

Current Samplers and Hamiltonian Monte Carlo: Notes for Users of CosmoSIS

Scott Ziegler

May-July 2019

These notes were produced by Scott Ziegler at Fermilab in the summer of 2019 for the purpose of examining the mathematics behind certain sampling modules in CosmoSIS. This research was supported in part by an appointment with the National Science Foundation (NSF) Mathematical Sciences Graduate Internship (MSGI) Program sponsored by the NSF Division of Mathematical Sciences. This program is administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and NSF. ORISE is managed for DOE by ORAU. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of NSF, ORAU/ORISE, or DOE.

The purpose of this document is to explore the mathematical and statistical background behind sampling methods implemented in CosmoSIS. The document is broadly split up into four sections: Random-walk Metropolis Hastings, Hamiltonian Monte Carlo, Gradient-free Hamiltonian Monte Carlo and MultiNest. Random-walk Metropolis Hastings and MultiNest were already in use in CosmoSIS at the time when this document was written, while the purpose of this project was in part to determine the viability of Hamiltonian Monte Carlo as a sampling method in CosmoSIS. These notes are meant to be self-contained but are not comprehensive or entirely rigorous. Further reading on selected topics will be highlighted throughout the notes.

1 Random-walk Metropolis Hastings

1.1 The Metropolis Hastings Algorithm

The general MH algorithm has been used widely for many years, so there is ample reading material to be found on the subject. A thorough summary of the MH algorithm is handled in [8], but we will provide a short summary below. Understanding the random-walk MH algorithm is necessary to understanding what makes HMC so effective.

The main goal behind any Markov Chain Monte Carlo (MCMC) method is to generate a sequence of points where the points are drawn from a probability distribution. If we can assure that we are drawing from the correct distribution and we draw enough points from that distribution, we are able to obtain a thorough understanding of what this probability distribution looks like. Random-walk MH is an example of a MCMC method.

The MH algorithm works by starting at an initial point x_0 (which may or may not be in the target space) and then generating a new proposal point x_1 using what is called a **transition kernel**. We will denote this transition kernel by $\mathcal{Q}(x_{n+1}|x_n)$. The actual definition of this transition kernel for an arbitrary parameter space is rather technical, but we can just think of \mathcal{Q} as being a conditional probability for x_{n+1} given x_n . By specifying this transition kernel and the initial point, we are completely describing the MH algorithm. This kernel is also what makes the resulting sequence of points a Markov chain. In every MCMC method, the kernel is said to have the **Markov Property**. This means that the probability of obtaining the next step in the chain given all previous steps of the chain is equal to the probability of obtaining the next step in the chain given *only* the previous step of the chain. While this property may seem somewhat inconspicuous, it actually contributes greatly to assuring the chain converges in probability to the target distribution that we wish to sample from. After generating a point x_{n+1} using the transition kernel, the MH algorithm proceeds by deciding how “good” of a proposal that point is and accepting or rejecting the point accordingly. Let’s take a look at how the MH algorithm constructs the transition kernel and also how it decides whether to accept or reject proposals generated by that transition kernel.

Let’s assume that our transition kernel can be written as $\mathcal{Q}(x_{n+1}|x_n) = r(x_{n+1}, x_n)g(x_{n+1}|x_n)$. Here $r(x_{n+1}, x_n)$ is known as the acceptance probability (or acceptance ratio) and $g(x_{n+1}|x_n)$ is known as the **proposal distribution**. Let $P(x)$ be the distribution we wish to sample from (the target distribution), then in the MH algorithm this acceptance probability is given by

$$r(x_{n+1}, x_n) = \min \left(1, \frac{P(x_{n+1}) g(x_n|x_{n+1})}{P(x_n) g(x_{n+1}|x_n)} \right). \quad (1)$$

The choice of this acceptance probability arises from our desire to have the transition kernel satisfy what is known as **detailed balance**. If the transition kernel does in fact satisfy detailed balance, then the Markov Chain generated by that transition kernel is guaranteed to have $P(x)$ as its **stationary distribution**. In other words, it guarantees that the Markov Chain will converge to the distribution $P(x)$.

If we assume that we can easily draw points from the proposal distribution $g(x_{n+1}|x_n)$ and that we can evaluate $P(x)$ at all points x_n in our chain, then the MH algorithm can be summarized as follows.

Algorithm 1. (MH Algorithm)

Given the current step x_n and proposal distribution $g(x_{n+1}|x_n)$, do:

1. draw candidate point $x^* \sim g(x^*|x_n)$
2. calculate $r(x^*, x_n) = \min\left(1, \frac{P(x^*)g(x_n|x^*)}{P(x_n)g(x^*|x_n)}\right)$
3. draw $u \sim U([0, 1])$. If $u \leq r(x^*, x_n)$, let $x_{n+1} = x^*$. Else, let $x_{n+1} = x_n$.

The last step here is important, because it allows for us to occasionally take steps in regions of lower probability. Without this, we would only accept points in regions of high probability and as a result we would not fully explore the probability distribution.

1.2 The Random-Walk MH Algorithm

The Random-walk MH algorithm is simply the MH algorithm in which we choose the proposal distribution $g(x_{n+1}|x_n)$ to be a normal distribution centered at the previous step of the chain. That is, we let

$$g(x_{n+1}|x_n) = \mathcal{N}(x_{n+1}|\mu = x_n, \sigma^2 = 1) = \mathcal{N}(x_{n+1}|x_n).$$

This leads to the name Random-walk, since at each step of the chain we simply take a step in a random direction. Then, the accept/reject phase of the MH algorithm checks to see if that step was in the direction of a higher probability region or a lower one.

Before presenting the Random-walk MH algorithm we note that this proposal distribution is symmetric, meaning $\mathcal{N}(x_{n+1}|x_n) = \mathcal{N}(x_n|x_{n+1})$. This simplifies the calculation of the acceptance probability and leads to the following algorithm.

Algorithm 2. (Random-walk MH Algorithm)

Given the current step x_n and proposal distribution $\mathcal{N}(x_{n+1}|x_n)$, do:

1. draw candidate point $x^* \sim \mathcal{N}(x_{n+1}|x_n)$
2. calculate $r(x^*, x_n) = \min\left(1, \frac{P(x^*)}{P(x_n)}\right)$
3. draw $u \sim U([0, 1])$. If $u \leq r(x^*, x_n)$, let $x_{n+1} = x^*$. Else, let $x_{n+1} = x_n$.

The simplification of the acceptance probability lets us see that if the next step of the random walk (x^*) is in a region of higher probability then it is always accepted, since in that case $r(x^*, x_n) = 1$. Again, step 3 of the Random-walk MH algorithm allows us to

occasionally make steps into regions of lower probability. This ensures we are able to fully explore the probability distribution.

1.2.1 Requirements

One of the best aspects of Random-walk MH is that we have almost no restrictions on what probability distribution we want to sample from. The only requirement is that we are able to evaluate a function *proportional* to the desired probability distribution $P(x)$. To see this, let $A(x) = cP(x)$ for some constant c . We claim that using $A(x)$ in the Random-walk MH algorithm is the same as using $P(x)$. To see this, look at step 2 of algorithm 2 and note that

$$\begin{aligned} r(x^*, x_n) &= \min\left(1, \frac{A(x^*)}{A(x_n)}\right) \\ &= \min\left(1, \frac{cP(x^*)}{cP(x_n)}\right) \\ &= \min\left(1, \frac{P(x^*)}{P(x_n)}\right). \end{aligned}$$

This is extremely useful when exploring posterior distributions obtained in Bayesian statistics, as we can often only obtain a distribution proportional to the posterior using Bayes' Law.

1.2.2 Limitations

The advantages of Random-walk MH arise from the simplicity of the proposal distribution. However, this also limits the algorithm from being effective as the dimension of the parameter (or state) space increases. The intuitive reason for this is that we are much less likely to randomly propose a step into a region of higher probability space in higher dimensions. Because of this, as the dimension of parameter space increases the number of accepted proposals in the algorithm decreases. The algorithm also struggles to explore multimodal distributions, as the random walk will often get stuck exploring one of the modes while neglecting the others.

2 Hamiltonian Monte Carlo

In this section we address the issue in Random-walk MH of inefficient sampling in higher dimensional probability space using a technique called **Hamiltonian Monte Carlo (HMC)**. HMC differs from Random-walk MH in that we have a more systematic and directed approach to picking proposal points. This means the number of accepted proposals in the accept/reject phase of the MH algorithm is initially higher than Random-walk MH and persists even as the dimension of the parameter space is increased. We summarize the technique below, and note that additional reading for HMC can be found at [2] and [3].

2.1 Hamiltonian Mechanics

In order to understand how HMC chooses its proposal points, we must diverge into a short discussion of Hamiltonian mechanics. While the reader is likely familiar with these concepts, we will re-present them to highlight important aspects for HMC and also to establish consistent notation. More in-depth reading on the subject can be found in [12].

In Hamiltonian mechanics, a system is described by a set of coordinates $\{(\mathbf{q}, \mathbf{p})\}$ where $\mathbf{q} \in \mathbb{R}^d$ is the position of the system and $\mathbf{p} \in \mathbb{R}^d$ is the momentum of the system. We next introduce the **Hamiltonian** of the system, denoted by $\mathcal{H}(\mathbf{q}, \mathbf{p})$. For us, the Hamiltonian will represent the total energy of the system. We will assume the system is closed, which means that the Hamiltonian can be written as

$$\mathcal{H}(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$$

where $U(\mathbf{q})$ is the potential energy of the system and $K(\mathbf{p})$ is the kinetic energy of the system. The system described by the Hamiltonian $\mathcal{H}(\mathbf{q}, \mathbf{p})$ evolves according to the Hamilton's equations:

$$\begin{aligned}\frac{d\mathbf{p}}{dt} &= -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \\ \frac{d\mathbf{q}}{dt} &= \frac{\partial \mathcal{H}}{\partial \mathbf{p}}.\end{aligned}$$

These equations allow us to determine the path that a particle takes through **phase space** $\mathbf{q} \times \mathbf{p}$. Note that the above equations imply that the Hamiltonian is constant (i.e. $\frac{d}{dt}\mathcal{H}(\mathbf{q}, \mathbf{p}) = 0$), meaning the path of a particle through phase space follows a level set of the Hamiltonian. Thus if we fix an initial point of a particle in phase space, we can explore other regions in phase space which produce equal values in the Hamiltonian by evolving the point according to Hamilton's equations. This will be particularly useful later when we discuss the HMC algorithm.

2.2 The Leapfrog Integrator

In order to produce numerical simulations of Hamiltonian dynamics through phase space, we must discretize Hamilton's equations. To do this, we will use the **leapfrog integrator** [3]. This method updates the position and momentum of a particle in time according to the following equations:

$$\begin{aligned}\mathbf{p}_i(t + \epsilon/2) &= \mathbf{p}_i(t) - (\epsilon/2)\nabla U(\mathbf{q}(t)) \\ \mathbf{q}_i(t + \epsilon) &= \mathbf{q}_i(t) + \epsilon\mathbf{p}_i(t + \epsilon/2) \\ \mathbf{p}_i(t + \epsilon) &= \mathbf{p}_i(t + \epsilon/2) - (\epsilon/2)\nabla U(\mathbf{q}(t + \epsilon)).\end{aligned}$$

There are two parameters in the leapfrog integrator that will be important to us later, so we will take a bit of time to highlight them. The first is the **step size** ϵ and the second is the **number of steps** L . The step size appropriately describes how large of a leap we make along a level curve of the Hamiltonian in phase space, while the number of steps describes how many leaps we make along that same level curve. Both the step size and the number of steps are important for simulating the Hamiltonian dynamics correctly and for ensuring that we end our dynamics in a desirable point in phase space. To highlight this, consider the sequence in figures 1 and 2.

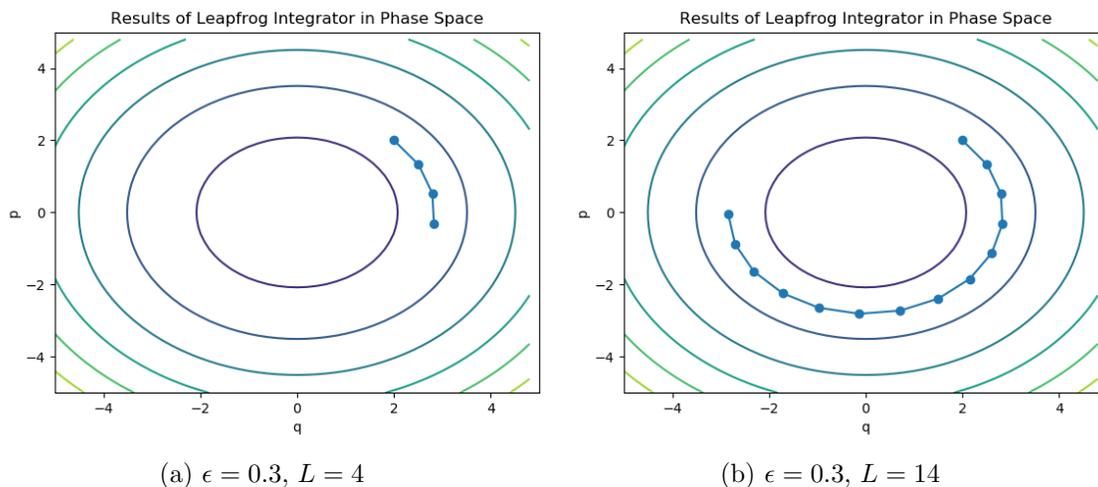


Figure 1: Hamiltonian Dynamics using the Leapfrog Integrator

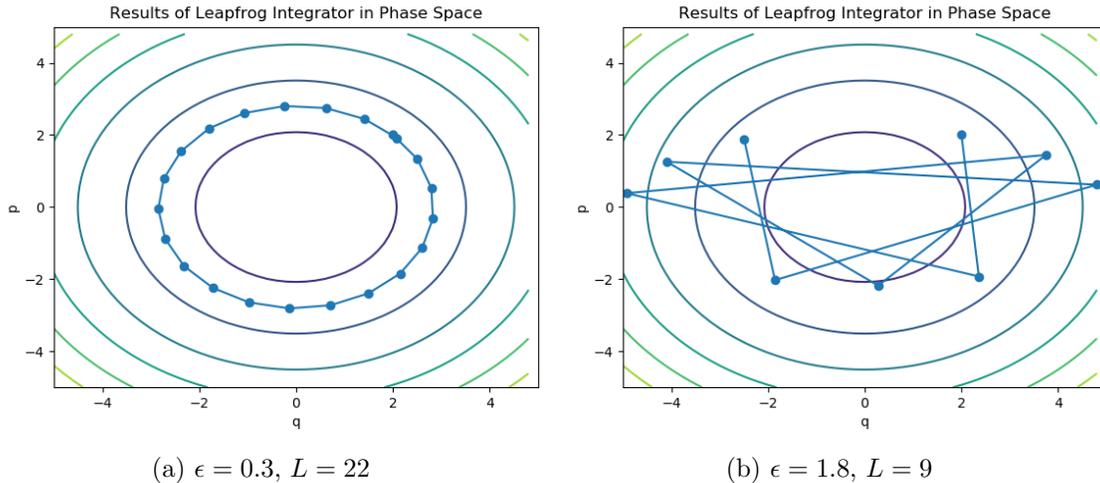


Figure 2: Hamiltonian Dynamics using the Leapfrog Integrator

These figures show the results of simulating the time evolution of a particle in phase space using the leapfrog integrator. In Figures 1(a),1(b) and 2(a) we see that if ϵ is sufficiently small, then we will accurately model the Hamiltonian dynamics by following a level set of the Hamiltonian. However if L is small as in 1(a) then we will not proceed far along the level set. Similarly if L is too big as in 2(a) then we will circle around the level set back to our original starting point. In figure 2(b) we see the importance of having ϵ be sufficiently small. If it is not, then the dynamics break down and our leapfrog integrator is useless. Figure 1(b) represents a good implementation of the leapfrog integrator for our purposes. ϵ is small enough to ensure accurate modeling while L is a good value for moving us sufficiently far in phase space from the starting point. We will revisit the step size and number of steps later when discussing the HMC algorithm.

2.3 The Canonical Distribution

We'll now introduce the concept of a **canonical distribution**, which originated in the field of statistical mechanics. Given an energy function $E(x)$ where x is a state of some physical system, the canonical distribution is the function

$$C(x) \propto \exp\left(-\frac{E(x)}{T}\right).$$

For us, the energy function in question will be the Hamiltonian $\mathcal{H}(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$. We will also make the assumption that $T = 1$. This means that our canonical distribution

takes the form

$$\begin{aligned} C(\mathbf{q}, \mathbf{p}) &\propto \exp(\mathcal{H}(\mathbf{q}, \mathbf{p})) \\ &= \exp(-U(\mathbf{q})) \exp(-K(\mathbf{p})). \end{aligned} \quad (2)$$

This canonical distribution is a joint probability distribution in phase space for the variables \mathbf{q} and \mathbf{p} . We will define the potential and kinetic energy as follows:

$$U(\mathbf{q}) = -\log(P(\mathbf{q})) \quad (3)$$

$$K(\mathbf{p}) = \frac{\mathbf{p}^T \mathbf{p}}{2}, \quad (4)$$

meaning our canonical distribution is

$$\begin{aligned} C(\mathbf{q}, \mathbf{p}) &\propto \exp(\log(P(\mathbf{q})) \exp\left(\frac{\mathbf{p}^T \mathbf{p}}{2}\right)) \\ &= P(\mathbf{q}) \exp\left(\frac{\mathbf{p}^T \mathbf{p}}{2}\right) \end{aligned}$$

Therefore, \mathbf{q} and \mathbf{p} are independent under the distribution $C(\mathbf{q}, \mathbf{p})$ and further the marginal distribution of \mathbf{q} under $C(\mathbf{q}, \mathbf{p})$ is our target distribution $P(\mathbf{q})$.

2.4 The Hamiltonian Monte Carlo Algorithm

The Hamiltonian Monte Carlo (HMC) algorithm makes use of the Hamiltonian dynamics and canonical distribution described above to propose samples in a more directed way as compared to Random-walk MH. The process starts by letting the elements of our target space be denoted by $\mathbf{q} \in \mathbb{R}^d$ and introducing auxiliary variables $\mathbf{p} \in \mathbb{R}^d$. We will now carry out the Metropolis-Hastings algorithm in phase space where we sample from the canonical distribution $C(\mathbf{q}, \mathbf{p})$ defined in the previous section using Hamiltonian mechanics to propose new points (\mathbf{q}, \mathbf{p}) in phase space. Then since \mathbf{q}, \mathbf{p} are independent under $C(\mathbf{q}, \mathbf{p})$ and the marginal distribution of \mathbf{q} under $C(\mathbf{q}, \mathbf{p})$ is $P(\mathbf{q})$, we can simply drop the elements \mathbf{p}_i to obtain samples of our target distribution P . Consider only this, we have the following naive algorithm.

Algorithm 3. (Naive HMC)

Given the current step $(\mathbf{q}_n, \mathbf{p}_n)$, step size ϵ and step length L , do:

1. let $(\mathbf{q}_i, \mathbf{p}_i) = (\mathbf{q}_n, \mathbf{p}_n)$. For $i = 1:L$ do

$$\begin{aligned}\mathbf{p}_i &= \mathbf{p}_i - (\epsilon/2)\nabla U(\mathbf{q}_i) \\ \mathbf{q}_i &= \mathbf{q}_i(t) + \epsilon\mathbf{p}_i \\ \mathbf{p}_i &= \mathbf{p}_i - (\epsilon/2)\nabla U(\mathbf{q}_i)\end{aligned}$$

2. calculate $r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L) = \min(1, \exp(-\mathcal{H}(\mathbf{q}_L, \mathbf{p}_L) + \mathcal{H}(\mathbf{q}_n, \mathbf{p}_n)))$

3. draw $u \sim U([0, 1])$. If $u \leq r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L)$, let $(\mathbf{q}_{n+1}, \mathbf{p}_{n+1}) = (\mathbf{q}_L, \mathbf{p}_L)$. Else, let $(\mathbf{q}_{n+1}, \mathbf{p}_{n+1}) = (\mathbf{q}_n, \mathbf{p}_n)$.

Before discussing why this algorithm fails (thus the name Naive HMC) we will make a couple of remarks. The first is that the expression for r in step 2 of the above algorithm is a result of the following simplifications:

$$\begin{aligned}r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L) &= \min\left(1, \frac{C(\mathbf{q}_L, \mathbf{p}_L)}{C(\mathbf{q}_n, \mathbf{p}_n)}\right) \\ &= \min\left(1, \frac{\exp(-\mathcal{H}(\mathbf{q}_L, \mathbf{p}_L))}{\exp(-\mathcal{H}(\mathbf{q}_n, \mathbf{p}_n))}\right) \\ &= \min(1, \exp(-\mathcal{H}(\mathbf{q}_L, \mathbf{p}_L) + \mathcal{H}(\mathbf{q}_n, \mathbf{p}_n))).\end{aligned}$$

Thus the proposal point $(\mathbf{q}_L, \mathbf{p}_L)$ is generated using Hamiltonian dynamics in step 1, and either accepted or rejected in steps 2 and 3 in the same way as the standard MH algorithm.

The second remark is that after simplifying we see that the ratio $r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L)$ is actually checking if our leapfrog integrator accurately simulated the Hamiltonian mechanics. If we simulated the mechanics exactly correct, then $\mathcal{H}(\mathbf{q}_L, \mathbf{p}_L) = \mathcal{H}(\mathbf{q}_n, \mathbf{p}_n)$ since we are simply following a level set of the Hamiltonian in phase space. This means the candidate point $(\mathbf{q}_L, \mathbf{p}_L)$ would always be accepted. Thus if our leapfrog integrator is tuned well (that is, ϵ and L are chosen well) we should expect the number of accepted points in the algorithm to be very high.

To see why this algorithm fails, consider how this algorithm acts on $(\mathbf{q}_n, \mathbf{p}_n)$ in phase space. At each step of the algorithm we simulate Hamiltonian dynamics on the point, meaning (assuming our simulation is accurate) we will always remain on the same level set of the Hamiltonian in phase space! This is not ideal, since it means we are not sufficiently exploring the distribution $C(\mathbf{q}, \mathbf{p})$. To correct for this, we will resample the momentum \mathbf{p}_n at each step of the algorithm from its distribution $\mathcal{N}(\mathbf{p}|\mathbf{0}, \mathbf{I}_d)$. This allows us to start at a new level set of the Hamiltonian in phase space at each iteration, then simulate Hamiltonian dynamics that moves us along that level set. We thus have the Hamiltonian Monte Carlo algorithm.

Algorithm 4. (HMC)

Given the current step \mathbf{q}_n , step size ϵ and step length L , do:

1. draw $\mathbf{p}_n \sim \mathcal{N}(\mathbf{p}|\mathbf{0}, \mathbf{I}_d)$.
2. let $(\mathbf{q}_i, \mathbf{p}_i) = (\mathbf{q}_n, \mathbf{p}_n)$. For $i = 1:L$ do

$$\mathbf{p}_i = \mathbf{p}_i - (\epsilon/2)\nabla U(\mathbf{q}_i)$$

$$\mathbf{q}_i = \mathbf{q}_i + \epsilon\mathbf{p}_i$$

$$\mathbf{p}_i = \mathbf{p}_i - (\epsilon/2)\nabla U(\mathbf{q}_i)$$

3. calculate $r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L) = \min(1, \exp(-\mathcal{H}(\mathbf{q}_L, \mathbf{p}_L) + \mathcal{H}(\mathbf{q}_n, \mathbf{p}_n)))$
 4. draw $u \sim U([0, 1])$. If $u \leq r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}_L, \mathbf{p}_L)$, let $\mathbf{q}_{n+1} = \mathbf{q}_L$. Else, let $\mathbf{q}_{n+1} = \mathbf{q}_n$.
-

2.5 Requirements

HMC can only sample from continuous distributions on finite dimensional space (\mathbb{R}^d) for which the density in question can be evaluated. In fact, we again only need to be able to evaluate a function proportional to the density in question. For HMC we must also be able to evaluate partial derivatives of the log of the density function in order to simulate the Hamiltonian dynamics.

2.6 Limitations

Similar to Random-walk MH, the HMC algorithm struggles to sample from multimodal distributions. This is especially true in higher dimensions when a large number of modes are present. The requirement of needing to calculate partial derivatives of the density in question is also problematic for implementation in CosmoSIS. Due to its modular framework, there is no clear method for extracting derivative information from a posterior distribution in CosmoSIS.

3 Gradient-free HMC

One of the main obstacles we will be dealing with in terms of implementing HMC in CosmoSIS is a lack of derivative information for posterior distributions. In this section we propose two possible solutions to this problem, each with benefits and admitted shortcomings.

3.1 Kernel Hamiltonian Monte Carlo

A gradient-free alternative to HMC was proposed in [11] called *Kernel Hamiltonian Monte Carlo (KMC)*. In order to understand this algorithm we will have to take some detours.

3.1.1 Pseudo-Marginal MCMC

Pseudo-marginal MCMC (PM MCMC) is a variation of the Metropolis-Hastings algorithm in which we replace the target density $P(x)$ by some unbiased, non-negative estimate $\hat{P}(x)$. That is, the PM MCMC algorithm is the MH algorithm with the following adaptation:

$$r(x_{n+1}, x_n) = \min \left(1, \frac{\hat{P}(x_{n+1}) g(x_n|x_{n+1})}{\hat{P}(x_n) g(x_{n+1}|x_n)} \right).$$

This adaptation is useful when the target distribution $P(x)$ is intractable. Obtaining the estimates $\hat{P}(x)$ can be rather involved and non-obvious for certain distributions. We will not concern ourselves with the process of finding these estimates in this summary, but further reading on this algorithm can be found at [1].

3.1.2 Reproducing Kernel Hilbert Spaces

We'll start with the definition of a reproducing kernel Hilbert space (RKHS) and then give a more intuitive feel of their structure.

Definition 1. Let X be an arbitrary set and denote by \mathbb{R}^X the vector space of all functions $f : X \rightarrow \mathbb{R}$ equipped with pointwise operations. A subspace $V \subset \mathbb{R}^X$ endowed with an inner product is a reproducing kernel Hilbert space if V is complete and, for every $x \in X$, the evaluation functional $f \rightarrow f(x) = l_x(f)$ on V is bounded.

It turns out that every RKHS has a unique **kernel**, which we define next.

Definition 2. If $V \subset \mathbb{R}^X$ is a RKHS then its kernel is the function $k : X \times X \rightarrow \mathbb{R}$ satisfying $\langle f, k(\cdot, y) \rangle = f(y)$ for all $f \in V$ and $y \in X$.

Note that in the second definition we are taking $k(\cdot, y)$ to be the function $x \rightarrow k(x, y)$, which is an element of V . This definition can look a little obtuse on first glance, so we will attempt to gain a more intuitive feel for these spaces. We can think of a RKHS as behaving very much like Euclidean space, even more so than a more general Hilbert space. One of the best ways to see this is by thinking of and comparing norms on a Euclidean space, Hilbert space, and RKHS. In Euclidean space all norms are equivalent, but in a Hilbert space different norms induce different topologies. This means that even if two elements (lets think of them as functions) of a Hilbert space are close with respect to the norm induced by the inner product, the point-wise evaluation of those two functions at some point $x \in X$ could be vastly different. A good example of this is the Hilbert space $L^2(\Omega)$. In this space two functions f, g are considered close if the norm

$$\|f - g\|_2 = \left(\int_{\Omega} |f(x) - g(x)|^2 dx \right)^{1/2}$$

is kept small. However this means nothing for point-wise evaluations of the functions f and g since the two functions could vary greatly on a set of measure zero and still be close in norm. A RKHS does not have this feature, since by definition the evaluation function for a RKHS is bounded (equivalently continuous). This ensures that if two functions f, g in a RKHS are close in norm, they are also close point-wise. Thus although different norms in a RKHS might not induce exactly the same topologies, the topologies those different norms do induce will not be as radically different as they often are in a Hilbert space.

An excellent reference on this topic can be found in [7].

3.1.3 Exponential Families

Definition 3. Suppose that our parameter space is $\Omega \subset \mathbb{R}^m$. The exponential family generated by probability density q_0 is a set of probability distributions parametrized by θ where each probability distribution in the set has the form

$$p(x|\theta) = q_0(x) \exp(\theta^T T(x) - A(\theta)).$$

We say that $\theta \in \Theta \subset \mathbb{R}^m$ is the *natural parameter*, $\Theta \subset \{\theta \in \mathbb{R}^m : A(\theta) < \infty\}$ is the natural parameter space, $A(\theta) := \log \int_{\Omega} e^{\theta^T T(x)} q_0(x) dx$, and $T : \Omega \rightarrow \mathbb{R}^m$ is the *sufficient statistic*.

For our purposes, we need a more general definition of an exponential family. Specifically we would like an infinite dimensional generalization of the previous definition. We will use the following as that generalization.

Definition 4. Let $(H, \langle \cdot, \cdot \rangle)$ be a reproducing kernel Hilbert space with k as its reproducing kernel. Let $\mathcal{F} = \{f \in H : e^{A(f)} < \infty\}$ with $A(f) := \log \int_{\Omega} e^{f(x)} q_0(x) dx$. An exponential family is a set of probability distributions parametrized by $f \in \mathcal{F}$ such that each probability distribution has the form

$$p(x|f) = q_0(x) \exp(f(x) - A(f)).$$

A few notes on this definition. The first is that we can equivalently write each probability distribution as

$$p(x|f) = q_0(x) \exp(\langle f, k(x, \cdot) \rangle_H - A(f))$$

due to the reproducing property of the kernel $f(x) = \langle f, k(x, \cdot) \rangle_H$. The second note is that the choice of \mathcal{F} being a subset of a RKHS is quite natural. This is because every finite dimensional exponential family can be constructed from the exponential family defined in definition 4 for some H a finite dimensional RKHS. Further reading on this subject can be found in [10].

With these concepts presented, we can now discuss the KMC algorithm. This algorithm assumes that we can evaluate neither the target density $P(x)$ nor the gradients $\nabla_x \log P(x)$ for any x in our target space. In the case of sampling from likelihoods produced using a CosmoSIS module, we do not have access to the gradients $\nabla_x \log P(x)$, but we do have access to evaluations of the target density $P(x)$. The key step around the need for derivatives is the introduction of a kernel induced surrogate whose gradients approximate the gradients $\nabla_x \log P(x)$. That is, we let

$$P(x) \approx \exp(\langle f, k(x, \cdot) \rangle_H - A(f)).$$

If we can find an $f \in H$ for which this approximation is in some way “close”, we have an approximation to the partial derivatives of potential energy required for the modeling of Hamiltonian dynamics. To see this, note that

$$\begin{aligned} -\nabla U &= \nabla_x \log(P(x)) \approx \nabla_x (\langle f, k(x, \cdot) \rangle_H - A(f)) \\ &= \nabla_x (f(x) - A(f)) \\ &= \nabla_x f(x). \end{aligned}$$

The goal now is clear: find an $f \in H$ such that $\exp(\langle f, k(x, \cdot) \rangle_H - A(f))$ is a good approximation to our distribution $P(x)$.

The approach taken by [11] to determining such an f is to use the concept of score matching [6]. We want to determine an $f \in \mathcal{H}$ such that

$$J(f) = \int_x \pi(x) \|\nabla_x \log P(x|f) - \nabla_x \log P(x)\|_2^2 dx$$

is as small as possible. I should note that the reason for the name “score matching” is that the terms $\psi(x|f) = \nabla_x \log P(x|f)$ and $\psi(x) = \nabla_x \log P(x)$ are called score functions corresponding to $P(x|f)$ and $P(x)$, respectively. Now we still don’t know $\nabla_x \log P(x)$ in order to evaluate the above integral, but surprisingly it was shown in [11] that we can write an approximation to the integral in $J(f)$ that is extremely accurate. If we have n observations (in dimension d) of the random vector x , then we have the approximation

$$\hat{J}(f) = \frac{1}{n} \sum_{j=1}^n \sum_{l=1}^d \left(\frac{\partial^2 \log P(x|f)}{\partial x_l^2} + \frac{1}{2} \left(\frac{\partial \log P(x|f)}{\partial x_l} \right)^2 \right).$$

The proof that this functional $\hat{J}(f)$ exists and accurately approximates $J(f)$ is unintuitive and unhelpful for our purposes here. The existence of this \hat{J} operator though does give hope in the ability to develop an algorithm in which we can use past elements of a Markov chain to approximate derivatives of $\nabla_x \log P(x)$.

3.1.4 KMC lite

The first approach to developing such an algorithm is known as **KMC lite** [11]. The idea is to form a proposal within MH using a random sub-sample of fixed size n from the Markov chain history. That is, we take $\mathbf{z} := \{z_i\}_{i=1}^n \subseteq \{x_i\}_{i=1}^t$. Next, we suppose that our function f which minimizes \hat{J} has the form

$$f(x) = \sum_{i=1}^n \alpha_i k(z_i, x)$$

where now $\alpha \in \mathbb{R}^n$ is the vector we seek from minimizing the functional \hat{J} . There is a unique minimizer for \hat{J} , which is given by the following proposition.

Proposition 1. *Given a set of samples $\mathbf{z} = \{z_i\}_{i=1}^n$ and assuming $f(x) = \sum_{i=1}^n \alpha_i k(z_i, x)$ for the Gaussian kernel of the form $k(x, y) = \exp(-\sigma^{-1} \|x - y\|_2^2)$, and $\lambda > 0$, the unique minimizer of the regularized functional \hat{J} is given by*

$$\hat{\alpha}_\lambda = -\frac{\sigma}{2} (C + \lambda I)^{-1} b,$$

where $b \in \mathbb{R}^n$ and $C \in \mathbb{R}^{n \times n}$ are given by

$$b = \sum_{l=1}^d \left(\frac{2}{\sigma} (Ks_l + D_{s_l}K\mathbf{1} - 2D_{x_l}Kx_l) - K\mathbf{1} \right)$$

and

$$C = \sum_{l=1}^d (D_{x_l}K - KD_{x_l})(KD_{x_l} - D_{x_l}K),$$

with $s_l := x_l \odot x_l$ and $D_x := \text{diag}(x)$.

There is obviously a lot to dig into with this, and the proof of this proposition is straightforward but tedious. A few clarifications and notes in this proposition are warranted. First, the operation $x_l \odot x_l$ means component wise multiplication of the entries of x_l . Also, all of the indexing here can be difficult to keep track of. Keep in mind that each z_i and x_i is actually a d -dimensional vector. This makes the dimensions of everything work out nicely. Lastly, the term λ is a regularization parameter and exists to account for any ill-conditioning in the matrix C .

We thus arrive at the following algorithm for KMC lite.

Algorithm 5. (KMC Lite)

Given $P(x)$ (or possibly a PM MCMC estimator $\hat{P}(x)$), σ , λ , current step x_i and history of the chain $\{x_i\}_{i=1}^t$, do:

1. update sub-sample $z \subseteq \{x_i\}_{i=1}^t$
2. re-compute b and C from Proposition 1
3. solve $\hat{\alpha}_\lambda = -\frac{\sigma}{2}(C + \lambda I)^{-1}b$
4. $\nabla f = \sum_{i=1}^n (\hat{\alpha}_\lambda)_i \nabla k(z_i, x)$
5. run HMC with $\nabla U = \nabla f$

The paper states that there are two free parameters in this algorithm, the Gaussian kernel bandwidth σ and the regularization parameter λ . While these are both certainly free parameters, the value of n is not made clear in the paper. We will treat this as another free parameter for now while acknowledging that it may in fact have a pre-prescribed value.

3.1.5 KMC Finite

The previous method was based on supposing that $f \in H$ has the form $f(x) = \sum_{i=1}^n \alpha_i k(z_i, x)$. The next method (known as **KMC finite**) instead begins with supposing that $f \in H$ has the form $f(x) = \langle \theta, \phi_x \rangle_H = \theta^T \phi_x$, where $\theta \in \mathbb{R}^m$, $\phi_x \in H$ is an embedding of a point $x \in \mathbb{R}^d$ into H_m , and $H_m = \mathbb{R}^m$ is an m -dimensional approximate feature space of H . Here we assume that ϕ_x is given (or more specifically we choose ϕ_x) and θ is found by minimizing the functional \hat{J} . Fortunately, there is again a proposition that gives us this minimizer in a relatively simple form:

Proposition 2. *Given a set of samples $\mathbf{x} = \{x_i\}_{i=1}^t$ and assuming $f(x) = \theta^T \phi_x$ for a finite dimensional feature embedding $x \mapsto \phi_x \in \mathbb{R}^m$ and $\lambda > 0$, the unique minimizer of the regularized functional \hat{J} is given by*

$$\hat{\theta}_\lambda := (C + \lambda I)^{-1} b,$$

where

$$b := -\frac{1}{n} \sum_{i=1}^t \sum_{l=1}^d \ddot{\phi}_{x_i}^l \in \mathbb{R}^m$$

and

$$C := \frac{1}{n} \sum_{i=1}^t \sum_{l=1}^d \dot{\phi}_{x_i}^l \left(\dot{\phi}_{x_i}^l \right)^T \in \mathbb{R}^{m \times m},$$

with $\dot{\phi}_{x_i}^l := \frac{\partial}{\partial x_l} \phi_x$ and $\ddot{\phi}_{x_i}^l := \frac{\partial^2}{\partial x_l^2} \phi_x$

While the intuition in this proposition is again relatively absent, the terms within the proposition are a bit easier to grasp. In this method we only need to specify the embedding ϕ_x and the regularization parameter λ . The paper presents the possible choice of ϕ_x as $\phi_x = \sqrt{\frac{2}{m}} [\cos(\omega_1^T x + u_1), \dots, \cos(\omega_m^T x + u_m)]$ with $\omega_i \sim \mathcal{N}(\omega)$ and $u_i \sim U[0, 2\pi]$. This proposition leads to the following algorithm for KMC.

Algorithm 6. (KMC Finite)

Given $P(x)$ (or possibly a PM MCMC estimator $\hat{P}(x)$), ϕ_x , λ , current step x_i and history of the chain $\{x_i\}_{i=1}^t$, do:

1. update C and b using proposition 2
2. calculate $(C + \lambda I)^{-1}$, likely using a rank-d update

3. let $\hat{\theta}_\lambda = (C + \lambda I)^{-1}b$
4. $\nabla f = [\nabla\phi_x]^T \hat{\theta}_\lambda$
5. run HMC with $\nabla U = \nabla f$

The use of a rank-d update in step 2 here is simply to keep the cost of the algorithm lower. One could invert the matrix $C + \lambda I$ at each step, but performing a rank-d update means we only need to compute a costly inversion once instead of at every iteration.

3.1.6 Limitations

We stated the requirements for each algorithm within the above sections, so we will instead jump to discussing the limitations of both KMC algorithms. The primary limitations are associated with the complexity of both algorithms. Both Random-walk MH and HMC are very appealing since they are intuitively clear and simple to present. This makes diagnosing any problem while running these algorithms relatively easy. However with the KMC algorithms most of the machinery is so obfuscated that it becomes very difficult to diagnose any problems that will undoubtedly arise. More work needs to be done to see if KMC is a realistic option for implementation in CosmoSIS.

3.2 “Gambling” Finite Difference Hamiltonian Monte Carlo

One other possible technique to circumvent the need for gradient information is to use a finite difference approximation to the partial derivatives in the simulation of the Hamiltonian dynamics. This is an appealing solution because it is simple and can be made quite accurate for most distributions. However, function evaluations in CosmoSIS are very expensive. So while we do in theory have access to evaluations of the probability density function in question, we cannot take a large number of evaluations without incurring a substantial computational cost.

A possible solution would be to reduce the number of steps in the leapfrog iterator to a single step with a larger than normal step-size. This runs the risk however of degrading the approximation of the Hamiltonian dynamics and as a result ruining the acceptance ratio. To account for this, we propose drawing the step-size ϵ from some distribution which gives more weight to step-sizes that preserve the dynamics while occasionally drawing a larger step-size. Thus we occasionally “gamble” and take a large jump through phase space to

avoid falling back to random-walk type behavior. The choice of the distribution for ϵ is considered a free parameter and will need to be tuned, but a normal or skewed normal distribution (with a step to ensure positivity) could be reasonable. We will let

$$\nabla_f U_h = \begin{bmatrix} \frac{U(q_1+h, q_2, \dots, q_d) - U(q_1, q_2, \dots, q_d)}{h} \\ \frac{U(q_1, q_2+h, \dots, q_d) - U(q_1, q_2, \dots, q_d)}{h} \\ \vdots \\ \frac{U(q_1, q_2, \dots, q_d+h) - U(q_1, q_2, \dots, q_d)}{h} \end{bmatrix}$$

be the finite difference approximation to the gradient of U . The process outlined above is described in the following algorithm.

Algorithm 7. Gambling Finite Difference HMC

Given μ, σ, h and current step q_n do:

1. draw $\mathbf{p}_n \sim \mathcal{N}(\vec{0}, \mathbf{I}_d)$
2. draw $\epsilon \sim \mathcal{N}(\mu, \sigma)$
if $\epsilon \leq 0$, redraw $\epsilon \sim \mathcal{N}(\mu, \sigma)$
- 3.

$$\begin{aligned} \mathbf{p}' &= \mathbf{p}_n - \frac{\epsilon}{2} \nabla_f U_h(\mathbf{q}_n) \\ \mathbf{q}' &= \mathbf{q} + \epsilon \mathbf{p}' \\ \mathbf{p}' &= \mathbf{p}' - \frac{\epsilon}{2} \nabla_f U_h(\mathbf{q}'). \end{aligned}$$

4. calculate $r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}', \mathbf{p}') = \min(1, \exp(-\mathcal{H}(\mathbf{q}', \mathbf{p}') + \mathcal{H}(\mathbf{q}_n, \mathbf{p}_n)))$
5. draw $u \sim U([0, 1])$. If $u \leq r(\mathbf{q}_n, \mathbf{p}_n, \mathbf{q}', \mathbf{p}')$, let $\mathbf{q}_{n+1} = \mathbf{q}'$. Else, let $\mathbf{q}_{n+1} = \mathbf{q}_n$.

In additions to valid concerns about the results of using a less refined approximation to the dynamics, this approach becomes less plausible in CosmoSIS as the dimension of the parameter space increases. At each iteration we require $2d$ more function evaluations than in random-walk MH (where d is the dimension of parameter space). Thus for large values of d this method is likely infeasible. Further testing and discussion must be done before this method is deemed valid for implementation in CosmoSIS.

4 MultiNest

We will now diverge from our discussion of MCMC methods to discuss another sampler in CosmoSIS. The sampler in CosmoSIS is called multinest, and it is based on the MultiNest algorithm. This algorithm is not intended for parameter estimation, but rather in the aid of Bayesian model selection. However, as a byproduct this algorithm also produces weighted samples from a posterior distribution. This summary will skip over many details of the algorithm especially in the later stages, but should provide an idea of how the algorithm works and when it is useful. Further details on this material can be found at [4].

4.1 Model Selection

In Bayesian inference problems the goal can be broadly summarized as either trying to estimate a set of parameters Θ or trying to determine an optimal model H . All of the algorithms above have centered on trying to estimate parameters of some model, but we would sometimes like to know whether a certain model predicts data better than another model. This is where the calculation of the Bayesian evidence is useful. Recall that Bayes' Theorem states

$$P(\Theta|D, H) = \frac{P(D|\Theta, H)P(\Theta|H)}{P(D|H)}$$

where $P(\Theta|D, H)$ is the posterior probability density of the model parameters, $P(D|\Theta, H) := \mathcal{L}(\Theta)$ is the likelihood function, $P(\Theta|H) := \pi(\Theta)$ is the prior distribution and $P(D|H) := \mathcal{Z}$ is the **Bayesian evidence**. This Bayesian evidence is given by

$$\mathcal{Z} = \int_{\Omega} \mathcal{L}(\Theta)\pi(\Theta)d\Theta$$

where Ω is the parameter space. Notice that the Bayesian evidence is independent of the parameter Θ , so it is typically ignored in parameter estimation. Notice also that if the dimension of the parameter space Ω is high, then this integral becomes very difficult to calculate.

The evidence is simply the likelihood function averaged over the prior. If we are trying to determine whether two models H_1 or H_2 fit data better, then a larger value of the evidence for H_1 tells us that H_1 is the better model. Mathematically, we can see this by noting that if $Pr(H|D)$ is the posterior probability for model H given data D then

$$\frac{Pr(H_1|D)}{Pr(H_2|D)} = \frac{Pr(D|H_1)Pr(H_1)}{Pr(D|H_2)Pr(H_2)} = \frac{\mathcal{Z}_1 Pr(H_1)}{\mathcal{Z}_2 Pr(H_2)}.$$

When neither model H_1 nor H_2 are initially given preference over one another, the value $Pr(H_1)/Pr(H_2) = 1$ and we see that the ratio of the posterior probability is entirely dependent on the Bayesian evidence.

4.2 Nested Sampling

The MultiNest algorithm is essentially just an extension of another algorithm used to calculate the Bayesian evidence called Nested Sampling [9]. The nested sampling algorithm is designed to calculate the Bayesian evidence. That is, nested sampling attempts to find the value of the (often) multi-dimensional integral

$$\mathcal{Z} = \int_{\Omega} \mathcal{L}(\Theta)\pi(\Theta)d\Theta.$$

The technique adopted by the Nested Sampling algorithm for evaluating this integral in a computationally efficient way is to transform the integral into a single dimensional integral over what is called the **prior mass**. The prior mass is defined as

$$X(\lambda) = \int_{\{\Theta:\mathcal{L}(\Theta)>\lambda\}} \pi(\Theta)d\Theta.$$

We could take the above line as a definition for the prior mass, but it might be helpful to think of the prior mass as the survival function of the likelihood function $\mathcal{L}(\Theta)$. The prior mass should be thought of as the amount of prior density contained within an iso-contour of the likelihood function.

At this step we will use a result from probability theory to state that the expected value of a non-negative random variable is equal to the integral of its survival function. Since the evidence is simply the expected value of the likelihood over the prior, this result allows us to write the evidence as

$$\mathcal{Z} = \int_0^{\infty} X(\lambda)d\lambda.$$

Now under a couple of reasonable assumptions on the likelihood $\mathcal{L}(\Theta)$ (namely that it is continuous and has connected support) we can say that the inverse function of $X(\lambda)$ exists. In order to be consistent with the literature, we will call this inverse function $\mathcal{L}(X)$. With this in hand we can write the evidence as

$$\mathcal{Z} = \int_0^1 \mathcal{L}(X)dX.$$

The following figure, taken from [5], does a nice job of displaying the transformed integrand.

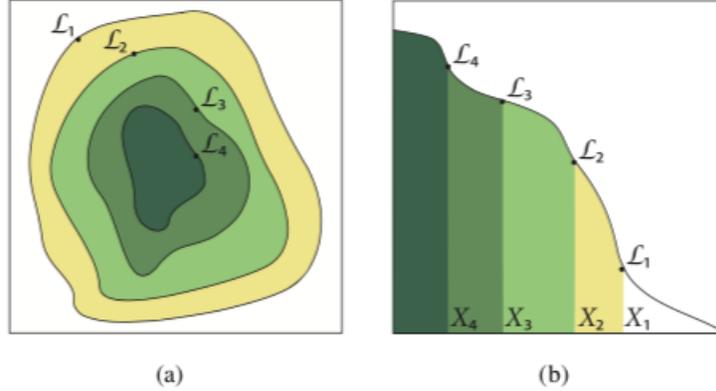


Figure 3: (a) The posterior of a two dimensional problem (b) The transformed $\mathcal{L}(X)$ function where the prior volumes X_i are associated with each likelihood \mathcal{L}_i

We will mention a couple of points about this transformed integral before we move on that reinforce the intuition and desirability of the steps above. With regards to the intuition note that the prior mass X simply computes the integral of the prior within some iso-likelihood contour of the likelihood. Thus X ranges from one to zero and is a decreasing function with respect to its argument θ . The inverse function $\mathcal{L}(X)$ then is also a decreasing function and maps the prior mass to some value of the likelihood function. Thus integrating this inverse function $\mathcal{L}(X)$ over the prior mass is exactly the same as averaging the likelihood over the prior in parameter space.

In reference to the desirability, we have taken a d -dimensional integral over parameter space and instead transformed it into a one dimensional integral over prior mass space. Thus if we are able to calculate $\mathcal{L}_i = \mathcal{L}(X_i)$ for a deterministic set of X_i values,

$$0 < X_N < \dots < X_2 < X_1 < X_0 = 1,$$

then we can approximate the evidence using some type of quadrature rule. That is, we would have

$$\mathcal{Z} \approx \sum_{i=1}^N \mathcal{L}_i \omega_i,$$

where ω_i come from the choice of quadrature rule.

At this point we are set up to either approximate the evidence or draw samples from the posterior. To proceed with approximating the evidence, we draw N “live points” $\theta_1, \theta_2, \dots, \theta_N$ from the prior density $\pi(\Theta)$. We evaluate $\mathcal{L}(\theta)$ at each of these N live points

and arrange them in decreasing value of \mathcal{L} . The point θ_i with the lowest likelihood value is then removed from the live points and we draw a new point from the prior density with the condition that the new point θ_{N+1} must be drawn from a region such that $\mathcal{L}(\theta_{N+1}) > \mathcal{L}(\theta_i)$. We then repeat this process until the desired stopping point, being sure to store the removed points for later calculations. In doing this evaluate and replace process, we are ensuring that at each step of the algorithm the live points are drawn from an iso-contour of the likelihood function with a larger value. Therefore at each step of the algorithm we are moving to an iso-contour of the likelihood function nested within the previous step. Hence the name Nested Sampling. This technique avoids the naive approach of randomly drawing points from the prior which might actually be in a region of negligible likelihood value. After this nested sampling is terminated the evidence can be approximated by typically using all previous points, even the points removed from the set of live points at each iteration. There is a bit of work to be done in determining the size reduction of the prior mass X_i at each iteration, but we will leave that discussion for [4].

If we instead wanted to sample from the posterior, we would start by following the procedure outlined above for estimating the evidence. At the end of this procedure we would have the set of N currently live points as well as all of the points we removed from this set. Each of these points is a draw from the posterior distribution with the weight

$$p_i = \frac{\mathcal{L}_i \omega_i}{\sum_j \mathcal{L}_j \omega_j}.$$

To see this, note that the posterior distribution is simply the prior distribution weighted by the likelihood. The points θ_i obtained by performing the Nested Sampling algorithm are first drawn by the prior and then given importance based on their evaluation in the likelihood function. Therefore these points are draws from the posterior assuming we give them the proper weight, which we are doing by assigning p_i to each point θ_i .

As stated above, the MultiNest algorithm is simply an extension of the Nested Sampling algorithm which addresses a few of the more difficult aspects of the algorithm. One of the main difficulties with the Nested Sampling algorithm is in drawing unbiased samples at each iteration such that the likelihood value of that sample is larger than the previously removed point. The MultiNest algorithm works around this by enclosing the N live points at each step by one or more ellipsoids meant to mimic the iso-contours of the likelihood function. We will leave the description of the construction of these ellipsoids to the paper itself [4].

4.3 Comparison with MCMC Methods

There is a bit of care that must be taken when discussing a comparison between MultiNest and MCMC methods such as Random-walk MH and HMC. There are MCMC methods that

exist to calculate the Bayesian evidence as opposed to sample from the posterior. So when most authors state that MultiNest is more efficient as compared to MCMC methods, they typically mean that MultiNest is more efficient in computing the Bayesian evidence. In [4] there is a section which analyzes how MultiNest and a tailored version of the MH algorithm compare with respect to quickly drawing from a posterior distribution in order to get an initial feel for the data. MultiNest is shown to compare favorably to the MCMC method in that setting, but it is unclear how MultiNest compares when a thorough analysis of the posterior is required. It seems (based on relatively little comparison in the literature) that MultiNest is a plausible alternative to MCMC methods in sampling from the posterior when the dimension of parameter space is low and when the posterior distribution is multimodal. How low the dimension needs to be for this algorithm to remain a plausible alternative is unclear at this stage. It does seem clear however that as the dimension of the parameter space grows the performance of MultiNest begins to break down. This is due to the fact that MultiNest samples uniformly over parameter space, so when the dimension of parameter space is large the algorithm will take longer to select points in a region of higher likelihood value. In this way the appeal of HMC for sampling in parameter spaces of higher dimensions remains.

References

- [1] Andrieu and Roberts. The pseudo-marginal approach for efficient monte carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.
- [2] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv:1701.02434 [stat.ME]*, Jan 2017.
- [3] Brooks, Melman, Jones, and Meng. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall, 2011.
- [4] Feroz, Hobson, and Bridges. Multinest: an efficient and robust bayesian inference tool for cosmology and particle physics. *Mon. Not. Roy. Astron. Soc.*, 2009.
- [5] Feroz, Hobson, Cameron, and Pettitt. Importance nested sampling and the multinest algorithm. *arXiv:1306.2144 [astro-ph.IM]*, 2013.
- [6] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.
- [7] Jonathan H. Manton. A primer on reproducing kernel hilbert spaces. *Foundations and Trends in Signal Processing*, 2015.
- [8] Sherlock, Fearnhead, and Roberts. The random walk metropolis: Linking theory and practice through a case study. *Statistical Science*, 25(2):172–190, 2010.
- [9] John Skilling. Nested sampling for general bayesian computation. *Bayesian Analysis*, 1(4):833–859, 2006.
- [10] Sriperumbudur, Fukumizu, Gretton, Hyvärinen, and Kumar. Density estimation in infinite dimensional exponential families. *Journal of Machine Learning Research*, 18:1–59, 2017.
- [11] Strathmann, Sejdinovic, Livingstone, Szabo, and Gretton. Gradient-free hamiltonian monte carlo with efficient kernel exponential families. *Advances in Neural Information Processing Systems*, 28, 2015.
- [12] John R. Taylor. *Classical Mechanics*. University Science Books, 2005.