The American Statistician

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/utas20

Learning Hamiltonian Monte Carlo in R

Samuel Thomas & Wanzhu Tu

To cite this article: Samuel Thomas & Wanzhu Tu (2021) Learning Hamiltonian Monte Carlo in R, The American Statistician, 75:4, 403-413, DOI: 10.1080/00031305.2020.1865198

To link to this article: https://doi.org/10.1080/00031305.2020.1865198

THE AMERICAN STATISTICIAN
2021, VOL. 75, NO. 4, 403–413: Statistical Computing and Graphics
https://doi.org/10.1080/00031305.2020.1865198

Learning Hamiltonian Monte Carlo in R

Samuel Thomas and Wanzhu Tu

Department of Biostatistics and Health Data Science, Indiana University School of Medicine, Indianapolis, IN

ABSTRACT

Hamiltonian Monte Carlo (HMC) is a powerful tool for Bayesian computation. In comparison with the traditional Metropolis–Hastings algorithm, HMC offers greater computational efficiency, especially in higher dimensional or more complex modeling situations. To most statisticians, however, the idea of HMC comes from a less familiar origin, one that is based on the theory of classical mechanics. Its implementation, either through Stan or one of its derivative programs, can appear opaque to beginners. A lack of understanding of the inner working of HMC, in our opinion, has hindered its application to a broader range of statistical problems. In this article, we review the basic concepts of HMC in a language that is more familiar to statisticians, and we describe an HMC implementation in R, one of the most frequently used statistical software environments. We also present `hmclearn`, an R package for learning HMC. This package contains a general-purpose HMC function for data analysis. We illustrate the use of this package in common statistical models. In doing so, we hope to promote this powerful computational tool for wider use. Example code for common statistical models is presented as supplementary material for online publication.

ARTICLE HISTORY
Received June 2020
Accepted December 2020

KEYWORDS
Bayesian computation; Hamiltonian Monte Carlo; MCMC; Stan

1. Introduction

Hamiltonian Monte Carlo (HMC) is one of the newer Markov chain Monte Carlo (MCMC) methods for Bayesian computation. An essential advantage of HMC over the traditional MCMC methods, such as the Metropolis–Hastings algorithm, is its greatly improved computational efficiency, especially in higher dimensional and more complex models. But despite the method's computational prowess and the existence of excellent introductions (Neal 2011; Betancourt 2017), practitioners still face daunting challenges in applying the method to their own applications. Difficulties mainly arise in three areas: (i) unfamiliarity with the theory behind the algorithm, (ii) lack of understanding of how the existing software works, (iii) inability to tune the HMC parameters. These difficulties have limited the use of HMC to those who understand the theory and have the programming skills to implement the algorithm. But it does not have to be so.

The emergence of modern Bayesian software such as Stan (Carpenter et al. 2017) has, to some extent, alleviated these difficulties. Stan is a powerful and versatile programming language that has a syntax similar to that of WinBUGS, but uses HMC instead of Gibbs sampling to generate posterior samples (Gelman, Lee, and Guo 2015). Stan translates its code to a lower-level language to maximize speed and efficiency. Importantly, it automates the tuning of HMC parameters and thus significantly reduces the burden of implementation. For R and Python users, packages have been created that allow Stan be called from those languages. For people who are familiar with WinBUGS and comfortable with programming in probabilistic terms, Stan is an ideal choice for HMC implementation. But for beginners who want to learn HMC, Stan can come across as a "black box". Other high-performance software, such as `PyMC` and `Edward` (Salvatier, Wiecki, and Fonnesbeck 2016; Tran et al. 2016), present similar challenges. While scalability and efficiency are often the foremost considerations in software development, a good understanding of the methodology is more essential to learners, as it instills confidence in the practical use of new methods.

The objectives of the current article are largely pedagogical, that is, helping practitioners learn HMC and its algorithmic ingredients. Toward that end, we developed a general-purpose R function `hmc` for the fitting of common statistical models. We also present details of HMC parameter tuning for those who are interested in writing and implementing their own programs. Multiple examples are presented, with accompanying R code. We have assembled all of the learning material, including the necessary HMC functions, example code, and data in an R package, `hmclearn`, for convenience of the readers.

2. Markov Chain Monte Carlo: The Basics

MCMC is a broad class of computational tools for integral approximation and posterior sample generation. In Bayesian analysis, MCMC algorithms are primarily used to simulate samples for approximation of the posterior distribution.

In Bayesian analysis, estimation and inference of the parameter of interest are made based on the observed data $\mathcal{D}$ together

CONTACT Wanzhu Tu *wtu1@iu.edu* Department of Biostatistics and Health Data Science, Indiana University School of Medicine, Indianapolis, IN.
Supplementary materials for this article are available online. Please go to *www.tandfonline.com/r/TAS*.

with the a priori information that one has on the parameters of interest $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)^T \in \mathbb{R}^k$. The posterior distribution $f(\boldsymbol{\theta}|\mathcal{D})$ combines both the data and prior information in accordance to the Bayes formula, and is proportional to the product of the likelihood function $f(\mathcal{D}|\boldsymbol{\theta})$ and the prior density $f(\boldsymbol{\theta})$ (Carlin and Louis 2008) ,

$$f(\boldsymbol{\theta}|\mathcal{D}) = \frac{f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})}{\int f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})d\boldsymbol{\theta}},$$
$$\propto f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta}).$$

The integral in the denominator is usually difficult to evaluate. But since the denominator is constant with respect to $\boldsymbol{\theta}$, one could work with the unnormalized posterior $f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})$. In the absence of an explicit expression of the posterior, approximating it with simulated samples following $f(\boldsymbol{\theta}|\mathcal{D})$ becomes a desirable alternative.

2.1. Metropolis–Hastings

Metropolis algorithm is the first widely used MCMC method for generating Markov Chain samples following $f(\boldsymbol{\theta}|\mathcal{D})$. The method originated from a physics application in the 1950s (Metropolis et al. 1953), and was further extended nearly two decades later by Hastings (1970), thus giving rise to the name of Metropolis–Hastings (MH) algorithm. We begin with a brief description of MH, as HMC was built on a similar concept.

MH generates a sequence of values of $\boldsymbol{\theta}$ that form a Markov chain, whose values can be used to approximate a posterior density $f(\boldsymbol{\theta}|\mathcal{D})$. For brevity, we drop $\mathcal{D}$ from the expression and write the posterior simply as $f(\boldsymbol{\theta})$. Values in the Markov chain $\boldsymbol{\theta}^{(t)}$ are indexed by $t = 0, 1, .., N$, where $\boldsymbol{\theta}^{(0)}$ is a user or program-specified starting value.

MH defines a transition probability that assures the Markov chain is *ergodic* and satisfies *detailed balance* and *reversibility* (Chib and Greenberg 1995). These technical conditions are put in place to ensure the chain samples from the full support of $\boldsymbol{\theta}$ without bias.

In MH, values of $\boldsymbol{\theta}^{(t)}$ in the chain are defined in part by a proposal density $q(\boldsymbol{\theta}^{\text{PROP}}|\boldsymbol{\theta}^{t-1})$, where $\boldsymbol{\theta}^{\text{PROP}}$ is a proposal for the next value in the chain. This proposal density is conditioned on the previous value $\boldsymbol{\theta}^{(t-1)}$. A variety of proposal functions can be used, with random walk proposals being the most common choice.

Algorithm 1 Metropolis–Hastings

procedure MH($\boldsymbol{\theta}^{(0)}, f(\boldsymbol{\theta}), q(\boldsymbol{\theta}^{(1)}|\boldsymbol{\theta}^{(2)}), N$)
  Calculate $f(\boldsymbol{\theta}^{(0)})$
  for $t = 1, \ldots, N$ **do**
    $\boldsymbol{\theta}^{\text{PROP}} \leftarrow q(\boldsymbol{\theta}^{\text{PROP}}|\boldsymbol{\theta}^{(t-1)})$
    $u \leftarrow U(0, 1)$
    $\alpha = \min\left(1, \frac{f(\boldsymbol{\theta}^{\text{PROP}})q(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^{\text{PROP}})}{f(\boldsymbol{\theta}^{(t-1)})q(\boldsymbol{\theta}^{\text{PROP}}|\boldsymbol{\theta}^{(t-1)})}\right)$
    If $\alpha < u$, then $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{\text{PROP}}$. Otherwise, $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
  end for
  return $\boldsymbol{\theta}^{(1)} ... \boldsymbol{\theta}^{(N)}$
end procedure

In MH, a proposal is accepted with probability

$$\alpha = \min\left(1, \frac{f(\boldsymbol{\theta}^{\text{PROP}})q(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^{\text{PROP}})}{f(\boldsymbol{\theta}^{(t-1)})q(\boldsymbol{\theta}^{\text{PROP}}|\boldsymbol{\theta}^{(t-1)})}\right), \tag{1}$$

When q is symmetric i.e., $q(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^{\text{PROP}}) = q(\boldsymbol{\theta}^{\text{PROP}}|\boldsymbol{\theta}^{(t-1)})$, this simplifies to

$$\alpha = \min\left(1, \frac{f(\boldsymbol{\theta}^{\text{PROP}})}{f(\boldsymbol{\theta}^{(t-1)})}\right),$$

which is used in the original Metropolis algorithm.

The denominator in the posterior is constant with respect to $\boldsymbol{\theta}$. As such, the ratio of posterior densities at two different points $\boldsymbol{\theta}^{\text{PROP}}$ and $\boldsymbol{\theta}^{(t-1)}$ can be compared even when the denominator is unknown, with the denominators being cancelled out. Because a derivation of the full posterior distribution (numerator and denominator) is not necessary to implement MH (and HMC, as we will see), data analysts have considerable flexibility to select models of their liking.

The acceptance rate α in (1) is an important gauge of the efficiency of an MH algorithm. A careful examination of α's roles gives a more intuitive understanding of the algorithm:

1. When $f(\boldsymbol{\theta}^{\text{PROP}}) \geq f(\boldsymbol{\theta}^{(t-1)})$, the proposal $f(\boldsymbol{\theta}^{\text{PROP}})$ represents a "more likely" value than the previous value $\boldsymbol{\theta}^{(t-1)}$, as quantified by the density functions. When this occurs, the proposal is always accepted (i.e., with probability 1).
2. When $f(\boldsymbol{\theta}^{\text{PROP}}) < f(\boldsymbol{\theta}^{(t-1)})$, the proposal $\boldsymbol{\theta}^{\text{PROP}}$ has a lower density in comparison to the previous value, and we accept the proposal at random with probability $\alpha \in (0, 1)$, which indicates the relative likelihood of observing $\boldsymbol{\theta}^{\text{PROP}}$ from f, as compared to $\boldsymbol{\theta}^{(t-1)}$. The larger the α, the greater the chance of accepting $\boldsymbol{\theta}^{\text{PROP}}$. If the proposal is not accepted, the proposal will be discarded and the chain will remain in place $\boldsymbol{\theta}^{(t)} := \boldsymbol{\theta}^{(t-1)}$, and we will start with a new proposal.

With such a scheme, the algorithm frequents regions of *higher* posterior density, while occasionally visiting the low density areas (e.g., tails in one-dimensional situations). Provided the algorithm satisfies the conditions for ergodicity (Tierney 1994) and runs a sufficient number of iterations, the empirical distribution of the MCMC chain samples should approximate the true posterior density. The simulated values can therefore be used for estimation and inference based on the posterior distribution. See Carlin and Louis (2008), Chib and Greenberg (1995), and Gelman et al. (2013) for additional details on MH.

2.2. Limitations of Metropolis–Hastings

The theoretical requirements for using MH are quite minimal, making it an attractive choice for Bayesian inference. Limitations of MH are primarily computational. With randomly generated proposals, it often takes a large number of iterations to get into areas of higher posterior density. Even efficient MH algorithms sometimes accept less than 25% of the proposals (Roberts et al. 1997). In lower dimensional situations, increased computational power may compensate the lower efficiency to some extent. But in higher dimensional and more complex modeling situations, faster computers alone are rarely sufficient to overcome the challenge.

Gibbs sampling can be a viable and more efficient alternative to MH in some situations (Geman and Geman 1984). In fact, several popular software platforms, such as WinBUGS and JAGS, use Gibbs to generate posterior samples (Lunn et al. 2000; Plummer 2003). Gibbs' requirement for explicitly expressed conditional posterior densities, however, has prevented it from being used in many practical situations. In addition to this restriction, Gibbs also has its own efficiency limitations (Robert 2007). It is in this context that HMC emerges as a preferred alternative for Bayesian analysis.

3. Hamiltonian Monte Carlo

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 toward 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 (Roberts et al. 1997).

It is important to note that although the HMC algorithm frequently samples in regions of higher density, referred to as the *typical set* (Betancourt 2017), it still samples the tail areas properly. While both MH and HMC produce ergodic Markov chains, the mathematics of HMC is substantially more complex than that of MH. In this article, we provide a less technical introduction of the ideas behind HMC. More technical expositions can be found elsewhere (Neal 2011; Betancourt 2017).

3.1. The Idea

The methods one uses to generate proposals strongly influences the efficiency of MCMC. Suppose $f(\theta)$ is a one-dimensional posterior density function, and $-\log f(\theta)$ assumes the shape of an inverse bell-shaped curve as depicted by Figure 1. To generate θ in a region of high posterior density, one needs to sample θ in the region corresponding to the lower values of $-\log f(\theta)$; the region can be reached with the guidance of the gradient of $-\log f(\theta)$. 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.

In a Hamiltonian system, the horizontal and vertical positions are given by (θ, p). In MCMC, we are interested in θ. The parameter p, which is often referred to as the *momentum*, is an auxiliary quantity that we use to simulate θ under the Hamiltonian equations.

3.2. The Hamiltonian Equations

We introduce HMC in a generic MCMC setting, where $\boldsymbol{\theta}$ follows the posterior density $f(\boldsymbol{\theta})$ of interest, and the momentum $\mathbf{p}$

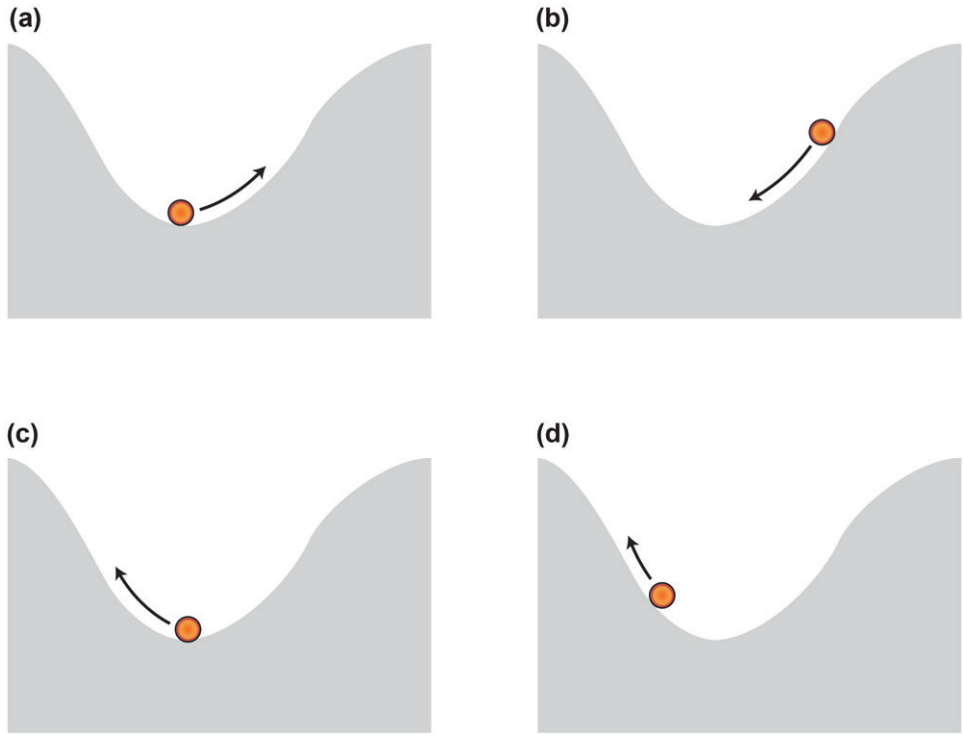

Figure 1. One-dimensional HMC example—movement of an object on a smooth, frictionless curve. (a) We apply a force with randomly generated direction and strength to the object. This object acquires a certain amount of kinetic energy, which makes it move in the direction of the applied force. The momentum, proportional to the object's velocity, changes throughout the path of the curve. When the object moves up along the curve, the velocity of the object and its momentum decrease. Its kinetic energy converts to potential energy, while the total energy remains constant. (b) The object will stop at a point when all of its kinetic energy is converted to potential energy. The potential energy then makes the object move in the opposite direction, converting its potential energy back to kinetic energy. (c) At the lowest point of the curve, all of the energy is in the kinetic form (peak velocity/momentum), which pushes the object up to the left side of the curve. (d) As the object goes up on the curve, its kinetic energy again converts to potential energy, until all is in the form of potential energy. Then, the object would stop and then slide back as guided by its potential energy. Since the surface is frictionless, the total energy remains constant throughout these repeated movements.

is generated from a parametric distribution. The momentum matches the dimensionality of $\boldsymbol{\theta}$ as a vector of length k.

We write the Hamiltonian function as $H(\boldsymbol{\theta}, \mathbf{p})$, which consists of *potential* energy $U(\boldsymbol{\theta})$ and *kinetic* energy $K(\mathbf{p})$: $H(\boldsymbol{\theta}, \mathbf{p}) = U(\boldsymbol{\theta}) + K(\mathbf{p})$, where $\mathbf{p}$ and $\boldsymbol{\theta} \in \mathbb{R}^k$.

In statistical applications of MCMC, we are primarily interested in generating $\boldsymbol{\theta}$ from a given distribution $f(\boldsymbol{\theta})$. To do so, we let $U(\boldsymbol{\theta}) := -\log f(\boldsymbol{\theta})$. Such a designation would ensure $\boldsymbol{\theta}$ generated from the Hamiltonian function follows the desired distribution. For momentum, we typically assume $\mathbf{p} \sim N_k(0, \mathbf{M})$, where $\mathbf{M}$ is a user-specified covariance matrix.

Under this formulation, we have

$$H(\boldsymbol{\theta}, \mathbf{p}) = -\log f(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}. \tag{2}$$

Over time, HMC travels on trajectories that are governed by the following first-order differential equations, known as the *Hamiltonian equations*

$$\begin{aligned} \frac{d\mathbf{p}}{dt} &= -\frac{\partial H(\boldsymbol{\theta}, \mathbf{p})}{\partial \boldsymbol{\theta}} = -\frac{\partial U(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta}), \\ \frac{d\boldsymbol{\theta}}{dt} &= \frac{\partial H(\boldsymbol{\theta}, \mathbf{p})}{\partial \mathbf{p}} = \frac{\partial K(\mathbf{p})}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}, \end{aligned} \tag{3}$$

where $\nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta})$ is the gradient of the log posterior density. A solution to the Hamiltonian equations is a function that defines the path of $(\boldsymbol{\theta}, \mathbf{p})$ from which specific values of $\boldsymbol{\theta}$ could be sampled. Within an MCMC iteration, we sample a value $\boldsymbol{\theta}$ from this path. The randomness of the MCMC samples comes from the momentum $\mathbf{p} \sim N_k(0, \mathbf{M})$ and the specific $\boldsymbol{\theta}$ value we choose.

3.3. Solving the Hamiltonian Differential Equations

Solving the Hamiltonian equations, therefore, becomes a critical step in HMC simulation. A standard approach for solving differential equations is Euler's method, which produces a discrete function that approximates the solution at each time t. Values of $(\boldsymbol{\theta}, \mathbf{p})$ that satisfy the Hamiltonian equations would be legitimate values for the HMC. But as Neal (2011) have noted, errors tend to accumulate in Euler's method, especially after a larger number of steps. In HMC, one often has to take a larger number of steps to ensure the new proposal is sufficiently far from the location of the previous sample.

The *leapfrog* method is a good alternative to the standard Euler's method for approximating the solutions to Hamiltonian equations (Ruth 1983). The leapfrog algorithm modifies Euler's method by using a discrete step size ϵ individually for $\mathbf{p}$ and $\boldsymbol{\theta}$, with a full step ϵ in $\boldsymbol{\theta}$ sandwiched between two half-steps $\epsilon/2$ for $\mathbf{p}$,

$$\begin{aligned} \mathbf{p}(t + \epsilon/2) &= \mathbf{p}(t) + (\epsilon/2)\nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta}(t)), \\ \boldsymbol{\theta}(t + \epsilon) &= \boldsymbol{\theta}(t) + \epsilon\mathbf{M}^{-1}\mathbf{p}(t + \epsilon/2), \\ \mathbf{p}(t + \epsilon) &= \mathbf{p}(t + \epsilon/2) + (\epsilon/2)\nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta}(t + \epsilon)). \end{aligned} \tag{4}$$

For HMC, multiple leapfrog steps are typically required to move a sufficient distance to the next proposal. Research has shown that discrete approximations remain accurate, even after many steps. The stability of the leapfrog algorithm is due to the leapfrog's symplectic property. (Channell and Scovel 1990; Betancourt 2017). Symplecticity ensures that the volume of the support is preserved when mapping from one point to another, such as through one or more consecutive iterations of the leapfrog algorithm (Neal 2011).

For a given momentum vector $\mathbf{p}$ within an HMC iteration, the path defined by the Hamiltonian equations is deterministic. Proposals generated from an exact solution of these equations, if achievable, would always be accepted. But since our solution from the leapfrog is an approximation, a Metropolis style accept/reject step is added to ensure the newly generated proposal does not deviate too far from the specified Hamiltonian $H(\boldsymbol{\theta}, \mathbf{p})$. The acceptance rate of HMC proposals is therefore less than 100%, but generally higher than that of the Metropolis algorithm.

3.4. HMC Algorithm

The flowchart in Figure 2 shows the key steps in HMC. Initial values for $\boldsymbol{\theta}$ and $\mathbf{p}$ are required to start the algorithm. With $\boldsymbol{\theta}^{(0)}$ and $\mathbf{p}^{(0)}$ specified, the leapfrog algorithm is used to find approximate solutions to the Hamiltonian equations. The leapfrog solutions define the path of $(\boldsymbol{\theta}, \mathbf{p})$ over time within an iteration.

Typically, multiple steps, each of length ϵ, are taken to generate an HMC proposal. Parameter L represents the number of steps. While L is often fixed to a positive integer value, some randomness can be introduced to ensure a valid exploration of the space of $(\boldsymbol{\theta}, \mathbf{p})$. A generic HMC is given in Algorithm 2.

Algorithm 2 Hamiltonian Monte Carlo

procedure HMC($\boldsymbol{\theta}^{(0)}, \log f(\boldsymbol{\theta}), \mathbf{M}, N, \epsilon, L$)
  Calculate $\log f(\boldsymbol{\theta}^{(0)})$
  for $t = 1, \ldots, N$ **do**
    $\mathbf{p} \leftarrow N(0, \mathbf{M})$
    $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}, \tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t-1)}, \tilde{\mathbf{p}} \leftarrow \mathbf{p}$
    for $i = 1, \ldots, L$ **do**
      $\tilde{\boldsymbol{\theta}}, \tilde{\mathbf{p}} \leftarrow \text{Leapfrog}(\tilde{\boldsymbol{\theta}}, \tilde{\mathbf{p}}, \epsilon, \mathbf{M})$
    end for
    $\alpha \leftarrow \min\left(1, \frac{\exp(\log f(\tilde{\boldsymbol{\theta}}) - \frac{1}{2}\tilde{\mathbf{p}}^T\mathbf{M}^{-1}\tilde{\mathbf{p}})}{\exp(\log f(\boldsymbol{\theta}^{(t-1)}) - \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p})}\right)$
    With probability α, $\boldsymbol{\theta}^{(t)} \leftarrow \tilde{\boldsymbol{\theta}}$ and $\mathbf{p}^{(t)} \leftarrow -\tilde{\mathbf{p}}$
  end for
  return $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(N)}$
  function LEAPFROG($\boldsymbol{\theta}^*, \mathbf{p}^*, \epsilon, \mathbf{M}$)
    $\tilde{\mathbf{p}} \leftarrow \mathbf{p}^* + (\epsilon/2)\nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta}^*)$
    $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^* + \epsilon\mathbf{M}^{-1}\tilde{\mathbf{p}}$
    $\tilde{\mathbf{p}} \leftarrow \tilde{\mathbf{p}} + (\epsilon/2)\nabla_{\boldsymbol{\theta}} \log f(\tilde{\boldsymbol{\theta}})$
    return $\tilde{\boldsymbol{\theta}}, \tilde{\mathbf{p}}$
  end function
end procedure

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 our HMC samples provide a valid representation of the posterior distribution. If we denote the transition probability from $\boldsymbol{\theta}^{(t)}$ to $\boldsymbol{\theta}^{(t+1)}$ as $T(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}^{(t+1)})$, then detailed balance requires that $f(\boldsymbol{\theta}^{(t)})T(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}^{(t+1)}) = f(\boldsymbol{\theta}^{(t+1)})T(\boldsymbol{\theta}^{(t+1)}, \boldsymbol{\theta}^{(t)})$. The HMC

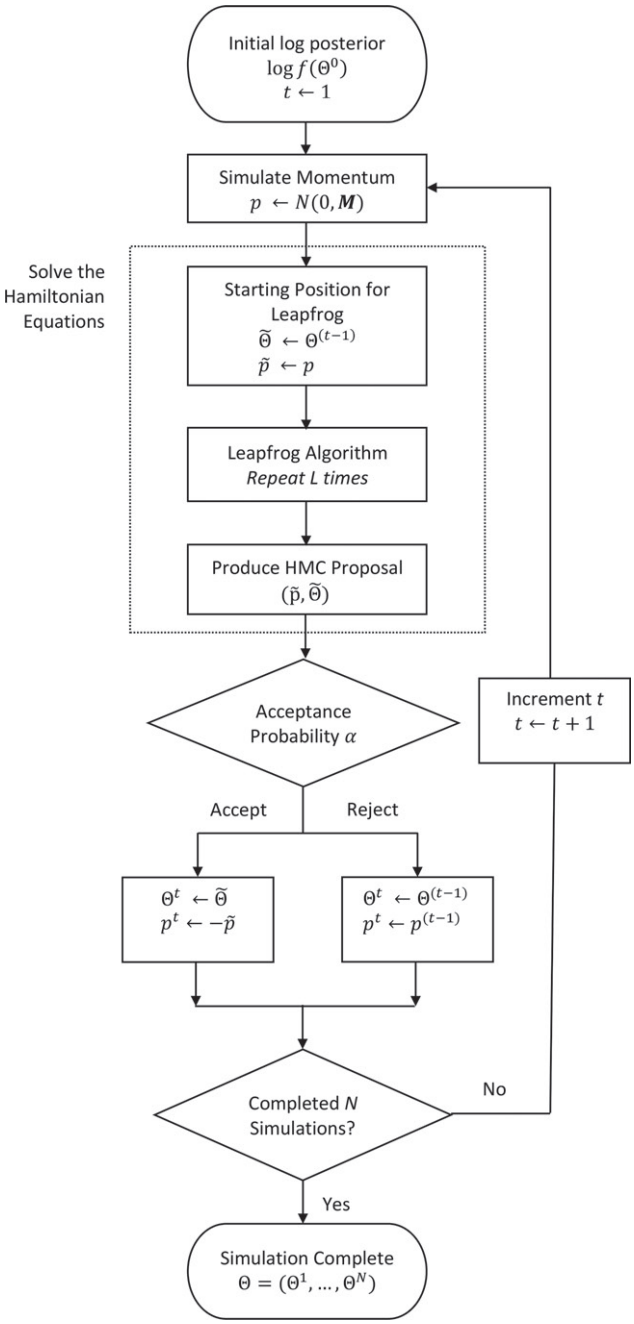

Figure 2. Main steps of the Hamiltonian Monte Carlo Method.

transition probability includes two components to ensure that detailed balance and reversibility hold true:

1. the accept/reject step, and
2. the negation of the momentum after the final leapfrog step.

The negated momentum illustrates the reversibility of HMC transitions, which can be demonstrated by stepping through the leapfrog from the proposed state to the original state. Tierney (1994) described the theoretical requirements for MCMC algorithms in general, while Betancourt (2017) provided a detailed exposition specific to HMC.

In Section 4, we describe a general-purpose function `hmc` in our proposed package. Within the package, the gradient functions for commonly used generalized linear mixed effect models under the default priors are provided. The `hmc` function can also take user-defined posterior density and gradient functions for non-standard statistical models. In situations where analytical derivation of gradient functions is infeasible, one could consider using numerical auto-differencing functions. Automated differencing libraries capable of calculating the gradient exactly such as the Stan math library (Carpenter et al. 2015), also called Autodiff, are appropriate for direct use in HMC applications.

3.5. HMC Tuning for Improved Efficiency

The efficiency of an HMC algorithm can be improved through parameter tuning and reparameterization. HMC tuning involves selection and adjustment of the various HMC parameters. Two parameters that need to be specified are the step size ϵ and the number of leapfrog steps L. Elements in the covariance matrix $\mathbf{M}$ may also be adjusted from the default identity matrix for efficiency improvement.

It is generally a good practice to set ϵ to a smaller value relative to the magnitude of the parameter of interest. A smaller ϵ results in closer approximations and thus higher acceptance rates. But a small ϵ must be coupled with a large L to ensure the trajectory length ϵL is large enough to move the simulated parameter to a distant point in the distribution. On the other hand, if ϵL is too large the trajectory is likely to circle back, causing waste in simulation. To tune ϵ and L is to find the right combinations of these values, which are usually chosen via monitoring the acceptance rate. Neal (2011) suggested an optimal acceptance rate is approximately 65%. At the same time, it is often helpful to examine the trace plots of the MCMC samples for signs of autocorrelation. Slow-moving chains with stronger autocorrelation often indicate insufficient ϵL. While ϵ and L can be tuned jointly, most analysts choose to select the step size first, then under a given step size, they fine-tune the number of steps per leapfrog L.

Additional adjustments may be made to the tuning parameters beyond these basic steps. For example, one could use different values of ϵ for each of the k parameters in $\boldsymbol{\theta}$ to increase the sampling efficiency. The `hmc` function in `hmclearn` allows setting ϵ to a vector instead of a single number to give analysts the flexibility to use different step sizes for different parameters. The parameter for the number of steps L must be a natural number. However, randomly chosen L could be used to guard against periodicity of the Markov chain. The step size ϵ may also be randomized. In the `hmc` function, random ϵ and L can be automatically applied via parameter setting. A useful algorithm known as the No U-Turn Sampler (NUTS) automatically selects L for each sample; NUTS is a commonly used alternative to manual parameter tuning (Hoffman and Gelman 2014).

The efficiency of sampling in the standard HMC algorithm can also be improved for multivariate models when the parameters have an orthogonal basis. One common method of ensuring an orthogonal basis involves applying QR decomposition (Voss 2013) . In many statistical models, especially linear models, the design matrix helps to define the model itself and is central to the model fitting computation. In HMC, QR decomposition is often applied to the design matrix to create the orthogonal basis for sampling. Applying this transformation in practice can improve

the computational efficiency of HMC for many models (Team 2017). After the simulation is complete, the MCMC samples are transformed back to the original basis for inference.

4. A Package for Learning HMC

HMC presents considerable challenges to beginners attempting to learn the algorithm. First, the method can be difficult to comprehend because its idea originated from physics applications of the Hamiltonian equations. Second, it is often difficult to learn the inner working of HMC from programs such as Stan, because they are not designed as teaching tools. In fact, Stan specifies models in a probabilistic syntax and shields users from the actual HMC steps.

In this article, we present an R package `hmclearn` to provide users with the software tools to *learn* the intricacies of the HMC algorithm, through explicit specification of log posterior and gradient functions, as well as parameter tuning. It is designed to give user a hands-on experience for implementing HMC analysis for a broad class of statistical models. Once users have understood and mastered the essential HMC steps, they could go on to write their own code for specific applications. To download `hmclearn`, go to *https://cran.r-project.org/web/packages/hmclearn/index.html*.

The core function in `hmclearn` is `hmc`, which is a general-purpose function for MCMC sample generation by using the HMC method. This function takes user-defined log posterior and gradient functions as inputs and produces MCMC samples. Here we do not ask for an explicit specification of prior $f(\boldsymbol{\theta})$ as an input function. Instead, we let users define their log posterior $\log f(\boldsymbol{\theta}|\boldsymbol{y}) = \log f(\boldsymbol{y}|\boldsymbol{\theta}) + \log f(\boldsymbol{\theta})$, which includes $f(\boldsymbol{\theta})$. Such a design reduces the number of required input functions, while preserving users' flexibility in choosing different priors.

Other input parameters to `hmc` include the number of samples N, the step size ϵ, the number of leapfrog steps L, and the Mass matrix $\mathbf{M}$. These are the essential elements to start an HMC simulation, but the user will typically need to adjust at least some of these parameters to tailor the simulation to their specific applications. Users are required to provide their own starting values for $\boldsymbol{\theta}$ when using the `hmc` function for their own applications. Examples of log posterior and gradient functions are provided in `hmclearn` for various generalized linear mixed effect models, which can be used as templates for less standard models.

Running multiple MCMC chains is often desirable to determine if each chain converges to the same distribution of $\boldsymbol{\theta}$. Since modern computers almost universally have multiple core processors, parallel processing can be an efficient way to run multiple chains at the same time. To that end, `hmclearn` includes parameters to enable parallel processing as well as multiple chains.

Finally, a variety of Bayesian graphical functions are provided based on the `bayesplot` package (Gabry and Mahr 2016). Functionalities incorporated in `hmclearn` include trace plots, histograms, density plots, and credible interval plots. The integrated functions comprise the core diagnostic plotting functions typical for MCMC applications. Additional diagnostics can be programmed directly or called based on the output of the `hmc` function.

5. HMC in Statistical Models

5.1. A General Process

In this section, we discuss the general steps of HMC implementation in statistical models. We describe the process through examples of generalized linear models. The major steps required to fit a statistical model are summarized in Figure 3. Following the steps illustrated in the diagram, one could generate HMC samples with user-specified posterior and gradient functions, by using the `hmc` function in the `hmclearn` package.

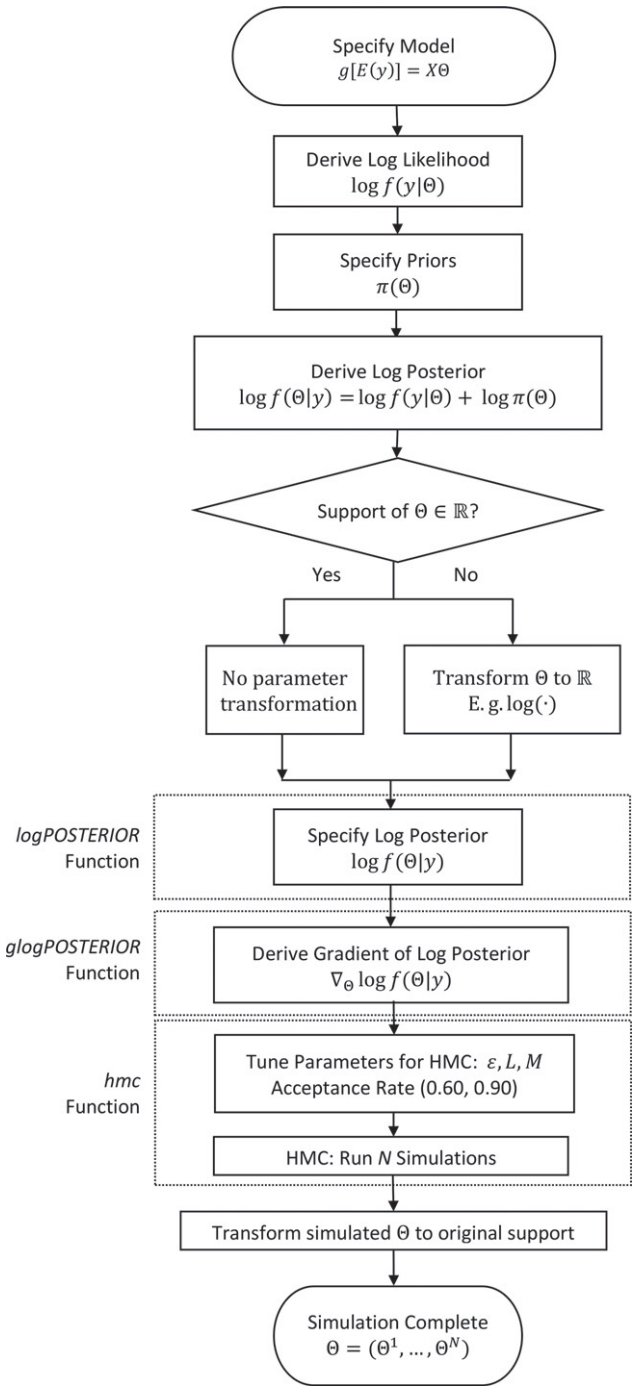

Figure 3. Major steps of HMC implementation.

5.2. Examples

We present three examples to illustrate how to fit various linear models using HMC. Our notation for these examples reflects the programming of the sample log posterior and gradient functions in **hmclearn**. This programming uses matrix and vector multiplication instead of for loops, which can be computationally slow in R.

5.2.1. Example 1: Linear Regression

We consider a linear regression model $y_i = \mathbf{x}_i^T\boldsymbol{\beta} + \epsilon_i$, where y_i is the response for the ith subject, $i = 1, \ldots, n$, and $\mathbf{y} = (y_1, \ldots, y_n)^T$ is a vector of responses. The covariate values for the ith subject are $\mathbf{x}_i^T = (x_{i0}, \ldots, x_{iq})$, where x_{i0} is frequently set to one as an intercept term for all subjects. We write the full design matrix as $\mathbf{X} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T)^T \in \mathbb{R}^{n\times(q+1)}$. The regression coefficients for the q covariates plus an intercept are written as $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_q)^T$. The error term for each subject is ϵ_i. All error terms $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^T$ are assumed to be independent and normally distributed with mean zero and constant variance σ_ϵ^2.

The log-likelihood for linear regression, omitting the constants, can be written as

$$\log f(\mathbf{y}|\boldsymbol{\beta}, \sigma_\epsilon^2) \propto -n\log\sigma_\epsilon - \frac{1}{2\sigma_\epsilon^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}).$$

We specify a multivariate normal prior for $\boldsymbol{\beta}$ with covariance matrix $\sigma_\beta^2\mathbf{I}$ where σ_β^2 is a hyperparameter set by the analyst, and an inverse gamma (IG) prior for σ_ϵ^2. The IG prior has hyperparameters a and b, which are also set by the analyst. We write

$$f(\boldsymbol{\beta}|\sigma_\beta^2) \propto \exp\left(-\frac{\boldsymbol{\beta}^T\boldsymbol{\beta}}{2\sigma_\beta^2}\right) \quad \text{and}$$
$$f(\sigma_\epsilon^2|a, b) = \frac{b^a}{\Gamma(a)}(\sigma_\epsilon^2)^{-a-1}\exp\left(-\frac{b}{\sigma_\epsilon^2}\right).$$

The support of σ_ϵ^2 is $(0, \infty)$. We apply a logarithmic transformation to expand the support to $\mathbb{R}$. We have

$$\gamma = \log\sigma_\epsilon^2, \quad \sigma_\epsilon^2 = g^{-1}(\gamma) = e^\gamma,$$
$$f(\gamma|a, b) = \frac{b^a}{\Gamma(a)}\exp\left(-a\gamma - \frac{b}{e^\gamma}\right),$$
$$\log f(\gamma|a, b) \propto -a\gamma - be^{-\gamma}.$$

The log posterior is proportional to the log-likelihood plus the log prior,

$$\log f(\boldsymbol{\beta}, \gamma|\mathbf{y}, \mathbf{X}, \sigma_\beta^2, a, b) \propto -\left(\frac{n}{2} + a\right)\gamma - \frac{e^{-\gamma}}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \frac{\boldsymbol{\beta}^T\boldsymbol{\beta}}{2\sigma_\beta^2} - be^{-\gamma}.$$

The parameters of interest are defined as $\boldsymbol{\theta} := (\beta_0, \ldots, \beta_q, \gamma)^T$, where $k = q + 2$. To fit this model using `hmc`, the user must provide a function for the log posterior where the first function parameter is a vector for the parameters of interest $\boldsymbol{\theta}$. Additional function parameters can be included for the data and hyperparameters. An example log posterior function for this model and specification of priors is included in `hmclearn`.

The Hamiltonian function (2) is composed of the log posterior and the log density function of the momentum, where $\mathbf{p} \sim N_k(0, \mathbf{M})$. Writing the Hamiltonian function for our linear regression model is straightforward once the log posterior is developed,

$$H(\boldsymbol{\theta}, \mathbf{p}) = H(\boldsymbol{\beta}, \gamma, \mathbf{p}) \propto \log f(\boldsymbol{\beta}, \gamma|\mathbf{y}, \mathbf{X}, \sigma_\beta^2, a, b) + \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}.$$

With the Hamiltonian function explicitly defined, we can write the Hamiltonian equations (3) for this particular model.

The steps of the leapfrog algorithm are integrated with `hmc` in a self-contained function. This function requires, as an input, a separate standalone function that returns a vector for the gradient of the log posterior. As with the log posterior function, the first function parameter must be a vector for $\boldsymbol{\theta}$. The gradient functions for the model in this example are also included in `hmclearn`,

$$\nabla_{\boldsymbol{\beta}}\log f(\boldsymbol{\beta}, \gamma|\mathbf{y}, \mathbf{X}, \sigma_\beta^2, a, b) \propto e^{-\gamma}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \boldsymbol{\beta}/\sigma_\beta^2,$$
$$\nabla_\gamma\log f(\boldsymbol{\beta}, \gamma|\mathbf{y}, \mathbf{X}, \sigma_\beta^2, a, b) \propto -\left(\frac{n}{2} + a\right) + \frac{e^{-\gamma}}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + be^{-\gamma}.$$

We now have everything we need to solve the Hamiltonian equations via the leapfrog algorithm and generate samples for the posterior $f(\boldsymbol{\theta})$. The main `hmc` function handles the details of the HMC sample generation process for the user. A description of the function parameters is in Section **A.1** of the appendix. Additional programming details are provided with the **hmclearn** package, including detailed vignettes with additional examples.

For a numerical example we use the *warpbreaks* dataset (Tippett 1950), which is one of the sample datasets included with base R. In this example, we estimate the associations between the yarn's type of wool and tension and the number of warp breaks per loom. We write the model as follows

$$\text{Breaks}_i = \beta_0 + \beta_1\text{woolB}_i + \beta_2\text{tensionM}_i + \beta_3\text{tensionH}_i + \beta_4\text{woolB}_i : \text{tensionM}_i + \beta_5\text{woolB}_i : \text{tensionH}_i + \epsilon_i,$$

where $y_i := \text{Breaks}_i$ and the ith row of $\mathbf{X}$ is $\mathbf{x}_i^T = (1, \text{woolB}_i, \text{tensionM}_i, \text{tensionH}_i, \text{woolB}_i : \text{tensionM}_i, \text{woolB}_i : \text{tensionH}_i)$.

To fit this model using `hmc`, we must first specify the initial values of $\boldsymbol{\theta}$ for the MCMC chain. The initial values are provided as a vector of length $k = 7$, including 6 for $\boldsymbol{\beta}$ and 1 for γ. We use the default hyperparameters for the sample log posterior and gradient functions in `hmclearn`, such that $\sigma_\beta^2 = 1\text{e}3$ and $a = b = 1\text{e}{-4}$.

The HMC simulation takes approximately 6 sec to run on a 2015 Macbook Pro with a 2.5GHz processor. Users have a number of options to summarize and visualize the HMC samples. The generic `summary` function provides quantiles from the posterior samples in a table. Many data visualization options

are available through direct integration with the `bayesplot` package (Gabry and Mahr 2016). Graphical options for visualizing the posterior samples include histograms, density plots, and credible interval plots. General MCMC diagnostics such as trace plots, autocorrelation plots, and $\hat{R}$ statistics are also readily available. Additional customized analyses can be performed using the posterior sample output from `hmc`.

The marginal posterior sample distributions for $f(\boldsymbol{\theta})$ are found to be well-behaved and similar to frequentist estimates. The R code for fitting the model is presented in Section **A.2** of the appendix.

5.2.2. Example 2: Logistic Regression

We consider a logistic regression model $P(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta}) = \left[1 + \exp(-\mathbf{x}_i^T\boldsymbol{\beta})\right]^{-1}$, where y_i is the binary response for the ith subject $i = 1, \ldots, n$, and $\mathbf{y} = (y_1, \ldots, y_n)^T$ is a vector of responses for all subjects. The covariate values for the ith subject are $\mathbf{x}_i^T = (x_{i0}, \ldots, x_{iq})$, where x_{i0} is frequently set to one as an intercept term for all subjects. Frequently, x_{i0} is set to one for all individuals as an intercept term. We write the full design matrix as $\mathbf{X} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T)^T \in \mathbb{R}^{n\times(q+1)}$. The regression coefficients for q covariates plus an intercept are a vector $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_q)^T$.

The log-likelihood for the logistic regression model is

$$\log f(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) = \boldsymbol{\beta}^T\mathbf{X}^T(\mathbf{y} - \mathbf{1}_n) - \mathbf{1}_n^T[\log(1 + e^{-\mathbf{x}_i^T\boldsymbol{\beta}})]_{n\times 1},$$

where $\boldsymbol{\beta}$ is the regression coefficient vector and the parameter we intend to estimate, and $[\log(1 + e^{-\mathbf{x}_i^T\boldsymbol{\beta}})]_{n\times 1}$ indicates an $n \times 1$ vector $\forall i = 1, \ldots, n$. We specify a multivariate normal prior for $\boldsymbol{\beta}$ with covariance matrix $\sigma_\beta^2\mathbf{I}$, where σ_β^2 is a hyperparameter set by the analyst.

The log posterior is proportional to the sum of the log-likelihood and log prior of $\boldsymbol{\beta}$. Excluding constants, we write the log posterior as

$$\log f(\boldsymbol{\beta}|\mathbf{y}, \mathbf{X}, \sigma_\beta^2) \propto \boldsymbol{\beta}^T\mathbf{X}^T(\mathbf{y} - \mathbf{1}_n) - \mathbf{1}_n^T[\log(1 + e^{-\mathbf{x}_i^T\boldsymbol{\beta}})]_{n\times 1} - \frac{\boldsymbol{\beta}^T\boldsymbol{\beta}}{2\sigma_\beta^2}.$$

The parameters of interest are defined as $\boldsymbol{\theta} := \boldsymbol{\beta} = (\beta_0, \ldots, \beta_q)^T$, where $k = q + 1$. To fit this model using `hmc`, the user must provide a log posterior function containing the parameters of interest $\boldsymbol{\theta}$, the observed data, and possibly additional hyperparameters. The log posterior function for this model and the specification of priors are described in **hmclearn**.

The Hamiltonian function (2) is composed of the log posterior and the log density function of the momentum $\mathbf{p} \sim N_k(0, \mathbf{M})$. Writing the Hamiltonian function for our example model is straightforward once the log posterior is specified,

$$H(\boldsymbol{\theta}, \mathbf{p}) = H(\boldsymbol{\beta}, \mathbf{p}) \propto \log f(\boldsymbol{\beta}|\mathbf{y}, \mathbf{X}, \sigma_\beta^2) + \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}.$$

With the Hamiltonian function explicitly defined, we can write the Hamiltonian equations for this particular model. To generate samples from $f(\boldsymbol{\theta})$, we then use the leapfrog method to find a discrete approximation. The leapfrog steps are integrated with `hmc` in a self-contained function, using user-supplied gradients.

$$\nabla_{\boldsymbol{\beta}} \log f(\boldsymbol{\beta}|\mathbf{y}, \mathbf{X}, \sigma_\beta^2) \propto \mathbf{X}^T\left(\mathbf{y} - \mathbf{1}_n + \left[\frac{e^{-\mathbf{x}_i^T\boldsymbol{\beta}}}{1 + e^{-\mathbf{x}_i^T\boldsymbol{\beta}}}\right]_{n\times 1}\right) - \boldsymbol{\beta}/\sigma_\beta^2.$$

With the gradient function specified, we can solve the Hamiltonian equations via the leapfrog algorithm, and generate posterior samples following $f(\boldsymbol{\theta})$. The main function `hmc` handles the implementation of the HMC sample generation process.

We analyzed data of 189 births at the U.S. hospital (Hosmer, Lemeshow, and Sturdivant 1989) to examine the risk factors of low birth weight. Data are available from the `MASS` package (Venables and Ripley 2002). We prepare the data for analysis as noted in the text.

The logistic regression model formulation for this application is

$$\begin{aligned}\text{logit}\,[P(\text{low}_i = 1)] = {} & \beta_0 + \beta_1\text{age}_i + \beta_2\text{lwt}_i + \beta_3\text{race2black}_i \\ & + \beta_4\text{race2other} + \beta_5\text{smoke}_i + \beta_6\text{ptd}_i \\ & + \beta_7\text{ht}_i + \beta_8\text{ui}_i + \beta_9\text{ftv21}_i \\ & + \beta_{10}\text{ftv22plus}_i.\end{aligned}$$

Here $\mathbf{x}_i^T = (1, \text{age}_i, \text{lwt}_i, \text{race2black}_i, \text{race2other}_i, \text{smoke}_i, \text{ptd}_i, \text{ht}_i, \text{ui}_i, \text{ftv21}_i, \text{ftv22plus}_i)$, where the elements indicate the mother's age in years *age*, mother's weight in pounds at last menstrual period *lwt*, black *race2black* and other races *race2other*, smoking during pregnancy *smoke*, premature birth *ptd*, hypertension *ht*, presence of uterine irritability *ui*, one physician visit during the first trimester *ftv21*, and two or more physician visits during the first trimester *ftv22plus*.

To fit this model using `hmc`, the user needs to set the initial values for $\boldsymbol{\beta}$, a vector of length $k = 11$, as well as the value of the hyperparameter σ_β^2, which we set at $1e3$. In this example, we set the step size parameter ϵ to different values for continuous and dichotomous variables.

The HMC simulation takes about 6 sec to run on a 2015 Macbook Pro with a 2.5GHz processor. The R code for fitting the model is presented in Section **A.3** of the appendix. The marginal posterior sample distributions for $f(\boldsymbol{\theta})$ are found to be well-behaved with central locations similar to frequentist estimates.

5.2.3. Example 3: Poisson regression with random subject effects

Finally, we consider a random effect model for count data

$$g[E(\mathbf{y}_i|u_i)] = \mathbf{X}_i\boldsymbol{\beta} + \mathbf{z}_i u_i,$$

for $i = 1, \ldots, n$ subjects, where each subject's response vector $\mathbf{y}_i = (y_{i1}, \ldots, y_{id})^T$ contains $j = 1, \ldots, d$ observations. Each individual has a subject-specific random intercept parameter u_i, and $\mathbf{u} = (u_1, \ldots, u_n)^T$. The fixed effects design matrix $\mathbf{X}_i = (\mathbf{x}_{i1}^T, \ldots, \mathbf{x}_{id}^T)^T \in \mathbb{R}^{d\times(q+1)}$, where the jth row of $\mathbf{X}_i$ contains the $q + 1$ covariate values of that observation, including a common intercept. The fixed effects regression coefficients for q covariates and a global intercept are a vector $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_q)^T$. The random intercept vector is $\mathbf{z}_i = (z_{i1}, \ldots, z_{id})^T = \mathbf{1}_d$, The distribution of $\mathbf{y}_i$ conditional on u_i follows a Poisson distribution with a log link function, where $\log[E(\mathbf{y}_i|u_i)] = \mathbf{X}_i\boldsymbol{\beta} + \mathbf{z}_i u_j$.

The subject-level response vectors are combined in a single vector, $\mathbf{y} = (\mathbf{y}_i^T, \ldots, \mathbf{y}_n^T)^T \in \mathbb{R}^{nd\times 1}$. The full fixed-effects design matrix for all subjects is $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_n)^T \in \mathbb{R}^{nd\times(q+1)}$, and the random effects design matrix is $\mathbf{Z} = \mathbf{I}_n \otimes \mathbf{1}_d \in \mathbb{R}^{nd\times n}$. The log-likelihood for the Poisson mixed effects model, omitting constants, can be written as

$$\log f(\mathbf{y}|\mathbf{X}, \mathbf{Z}, \boldsymbol{\beta}, \mathbf{u}) \propto -\mathbf{1}_{nd}^T \left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+z_{ij}u_i}\right]_{nd\times 1} + \mathbf{y}^T(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u}),$$

where $\boldsymbol{\beta}$ is the fixed-effect coefficient vector, u_i is the random intercept, and $\left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+z_{ij}u_i}\right]_{nd\times 1}$ is an $nd \times 1$ vector $\forall i = 1, \ldots, n$ and $j = 1, \ldots d$. We specify multivariate normal priors $\boldsymbol{\beta}|\sigma_\beta^2 \sim N(0, \sigma_\beta^2\mathbf{I})$ and $\mathbf{u} \sim N(0, \mathbf{G})$, where σ_β^2 is a hyperparameter set by the analyst and $\mathbf{G}$ is parameterized for efficient Bayesian computation.

We parameterize the covariance matrix of $\mathbf{G}$ for efficient sampling of hierarchical models such that $\mathbf{G}^{1/2} := \lambda\mathbf{I}\boldsymbol{\tau}$, where $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)^T \sim N(0, \mathbf{I}_n)$ (Betancourt and Girolami 2013). For λ, we assign a 2-parameter half-t prior per the recommendation of Gelman (2006) for hierarchical models.

One final parameter transformation is necessary before applying HMC. Since the support of λ is $(0, \infty)$, we apply a logarithmic transformation to expand the support to $\mathbb{R}$. We write

$$\xi = \log\lambda, \quad \lambda = g^{-1}(\xi) = e^\xi,$$

$$f(\xi|a, b) \propto \left(1 + \frac{1}{\nu_\xi}\left(\frac{e^\xi}{A_\xi}\right)^2\right)^{-(\nu_\xi+1)/2} e^\xi,$$

$$\log f(\xi|a, b) \propto -\frac{\nu_\xi + 1}{2}\log\left(1 + \frac{1}{\nu_\xi}\left(\frac{e^\xi}{A_\xi}\right)^2\right) + \xi,$$

where ν_ξ and A_ξ are hyperparameters set by the analyst.

Omitting constants, we write the log posterior as

$$\begin{aligned}\log f(&\boldsymbol{\beta}, \boldsymbol{\tau}, \xi|\mathbf{y}, \mathbf{X}, \mathbf{Z}, \sigma_\beta^2, \nu_\xi, A_\xi)\\ &\propto -\mathbf{1}_{nd}^T\left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+e^\xi z_{ij}\tau_i}\right]_{nd\times 1} + \mathbf{y}^T(\mathbf{X}\boldsymbol{\beta} + e^\xi\mathbf{Z}\boldsymbol{\tau}) - \frac{\boldsymbol{\beta}^T\boldsymbol{\beta}}{2\sigma_\beta^2}\\ &\quad - \frac{\nu_\xi + 1}{2}\log\left(1 + \frac{1}{\nu_\xi}\left(\frac{e^\xi}{A_\xi}\right)^2\right) + \xi - \frac{1}{2}\boldsymbol{\tau}^T\boldsymbol{\tau},\end{aligned}$$

where the parameters of interest can be written as $\boldsymbol{\theta} := (\beta_0, \ldots, \beta_q, \tau_1, \ldots, \tau_n, \xi)^T$, with $k = q + n + 2$.

Assuming $\mathbf{p} \sim N_k(0, \mathbf{M})$, we write the Hamiltonian function as,

$$\begin{aligned}H(\boldsymbol{\theta}, \mathbf{p}) = H(\boldsymbol{\beta}, \boldsymbol{\tau}, \xi, \mathbf{p}) &\propto \log f(\boldsymbol{\beta}, \boldsymbol{\tau}, \xi|\mathbf{y}, \mathbf{X}, \mathbf{Z}, \sigma_\beta^2, \nu_\xi, A_\xi)\\ &\quad + \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p},\end{aligned}$$

from which we can derive the Hamiltonian equations, and then use the leapfrog method to find approximate solutions.

We write the gradient functions for readers' convenience,

$$\begin{aligned}\nabla_{\boldsymbol{\beta}}&\log f(\boldsymbol{\beta}, \xi, \boldsymbol{\tau}|\mathbf{y}, \mathbf{X}, \mathbf{Z}, \sigma_\beta^2, \nu_\xi, A_\xi)\\ &\propto \mathbf{X}^T\left(-\left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+e^\xi z_{ij}\tau_i}\right]_{nd\times 1} + \mathbf{y}\right) - \boldsymbol{\beta}/\sigma_\beta^2,\\ \nabla_\xi&\log f(\boldsymbol{\beta}, \xi, \boldsymbol{\tau}|\mathbf{y}, \mathbf{X}, \mathbf{Z}, \sigma_\beta^2, \nu_\xi, A_\xi)\\ &\propto e^\xi\boldsymbol{\tau}^T\mathbf{Z}^T\left(-\left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+e^\xi z_{ij}\tau_i}\right]_{nd\times 1} + \mathbf{y}\right) - \frac{\nu_\xi + 1}{1 + \nu_\xi A_\xi^2 e^{-2\xi}} + 1,\\ \nabla_{\boldsymbol{\tau}}&\log f(\boldsymbol{\beta}, \xi, \boldsymbol{\tau}|\mathbf{y}, \mathbf{X}, \mathbf{Z}, \sigma_\beta^2, \nu_\xi, A_\xi)\\ &\propto e^\xi\mathbf{Z}^T\left(-\left[e^{\mathbf{x}_{ij}^T\boldsymbol{\beta}+e^\xi z_{ij}\tau_i}\right]_{nd\times 1} + \mathbf{y}\right) - \boldsymbol{\tau}.\end{aligned}$$

For numerical example, we consider data generated by a study on gopher tortoises (Ozgul et al. 2009; Fox et al. 2015; Bolker 2018). The mortality of the tortoise populations is measured by the number of shells. We estimate the associations of the number of shells to year (2004, 2005, 2006) and seroprevalence of bacterium *Mycoplasma agassizii*. The random effects are the intercepts for each of $n = 10$ sites in Florida. Each site has $d = 3$ observations, one for each year. The fixed effects are a global intercept, two indicator variables for the three years, and seroprevalence of *M. agassizii*.

The poisson mixed effects model can be written as

$$\begin{aligned}\log[E(\mathbf{shells})] \propto \sum_{i=1}^{10}\sum_{j=1}^{3}\Big[&-e^{[1,I(2005)_{ij},I(2006)_{ij},\text{prev}_{ij}]\boldsymbol{\beta}+e^\xi z_{ij}\tau_i} +\\ &y_{ij}\left([1, I(2005)_{ij}, I(2006)_{ij}, \text{prev}_{ij}]\boldsymbol{\beta} + e^\xi z_{ij}\tau_i\right)\Big] -\\ &\frac{\boldsymbol{\beta}^T\boldsymbol{\beta}}{2\sigma_\beta^2} - \frac{\nu_\xi + 1}{2}\log\left(1 + \frac{1}{\nu_\xi}\left(\frac{e^\xi}{A_\xi}\right)^2\right) + \xi - \frac{1}{2}\boldsymbol{\tau}^T\boldsymbol{\tau},\end{aligned} \tag{5}$$

where $\mathbf{y} := (\mathbf{shells}_1, ..., \mathbf{shells}_{10})^T$ and $\mathbf{shells}_i = (\text{shells}_1, \text{shells}_2, \text{shells}_3)^T$. The fixed effects design matrix is composed from $\mathbf{x}_{ij}^T = [1, I(2005)_{ij}, I(2006)_{ij}, \text{prev}_{ij}]$, and the random effects design matrix from $z_{ij} = 1$ for site i and 0 otherwise, for all observations $j = 1, 2, 3$.

To fit this model using `hmc`, we first specify the initial values of $\boldsymbol{\theta}$ in a vector of length $k = 15$ and use the default hyperparameters $\sigma_\beta^2 = 1e3, \nu_\xi = 1,$ and $A_\xi = 25$. The step sizes are selected as part of the tuning process.

The HMC simulation takes about 23 sec to run on a 2015 Macbook Pro with a 2.5GHz processor. The marginal posterior sample distributions for $f(\boldsymbol{\theta})$ are found to be well-behaved with central locations similar to frequentist estimates. The R code for fitting the model is presented in Section A.4 of the appendix.

In each of the above examples, we set $N = 2000$ HMC samples including a short burn-in period. The $\hat{R}$ statistics for each of the simulations is close to one, indicating that multiple chains converged to the same distribution for each example. Informally, the relatively low number of HMC simulations illustrates the efficiency benefits of this algorithm over traditional MCMC methods, such as the Metropolis algorithm, which often require many thousands of simulations to achieve a converge. A substantially larger number of simulations can push Metropolis to have a longer runtime than *hmc* in *hmclearn*, even when Metropolis is programmed in an efficient compiled language like C++.

6. Discussion

Since its becoming of a general-purpose computational method in the early 1990s, MCMC has fundamentally changed the landscape of Bayesian data analysis (Robert and Casella 2011). Previous confinement to the conjugate families of distributions has been lifted, and analysts have been freed from the burden of explicitly deriving the posteriors. Over the past three decades, tremendous progress has been made in refining the MCMC methods; models are becoming more flexible, algorithms more comprehensive, and software easier to use. Despite the progress, however, as analysts begin to take on increasingly complex statistical models, suboptimal efficiency has become a predominant concern, especially in models involving high dimensional parameters. In many of those situations, traditional MCMC is often too slow to be practically useful.

One of the newer variants of MCMC algorithms designed to address the efficiency problem is HMC. With the aid of the posterior gradient functions and the Hamiltonian equations, HMC tends to converge to regions of higher posterior density more quickly in comparison with Metropolis–Hastings. For example, Section 5.7.1 of Agresti (2015) uses `MCMCpack` (Martin, Quinn, and Park 2011) to fit a logistic regression model using Metropolis–Hastings. The compiled C++ code from this package is computationally advantageous compared to the fully R-based `hmclearn`. However, the run-time of this example with `MCMCpack` is approximately 2.6 min on a 2015 Macbook Pro with a 2.5GHz processor, versus 40 sec with `hmclearn` on the same computer, a 5x difference. The code for this example is detailed in the Logistic Regression vignette provided for `hmclearn` on CRAN. Analysts who require efficient HMC computation without the need for manually computing gradients or tuning parameters may consider `Stan` (Carpenter et al. 2017) for practical use. Stan translates BUGS-like (Spiegelhalter et al. 1999) code to C++ code for efficient computation.

These exciting developments, however, have not been translated into analytical practice. Many statistical practitioners remain unfamiliar with these powerful tools and, thus, hesitant to use them. Some have attempted to generate HMC samples by mimicking the Stan code, but in the absence of an in-depth understanding of the method and the ideas behind it, many analysts have not acquired a level of comfort to write HMC code for less standard analyses. We contend that the best way to learn a new method is through hands-on data analysis, with common statistical models on a familiar computational platform. With this in mind, we have put forward an introductory level description of HMC, not with the original terminology of classical mechanics, but in a more familiar language of statistics. We have disseminated the components of the HMC algorithm and discussed the implementation details, from prior specification, posterior and gradient function derivation, to solving the Hamiltonian differential equations, and to the tuning of HMC parameters. Herein, we present an R package `hmclearn` to help beginners to experiment with HMC in a familiar computing environment. The main function of this package, `hmc` is designed for general use – analysts could use it to produce MCMC samples by using user-supplied posterior functions. We have provided many concrete data examples, in the package as well as in this article, to help learners study and appreciate the inner workings of the algorithm. In comparison with commonly used Bayesian data analysis software such as Stan, our package `hmclearn` is designed primarily as a teaching tool. As such, the input functions require hands-on programming, so that the data-generation process is made more transparent to its users. This said, we would not trivialize the potential challenges in implementing a successful HMC program. The tuning of parameters, for example, often requires much practice and experience. Notwithstanding such limitations, we hope that this article provides an intuitive introduction of a powerful and yet intricate computational tool.

Supplementary Materials

Appendix: R code for HMC examples. (pdf file type)

R-package for learning HMC: R-package `hmclearn` contains a general-purpose function as well as utility functions for the model fitting methods described in the article. Example datasets and code are also made available in the package. The package `hmclearn` can be accessed at *https://cran.r-project.org/web/packages/hmclearn/index.html*.

Acknowledgments

The authors thank the Editor, Associate Editor, and two reviewers for their many insightful comments.

Funding

This work is partially supported by National Institutes of Health grants R01AA025208 and U24 AA026969.

References

Agresti, A. (2015), *Foundations of Linear and Generalized Linear Models*, Hoboken, NJ, Wiley. [412]

Betancourt, M. (2017), "A Conceptual Introduction to Hamiltonian Monte Carlo," arXiv preprint arXiv:1701.02434, pp. 1–60. *http://arxiv.org/abs/1701.02434* [403,405,406,407]

Betancourt, M. J., and Girolami, M. (2013), "Hamiltonian Monte Carlo for Hierarchical Models," *http://arxiv.org/abs/1312.0906* [411]

Bolker, B. (2018), "Glmm Worked Examples," *https://bbolker.github.io/mixedmodels-misc/ecostats_chap.html*. Accessed: 2019-12-09. [411]

Carlin, B. P., and Louis, T. A. (2008), *Bayesian Methods for Data Analysis*, Boca Raton, FL: CRC Press. [404]

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017), "Stan: A Probabilistic Programming Language," *Journal of Statistical Software*, 76, 1–32. [403,412]

Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., and Betancourt, M. (2015), "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++," arXiv preprint arXiv:1509.07164 . [407]

Channell, P. J., and Scovel, C. (1990), "Symplectic Integration of Hamiltonian Systems," *Nonlinearity* 3, 231. *http://stacks.iop.org/0951-7715/3/i=2/a=001* [406]

Chib, S., and Greenberg, E. (1995), "Understanding the Metropolis-Hastings Algorithm," *The American Statistician*, 49, 327–335. [404]

Fox, G. A., Negrete-Yankelevich, S., and Sosa, V. J. (2015), *Ecological Statistics: Contemporary Theory and Application*, Oxford, UK: Oxford University Press. [411]

Gabry, J., and Mahr, T. (2016), bayesplot: Plotting for Bayesian Models. R package version 1.1.0. *https://mc-stan.org/bayesplot* [408,410]

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013), *Bayesian Data Analysis*, Boca Raton, FL: CRC Press. [404]

Gelman, A., Lee, D. and Guo, J. (2015), "Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization," *Journal of Educational and Behavioral Statistics*, 40, 530–543. [403]

Gelman, A. (2006), "Prior Distributions for Variance Parameters in Hierarchical Models (Comment on Article by Browne and Draper)," *Bayesian Analysis*, 1, 515–534. [411]

Geman, S., and Geman, D. (1984), "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 721–741. [405]

Hastings, W. K. (1970), "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, 57, 97–109. *https://doi.org/10.1093/biomet/57.1.97* [404]

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, 1593–1623. [407]

Hosmer, D. W., Lemeshow, S., and Sturdivant, R. X. (1989), "The Multiple Logistic Regression Model," *Applied Logistic Regression*, 1, 25–37. [410]

Lunn, D. J., Thomas, A., Best, N., and Spiegelhalter, D. (2000), 'WinBUGS-a Bayesian Modelling Framework: Concepts, Structure, and Extensibility," *Statistics and Computing*, 10, 325–337. [405]

Martin, A. D., Quinn, K. M., and Park, J. H. (2011), "MCMCpack: Markov Chain Monte Carlo in R," *Journal of Statistical Software*, 42, 22. *http://www.jstatsoft.org/v42/i09/* [412]

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953), "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, 21, 1087–1092. [404]

Neal RM. (2011) "MCMC Using Hamiltonian Dynamics," in *Handbook of Markov Chain Monte Carlo*, eds. S. Brooks, A. Gelman, G. Jones, and X. L. Meng, Boca Raton, FL: CRC Press. [403,405,406,407]

Ozgul, A., Oli, M. K., Bolker, B. M., and Perez-Heydrich, C. (2009), "Upper Respiratory Tract Disease, Force of Infection, and Effects on Survival of Gopher Tortoises," *Ecological Applications*, 19, 786–798. [411]

Plummer, M. (2003), "Jags: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling," in Proceedings of the 3rd international workshop on distributed statistical computing, Vol. 124, Vienna, Austria, p. 1. [405]

Robert, C. (2007), *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, New York, Springer Science & Business Media. [405]

Robert, C., and Casella, G. (2011), "A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data," *Statistical Science*, 26, 102–115. [412]

Roberts, G. O., Gelman, A., Gilks, W. R. (1997), ' Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms," *The Annals of Applied Probability*, 7, 110–120. [404,405]

Ruth, R. D. (1983), "A Canonical Integration Technique," *IEEE Transactions on Nuclear Science*, 30, 2669–2671. [406]

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016), "Probabilistic Programming in Python Using PyMC3," *PeerJ Computer Science*, 2, e55. [403]

Spiegelhalter, D., Thomas, A., Best, N., and Gilks, W. (1999), "BUGS: Bayesian Inference Using Gibbs Sampling, version 0.5 (version ii)'. [412]

Team, S. D. (2017), Stan Modeling Language Users Guide and Reference Manual. *http://mc-stan.org* [408]

Tierney, L. (1994), "Markov Chains for Exploring Posterior Distributions," *Annals of Statistics*, 22, 1701–1728. *https://doi.org/10.1214/aos/1176325750* [404,407]

Tippett, L. H. C. (1950), *Technological Applications of Statistics*, New York: Wiley. [409]

Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D. and Blei, D. M. (2016), "Edward: A Library for Probabilistic Modeling, Inference, and Criticism," arXiv preprint arXiv:1610.09787 . [403]

Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, (4th edition), New York: Springer Science & Business Media. [410]

Voss, J. (2013), *An Introduction to Statistical Computing: A Simulation-Based Approach*, Hoboken, NJ: Wiley. [407]