STAT 547: Bayesian Workflow

Charles C. Margossian
University of British Columbia
Winter 2026
`https://charlesm93.github.io/stat_547/`

DRAFT

4 Hamiltonian Monte Carlo

We've discussed MCMC in a fairly general setting. In this section, we examine a specific subclass of MCMC, called Hamiltonian Monte Carlo (HMC).

HMC has completely modernized MCMC in the 2010's and it arguably remains the most popular "off-the-shelf" inference algorithm for Bayesian analysis—although it certainly does not solve every problem, nor is it the only good candidate you should consider.

Common HMC algorithms tend to yield good results for a broad range of high-dimensional targets with a reasonable geometry, i.e. targets with finite curvature and a single mode (or sometimes multiple modes that are not too disconnected). Certain classes of HMC can handle non-finite curvature and, in general, algorithms designed for more intricate geometries can leverage HMC, for example to do location exploration within a mode.

One motivation for using HMC is that it often scales better in dimension than random-walk Metropolis algorithms, a fact that is often observed in practice. There is also a formal argument for this. Consider a d-dimensional target distribution. Then, under somewhat idealized conditions, the computational cost of HMC scales as $\mathcal{O}(d^{5/4})$, rather than the $\mathcal{O}(d^2)$ cost characteristic of random-walk algorithms.[1]

Interestingly, HMC itself is fairly old: it was introduced in a 1987 paper on quantum chromodynamics and its original name was *hybrid Monte Carlo* [Duane et al., 1987]. Adoption of the technique in the applied statistics community was slow because:

(i) the algorithm requires gradient calculations,

(ii) the algorithm is difficult to tune.

The method had some success in the 1990's and 2000's, thanks to Radford Neal's pioneering work on Bayesian neural networks. Around 2012, the creators of Stan popularized the method by creating a probabilistic language with automated gradient calculation[2] and a self-tuning HMC algorithm, called the No-U-Turn sampler (NUTS, Hoffman and Gelman [2014]).

[1] Consider a d-dimensional target distribution p and suppose this target factorizes, with each factor equal, $p(z) = \prod_{i=1}^d p(z_i)$. (Naturally, this scenario is simpler than what we encounter in practice, although it can be generalized a bit.) Assume the Markov chain is already stationary. Then in this setting, the computational cost of increasing the ESS by 1 with a random-walk Metropolis algorithm is $\mathcal{O}(d^2)$, but only $\mathcal{O}(d^{5/4})$ for HMC. For further discussion, see Neal [2011] and references therein.

[2] The efficient calculation of gradients is done using *automatic differentiation*, a broad class of techniques to calculate derivatives of functions specified as computer programs. The *reverse-mode* of automatic differentiation is known as backward propagation and underlies much of machine learning. See e.g., Baydin et al. [2018], Margossian [2019] for an introduction on the topic.

4.1 Ideal HMC

Suppose we want to construct an MCMC algorithm to sample from a target density $\pi(z)$ defined over $\mathbb{R}^d$. We'll assume that π is differentiable.

A standard[3] HMC transition kernel proceeds as follows:

1. Start at the Markov chain's current position $z_0 = z^{(i)}$.
2. Draw an auxiliary "momentum" variable, from a normal with covariance matrix M,

$$\rho_0 \sim \text{Normal}(0, M). \tag{1}$$

3. Simulate a trajectory over time T by solving Hamilton's equations of motions:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} z_t &= -\nabla_\rho \log \pi(\rho_t) = M^{-1} \rho_t & (2) \\ \frac{\mathrm{d}}{\mathrm{d}t} \rho_t &= \nabla_z \log \pi(z_t), & (3) \end{aligned}$$

   with initial conditions at time $t_0 = 0$ given by (z_0, ρ_0).

4. Update the sate of the Markov chain, $z^{(i+1)} = z_t$.

Showing *why* this algorithm is effective takes a little bit of work.

Intuition. Hamilton's equations can be interpreted as describing the movement of a particle over time, with the position of the particle given by z_t and its momentum by ρ_t. The particle is subject to a potential, determined by $-\log \pi(z)$, and this potential determines how the particle accelerates.

To simplify things, take $M = I$ (the identity matrix). Then, the change in position over time, $\partial z_t / \partial t$—or "velocity"—is given by the momentum ρ_t (eq. (2)). It turn, the change in momentum, $\partial \rho_t / \partial t$—or "acceleration"—is given by $\nabla_z \log \pi(z)$(eq. (3)). That is, the particle accelerates when it moves in a direction with a positive gradient and it decelerates when the gradient is negative.

Tuning. A critical question is how large T should be: that is, for how long do we simulate a Hamiltonian trajectory before we resample the momentum and start the next iteration of MCMC?

- If T is too small, then at each iteration, the Markov chain doesn't travel far in the parameter space and the MCMC samples are strongly autocorrelated.
- But increasing T beyond a certain threshold does not generate less correlated samples (e.g. as the trajectory starts to backtrack) and leads to a large computational cost per iteration.

Demo. Comparison between HMC, Gibbs and Metropolis-Hastings on a high-dimensional and ill-conditioned Gaussian target.

We'll now show that HMC has the correct stationary distribution. To do so, we first review key properties of Hamiltonian trajectories.

[3] The algorithm can be specified in a more general way but here we'll focus on the standard implementation.

We denote H the *Hamiltonian* of the system, defined as the negative log joint density over z and ρ, that is,

$$H(z_t, \rho_t) = -\log \pi(z_t, \rho_t) = -\log \pi(z_t) - \log \pi(\rho_t). \tag{4}$$

With this notion, the equations of motion can be rewritten in their general form,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} z_t &= \frac{\partial H}{\partial \rho} \\ \frac{\mathrm{d}}{\mathrm{d}t} \rho_t &= -\frac{\partial H}{\partial z}. \end{aligned} \tag{5}$$

Next, we denote $\Phi_T : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$, the function that maps (z_0, ρ_0) to (z_t, ρ_t).

Lemma 1. *(Properties of the Hamiltonian trajectory) The Hamiltonian trajectory Φ_T verifies the following properties.*

- *For any T, Φ_T preserves the Hamiltonian,*

$$H(z_t, \rho_t) = H(z_0, \rho_0). \tag{6}$$

  Equivalently, the joint distribution over (z, ρ) is preserved.

- *(Louiville's theorem) The Hamiltonian map Φ_T is volume preserving, that is the determinant of the Jacobian Φ_T is 1.*

- *The Hamiltonian map Φ_T is time-reversible. That is, it admits an inverse Φ_T^{-1} which is obtained by negating the time derivative in eq.* (2)–(3). *Furthermore, we can obtain the inverse map Φ_T^{-1} by negating ρ, applying Φ_T and negating ρ again.*

Here, I'll only provide a proof of the first item and the proof for the other properties is omitted.

Proof. (Conservation of the Hamiltonian) Taking the derivative of the Hamiltonian with respect to time,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} H(z_t, \rho_t) &= \sum_{i=1}^{d} \frac{\partial H}{\partial z_i} \frac{\mathrm{d}z_i}{\mathrm{d}t} + \frac{\partial H}{\partial \rho_i} \frac{\mathrm{d}\rho_i}{\mathrm{d}t} \\ &= \sum_{i=1}^{d} \frac{\partial H}{\partial z_i} \frac{\partial H}{\partial \rho_i} - \frac{\partial H}{\partial \rho_i} \frac{\partial H}{\partial z_i} = 0, \end{aligned} \tag{7}$$

where on the second line, we plugged in (5).

□

We now show that HMC has the correct stationary distribution.

Theorem 2. *The transition kernel for HMC admits $\pi(z)$ as its stationary distribution.*

There are multiple ways to prove this result. The first way is to directly show that HMC has the desired stationary distributions. The second way is to show reversibility (reversibility). While the second approach is less direct, it lays the foundation for showing that non-idealized versions of HMC have the right stationary distribution.

Here, we'll go over both proofs.

Proof. (Direct approach)

We will show that the joint distribution $\pi(z, \rho)$ is stationary and from this it will follow that the marginal distribution $\pi(z)$ is also stationary.

First, we start with $(z, \rho) \sim \pi$. The first step of HMC is to refresh the momentum, that is draw $\rho^* \sim \pi$. By construction $\pi(z, \rho) = \pi(z)\pi(\rho)$ and so z and ρ are independent. Hence, it suffices that z and ρ^* are marginally distributed according to π to have (z, ρ^*) be jointly distributed according to π and so the first step of HMC leaves the distribution π invariant.

The next step is to simulate a Hamiltonian trajectory. For ease of notation, we denote $x = (z, \rho^*)$. We need to show that for any measurable set A over $\mathbb{R}^{2d}$, $P(x \in A) = P(\Phi_T(x) \in A)$. Notice that $\{\Phi_T(x) \in A\} \iff \{x \in \Phi_T^{-1}(A)\}$ and so we must show $P(x \in A) = P(x \in \Phi_T^{-1}(A))$.

Now,

$$P(x \in A) = \int_A \pi(x)\mathrm{d}x. \tag{8}$$

Let $y = \Phi_T^{-1}(x)$. Doing a change of variable,

$$\int_A \pi(x)\mathrm{d}x = \int_{\Phi_T^{-1}(A)} \pi(\Phi_T(y))|J_\Phi(y)|\mathrm{d}y, \tag{9}$$

where $|J_\phi(y)|$ is the determinant of the Jacobian of Φ_T.

By Lemma 1, the Hamiltonian is conserved and so $\pi(\Phi_T(y)) = \pi(y)$. Furthermore, $|J_\phi(y)| = 1$.

Therefore,

$$\int_{\Phi_T^{-1}(A)} \pi(\Phi_T(y))|J_\Phi(y)|\mathrm{d}y = \int_{\Phi_T^{-1}(A)} \pi(y)\mathrm{d}y. \tag{10}$$

But the integral on the right-hand-side is exactly $P(x \in \Phi_T^{-1})$, and so $P(x \in A) = P(x \in \Phi_T^{-1}(A))$, as desired.

Thus both steps of HMC preserve the distribution $\pi(x)$ and thence the marginal distribution $\pi(z)$.

□

Next, we consider a proof that shows reversibility.

Proof. (using reversibility)

As in the previous proof, we have that the first step of HMC (momentum refreshment) leaves π invariant.

For the second step, we introduce a modification of the HMC transition kernel: namely, after time T we flip the momentum, so that the final state of the transition is

$$(z', \rho') = (z_T, -\rho_T). \tag{11}$$

This does not actually induce any algorithmic change, since the momentum is refreshed at the beginning of the next iteration and, ultimately, we're only interested in the position variable z. However the momentum flip ensures the algorithm maintains reversibility.

To see this, denote (z_0, ρ_0) the initial state of the trajectory. The Hamiltonian map induces a conditional probability,

$$p(z, \rho \mid z_0, \rho_0) = \delta(z = z_T, \rho = -\rho_T), \tag{12}$$

where δ is the Dirac delta function (meaning all the probability mass concentrates at a single point). Conversely, it follows from the reversibility of the Hamiltonian trajectory that,

$$p(z, \rho \mid z', \rho') = \delta(z = z_0, \rho = -\rho_0). \tag{13}$$

Therefore,

$$\mathbb{P}(z', \rho' \mid z_0, \rho_0) = \mathbb{P}(z_0, \rho_0 \mid z', \rho'). \tag{14}$$

In words, if (z', ρ') and (z, ρ) are each other's "reciprocal" on a Hamiltonian trajectory of length T, meaning $(z' = z_T, \rho' = -\rho_T)$ and $(z = z_0, \rho = \rho_0)$, then the probability on either side is 1, else it is 0.

Next, recall that the Hamiltonian is conserved and so $H(z_T, \rho_T) = H(z_0, \rho_0)$. Furthermore, $H(z, \rho) = -\log \pi(z, \rho) = -\log \pi(z) - \log \pi(\rho)$. Since $\pi(\rho)$ is a symmetric distribution, $\pi(-\rho) = \pi(\rho)$. Therefore $H(z', \rho') = H(z_T, \rho_T) = H(z_0, \rho_0)$ and moreover,

$$\pi(z', \rho') = \pi(z_0, \rho_0). \tag{15}$$

Combining equations (14) and (15), we have

$$\pi(z_0, \rho_0)\mathbb{P}(z', \rho' \mid z_0, \rho_0) = \pi(z', \rho')\mathbb{P}(z_0, \rho_0 \mid z', \rho'), \tag{16}$$

which is reversibility. Thus, the Hamiltonian map leaves π invariant, which completes the proof.

□

4.2 Discretized HMC

In practice, we cannot solve Hamilton's equations of motion exactly and so we resort to a numerical integrator. The most elementary integrator for solving differential equations is *Euler's method*, which uses a tangent approximations.

Specifically, at each iteration of Euler's method, we increment (z, ρ) by a step ϵ in the direction of the tangent $(\mathrm{d}z/\mathrm{d}t, \mathrm{d}\rho/\mathrm{d}t)$. If integrating from 0 to T, we repeat this process for L steps, such that $L\epsilon = T$. That is, we compute a discretized trajectory over $\{0, \epsilon, 2\epsilon, \cdots, L\epsilon\}$.

(Euler's method)

```
for $i$ in $\{1, \cdots, L\}$ do
    $\rho(t + \epsilon) \leftarrow \rho_t + \epsilon \nabla \log \pi(z_t)$
    $z(t + \epsilon) \leftarrow z_t + \epsilon M^{-1} \rho_t$
end for
```

In practice, Euler's method works poorly and accumulates a large error as L increases.

```
(Leapfrog integrator)
  for $i$ in $\{1, \cdots, L\}$ do
      $\rho_{t+\epsilon/2} \leftarrow \rho_t + \frac{\epsilon}{2} \nabla \log \pi(z_t)$
      $z_{t+\epsilon} \leftarrow z_t + \epsilon M^{-1} \rho_{t+\epsilon/2}$
      $\rho_{t+\epsilon} \leftarrow \rho_{t+\epsilon/2} + \frac{\epsilon}{2} \nabla \log \pi(z_{t+\epsilon})$
  end for
```

A simple but surprisingly effective modification leads to the *leapfrog integrator*, which is what is used in practice to run HMC.

Question: How many gradient evaluations per step does the leapfrog integrator require?

Tuning problem. The leapfrog integrator requires choosing a step size ϵ. If ϵ is too large, then we do not simulate accurate Hamiltonian trajectories. On the other hand, a small ϵ means we require more steps (i.e. a larger L) in order to perform integration over $[0, T]$.

One way to verify the accuracy of the leapfrog integrator is to check that the Hamiltonian $H(z_t, \rho_t)$ is indeed conserved over time. But even for an adequate ϵ, the error remains non-zero, which invalidates our argument that the Hamiltonian map verifies reversibility.

Metropolis adjustment. To ensure that the discretized HMC verifies reversibility, we can perform a Metropolis correction. Starting at a state (z, ρ), we obtain a new state (z_t, ρ_t) by simulating the Hamiltonian for time T. We then generate a proposal,

$$(z^*, \rho^*) = (z_t, -\rho_t). \tag{17}$$

Notice the sign change in ρ! We then accept the proposal with probability,

$$\alpha = \min\left(1, e^{-H(z^*, \rho^*) + H(z, \rho)}\right). \tag{18}$$

Computing the exponentiated difference in the Hamiltonian is equivalent to evaluating a ratio of the densities, $\pi(z^*, \rho^*)/\pi(z, \rho)$, usually seen in the Metropolis acceptance rule.

In practice, we can ignore the sign flip in ρ. Since ρ is distributed according to a normal centered at 0, ρ_t and $-\rho_t$ have the same density. In other words, flipping the momentum does not change the Hamiltonian. Furthermore, the new state ρ^* can be ignored because the first step of HMC is to refresh the momentum. Still, the sign flip is a useful theoretical tool which lets us prove reversibility.

Other strategies. There exist other strategies to correct the numerical error introduce by the leapfrog integrator. For example:

- **Multinomial sampling:** draw a sample from the *entire* discrete trajectory with the probability of drawing any sample weighted by $\exp(-H(z_t, \rho_t))$. This can be effective if the integration error at (z_T, ρ_T) is large, even though it is acceptable along other points on the trajectory.
- **Unadjusted sampling:** In some cases, the error introduced by the leapfrog integrator is sufficiently small that the correction is ineffective and it is better to accept aggressively. This leads to *unadjusted* samplers.

4.3 Adaptive Hamiltonian Monte Carlo

There are three tuning parameters in HMC: the step size ϵ of the leapfrog integrator, the number L of leapfrog steps (with the trajectory length given by $T = L\epsilon$), and the mass matrix M.

4.3.1 Adaptively setting the path length L

We start by assuming ϵ is fixed and to simplify the notation, we take $M = I$.

No-U Turn criterion. Conceptually, large steps in the state space reduce the autocorrelation between samples in the Markov chain and, at stationarity, leads to a larger ESS per iteration. Therefore, running a Hamiltonian trajectory for a longer time is useful in so far as it increases the distance from the initial point. Once the trajectory starts backtracking, the benefits of running a longer trajectory decreases and we get less out of the computational work we do to simulate the Hamiltonian trajectory.

This reasoning motivates the *No-U Turn* criterion, whereby we monitor whether increasing the trajectory length T increases the distance from the initial point. This idea is at the heart of the *No-U Turn Sampler* (NUTS) which underlies the MCMC algorithm in Stan, PyMC and other probabilistic languages.

Fortunately, there is a straightforward way to monitor whether a longer trajectory would increase the distance from the starting point. Let z_0 be the initial position and z_t the current position. Then,

$$\frac{\mathrm{d}}{\mathrm{d}t}||z_0 - z_t||^2 = \frac{\mathrm{d}}{\mathrm{d}t}(z_0 - z_t)^T(z_0 - z_t) = 2(z_t - z_0)^T \rho_t. \tag{19}$$

A natural idea then would be to simulate a Hamiltonian trajectory until $(z_t - z_0)^T \rho_t < 0$, at which point we stop in order to not backtrack closer to our starting point.

Reversibility. Unfortunately, the map that simulates a Hamiltonian trajectory with a dynamic trajectory length is not time-reversible (Figure 1) and as a result, we cannot guarantee that the resulting MCMC algorithm is reversible and has the right stationary distribution.

A more sophisticated scheme can address this problem. To make the notation more compact, let $x = (z, \rho)$. The key idea is to stochastically generate a trajectory or *orbit* $\mathcal{I}$ from a starting point x_0, such that

$$\mathbb{P}(\mathcal{I} \mid x_0) = \mathbb{P}(\mathcal{I} \mid x_t), \tag{20}$$

for any $x_t \in \mathcal{I}$. In words, the probability of generating a particular orbit $\mathcal{I}$ is the same no matter which point in the orbit we start from.

Once this orbit is constructed, we set the new point x' in our Markov chain to a random point from the orbit, for example using a multinomial distribution,

$$\mathbb{P}(x' = x_t \mid x_t \in \mathcal{I}) \propto \exp(-H(z_t, \rho_t)) \propto \pi(z_t, \rho_t) = \frac{\pi(x_t)}{\sum_{t' \in \mathcal{I}} \pi(x_{t'})}. \tag{21}$$

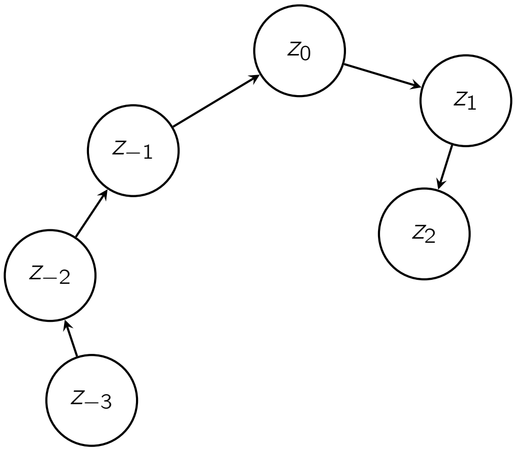

Figure 1: *Example of a (discrete) Hamiltonian trajectory. If starting at point z_0, the algorithm would simulate a trajectory forward in time all the way to z_2, where a U-Turn is detected. On the other hand, starting a trajectory from z_2 (with flipped momentum) may not lead back to z_0 since a U-turn may only occur later, for example at z_{-3}.*

We can explicitly check that this scheme verifies reversibility,

$$\begin{aligned}\pi(x)p(x' \mid x) &= \pi(x)\mathbb{P}(\mathcal{I} \mid x)\mathbb{P}(x' \mid \mathcal{I}) \\ &= \pi(x)\mathbb{P}(\mathcal{I} \mid x)\frac{\pi(x')}{\sum_{t\in\mathcal{I}}\pi(x_t)} \\ &= \pi(x')\mathbb{P}(\mathcal{I} \mid x')\frac{\pi(x)}{\sum_{t\in\mathcal{I}}\pi(x_t)} \\ &= \pi(x')\mathbb{P}(\mathcal{I} \mid x')\mathbb{P}(x \mid \mathcal{I}) \\ &= \pi(x')p(x \mid x'). \end{aligned} \tag{22}$$

Of course, this begs the question of *how* do we construct an orbit which verifies eq. (20)? First, it should be clear that any valid orbit must expand the Hamiltonian trajectory from a starting point z_0 both forward and backward in time.

Additive expansion. A natural if naive way to construct an orbit is with an *additive expansion*.

At each iteration of the expansion, we expand the trajectory forward or backward in time by one leapfrog step. The probability of choosing either direction in time is 1/2. We then stop expanding the trajectory, when a termination criterion based on the No-U-Turn condition is met.

Specifically, each time a new point z_t is added, we check whether it induces a U-Turn with any other existing point in the trajectory. Since we can evolve forward and backward in time, the U-Turn condition is checked at both extremities of each subtrajectory,

$$(z_+ - z_-)^T \rho_+ < 0 \ \text{ AND } \ (z_- - z_+)^T \rho_- < 0. \tag{23}$$

Unfortunately, the additive expansion can be quite expansive for high-dimensional models, because

(i) each extension of the orbit requires us to check the U-Turn condition against all points on the trajectory.

(ii) we need to store every state (z_t, ρ_t), which is memory intensive.

Efficient NUTS. Practical implementation of NUTS rely on a more sophisticated expansion, which I'll only briefly review here. For more details, you may consult Betancourt [2017, Appendix A] and Hoffman and Gelman [2014].

Here's a summary of the strategy used in Stan:

- Starting from an initial point z_0, we randomly pick a direction: backward or forward in time. When then compute one leapfrog step in the chosen direction.
- We repeat this process, but at each step, we double the length of the simulated sub-trajectory. For example, the growing trajectory may look as follows:

  (1) `forward=1`, orbit: $\{x_0, x_1\}$.

  (2) `forward=1`, orbit: $\{x_0, x_1, x_2, x_3\}$

  (3) `forward=0`, orbit: $\{x_{-4}, x_{-3}, x_{-2}, x_{-1}, x_0, x_1, x_2, x_3\}$,

  where at each step the new sub-trajectory is colored in orange. This construction admits a representation as a binary tree.
- At each step, we check for U-turns within the new sub-trajectory and eliminate points which verify the termination criterion in eq. (23). We then sample a candidate (z_t, ρ_t) from the new sub-trajectory, assign a probability weight (which accounts for $H(z_{t'}, \rho_{t'})$ for all points in the sub-trajectory), and only save that particular point.
- Once we've constructed $\mathcal{I}$, we sample from the candidate sample retained from each sub-trajectory using a multinomial distribution. The weight of each sub-trajectory candidate can be chosen to match the distribution in eq. (21). Alternatively, we can favor newer trajectories in order to increase the distance from the starting point z_0. This approach is termed *Biased Progressive Sampling*.

4.3.2 Adaptively setting the step size ϵ

The step size ϵ must be chosen to ensure that the leapfrog integrator is both sufficiently accurate and efficient. Ultimately, this balance ideally results in a Monte Carlo estimator that achieves a target accuracy as quickly as possible.

Step size in Metropolis algorithms. In random-walk Metropolis, we need to tune the size of the random step taken at each iteration and we must navigate the following trade-off: A small step leads to a higher acceptance probability but more correlated samples. Conversely, a large step leads to a lower acceptance probability but less correlated samples. Under certain conditions, it can be shown that a step size with an average acceptance probability of $\alpha = 0.234...$ achieves an optimal ESS/per iteration [Roberts et al., 1997].

Likewise, adaptive HMC targets the acceptance probability α of the Metropolis step in eq. (18). This acceptance probability is driven by fluctuations in the Hamiltonian $H(z_t, \rho_t)$. In the limit $\epsilon \to 0$, the leapfrog integrator simulates exact Hamiltonian trajectories and $\alpha \to 1$.

Step size adaptation as stochastic optimization. The problem of adapting ϵ can be framed as a *stochastic optimization problem*. To see this, we first define the expected acceptance,

$$h(\epsilon) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}(\alpha_n \mid \epsilon), \tag{24}$$

where α_n is the acceptance probability at the n^{th} sampling iteration. Then our goal is to drive,

$$h(\epsilon) \to \delta, \tag{25}$$

for some target acceptance probability δ. For example, Stan's default is $\delta = 0.8$. But the optimal δ can vary largely depending on the problem. The default is a battle-tested heuristic but it may be necessary to adjust δ after a first attempt at running MCMC. In practice, $h(\epsilon)$ cannot be evaluated exactly, rather it is approximated by Monte Carlo using α_n's computed as we run MCMC.

The above task is equivalent to a stochastic optimization problem where our goal is to drive the gradient of an objective function—in this case $h(\epsilon) - \delta$—to 0 and where this gradient can only be evaluated stochastically. This means stochastic optimization algorithms can be used to update ϵ. A common choice for HMC is *dual averaging*.

Strictly speaking, NUTS does not use an accept/reject Metropolis step. Instead, α_n is computed by averaging hypothetical acceptance probabilities over the final sub-trajectory computed when constructing the orbit $\mathcal{I}$.

Freezing adaptation. One final and important caveat is that step size adaptation can alter the stationary distribution. Therefore, it is common practice to only do adaptation during the *warmup* and freeze adaptation (meaning ϵ no longer changes) during the sampling phase.

Certain algorithms, such as delayed-rejection HMC, try to adaptively find an optimal ϵ at each MCMC iteration, including during the sampling phase. But just as with adaptive trajectory lengths, such algorithms required careful constructions to ensure reversibility.

4.3.3 Adapting the mass matrix

The last tuning parameter to consider is the mass matrix M. In practice, we focus on the inverse-mass matrix, M^{-1}, since this is the quantity that appears in the leapfrog integrator. Specifically, the relevant step is,

$$z_{t+\epsilon} \leftarrow z_t + \epsilon M^{-1} \rho_{t+\epsilon/2}. \tag{26}$$

Let's first consider a diagonal mass matrix. Then, from eq. (26), we can interpret M^{-1} as a rescaling of the step size ϵ along each dimension.

Such a rescaling can make sense when the distribution has wildly different scales along each dimension. If we fix the mass matrix to I, then ϵ must typically be small enough to accommodate the dimension with the smallest scale and the integrator can be wildly inefficient along dimensions with a larger scale, where a less granular discretization of the Hamiltonian trajectory may be required.

With this conceptual motivation in mind, Stan uses the inverse sample variance based on samples collected during the warmup phase,

$$M_{ii}^{-1} = \frac{1}{\widehat{\text{var}}(z_i)}. \tag{27}$$

(As with step size, the mass matrix adaptation is frozen after the warmup phase.) That said, finding an optimal adaptation strategy for the mass matrix remains an open question.

Sometimes, the direction along which ϵ needs to be rescaled does not correspond to a direction along one particular dimension. Think for instance of a distribution with strongly correlated variables. In that case, we may use the a non-diagonal mass matrix and set it to the sample covariance matrix.

But this choice can be costly for high-dimensional targets. Indeed, the matrix-vector multiplication in eq. (26) costs $\mathcal{O}(d^2)$ for a dense matrix. By contrast, with a diagonal mass matrix, the cost of this operation reduces to $\mathcal{O}(d)$. Hence, the benefits of using a dense mass matrix must justify the more expensive leapfrog step. Sometimes, a compromised can be struck by using a mass matrix structured as a diagonal matrix + a low-rank matrix.

For some targets, the correct mass matrix depends on where in the target space we are simulating a Hamiltonian trajectory. (The most notorious example of this may well be Neal's funnel, which arises in hierarchical models.) In this case, the mass matrix can be set locally. One choice is to use the *curvature* of the target distribution,

$$\mathcal{C}(z) = -\nabla^2 \log \pi(z). \tag{28}$$

For justification for this choice, see e.g., Girolami and Calderhead [2011]. Unfortunately, this approach tends to be costly, notably through the requirement to compute and store higher-order derivatives of $\log \pi(z)$.

Notice that if the target is Gaussian, then $\mathcal{C}(z) = \Sigma^{-1}$, which resonates with the heuristic of using the covariance matrix to set the mass matrix.

References

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

M. Betancourt. A conceptual introduction to hamiltonian monte carlo, 2017.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987. doi: 10.1016/0370-2693(87)91197-X.

M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011. doi: 10.1111/j.1467-9868.2010.00765.x.

M. D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.

C. C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019. doi: 10.1002/WIDM.1305.

R. M. Neal. Mcmc using hamiltonian dynamics. In S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman and Hall/CRC, 2011.

G. O. Roberts, A. Gelman, and W. R. Gilks. Weak convergence and optimal scaling of random walk metropolis algorithms. *The Annals of Applied Probability*, 7(1):110–120, 1997.