. Both MH and HMC produce ergodic Markov chains, but the mathematics of HMC is substantially more complex than MH. We provide a brief overview of the theoretical basis of HMC here, and refer to other sources for more detailed expositions (refer to Wiley paper, Betancourt).

Hamiltonian Monte Carlo Concepts and Theory

As with MH, our objective in HMC is to simulate the posterior $p(\Theta|\boldsymbol{X})$. The mathematical basis of HMC is the Hamiltonian function

$$\begin{aligned} H(\boldsymbol{p}, \Theta) &= K(\boldsymbol{p}, \Theta) + U(\Theta) \\ &= -\log p(\boldsymbol{p}) - \log p(\Theta). \end{aligned}$$

HMC introduces a latent parameter $\boldsymbol{p} = (p_1, ..., p_k)$ of the same length k as the parameter of interest $\Theta = (\theta_1, ..., \theta_k)$. The latent parameter $\boldsymbol{p}$ is often called the *momentum* based on its original application to physical laws of motion. The incorporation of the momentum provides the geometrical structure of the space through which the Markov chain travels. The purpose of the momentum is to ensure that the MCMC simulation is ergodic, covering the entire space of Θ.

The distribution of $\boldsymbol{p}$ is most often specified to be multivariate Normal with covariance M. The structure of M is often diagonal, but does not have to be.

$$\begin{aligned} \boldsymbol{p} &\sim N_k(0, M) \\ \log p(\boldsymbol{p}) &\propto \frac{1}{2}\boldsymbol{p}^T M^{-1}\boldsymbol{p} \end{aligned}$$

The Hamiltonian function represents the joint distribution of the multivariate Normal $\boldsymbol{p}$ and the log posterior $p(\Theta|\boldsymbol{X})$, whose notation we simplify to $p(\Theta)$.

$$H(\boldsymbol{p}, \Theta) = -\frac{1}{2}\boldsymbol{p}^T M^{-1}\boldsymbol{p} - \log p(\Theta)$$

The trajectories over which HMC travels in time are defined by first-order differential equations,

$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H(\Theta, \boldsymbol{p})}{\partial \Theta} = -\frac{\partial U(\Theta)}{\partial \Theta} = \nabla_\Theta \log p(\Theta)$$
$$\frac{d\Theta}{dt} = \frac{\partial H(\Theta, \boldsymbol{p})}{\partial \boldsymbol{p}} = \frac{\partial K(\boldsymbol{p})}{\partial \boldsymbol{p}} = M^{-1}\boldsymbol{p}$$

Here we see the explicit formulation of the gradient of the log posterior $\nabla_\Theta \log p(\Theta)$. The combination of these two equations forms deterministic paths for the MCMC chain to travel through the joint distribution $(\boldsymbol{p}, \Theta)$. Note that while the MCMC produces simulations of both $\boldsymbol{p}$ and Θ, only the values of Θ are of interest for statistical inference.

A solution for these differential equations is necessary to produce a practical MCMC simulation. Since no exact solution exists, a discrete approximation is needed to form the algorithm. The most commonly used solution is called the leapfrog method. The leapfrog defines a discrete step size ϵ individually for $\boldsymbol{p}$ and Θ.

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

The full step ϵ in Θ is sandwiched by half-steps $\epsilon/2$ for $\boldsymbol{p}$. The number of steps ϵ in an actual HMC algorithm is typically defined by a tuning parameter $L > 1$. This leapfrog approximation, though slightly more complex than Euler approximations, provides a more accurate approximation of the solution over the many samples that HMC requires.

The paths defined by the Hamiltonian equations are deterministic once $\boldsymbol{p}$ is defined. An exact solution would always be accepted in a MCMC algorithm. However, since the solution to the Hamiltonian equations is an approximation, a Metropolis-Hastings style acceptance ratio is used to formally determine whether a proposal is accepted or rejected.

With the overview of the major concepts in HMC complete, we turn to the formulation of the Hamiltonian Monte Carlo algorithm itself.

Hamiltonian Monte Carlo Algorithm

The HMC algorithm is presented in full here.

One may notice some similarities with MH:

1. An initial set of values $\Theta^{(0)}$ is required for both MH and HMC.

Algorithm 2 Euclidean Hamiltonian Monte Carlo

procedure EHMC($\Theta^{(0)}, \log \pi(\Theta), M, N, \epsilon, L$)
  Calculate $\log \pi(\Theta^{(0)})$ ▷ Initial value for log posterior
  for $t = 1, ..., N$ **do** ▷ Repeat simulation N times
    $p^0 \leftarrow N(0, M)$ ▷ Randomly sample momentum from MVN
    $\Theta^{(t)} \leftarrow \Theta^{(t-1)}, \tilde{\Theta} \leftarrow \Theta^{(t-1)}, \tilde{p} \leftarrow p^{(0)}$ ▷ Randomly sample from a uniform density, 0 to 1
    for $i = 1, ..., L$ **do** ▷ Run Leapfrog L times
      $\tilde{\Theta}, \tilde{p} \leftarrow \text{Leapfrog}(\tilde{\Theta}, \tilde{p}, \epsilon)$
    end for
    $\alpha = \min\left(1, \frac{\exp(\log \tilde{\Theta}) - \frac{1}{2}\tilde{p} \cdot \tilde{p}}{\exp(\log \tilde{\Theta}^{(t-1)}) - \frac{1}{2} p^0 \cdot p^0}\right)$ ▷ Calculate acceptance proposal probability
    With probability α, $\Theta^{(t)} \leftarrow \tilde{\Theta}$ and $p^{(t)} \leftarrow -\tilde{p}$
  end for
  return $\Theta^{(1)} ... \Theta^{(N)}$ ▷ Return simulated values of Θ from the unnormalized posterior
  function LEAPFROG(Θ, p, ϵ)
    $\tilde{p} \leftarrow p + (\epsilon/2)\nabla_\Theta \log \pi(\Theta)$
    $\tilde{\Theta} \leftarrow \Theta + \epsilon \tilde{p}$
    $\tilde{p} \leftarrow \tilde{p} + (\epsilon/2)\nabla_\Theta \log \pi(\Theta)$
    return $\tilde{\Theta}, \tilde{p}$
  end function
end procedure

2. The random walk Metropolis proposal is often Multivariate Normal. HMC simulates from the momentum $\boldsymbol{p}$ from a Multivariate Normal.

3. Proposals are accepted or rejected based on a ratio of log posteriors in both MH and HMC.

The key functional difference between the two algorithms is the leapfrog update process. HMC generates a proposal $(\boldsymbol{p}^t, \Theta^t)$ based on the randomly selected values of $\boldsymbol{p}$ and the number of leapfrog steps L. In contrast, MH selects a proposal from one simple randomly selected value from a proposal distribution. MH is therefore computationally simpler, but undirected. HMC's proposal combines the randomization provided by $\boldsymbol{p}$ with trajectories defined the by the gradient of the log posterior $\nabla_\Theta \log \pi(\Theta(t))$.

From an implementation standpoint, the additional programming required for HMC is fairly minimal. Manually determiming the gradient may be tedious, but the derivation is typically not difficult. Once the gradient is provided and coded, programming the leapfrog loop itself is straightforward.

The challenge of practical HMC implementation is related to implementation of MH - tuning. While tuning MH involves adjusting a proposal density and little else, HMC requires the selection and adjustment of the mass matrix M, number of leapfrog steps L, and step size ϵ.

While M can be fixed as an identity matrix, some modification of the diagonal, at a minimum, can provide notable improvements in computational efficiency. The selection of L is also an essential step in HMC implementation. Selecting L too low will produce a Markov chain that is inefficient, and may behave in

pattern similar to random walk Metropolis, but with the penalty of additional computation. Setting L too high can also be inefficient, where the chain may travel through the entire deterministic trajectory multiple times before a proposal is selected. Finally, ϵ can be set as a single constant for all k parameters, but varying ϵ for each individual parameter may improve computational efficiency.

Developing an intuition and appreciation for how HMC works can be challenging without direct experience working with the algorithm. The next section provides R code and some simple examples for the interested reader to gain practical, hands-on experience with implementing and tuning HMC.

HMC Implementation

We introduce how to use the *hmcR* package in this section. The software is designed with the flexibility to fit a wide variety of types of statistical models and priors with HMC. General requirements are described here. Additional detail for interested readers is provided in the Appendix.

Main function: *hmc*

The function *hmc* runs the main HMC algorithm. Many of the parameters have defaults to the most frequently used values. However, any of the defaults can be overriden with a custom specification.

- N: number of MCMC samples. Default is 10,000
- *theta.init*: initial values for model parameters
- *epsilon*: Default 0.01. stepsize tuning parameter for HMC
- L: Default 10. number of Leapfrog steps tuning parameter for HMC
- *logPOSTERIOR*: function to return the log posterior depending on Θ, p, and data provided in ....
- *glogPOSTERIOR*: function to return the gradient of the log posterior depending on Θ, p, and data provided in ....
- *varnames*: optional vector of variable names in the model
- *randlength*: default FALSE. Boolean value on whether to apply some randomness to the number of Leapfrog steps L
- *Mdiag*: optional vector for the diagonal of the Mass matrix M
- *constrain*: optional vector of which variables are bounded as positive only.

- *verbose*: default FALSE. Boolean on whether to print status updates of *hmc*
- *...*: Parameters, including data, passed to *logPOSTERIOR* and *glogPOSTERIOR*

The output of *hmc* is a single list containing the model results. The most important values provided in the output are the simulated values *thetaDF* and number of accepted proposals *accept*.

One might notice that none of the above parameters explicitly specify the data or priors. In this package, the data is passed to the functions indicated with the *logPOSTERIOR* and *glogPOSTERIOR* parameters via the *...* parameter. Priors should be defined either within the posterior functions or in separate function calls. The purpose of this design is to maximize flexibility regarding the types of models to fit using this package.

The default number of simulations is set to 10,000. However, the user may elect to start with a smaller number during tuning. Initial values for the parameters must be provided by the user. Caution should be used to ensure that these initial values are in the support of Θ.

Defaults are provided for the stepsize ϵ and leapfrog steps L. These likely will need to be adjusted depending on the particular application. One simple indicator of whether HMC is sampling correctly is the acceptance rate of the MCMC chain. In general, acceptance rates between 60% and 95% are sufficient for this algorithm.

The acceptance rate can be calculated from the *hmc* function from the number of accepted proposals divided by the number of samples *N*. Both of these values are included in the output of *hmc*.

The MCMC simulation values are provided in list and dataframe forms. In most cases, the dataframe *thetaDF* will be most convenient for exploring posterior simulation values. The number of columns in the dataframe is equal to the number of parameters defined in the model.

Simple example

To illustrate the functionality of the HMC software, we begin with a simple example fitting a Gamma distribution with two parameters α and β. The pdf is specified

$$f(x) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} I_x(0, \infty)$$

with likelihood and log likelihood

$$
\begin{aligned}
L(\alpha, \beta; x) &= \prod_{i=1}^{n} \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x_i^{\alpha-1} e^{-x_i/\beta} \\
l(\alpha, \beta; x) &= \sum_{i=1}^{n} -\alpha \log \beta - \log \Gamma(\alpha) + (\alpha - 1) \log x_i - x_i/\beta \\
&= -n\alpha \log \beta - n \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \frac{1}{\beta} \sum_{i=1}^{n} x_i.
\end{aligned}
$$

Now that the log likelihood has been specified, we can specify our priors. Both parameters α and β are strictly positive. As such, we can specify half-Normal priors for each of the parameters, with hyperpriors η_1 and η_2.

$$
\begin{aligned}
p(\alpha|\eta_1) &= \frac{2\eta_1}{\pi} \exp\left(-\frac{\alpha^2\eta_1^2}{\pi}\right) \\
p(\beta|\eta_2) &= \frac{2\eta_2}{\pi} \exp\left(-\frac{\beta^2\eta_2^2}{\pi}\right)
\end{aligned}
$$

The log posterior is the sum of the log likelihood and the log prior.

$$
\begin{aligned}
\log p(\alpha, \beta|x) &\propto l(\alpha, \beta; x) + \log p(\alpha, \beta) \\
&\propto -n\alpha \log \beta - n \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \frac{1}{\beta} \sum_{i=1}^{n} x_i - \frac{\alpha^2\eta_1^2}{\pi} - \frac{\beta^2\eta_2^2}{\pi}
\end{aligned}
$$

Note that we disregard any terms that are not dependent on our parameters of interest α and β. These terms are absorbed into the normalizing constant and do not impact the gradient calculation.

The R functions for the log likelihood and log posterior are provided here. For our application, our parameter of interest is defined $\Theta := (\alpha, \beta)$.

```r
# log likelihood of gamma
llgamma <- function(theta, X, y=NULL, Z=NULL) {
  alpha <- theta[1]
  beta <- theta[2]
  n <- length(X)
  -n*alpha*log(beta) - n*log(gamma(alpha)) + (alpha-1)*sum(log(X)) - sum(X)/beta
}

# log posterior of gamma
```

```
gamma.lposterior <- function(theta, X, eta1, eta2, y=NULL, Z=NULL) {
  alpha <- theta[1]
  beta <- theta[2]
  llgamma(theta, X) - beta^2 * eta1^2/pi - alpha^2 * eta2^2/pi
}
```

Since our model has two parameters, the partial derivatives must be calculated with respect to α and β.

$$\nabla_\alpha \log p(\alpha, \beta|x) = -n \log \beta - n\psi(\alpha) + \sum_{i=1}^{n} \log x_i - 2\alpha\eta_1^2/\pi$$

$$\nabla_\beta \log p(\alpha, \beta|x) = -\frac{n\alpha}{\beta} + \frac{1}{\beta^2} \sum_{i=1}^{n} x_i - 2\beta\eta_2^2/\pi$$

The function for the gradient of the log posterior is provided here.

```
# derivative of posterior
g.gamma.lposterior <- function(theta, X, eta1, eta2) {
  alpha <- theta[1]
  beta <- theta[2]
  n <- length(X)
  dalpha <- -n*log(beta) -n*digamma(alpha) + sum(log(X)) - 2*alpha*eta1^2/pi
  dbeta <- -n*alpha/beta + sum(X)/beta/beta - 2*beta*eta2^2/pi
  c(dalpha, dbeta)
}
```

Now that the log posterior and gradient have been derived and coded in R, we can run the *hmc* function to fit the model.

First, we simulate data based on a gamma distribution. Here, we simulate 1000 data points with $\alpha = 2$ and $\beta = 3$.

```
# simulate data
set.seed(312)
X <- rgamma(1000, 2, 1/3)
```

Next, we run our *hmc* function on our simulated data. For this example, we set the initial values $\Theta^{(0)}$ sufficiently away from zero with pre-selected parameters ϵ and L. This is to prevent the MCMC chain from moving to values less than zero.

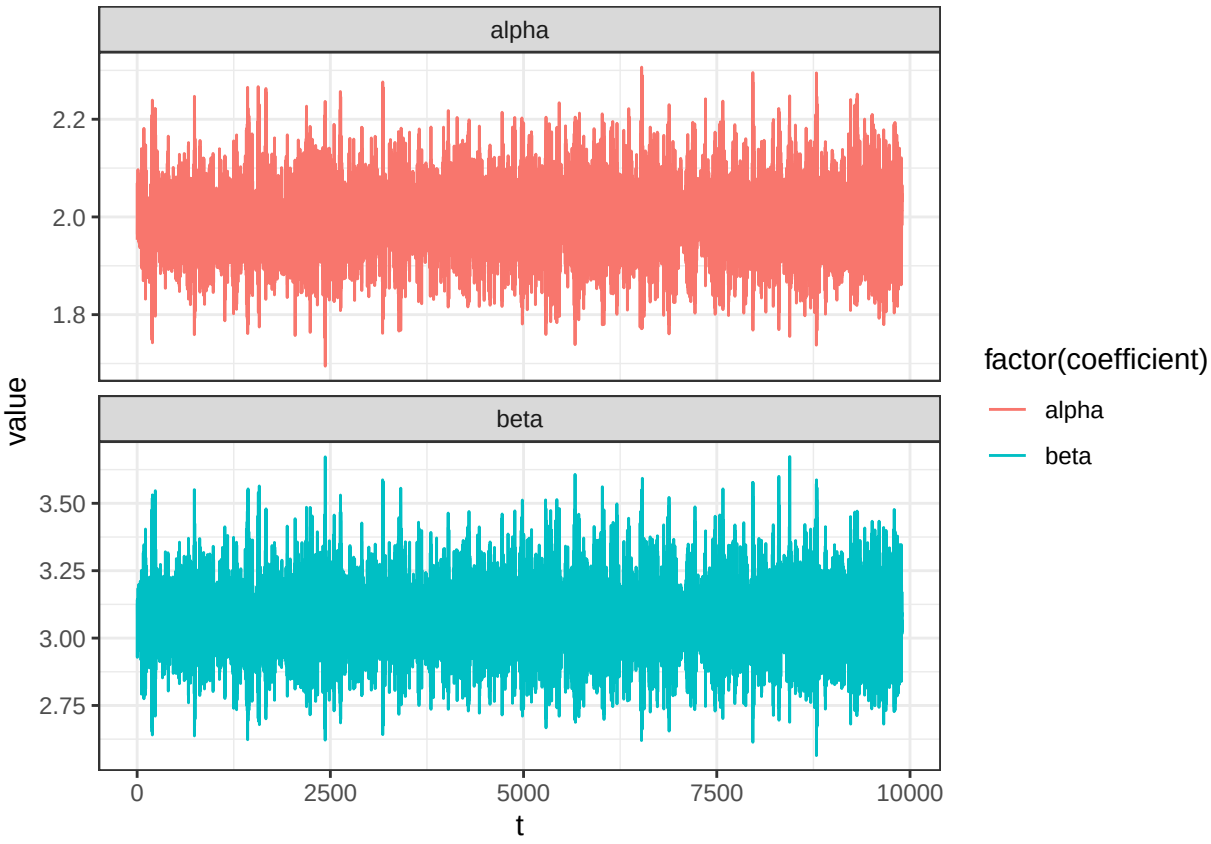
In production software, a transformation is typically performed to the parameters to allow values from the entire real number line (e.g. log transform). Alternatively, a *constrain* parameter can be set to designate α and β as strictly positive.

```
N <- 10000
set.seed(143)
ex1 <- hmc(N, theta.init = c(4, 4), epsilon = 2e-2, L = 22,
                    logPOSTERIOR = gamma.lposterior,
                    glogPOSTERIOR = g.gamma.lposterior, X=X,
                    varnames = c("alpha", "beta"),
                    eta1 = 1e-4, eta2 = 1e-4)
```

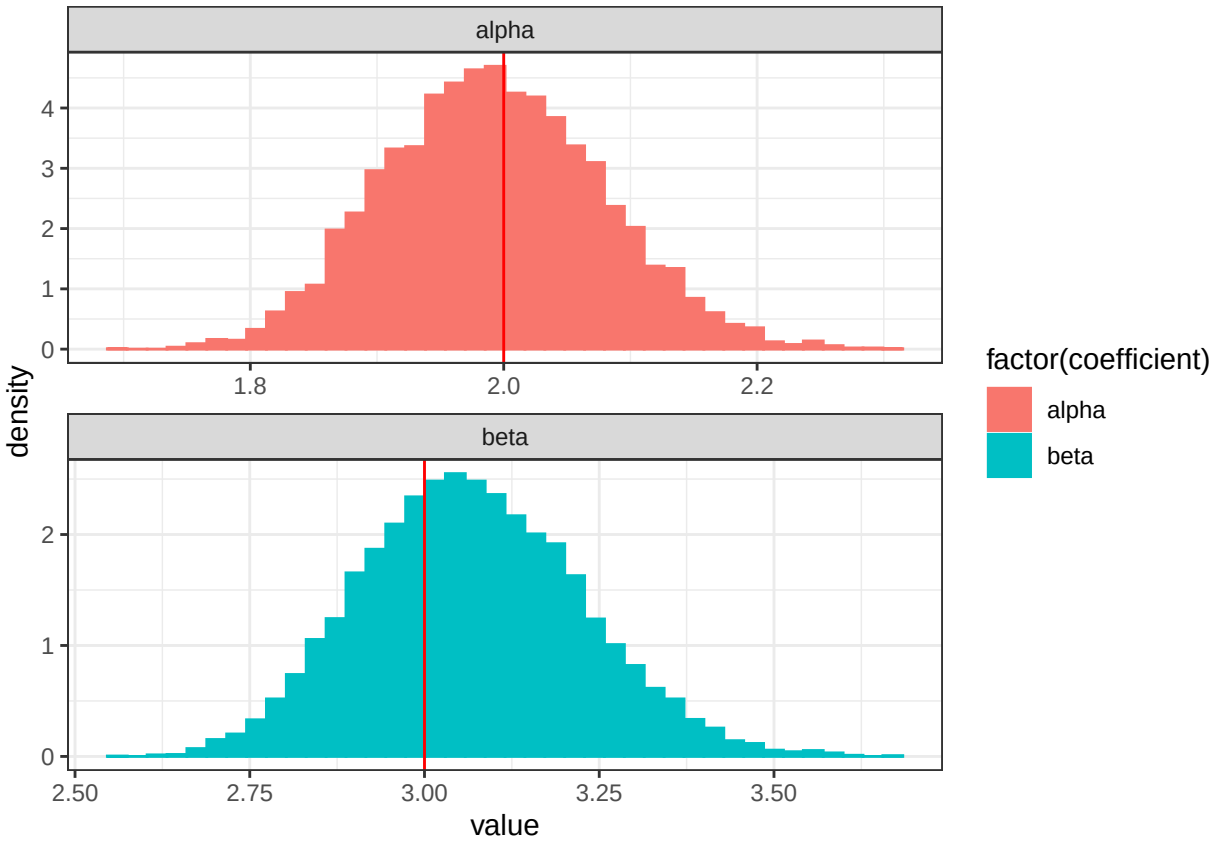
The *hmclearn* package has a diagnostic plot function to show trace plots and histograms from the simulation. An optional parameter *actual.mu* can be used to input the true values of the parameters from simulated data. A default *burnin* period of 100 samples is also selected for these plots.

Without performing rigorous diagnostics, the trace plots appear stationary with the given burnin period. The histograms of the simulations also appear well within the range of highest posterior density.

```
plot(ex1, actual.mu = c(2, 3))
#> [[1]]
```


alpha
beta
value
2.2
2.0
1.8
3.50
3.25
3.00
2.75
0
2500
5000
7500
10000
t
factor(coefficient)
alpha
beta

```
#>
#> [[2]]
```



A *summary* confirms that the actual parameter values of Θ fall well within the simulated posterior values.

```
summary(ex1)

#> Summary of HMC simulation

#>              5%      25%      50%      75%      95%
#> alpha 1.857580 1.931925 1.990089 2.049675 2.135091
#> beta  2.819241 2.957219 3.060700 3.172387 3.334479
```

In the next section, we will discuss some of the main practical considerations in fitting models using HMC, focusing on parameterization and tuning.

HMC Practical Considerations

Parameters for some statistical models are bounded. When parameters are bounded, the MCMC chain is restricted on simulated values. Since the HMC algorithm approximates the trajectories through the joint space of the momentum and Θ, it is possible that the chain may move to a location outside of the support of Θ.

Constrained Parameters

Parameters for many statistical models are limited to strictly positive numbers. This was true for the first example using HMC to fit α and β from a Gamma distribution.

We continue our previous example by implementing log transformations for each of these parameters.

$$\begin{aligned} a &= \log \alpha \\ p_a(a) &= p_\alpha(g^{-1}(a)) \left| \frac{d\alpha}{da} \right| \\ &= p_\alpha(e^a) e^a \\ \log p_a(a) &= \log p_\alpha(e^a) + a \\ &= \log p_\alpha(\alpha) + \log \alpha \end{aligned}$$

The Jacobian term $\log \alpha$ must be included with the prior when employing the log transformation of the original parameter. A similar transformation is necessary for β.

$$\begin{aligned} b &= \log \beta \\ \log p_b(b) &= \log p_\beta(\beta) + \log \beta \end{aligned}$$

The log posterior can be re-written with the new parameterization.

$$\begin{aligned} \log p(a, b|x) &\propto l(\alpha, \beta; x) + \log p(\alpha, \beta) \\ &\propto l(\alpha, \beta; x) + \log p(\alpha) + \log \alpha + \log p(\beta) + \log \beta \\ &\propto -n\alpha \log \beta - n \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \frac{1}{\beta} \sum_{i=1}^{n} x_i - \frac{\alpha^2 \eta_1^2}{\pi} - \frac{\beta^2 \eta_2^2}{\pi} + \log \alpha + \log \beta \end{aligned}$$

The gradient includes the partial derivatives of the Jacobian terms as well.

$$\begin{aligned} \nabla_\alpha \log p(a, b|x) &= -n \log \beta - n\psi(\alpha) + \sum_{i=1}^{n} \log x_i - 2\alpha\eta_1^2/\pi + \frac{1}{\alpha} \\ \nabla_\beta \log p(a, b|x) &= -\frac{n\alpha}{\beta} + \frac{1}{\beta^2} \sum_{i=1}^{n} x_i - 2\beta\eta_2^2/\pi + \frac{1}{\beta} \end{aligned}$$

Revised R functions are required for the log posterior and gradient.

```r
# log likelihood of gamma with log-transformed parameters
llgamma2 <- function(theta, X, y=NULL, Z=NULL) {
  a <- theta[1]
  b <- theta[2]
  alpha <- exp(a)
  beta <- exp(b)
  n <- length(X)
  -n*alpha*log(beta) - n*log(gamma(alpha)) + (alpha-1)*sum(log(X)) - sum(X)/beta
}

# log posterior of gamma with log-transformed parameters
gamma.lposterior2 <- function(theta, X, eta1, eta2, y=NULL, Z=NULL) {
  a <- theta[1]
  b <- theta[2]
  alpha <- exp(a)
  beta <- exp(b)
  llgamma2(theta, X) - beta^2 * eta1^2/pi - alpha^2 * eta2^2/pi + log(alpha) + log(beta)
}

# derivative of posterior with log-transformed parameters
g.gamma.lposterior2 <- function(theta, X, eta1, eta2) {
  a <- theta[1]
  b <- theta[2]
  alpha <- exp(a)
  beta <- exp(b)
  n <- length(X)
  dalpha <- -n*log(beta) -n*digamma(alpha) + sum(log(X)) - 2*alpha*eta1^2/pi + 1/alpha
  dbeta <- -n*alpha/beta + sum(X)/beta/beta - 2*beta*eta2^2/pi + 1/beta
  c(dalpha, dbeta)
}
```

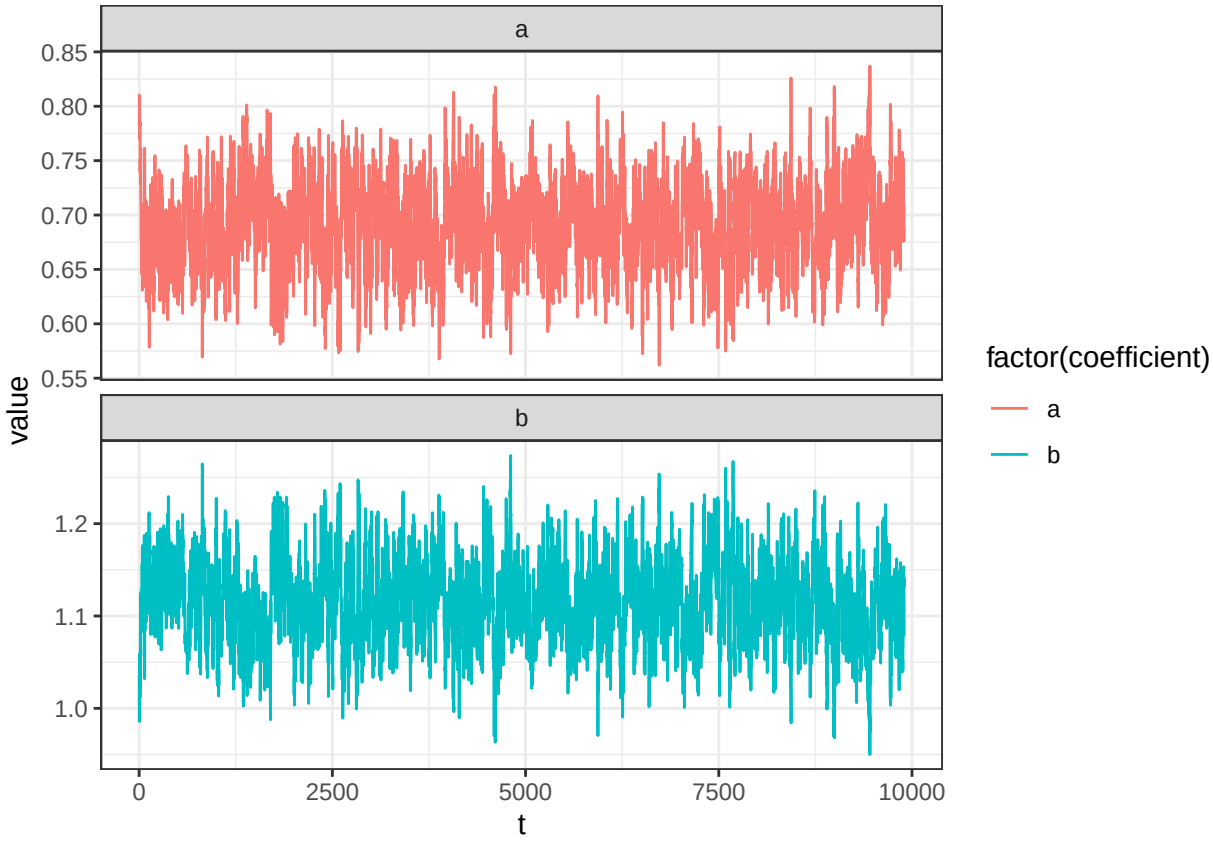
Now, we can safely re-fit the model with parameters that span the full real number line. Note that we are

re-defining $\tilde{\Theta} := (a, b) = (\log \alpha, \log \beta)$ with initial values at zero.

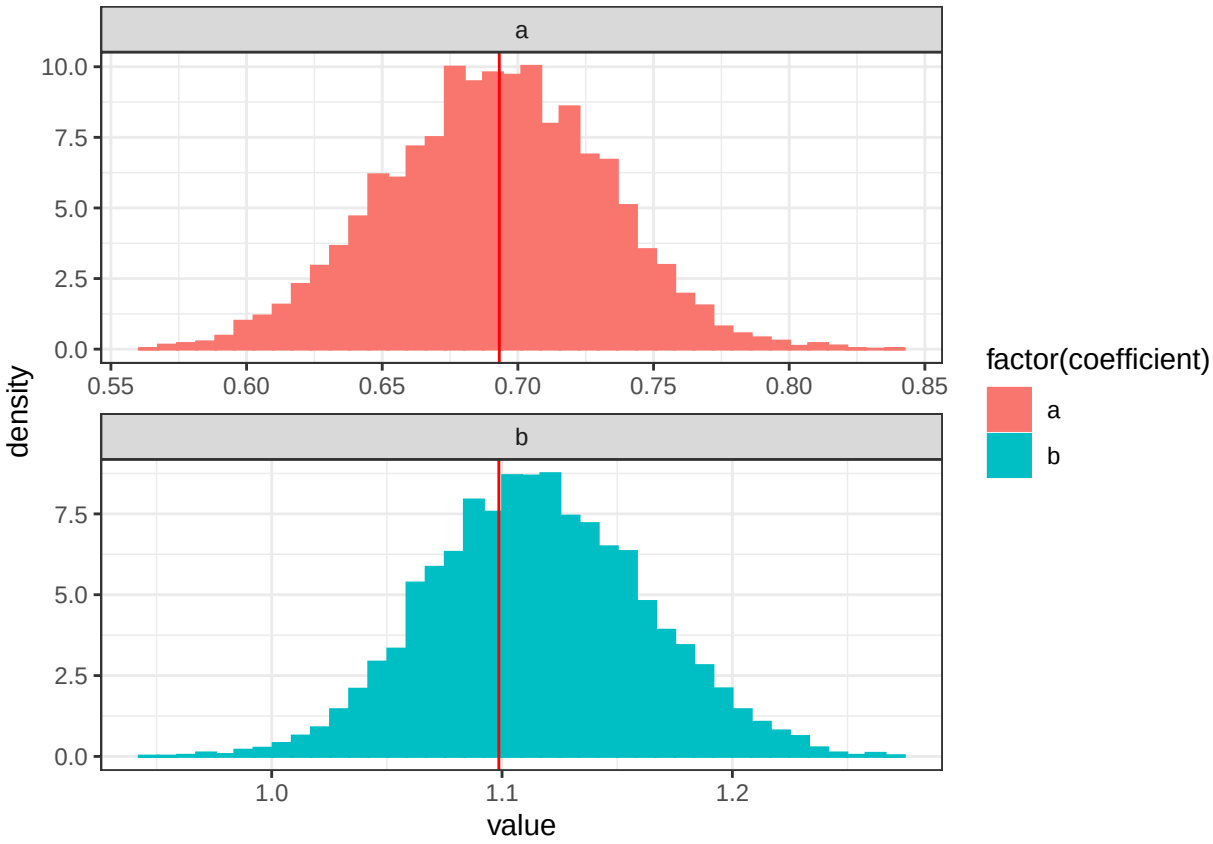
```
N <- 10000

set.seed(143)

ex1b <- hmc(N, theta.init = c(0, 0), epsilon = 1e-3, L = 30,

                    logPOSTERIOR = gamma.lposterior2,

                    glogPOSTERIOR = g.gamma.lposterior2, X=X,

                    varnames = c("a", "b"),

                    eta1 = 1e-4, eta2 = 1e-4)
```

HMC accurately fits the model using the log-transformed parameters, as we can see through the diagnostic plots and summary results.

```
plot(ex1b, actual.mu = c(log(2), log(3)))

#> [[1]]
```




```
#>

#> [[2]]
```




```
summary(ex1b)
#> Summary of HMC simulation
#>          5%       25%       50%       75%       95%
#> a 0.6239463 0.6636221 0.691753 0.7190035 0.7547585
#> b 1.0417932 1.0838178 1.114457 1.1472072 1.1938255
```

The exponential of the results confirms that the HMC converges to the correct parameters.

```
exp(summary(ex1b))
#> Summary of HMC simulation
#>          5%      25%      50%      75%      95%
#> a 1.866278 1.941813 1.997214 2.052387 2.127098
#> b 2.834295 2.955943 3.047914 3.149385 3.299680
```

Parameter transformation such as the *log* transform shown here is the most common method of ensuring that the MCMC simulation is remains in the support of Θ. An alternate approach to constrained variables was proposed by Neil (cite). This approach modifies the *leapfrog* function to ensure the proposed Θ remains within the appropriate support. A simplifed version of this method is provided in the *hmc* function for strictly

positive parameters (i.e. with support $(0, \infty)$).

Initial momentum half-step: $\boldsymbol{p}'(t + \epsilon/2) = \boldsymbol{p}(t) + (\epsilon/2)\nabla_\Theta \log \pi(\Theta(t))$

Initial full-step of Θ: $\Theta'(t + \epsilon) = \Theta(t) + \epsilon M^{-1}\boldsymbol{p}'(t + \epsilon/2)$

Check lower constraint of 0:

If $\Theta'(t + \epsilon) < 0$ then $p(t + \epsilon/2) = -p'(t + \epsilon/2)$ and $\Theta(t + \epsilon) = -\Theta'(t + \epsilon)$. Otherwise, keep the proposed momentum and Θ as simulated

Tuning

Metropolis-Hastings and HMC each utilize parameters that must be tuned to efficiently simulate from the posterior distribution. This is a matter of more practical importance than theoretical necessity. In the limit, both MCMC algorithms will converge to the true posterior given infinite samples. However, the converge rate of these algorithms varies substantially depending on how well the tuning parameters fit the application.

Tuning for random-walk Metropolis (RWM) typically involves adjusting the variance parameterization of the proposal density. Standardizing the design matrix to a common mean and variance can significantly simplify the tuning exercise. For example, a multivariate Normal proposal may simply involve adjusting a scaler for the identity covariance matrix. The analyst typically examines preliminary trace plots along with acceptance rate calculations to assess tuning effectiveness.

HMC can be significantly more complex than RWM due to the number and variety of tuning parameters in the HMC algorithm. Like RWM with a multivariate Normal proposal, HMC uses a multviariate Normal parameterization for the latent momentum variable p. While HMC may use an identity covariance matrix for M, tuning the diagonal and even off-diagonal parameters can improve the efficiency of sampling.

In addition to the covariance matrix M, HMC requires parameter selections for the step size ϵ and number of leapfrog steps L. Setting ϵ and L to low values, relative to the particular application, typically result in a high acceptance rate (e.g. greater than 95%). On the surface, accepting the vast majority of proposals may appear to be desirable. However, the combination of a small step size with few leapfrog steps can create a MCMC chain that traverses the support of Θ very slow. The evidence of inappropriately small values for these tuning parameters is a trace plot that exhibits a high degree of autocorrelation.

Generally, the step size should be set as high as possible to balance the acceptance rate and minimize autocorrelation. An acceptance rate of approximately 60 - 90% often produces MCMC chains that converge sufficiently quickly. While ϵ and L may be tuning jointly, the step size is often selected first, followed by

fine-tuning with the number of steps per leapfrog L.

A general approach to tuning HMC is provided here:

1. Set the stepsize to an initial value such as $\epsilon = 1e-2$ with $L = 10$ and M unit diagonal
2. Run a preliminary HMC chain and compute the acceptance rate. Adjust ϵ until the acceptance is between 0.6 and 0.9.
3. Check the autocorrelation for each parameter either visually or via direct calculation. For parameters with high autocorrelation, reduce the relevant value in the diagonal of M. This adjustment may decrease the acceptance rate.
4. If necessary, increase L to further reduce autocorrelation in the simulation

Additional adjustments may be made to the tuning parameters beyond these steps. The value of ϵ may be constant for each parameter in $(\theta_1, ..., \theta_k)$ in Θ, but does not need to be. The analyst may instead provide a vector $\epsilon := \epsilon_1, ..., \epsilon_k$ with a different value for each $\epsilon_i \forall\ i \in 1...k$. In the *hmclearn* package, the *hmc* function accepts single values and vectors for the parameter *epsilon*.

The parameter for the number of steps L must be a natural number. However, the number of steps may be randomized in each simulation $1...N$ to ensure against a periodic Markov chain (which would violate the regularity conditions for ergodicity). Further, some randomness may be applied to ϵ in each simulation step $1...N$ for additioanl assurance of an aperiodic chain. Randomness can be automatically applied for ϵ and L with logical parameter *randlength* in the *hmc* function.
In high performance HMC software such as STAN, the popular No-U-Turn Sampler (NUTS) algorithm (cite paper) is a popular automated option for selecting L. We avoid using NUTS in this paper as the emphasis here is more pedagogical. The intent is to provide interested readers with tools to experiment with the core HMC algorithm to gain intuition on the particulars of the algorithm.

Finally, the gradients of the log posterior must be developed in this software. However, automated differentiation software such as Autodiff is available to accurately and efficiently compute gradients for applications in HMC. Other automated gradient computation software is provided in deep learning platforms such as Tensorflow.

QR Decomposition

The efficiency of sampling in the standard HMC algorithm can be improved significantly for multivariate models when the parameters $(\theta_1, ..., \theta_k) \in \Theta$ have an orthogonal basis. One common method of ensuring an

orthogonal basis involves applying QR decomposition to the design matrix prior to applying HMC.

For example, Stan scales QR decomposition with a factor of $\sqrt{n-1}$ to ensure unit variance for continuous parameters (cite stan manual). Here X represents the design matrix for the model

$$\begin{aligned} X &= QR = Q^* R^* \\ Q^* &= \sqrt{n-1}Q \\ R^* &= \frac{1}{\sqrt{n-1}}R \\ X\Theta &= Q^* R^* \Theta \\ &= Q^* \widetilde{\Theta} \end{aligned}$$

The scaled Q matrix is used in the HMC simulation in place of the raw design matrix X. The simulated parameters will then be from $\widetilde{\Theta}$. To transform back to the original scale, we multiply by the inverse of R^*

$$\begin{aligned} R^* \Theta &= \widetilde{\Theta} \\ \Theta &= R^{*-1} \widetilde{\Theta} \end{aligned}$$

In R software, the base functions *qr* can be used to perform QR decomposition. The Q and R matrices can be obtained from the R methods $qr.Q$ and $qr.R$, respectively, applied to the resulting object.

HMC in Statistical Models

With a background of the HMC algorithm and its application to simple examples complete, we turn to more practical, real-world examples. Generalized Linear Models (GLM) are a broad class of statistical models that use a linear function to relate the mean response to a set of independent variables. Nonlinear functions of the mean can be accommodated through a specified link function. While there are the number and types of models to which HMC can be applied are limitless, we will focus on GLM's in this introductory article.

The major steps required to fit a statistical model are summarized below. We will elaborate on what is required for each step in this process.

1. Specify model
2. Derive log posterior
3. Re-parameterize parameters as needed (i.e. log transforms)

4. Derive gradient of the log posterior
5. Code functions for log posterior and gradient
6. Tune HMC parameters with trial runs
7. Run HMC

Step 1: Specify model

The general form of a GLM can be specified (cite Agresti GLM book)

$$g[E(y)] = \boldsymbol{X}\boldsymbol{\beta}$$

Here, we consider $\boldsymbol{\beta}$ our parameter of interest, such that $\Theta := \boldsymbol{\beta}$. The linear predictor of the GLM is the product of the design matrix $(\boldsymbol{x}_1, ..., \boldsymbol{x}_k) \in \boldsymbol{X}$ for $i = 1, .., n$ observations, and the parameter vector $(\beta_1, ..., \beta_k) \in \boldsymbol{\beta}$. A link function $g(\cdot)$ relates the linear predictor to the mean response. For a linear regression model, $g(\cdot)$ is the identity function, such that $E(y) = \boldsymbol{X}\boldsymbol{\beta}$.

Step 2: Derive log posterior

The log posterior is comprised of the sum of the log likelihood of the model and the log prior. In general, the likelihood can be written based on the inverse of the selected link function.

$$f(y; \boldsymbol{\beta}) = g^{-1}(\boldsymbol{X}\boldsymbol{\beta}) = p(y|\boldsymbol{X}\boldsymbol{\beta})$$

The likelihood functions for standard GLM's are readily available from statistical texts on the subject.

Once the likelihood is defined, the prior must be selected for the parameter of interest. Unlike more restrictive algorithms such as Gibbs, the analyst has a substantial degree of flexibility in prior selection for HMC. Depending on the application, the form of the prior may be defined as restricted within a certain range when known in advance.

Alternatively, the prior is defined as vague over the support of $\boldsymbol{\beta}$. For example, when $\boldsymbol{\beta}$ may be any real number, a vague prior may be a multivariate Normal with high variance. The resulting prior density would be flat over all practical values of the parameter of interest.

Once the log likelihood is defined and the prior is selected, the log posterior may be written

$$\log p(\boldsymbol{\beta}|y, \boldsymbol{X}) = \log p(y|\boldsymbol{X}\boldsymbol{\beta}) + \log p(\boldsymbol{\beta})$$

Step 3: Re-parameterize parameters as needed (i.e. log transforms)

This step should always be considered, but may not be necessary depending on the particular model. Typically, any standard deviation or variance parameters would be restricted to positive real numbers. These would need to either be re-parameterized or the constrained HMC leapfrog would need to be used.

A logistic regression model, for example, would only have the parameter vector for $\boldsymbol{\beta}$, for which the support can be the entirely real number line. A linear regression model has an error term σ^2 which is strictly positive. Fitting an HMC without considering the support of σ^2 could result in the MCMC chain selecting a negative sample. In such a case, the samples would no longer be valid for the parameter of interest and the algorithm would likely produce an error.

When re-parameterization is necessary, it is important to include the Jacobian adjustment for new parameter. Detailed explanation of a change of variables can be found in standard statistical texts.

Step 4: Derive gradient of the log posterior

Since the gradient is a linear operator, the gradient of the log posterior is equivalent to the sum of the gradient of the log likelihood and the gradient of the log prior. One may derive the gradient of the log likelihood and log prior each individually, in preparation for coding functions for these elements individually.

Note that if testing multiple prior distributions is desired, the gradient must be derived for each prior distribution to be tested.

Step 5: Code functions for the log posterior and gradient

Once the log posterior and gradient are formed, they must be coded into separate R functions. This process often requires some iteration and debugging, particularly for complex likelihood functions.

While the forms of the log likelihood and log prior can often be found from standard sources, the gradient may not be as readily available. As such, the analyst should test the gradient function at several values to ensure that it is returning the correct result.

One approach to testing the accuracy of a custom gradient function is to use a package that uses finite differencing or some variant to approximate the gradient given a general function. The R package *pracma*

contains functions that can be useful in this regard.

While finite differencing is appropriate for checking the accuracy of coding the gradient, such approximations are typically not sufficiently accurate for use in an HMC simulation. Further, gradient approximation is typically slower than a direct gradient function. Since the gradient must be calculated for each iteration of the leapfrog algorithm, a slower approximation function can substantially increase the computation time for HMC.

Step 6: Tune HMC parameters with trial runs

Tuning HMC involves adjusting M, ϵ, and L for the particular application. The tuning process detailed in the previous section can be used as a guide to set the parameters.

Trial runs will have fewer samples than the full simulation, but should have sufficient number of samples to assess the acceptance rate and stationarity of the simulation. Note that the acceptance rate in trial runs may be higher than the rate during the full simulation. This can occur when the initial values are distant from the region of high posterior density.

One common approach in any MCMC is to allow a certain burn-in period before assessing the stationarity of the simulation. This period can typically be determined with trial trace plots. The required number of simulations in the burn-in period will typically be inversely related to the combined length defined by the tuning parameters. Higher values of ϵ and L will move the chain quickly to the highest posterior region, while lower values will approach more slowly.

Step 7: Run HMC

After the parameters have been tuned, a full HMC simulation can be run. Analysts often run multiple chains with different initial values. Most production software packages allow these chains to be run in parallel. For *hmclearn*, parallelization can be accomplished using contributed R packages for the user's particular platform (e.g. Windows, mac OS, linux).

Once the simulation is complete, readily available diagonstics can be used to assess the convergence of the Markov chain. In R, the *coda* package provides a number of standard diagnostics, such as the Geweke (1992) and Gelman and Rubin (1992) diagnostics. The *bayesplot* package also provides graphical functions to assess convergence.

After the HMC simulation is complete and convergence has been assessed, the simulations can be used for statistical inference.

HMC Examples

We demonstrate fitting HMC models in R with three examples of increasing complexity.

Linear Regression

The linear regression model is this example is specified

$$y = X\boldsymbol{\beta} + \sigma$$

The dependent variable is the vector $(y_1, ..., y_n) \in y$. The independent variables are contained in the design matrix $\boldsymbol{x}_1, ..., \boldsymbol{x}_p$, with dimensions $N \times p$.

The model contains two parameters of interest to be fit via HMC, $(\beta_1, ..., \beta_p) \in \boldsymbol{\beta}$ and σ.

Next, we specify the likelihood for a linear regression model.

$$p(y|X, \beta; \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}(y - X\beta)^T(y - X\beta)\right)$$

Priors are needed for $\boldsymbol{\beta}$ and σ. We specify a flat uniform prior for $\boldsymbol{\beta}$ and Inverse Gamma (IG) for σ^2. The IG prior has two hyperpriors a and b that will need to be specified.

$$\begin{aligned} p(\beta) &\propto const \\ p(\sigma^2) &\sim IG(a, b) \\ &= \frac{b^a}{\Gamma(a)}(\sigma^2)^{-a-1} \exp\left(-\frac{b}{\sigma^2}\right) \end{aligned}$$

The support of σ^2 is $(0, \infty)$. We apply a log transformation to expand the support to all real numbers. The transformed parameter is γ which is derived using a single variable transformation.

$$\begin{aligned} \gamma &= \log \sigma^2 \\ \sigma^2 &= g^{-1}(\gamma) = e^{\gamma} \\ p_{\gamma}(\gamma) &= \frac{b^a}{\Gamma(a)} e^{-a\gamma} \exp\left(-\frac{b}{e^{\gamma}}\right) \\ \log p(\gamma) &\propto -a\gamma - be^{-\gamma} \end{aligned}$$

The log posterior for β and γ is proportional to the log likelihood plus the log prior.

$$\log p(\beta, \gamma | y, X) \propto -\left(\frac{n}{2} + a\right)\gamma - \frac{e^{-\gamma}}{2}(y - X\beta)^T(y - X\beta) - be^{-\gamma}$$

The gradient of log posterior is needed for the leapfrog function

$$\nabla_\beta \log p(\beta, \gamma | y, X) \propto e^{-\gamma} X^T (y - X\beta)$$
$$\nabla_\gamma \log p(\beta, \gamma | y, X) \propto -\left(\frac{n}{2} + a\right)$$

We have now derived everything that is needed to fit a linear regression model. The dataset *warpbreaks* is available standard with *R*.

```
head(warpbreaks)
#>   breaks wool tension
#> 1     26    A       L
#> 2     30    A       L
#> 3     54    A       L
#> 4     25    A       L
#> 5     70    A       L
#> 6     52    A       L
summary(warpbreaks)
#>      breaks       wool   tension
#>  Min.   :10.00   A:27   L:18
#>  1st Qu.:18.25   B:27   M:18
#>  Median :26.00          H:18
#>  Mean   :28.15
#>  3rd Qu.:34.00
#>  Max.   :70.00
```

The dependent variable is stored in y. The design matrix can be constructed using a standard function in *R*.

```
y <- warpbreaks$breaks
X <- model.matrix(breaks ~ wool*tension, data=warpbreaks)
```

The vector of parameters of interest is $(\boldsymbol{\beta}, \gamma) \in \theta$. The initial values specified in a vector of length 6 for β plus 1 for γ. The step size is a factor of 10 higher for $\boldsymbol{\beta}$ than for log transformed variance.

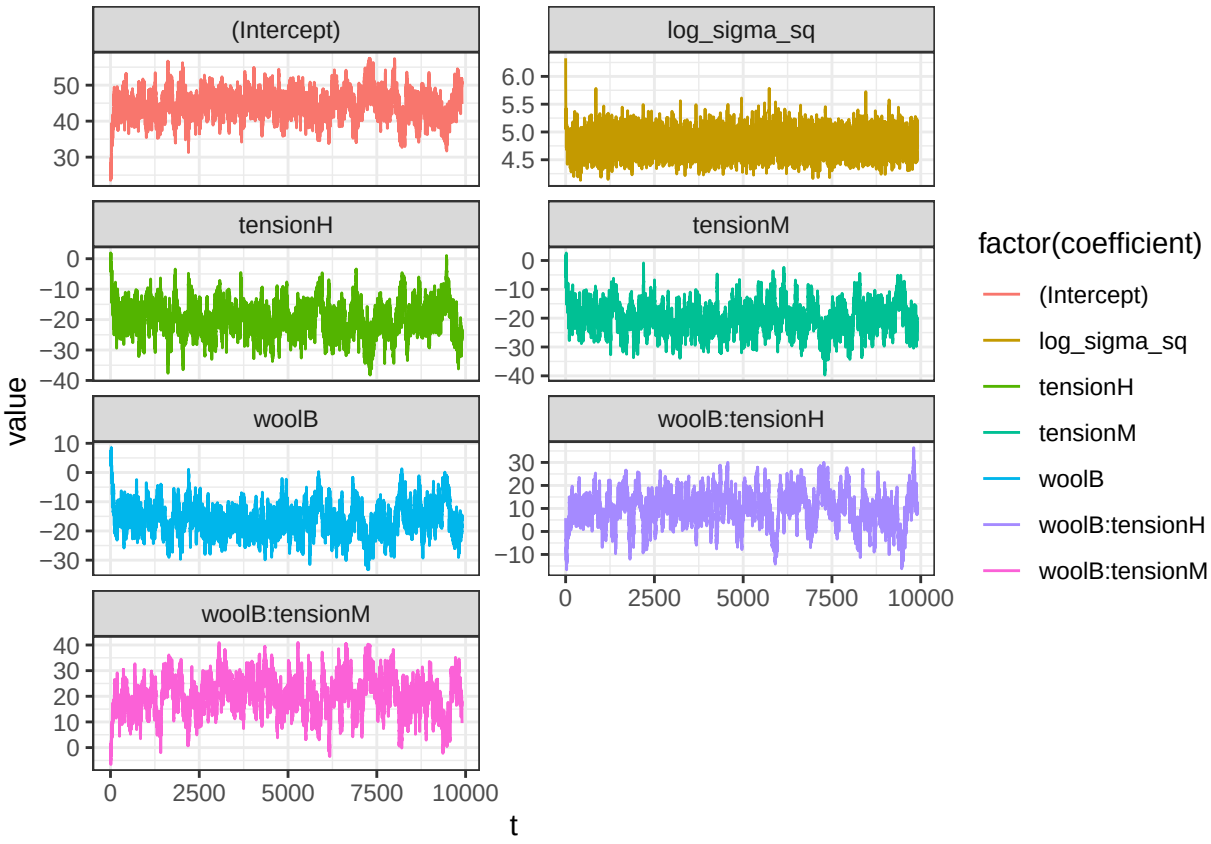
```r
N <- 10000
set.seed(143)


eps_vals <- c(rep(2e-1, 6), 2e-2)


t1 <- Sys.time()
fm1_hmc <- hmc(N, theta.init = c(rep(0, 6), 1), epsilon = eps_vals, L = 10,
                   logPOSTERIOR = linear_posterior,
                   glogPOSTERIOR = g_linear_posterior, y=y, X=X,
                   varnames = c(colnames(X), "log_sigma_sq"))
t2 <- Sys.time()
t2 - t1
#> Time difference of 10.08346 secs
```

This linear regression model takes approximately 10 seconds to fit on a 2015 Macbook Pro with 2.5GHz i7 processor and 16gb of RAM. The acceptance rate of this model fit is 96%, which is appropriate for a relatively simple model such as this one.

```r
fm1_hmc$accept/N
#> [1] 0.9573
```

Finally, we can summarize the results and plot the histograms of the simulated posteriors.

```r
summary(fm1_hmc)
#> Summary of HMC simulation
#>                         5%        25%        50%        75%        95%
#> (Intercept)     38.196614  42.094816  44.491192  46.921073  50.465719
#> woolB          -24.541405 -19.698288 -16.241622 -12.623061  -6.763026
#> tensionM       -28.310355 -23.583943 -20.218948 -16.904570 -11.289733
#> tensionH       -28.844635 -23.322128 -19.959562 -16.494869 -10.999875
#> woolB:tensionM   7.967271  15.406752  20.499092  25.151921  31.690727
#> woolB:tensionH  -3.530470   5.417325  10.480679  15.190497  22.061110
#> log_sigma_sq     4.475567   4.657704   4.789579   4.928734   5.150569
plot(fm1_hmc)[[1]]
```



Note that the Inverse Gamma distribution is not always an optimal prior when the support is strictly positive. This prior can create problematic results when the true value of the parameter is close to zero. Half-t distributions can be used as a more stable alternative to Inverse Gamma (Gelman 2006).

Logistic Regression

Our next example is a slightly more complicated generalized linear model with binary response. The model is specified

$$Pr(Y = 1|X) = [1 + e^{-X\beta}]^{-1}$$

As in the previous example, the dependent variable is $(y_1, ..., y_n) \in \boldsymbol{y}$ with an $N \times p$ design matrix $(x_1, ..., x_p) \in \boldsymbol{X}$.

The log-likelihood for a logistic regression model is

$$l(\beta; X, y) = \sum_{i=1}^{n} X_i\beta(y_i - 1) - \log(1 + e^{-X_i\beta})$$

In our prior example, we assigned a uniform prior to β. To demonstrate the flexibility of the software, we specify a multivariate Normal prior with hyperparameter B. The hyperprior should be set to a high value such as $1e3$ for a relatively uninformative prior.

$$\begin{aligned}\beta &\sim N(0, \Sigma_\beta) \\ &\sim N(0, BI)\end{aligned}$$

The log posterior is proportional to the sum of the log likelihood and log prior of $\boldsymbol{\beta}$. Note that constants are excluded to simplify the function.

$$\log p(\beta|X, y) \propto \sum_{i=1}^{n} X_i\beta(y_i - 1) - \log(1 + e^{-X_i\beta}) - \frac{1}{2}\beta^T\Sigma_\beta^{-1}\beta$$

The gradient of the log posterior is required for the leapfrog function.

$$\nabla_\beta \log p(\beta|X, y) \propto (y - 1)^T X + \left(\frac{e^{-X\beta}}{1 + e^{-X\beta}}\right)^T X - \Sigma_\beta^{-1}\beta$$

This example is from Hosmer and Lemeshow (1989) on a dataset of 189 births at a U.S. hospital. The dependent variable is an indicator of low birth weight. Data is available from the MASS package (Modern Applied Statistics with S by Venables and Ripley). We prepare the data for analysis as noted in the text.

```
birthwt2 <- birthwt


# label race variable
birthwt2$race2 <- factor(birthwt2$race, labels = c("white", "black", "other"))


# reduce to indicator variable for positive number of premature labors
birthwt2$ptd <- ifelse(birthwt2$ptl > 0, 1, 0)


# reduce to three levels
birthwt2$ftv2 <- factor(ifelse(birthwt2$ftv > 2, 2, birthwt2$ftv),
                        labels = c("0", "1", "2+"))


# create design matrix
```

```
X <- model.matrix(low ~ age + lwt + race2 + smoke + ptd + ht + ui + ftv2,
                  data = birthwt2)
y <- birthwt2$low
```

Variables of interest in this dataset are:

- low: birth weight less than 2.5kg (0/1)
- age: age of mother (yrs)
- lwt: weight of mother (lbs)
- race2: factor white/black/other
- smoke: smoking indicator (0/1)
- ptd: premature labor indicator (0/1)
- ht: history of hypertension indicator (0/1)
- ui: uterine irritability indicator (0/1)
- ftv2: number of physician visits factor (0, 1, 2 or more)

Two of the independent variables are continuous with wide ranges of values. The other nine variables are all dichotomous. In tuning this model, the step size ϵ is tuned separately to each of these types of variables. This example illustrates the need to set the tuning parameters for the specific HMC application.

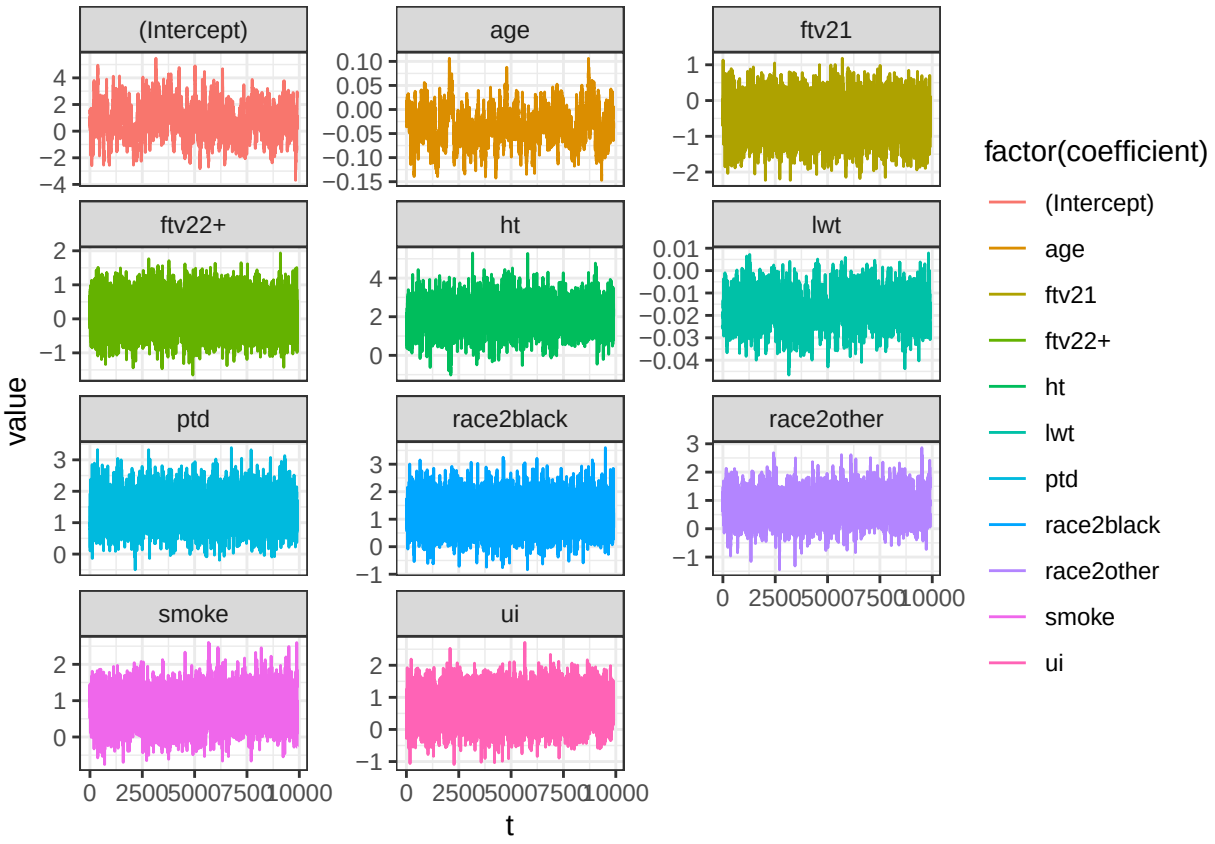
The log posterior and gradient functions are based on the likelihood and prior choices in this example.

```
N <- 10000
set.seed(143)

# use different epsilon values for continuous and dichotomous variables
continuous_ind <- c(FALSE, TRUE, TRUE, rep(FALSE, 8))
eps_vals <- ifelse(continuous_ind, 1e-3, 5e-2)

t1 <- Sys.time()
fm2_hmc <- hmc(N, theta.init = rep(0, 11), epsilon = eps_vals, L = 10,
                   logPOSTERIOR = logistic_posterior,
                   glogPOSTERIOR = g_logistic_posterior, y=y, X=X,
```

```
                    varnames = colnames(X))
t2 <- Sys.time()
t2 - t1
#> Time difference of 23.35309 secs
```

This example is slightly larger than the linear regression, taking approximately 18 seconds to run. The acceptance rate for this example is 83%.

```
fm2_hmc$accept/N
#> [1] 0.8303
summary(fm2_hmc)
#> Summary of HMC simulation
#>                        5%         25%         50%          75%          95%
#> (Intercept) -1.24074896 -0.08267908  0.77338815  1.652442249  2.836659672
#> age         -0.09537580 -0.05979888 -0.03496226 -0.008124015  0.028271813
#> lwt         -0.02937142 -0.02155002 -0.01663059 -0.011781228 -0.005280389
#> race2black   0.32046796  0.87283493  1.24408320  1.620991727  2.161785876
#> race2other   0.03091850  0.48190137  0.79086845  1.110211833  1.595168332
#> smoke        0.07279650  0.49078520  0.78178071  1.074807647  1.524820570
#> ptd          0.64279200  1.10038237  1.42579602  1.756948889  2.253508212
#> ht           0.84317318  1.55629614  2.03617908  2.508569224  3.277173581
#> ui          -0.08995582  0.38826658  0.71465278  1.031640514  1.466903344
#> ftv21       -1.33328622 -0.80152768 -0.47702258 -0.169416703  0.292104456
#> ftv22+      -0.60631860 -0.15425786  0.15610194  0.487772129  0.946406235
plot(fm2_hmc)[[1]]
```



Mixed effects model

This example demonstrates fitting a more complex higher dimension model using HMC. The distribution is Poisson with the link functon

$$\mu := E(Y|X,Z) = e^{X\beta+Zu}$$

$$\log \mu = X\beta + Zu$$

We have $i = 1...N$ observations and $j = 1...M$ groups. The fixed effects design matrix is X with dimension $N \times p$. When modeling a random intercept for each group, the random effects design matrix is Z with dimension $N \times M$.

The likelihood and log likelihood for the Mixed Effects model are based on the Poisson distribution.

$$L(\beta; y, X) = \prod_{i=1}^{n} \prod_{j=1}^{m} \frac{e^{-e^{X_i\beta+Z_{ij}u_{ij}}} e^{y_i(X_i\beta+Z_{ij}u_{ij})}}{y_i!}$$

$$l(\beta; y, X) = -e^{\sum_{ij} X\beta+Zu} + y^T(X\beta + Zu)$$

Next we select priors for our parameters of interest β and u.

We set a multivariate Normal prior for β with diagonal coveriance and hyperparameter B. Let $B = 1e4$ for instance, as a relatively uniformative prior. The relevant portions of the log prior for *beta* can be specified.

$$\log p(\beta) \propto -\frac{1}{2}\beta^T (BI)^{-1}\beta$$

The distribution of the random effects are defined as normal with a mean of zero. In this example, we assume that the covariance G is diagonal.

$$\begin{aligned} u &\sim N(0, G) \\ &:= D^{1/2}\tau \\ &\sim N(0, D^{1/2} I (D^{1/2})^T) \\ &\sim N(0, D^{1/2} D^{1/2}) \\ &\sim N(0, G) \end{aligned}$$

Let λ_k where $k = 1, ...p$ denote the diagonal entries of $G^{1/2}$.

$$G^{1/2} := \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & 0... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & \lambda_M \end{pmatrix}$$

Per Betancourt, Girolami (2013), we re-parameterize u using a standard normal parameterization we define as $\tau = (\tau_1, ..., \tau_M)$. Here, u is a deterministic function of D and τ. The intent of our parameterization is to allow D and τ to be largely independent in the MCMC sampling.

$$\begin{aligned} \tau &\sim N(0, I_M) \\ u &:= D^{1/2}\tau \\ &\sim N(0, D^{1/2} I (D^{1/2})^T) \\ &\sim N(0, D^{1/2} D^{1/2}) \\ &\sim N(0, G) \end{aligned}$$

We assign half-t priors for each of the parameters $\lambda_1, ..., \lambda_M$ (Gelman 2006) with hyperparameters ν and A.

$$p(\lambda_k) \sim \left(1 + \frac{1}{\nu}\left(\frac{\lambda_k}{A}\right)^2\right)^{-(\nu+1)/2}$$

We apply a log transformation to λ_k where $\xi_k := \log \lambda_k$ where $k = 1...M$.

$$\begin{aligned}
p(\lambda_k) &\sim \left(1 + \frac{1}{\nu}\left(\frac{\lambda_k}{A}\right)^2\right)^{-(\nu+1)/2} \\
p_{\xi_k}(\xi_k) &= p_{\lambda_k}(g^{-1}(\xi_k)) \left|\frac{d\lambda_k}{d\xi_k}\right| \\
&= p_{\lambda_k}(e^{\xi_k})|e^{\xi_k}| \\
&\propto \left(1 + \frac{1}{\nu}\left(\frac{e^{2\xi_k}}{A^2}\right)\right)^{-(\nu+1)/2} e^{\xi_k} \\
\log p(\xi_k) &\propto -\frac{\nu+1}{2}\log\left(1 + \frac{1}{\nu}\left(\frac{e^{2\xi_k}}{A^2}\right)\right) + \xi_k
\end{aligned}$$

With the log likelihood and priors defined, we can specify the log posterior for HMC.

$$\begin{aligned}
\log p(\beta, \gamma, \tau, \xi | y, X, Z) &\propto \log p(y|\beta, \tau, \gamma, \xi) + \log p(\beta) + \log p(\gamma) + \log p(\tau) + \log p(\xi) \\
&\propto -e^{\sum_{ij} X\beta + ZD^{1/2}\tau} + y^T(X\beta + ZD^{1/2}\tau) - \frac{1}{2}\beta^T(BI)^{-1}\beta \\
&- \sum_{k=1}^{M}\left(\frac{\nu_{\lambda_k}+1}{2}\log\left(1 + \frac{1}{\nu_{\lambda_k}}\frac{e^{2\xi_k}}{A_{\lambda_k}^2}\right) + \xi_k\right) - \frac{1}{2}\tau^T\tau
\end{aligned}$$

We need to derive the gradient of the log posterior for the leapfrog function. Let J^{kk} be the single element version of the diagonal matrix $D^{1/2}$ retaining only a single e^{ξ_k} and the remainder of the diagonal zero.

$$\begin{aligned}
\frac{\partial l}{\partial \beta} &\propto -\left(e^{\sum_{ij} X\beta + ZLD^{1/2}\tau}\right)^T X + y^T X - \Sigma_\beta^{-1}\beta \\
\frac{\partial l}{\partial \tau} &\propto -\left(e^{\sum_{ij} X\beta + ZLD^{1/2}\tau}\right) ZLD^{1/2} + y^T ZLD^{1/2} - \tau \\
\frac{\partial l}{\partial \xi_k} &\propto -\left(e^{\sum_{ij} X\beta + ZLD^{1/2}\tau}\right) ZLJ^{kk}\tau + y^T ZLJ^{kk}\tau - \frac{\nu_{\lambda_k}+1}{1 + \nu_{\lambda_k}A_{\lambda_k}^2 e^{-2\xi_k}} + 1
\end{aligned}$$

Our example comes from a study on gopher tortoises

Ozgul, A., Oli, M. K., Bolker, B. M., & Perez-Heydrich, C. (2009). Upper respiratory tract disease, force of infection, and effects on survival of gopher tortoises. Ecological Applications, 19(3), 786–798. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/19425439

https://bbolker.github.io/mixedmodels-misc/ecostats_chap.html#gopher-tortoise

As a comparison, we fit the model using the Frequentist package *lme4*

```
library(lme4)
#> Loading required package: Matrix

# Gopher data:  Gdat dataframe
data(gopherdat2)

# omit area offset
fm3 <- glmer(shells~prev+factor(year)+(1|Site),
      family=poisson,data=Gdat,
      control=glmerControl(optimizer="bobyqa",
                           check.conv.grad=.makeCC("warning",0.05)))



summary(fm3)
#> Generalized linear mixed model fit by maximum likelihood (Laplace
#>   Approximation) [glmerMod]
#>  Family: poisson  ( log )
#> Formula: shells ~ prev + factor(year) + (1 | Site)
#>    Data: Gdat
#> Control:
#> glmerControl(optimizer = "bobyqa", check.conv.grad = .makeCC("warning",
#>     0.05))
#>
#>      AIC      BIC   logLik deviance df.resid
#>    100.3    107.3    -45.1     90.3       25
#>
#> Scaled residuals:
#>     Min      1Q  Median      3Q     Max
#> -1.1282 -0.6135 -0.1528  0.2421  1.7940
#>
```

```
#> Random effects:
#>  Groups Name        Variance Std.Dev.
#>  Site   (Intercept) 0.3887   0.6235
#> Number of obs: 30, groups:  Site, 10
#>
#> Fixed effects:
#>                   Estimate Std. Error z value Pr(>|z|)
#> (Intercept)      -0.057791   0.397499  -0.145  0.88441
#> prev              0.022301   0.007715   2.891  0.00385 **
#> factor(year)2005 -0.653685   0.357270  -1.830  0.06730 .
#> factor(year)2006 -0.373511   0.322775  -1.157  0.24720
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Correlation of Fixed Effects:
#>             (Intr) prev   f()2005
#> prev        -0.592
#> fctr(y)2005 -0.333  0.007
#> fctr(y)2006 -0.243 -0.209  0.380
```

Next, we store Frequentist estimates in R variables.

```
truevals_fixed <- c(fixef(fm3))
truevals_fixed <- truevals_fixed[c(1, 3, 4, 2)]
truevals_random <- as.numeric(ranef(fm3)$Site[, 1])
truevals <- c(truevals_fixed, truevals_random,
              log(sqrt(as.numeric(VarCorr(fm3)$Site[, 1]))))
  # sqrt(as.numeric(VarCorr(fm3)$Site[, 1])))
nvar <- length(truevals)
```

The design matrices X and Z must be setup for *hmc*.

```
library(Matrix)


##########
```

```r
# block diagonal
Zi.lst <- split(rep(1, nrow(Gdat)), Gdat$Site)
Zi.lst <- lapply(Zi.lst, as.matrix)
Z <- bdiag(Zi.lst)
Z <- as.matrix(Z)
X <- model.matrix(~ factor(year), data=Gdat)
X <- cbind(X, Gdat$prev)
colnames(X)[ncol(X)] <- "prev"
colnames(X) <- make.names(colnames(X))
colnames(X)[1] <- "intercept"
y <- Gdat$shells
p <- ncol(X)
```

Finally, we run HMC for the Poisson mixed effects model. Note that we use a slightly informative prior on the random effects variance, a half-t prior with 4 degrees of freedom and scale parameter of 1. This facilitates converge of the HMC while allowing for a range of all reasonable values of the random effects variance.

Since this is a more complex example, we increase the number of samples to allow the MCMC chain more time to converge.

```r
N <- 1e5

set.seed(412)
initvals <- c(rep(0, 4),
              rnorm(10, mean=0, sd=1e-3),
              0)


M_vals <- c(1e-3, 1e-3, 1e-3, 1,
            rep(1e-3, 10),
            1e-3)
```

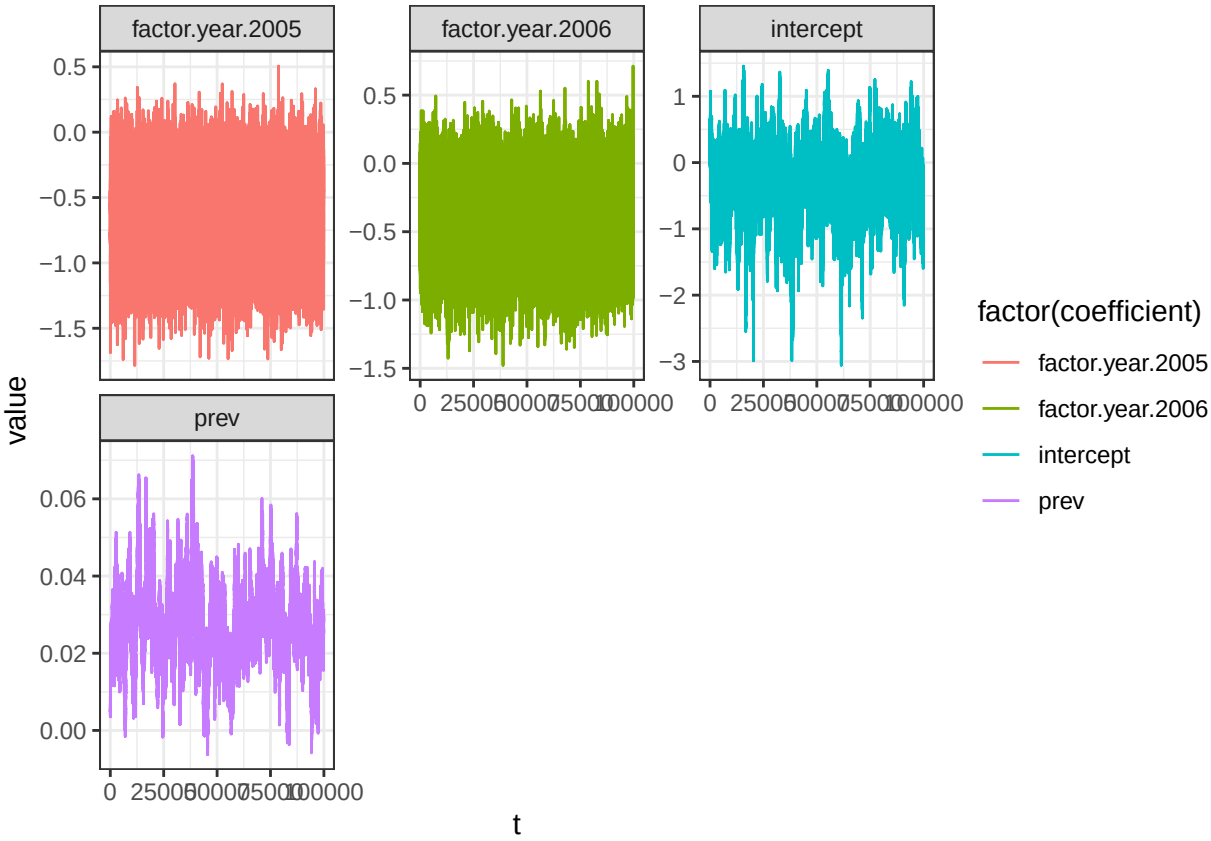
```r
t1.hmc <- Sys.time()

eps_vals <- c(rep(1e-3, 3), 1e-4, rep(2e-4, 11))

res <- hmc(N = N, theta.init = initvals, epsilon = eps_vals, L = 10,
                    logPOSTERIOR  = glmm_poisson_posterior,
                    Mdiag = M_vals,
                    varnames=c(colnames(X), paste0("u", 1:ncol(Z)), "lambda"),
                    glogPOSTERIOR  = g_glmm_poisson_posterior,
                    y = y, X=X, Z=Z, m=10, nuxi=1, Axi=25)

mypath <- '/Users/samthomas/biostat'
# saveRDS(res, file = paste(mypath, "example3.RData", sep="/"))

t2.hmc <- Sys.time()
t2.hmc - t1.hmc
#> Time difference of 13.04418 mins
res$accept/N
#> [1] 0.63217
```

```r
summary(res)
#> Summary of HMC simulation
#>                            5%         25%         50%         75%         95%
#> intercept        -1.186319574 -0.61239623 -0.27706979  0.03977338  0.47476008
#> factor.year.2005 -1.064248861 -0.81172537 -0.63486711 -0.46470017 -0.22114497
#> factor.year.2006 -0.811313951 -0.56549226 -0.40161070 -0.24045300 -0.01073622
#> prev              0.009897522  0.02046962  0.02698542  0.03402458  0.04672949
#> u1               -1.716241867 -1.22492526 -0.88069573 -0.51599419 -0.05109184
#> u2               -0.849965984 -0.36248626 -0.09236453  0.16159762  0.50171450
#> u3               -1.146234990 -0.69324762 -0.43225270 -0.20078158  0.12660727
#> u4               -0.326973726  0.03170525  0.32216520  0.64502256  1.19162713
#> u5               -0.608430259 -0.26523408 -0.03504051  0.19527744  0.51793162
#> u6                0.258950194  0.60536512  0.87234866  1.14751738  1.53282725
```

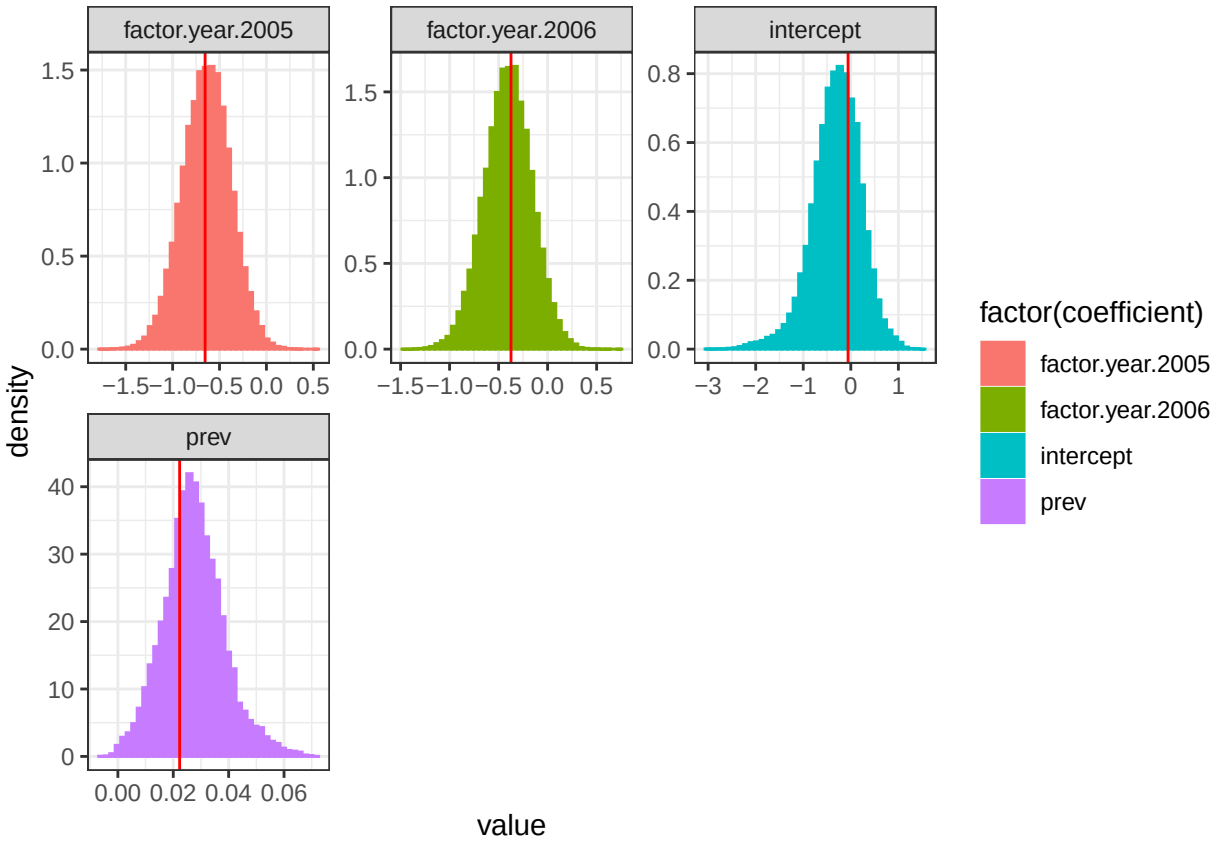
```
#> u7          -0.304127775 -0.01959770  0.17770746  0.38745935  0.71467070
#> u8          -0.761340817 -0.40865255 -0.16596744  0.07131674  0.41952747
#> u9           0.149307785  0.51102619  0.78041796  1.05675279  1.47831599
#> u10         -1.607109211 -1.13445334 -0.83640010 -0.57793229 -0.23290356
#> lambda      -0.229571978  0.08707938  0.32827142  0.59176781  0.97378004
```

```
N <- 1e5
mypath <- '/Users/samthomas/biostat'
res <- readRDS(paste(mypath, 'example3.RData', sep='/'))
```

```
# plot(res, burnin=round(0.3*N), actual.mu=truevals_fixed, cols=1:ncol(X))


plot(res, actual.mu = truevals_fixed, cols=1:4)
#> [[1]]
```




```
#>
#> [[2]]
```




```
# plot(res)
```

Discussion

Hamiltonian Monte Carlo is a flexible MCMC algorithm that efficiently samples from even complex posterior distributions. While Gibbs requires at least conditional conjugate priors, HMC may be applied to virtually any model with well-behaved properties such as a twice differentiable continuous distribution. Analysts are, as a result, able to specify models and priors based entirely on the particular application without artificial mathematical restrictions.

While there are many algorithms that use the gradient for optimization (e.g. stochastic gradient descent), HMC uniquely applies the gradient to sample from the entire distribution. The result is an accurate estimate of the full posterior distribution, not merely a single point. This distribution can be intuitively applied to answer a variety of inference, estimation, and prediction questions.

This software can be used used to fit a wide variety models using HMC provided functions for the log posterior and its gradient can be developed. Since the software is open source, analysts with a background in R can easily examine the functions to increase their understanding of the details behind HMC. This intuition can

be beneficial when using STAN and other HMC software with high-performance compiled code that may be more difficult to interpret.

The HMC algorithm as applied in this software represents only the beginning of its potential application for big data analysis. This algorithm uses only the first derivative in guiding the Markov chain. A more advanced version called Riemannian HMC (Girolami, Calderhead 2011) incorporates the second derivative along with a more sophisticated leapfrog function. This algorithm can be used to complex, correlated spaces without the need for transformations such as QR decomposition to produce an orthogonal space. As of this writing, the STAN team has already developed a substantial amount of development for generalized Riemannian HMC software.

Finally, on the computational hardware side parallelization for most available software is currently limited to the embarrassingly parallel application of running multiple HMC chains over the available CPU's. The incorporation of GPU's in improving the efficiency of HMC represents a new frontier in high performance statistical computation. One study found that GPU's could be used to reduce computation time when evaluating the posterior as well as its gradient (Fast HMC using GPU Computing, Beam Ghosh Doyle). NVIDIA is sponsoring a grant to support the STAN team in its HMC software development using GPU's.