Tuning-Free Generalized Hamiltonian Monte Carlo

Matthew D. Hoffman **Pavel Sountsov**
Google Research

Abstract

Hamiltonian Monte Carlo (HMC) has become a go-to family of Markov chain Monte Carlo (MCMC) algorithms for Bayesian inference problems, in part because we have good procedures for automatically tuning its parameters. Much less attention has been paid to automatic tuning of *generalized* HMC (GHMC), in which the auxiliary momentum vector is partially updated frequently instead of being completely resampled infrequently. Since GHMC spreads progress over many iterations, it is not straightforward to tune GHMC based on quantities typically used to tune HMC such as average acceptance rate and squared jumped distance. In this work, we propose an ensemble-chain adaptation (ECA) algorithm for GHMC that automatically selects values for all of GHMC's tunable parameters each iteration based on statistics collected from a population of many chains. This algorithm is designed to make good use of SIMD hardware accelerators such as GPUs, allowing most chains to be updated in parallel each iteration. Unlike typical adaptive-MCMC algorithms, our ECA algorithm does not perturb the chain's stationary distribution, and therefore does not need to be "frozen" after warmup. Empirically, we find that the proposed algorithm quickly converges to its stationary distribution, producing accurate estimates of posterior expectations with relatively few gradient evaluations.

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

1 INTRODUCTION

Hamiltonian Monte Carlo (HMC; Duane et al., 1987; Neal, 2011) is a workhorse of Bayesian inference. Its use of gradients and auxiliary momentum variables allows it to sample from high-dimensional, poorly conditioned distributions relatively efficiently. Automatic-differentiation systems such as Stan (Carpenter et al., 2017) eliminate the need to manually code gradients, and adaptive HMC variants such as the no-U-turn sampler (NUTS; Hoffman and Gelman, 2011) eliminate hard-to-tune parameters.

Generalized HMC (GHMC; Horowitz, 1991) has received less attention. Where HMC uses many leapfrog steps to generate a single Metropolis proposal, GHMC can use as little as one leapfrog step per iteration, and suppresses random-walk behavior by only partially updating the auxiliary momentum vector. This makes GHMC cheaper to interleave with other algorithms; for example, Neal (2020) interleaves GHMC updates with Gibbs updates for discrete latent variables, and Sohl-Dickstein and Culpepper (2012) use GHMC within annealed importance sampling (Neal, 2001) to allow for a finer annealing schedule. GHMC is also interesting for its connection with the underdamped Langevin SDE, which has received much attention from theorists and practitioners recently (e.g., Dalalyan and Riou-Durand, 2020; Wenzel et al., 2020, among many others). Finally, GHMC's SGD-like structure makes it well suited to modern vectorized hardware accelerators and software, unlike the control-flow-heavy NUTS algorithm.

One downside to the original GHMC scheme proposed by Horowitz (1991) is that it requires a much smaller step size than standard HMC, but recently Neal (2020) proposed a non-reversible slice-sampling scheme that makes GHMC much more competitive.

But there is another issue standing between GHMC and widespread adoption: it is not clear how to automatically tune its parameters. This is the problem we tackle in this work.

We consider a version of GHMC with tunable parameters controlling step size, damping, slice drift speed, and a diagonal preconditioning matrix. Working within the framework of ensemble-chain adaptation (ECA; Gilks et al., 1994), which allows for tuning of MCMC parameters while maintaining the correct stationary distribution, in Section 4 we derive heuristics for setting all of GHMC's parameters. In Section 5, we demonstrate that the resulting algorithm is competitive with other turnkey HMC methods such as ChEES-HMC (Hoffman et al., 2021) and NUTS (Hoffman and Gelman, 2011). In summary, our contributions include:

- We discuss how to derive ECA-MCMC algorithms that can take full advantage of parallel-compute resources such as GPUs. These algorithms can automatically select appropriate values for their tunable parameters while maintaining the correct stationary distribution.
- We propose ECA-friendly heuristics for selecting appropriate step size, damping, and slice drift speed parameters for GHMC with the slice-sampling scheme of Neal (2020). We also show how to apply the heuristics in conjunction with ECA preconditioning.
- We propose a heuristic for estimating the largest eigenvalue of a matrix from a noisy, low-rank estimate of that matrix.
- Putting these pieces together, we demonstrate empirically that the proposed tuning-free algorithm can compete with strong adaptive-HMC algorithms like NUTS and ChEES-HMC.

2 BACKGROUND AND RELATED WORK

In this section we review the tools we will build on to derive a self-tuning generalized HMC (GHMC) algorithm: HMC and GHMC; a slice-sampling extension that makes GHMC much more efficient; and the framework of ensemble-chain adaptation, which lets an ensemble of states inform each others' proposals. Throughout, we will assume that we are interested in sampling from an differentiable unnormalized distribution $p(\theta)$ over some $\theta \in \mathbb{R}^D$.

2.1 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC; Duane et al., 1987; Neal, 2011) generates proposals by first introducing an auxiliary vector of "momentum" variables m such that $p(\theta, m) = p(\theta)\mathcal{N}(m; 0, I)$. Interpreting θ as the position of a hypothetical particle and treating $-\log p(\theta)$ as a potential energy function and $-\log \mathcal{N}(m; 0, I)$ as a kinetic energy, we can simulate the Hamiltonian dynamics of that particle. Since Hamiltonian dynamics are reversible, preserve volume, and conserve energy (and therefore conserve $\log p(\theta, m)$), evolving the state θ, m according to the exact dynamics and negating the momentum would yield a reversible, deterministic Metropolis proposal (Metropolis et al., 1953) with acceptance probability 1. We can then resample the momentum m from its standard-normal distribution, and repeat.

In practice, we must discretize the Hamiltonian dynamics, and we typically use the leapfrog integrator, which requires one evaluation of the gradient $\nabla \log p$ per integrator step (with gradient caching), is reversible, and preserves volume. However, it does not conserve energy exactly, and so we must apply a Metropolis correction to ensure detailed balance; if we propose a new state θ', m', then we reject this move with probability $\max\{0, 1 - \frac{p(\theta', m')}{p(\theta, m)}\}$. Using smaller step sizes ϵ leads to smaller energy changes and higher acceptance rates, but increases the number of steps we must take to make a given amount of progress.

While the standard leapfrog integrator has a scalar step-size parameter ϵ, one can also implement it using per-dimension step sizes $\epsilon_{1:D}$; this is the integrator described in Algorithm 1. Neal (2011) shows that this is equivalent to either using a diagonal covariance matrix (sometimes called a mass matrix) for $p(m)$ or linearly rescaling the dimensions of θ to $\tilde{\theta}_d = \theta_d / \epsilon_d$. Such a rescaling can improve HMC's efficiency if it improves the conditioning of $\log p(\theta)$ (Langmore et al., 2019).

HMC's performance also depends strongly on the number of leapfrog steps taken between momentum-resampling steps. Too few steps and the chain will explore the space by a slow random walk; too many and we waste computation. A popular extension of HMC, the no-U-turn sampler (NUTS; Hoffman and Gelman, 2011) automatically decides when to resample the momentum based on when the dynamics start to double back and make a "U turn". However, Hoffman et al. (2021) observe that NUTS chains can be expensive to run in parallel on modern hardware accelerators such as GPUs and TPUs, and propose an alternative adaptive-MCMC strategy (called ChEES-HMC) to tune HMC's number-of-leapfrog-steps parameter.

2.2 Generalized Hamiltonian Monte Carlo

Horowitz (1991) developed a generalization of HMC (GHMC) in which the momentum is only partially updated each iteration. At the beginning of each iteration, instead of resampling $m \sim \mathcal{N}(0, I)$, we apply the update $m \sim \mathcal{N}(m\sqrt{1-\alpha}, \alpha I)$, where $\alpha \in (0, 1]$

is a scalar that controls how much m changes. This leaves the standard-normal distribution over m invariant, since it effectively adds a zero-mean normal with variance α to one with variance $1-\alpha$.

Next, as in HMC, we propose updating the state θ, m by applying one or more leapfrog updates and then negating the momentum (to make the proposal reversible), and accept or reject this move according to the usual Metropolis ratio. Finally, we unconditionally negate the momentum; if we accepted the leapfrog move, this will undo the negation from the previous step, but if we rejected the leapfrog move then it will make the chain "bounce" and reverse course. In standard HMC ($\alpha = 1$) this would be unnecessary, since m would immediately be completely resampled. But in GHMC, where we only partially update the momentum, allowing the momentum to be negated would cause the chain to reverse direction and undo some of the progress that it made on the previous iteration.

If we take one leapfrog step per iteration (as we will assume from here onward), in the limit where the step size ϵ and update amount α become small, all proposals are accepted and the dynamics simulate the underdamped Langevin SDE:

$$d\theta = mdt; \quad dm = (\nabla \log p(\theta) - \gamma m)dt + \sqrt{2\gamma} dW(t), \tag{1}$$

where we define the damping coefficient $\gamma = \frac{\alpha}{2\epsilon}$. Discretizations of underdamped Langevin dynamics resemble sampling analogs of gradient descent with momentum, and have receieved much positive attention from theorists recently (e.g., Cheng et al., 2018; Dalalyan and Riou-Durand, 2020; Ma et al., 2021). Like HMC, underdamped Langevin dynamics use momentum to suppress inefficient random-walk behavior.

So why is GHMC not as widely used in practice as HMC variants that completely resample the momentum each iteration? One answer has to do with rejections. Suppose both HMC and GHMC must simulate their dynamics for T leapfrog steps without a rejection to make optimal progress, and suppose that for both algorithms the energy after t steps is E_t. HMC faces a single accept-reject decision based on the total energy, whereas GHMC must endure a gauntlet of possible rejections:

$$\begin{aligned}\mathbb{P}^{\text{accept}}_{\text{GHMC}} &= \textstyle\prod_t \min\{1, e^{E_{t-1}-E_t}\} \\ &= \min\{1, \textstyle\prod_t \min\{1, e^{E_{t-1}-E_t}\}\} \\ &= \min\{1, \exp\{\textstyle\sum_t \min\{0, E_{t-1}-E_t\}\}\} \\ &\leq \min\{1, \exp\{\textstyle\sum_t E_{t-1}-E_t\}\} \\ &= \min\{1, e^{E_0-E_T}\} = \mathbb{P}^{\text{accept}}_{\text{HMC}}.\end{aligned} \tag{2}$$

So GHMC pays a price each time the energy increases, even if that energy is offset by a subsequent decrease, whereas HMC is robust to energy fluctuations around a stable mean. Symplectic integrators such as the leapfrog are celebrated for their tendency to produce stable fluctuations over long trajectories rather than accumulating energy errors (Hairer et al., 2006). Unlike HMC, GHMC must aggressively control these fluctuations by using a very small step size if it is to avoid random-walk behavior caused by rejections.

Fortunately, Neal (2020) recently proposed a solution to this problem based on slice sampling (Neal, 2003). We can augment the system with an auxiliary scalar slice variable $s \sim \text{Uniform}(0, p(\theta, m))$, so that the joint probability is uniform: $p(\theta, m, s) \propto \mathbb{I}[s \leq p(\theta, m)]$.

Now, holding s fixed, we can propose a new $\theta', -m' = \text{leapfrog}(\theta, m; \epsilon)$, which we will accept as long as $s \leq p(\theta', m')$. If we resampled s each iteration, this would be equivalent to the usual GHMC proposal. But if we leave s fixed for T steps, then we will accept all T steps as long as the energy never increases so much that $E_t - E_0 \geq \log p(\theta, m) - \log s$. This will cause our accept and reject decisions to cluster in time—when we sample a relatively "permissive" s (i.e., when s is significantly less than $p(\theta, m)$), we will tend to accept many steps in a row even with a large step size.

Rather than fully resample s periodically, Neal (2020) proposes a non-reversible update scheme for the reparameterized slice variable $u \triangleq \frac{s}{p(\theta,m)}$ that in the absence of energy fluctuations will cause u to trace out a triangle wave whose frequency is controlled by a free parameter δ. A GHMC update using Neal's persistent-Metropolis scheme is outlined in algorithm 1.

Algorithm 1 Persistent-MH Generalized HMC

```
 1: function LEAPFROG($\theta, m; \epsilon$)
 2:   For each $d$, set $m'_d := m_d + \frac{\epsilon_d}{2}\nabla_{\theta_d} \log p(\theta)$.
 3:   For each $d$, set $\theta'_d := \theta_d + \epsilon_d m'_d$.
 4:   For each $d$, set $m''_d := m'_d + \frac{\epsilon_d}{2}\nabla_{\theta_d} \log p(\theta')$.
 5:   return $\theta'$, $m''$.
 6: end function
 7:
 8: function PERS_GHMC($\theta, m, u; \epsilon, \alpha, \delta$)
 9:   Sample $\tilde{m} \sim \mathcal{N}(m\sqrt{1-\alpha}, \alpha I)$.
10:   Set $\tilde{u} := ((u + 1 + \delta) \mod 2) - 1$.
11:   Set $\theta', m' :=$ LEAPFROG($\theta, \tilde{m}; \epsilon$).
12:   if $|\tilde{u}| \leq \frac{p(\theta')\mathcal{N}(m';0,I)}{p(\theta)\mathcal{N}(\tilde{m};0,I)}$ then
13:     return $\theta'$, $m'$, $\tilde{u}\frac{p(\theta)\mathcal{N}(\tilde{m};0,I)}{p(\theta')\mathcal{N}(m';0,I)}$.
14:   else
15:     return $\theta$, $-\tilde{m}$, $\tilde{u}$.
16:   end if
17: end function
```

While this scheme makes GHMC competitive with standard HMC, there is another barrier to GHMC's wide adoption: the need to manually tune its parameters ϵ (step size), α (damping), and now δ (slice drift). Heuristics used to tune HMC do not immediately translate to GHMC. In HMC, the step size ϵ is adapted to achieve some target acceptance rate, but if the damping α is small then GHMC may require a much higher nominal acceptance rate than HMC; the persistent-Metropolis scheme further complicates the interpretation of acceptance rate. Heuristics used to control HMC's trajectory length such as NUTS and ChEES look at variants on expected squared jumped distance (ESJD; Pasarica and Gelman, 2010), but this must be computed with respect to a reference state at the beginning of a leapfrog trajectory; it is not clear what the analogous reference state is for GHMC. Finally, the slice-drift parameter δ is not needed in HMC.

In Section 4 we derive principled ways of setting all of these parameters based on statistics aggregated across multiple chains. But first, we will review a framework that lets us do this tuning without perturbing the chains' stationary distribution.

2.3 Ensemble-chain adaptation

In MCMC, using the state of a Markov chain to control the parameters ϕ of an transition kernel T is not generally valid; formally, the invariance relation $\int_\theta p(\theta)T(\theta' \mid \theta; \phi)d\theta = p(\theta')$ with ϕ fixed does not imply that $\int_\theta p(\theta)T(\theta' \mid \theta; \phi(\theta))d\theta = p(\theta')$ when $\phi(\theta)$ is a function of the current state θ.

However, if we are running multiple chains in parallel, we are allowed to update one of those chains using parameters that depend on the other chains. Such updates can be justified by treating the ensemble of chains as a single meta-chain. Formally, if we denote the kth chain's state θ_k and the set of other chains $\theta_{\setminus k}$, and we want to sample from the product distribution $p(\theta_{1:K}) = \prod_k p(\theta_k)$, then each update to a chain $\theta_k \mid \theta_{\setminus k}$ leaves the stationary distribution invariant:

$$\begin{aligned}&\textstyle\int_\theta (\prod_i p(\theta_i))T(\theta'_k \mid \theta_k; \phi(\theta_{\setminus k}))d\theta \\ &= \textstyle\int_\phi p(\phi) \int_{\theta_k} p(\theta_k)T(\theta'_k \mid \theta_k; \phi)d\theta d\phi = p(\theta'_k).\end{aligned} \tag{3}$$

MCMC procedures with this flavor have a long history, going back at least to Gilks et al. (1994) and the snooker algorithm, and termed "ensemble-chain adaptation" (ECA) by Zhang and Sutton (2011). In the context of (G)HMC algorithms, they have mostly been used to obtain preconditioners (e.g., Zhang and Sutton, 2011; Leimkuhler et al., 2018) (the methods in this paper are complementary to and could be integrated with such quasi-Newton-inspired methods).

ECA algorithms clearly require that we run multiple chains. ECA therefore seems well positioned to exploit the availability of vectorized hardware (such as GPUs and TPUs) and software (such as TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2019), and JAX (Bradbury et al., 2018)), which can cheaply run many chains in parallel. However, if we really must update each state θ_k holding the other states $\theta_{\setminus k}$ fixed, then these parallel resources are wasted. In the next section, we consider the question of how to design ECA algorithms to enable maximum parallelism.

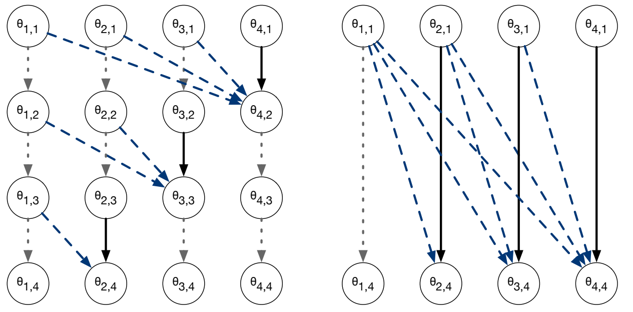

Figure 1: Graphical models illustrating two equivalent ECA procedures. $\theta_{k,t}$ denotes state k at notional iteration t. Solid black lines denote a transition kernel that leaves $p(\theta_k)$ invariant. This kernel may be controlled by some parameters $\phi_k(\theta_1, \ldots, \theta_{k-1})$ that depend on other states; this dependence is denoted by dashed blue lines. Dotted gray lines denote an identity map between $\theta_{k,t}$ and $\theta_{k,t+1}$. In the example above, we can perform the three updates (left) that take us from the joint state $\theta_{1:K,1}$ to $\theta_{1:K,4}$ in parallel (right).

3 PARALLELIZABLE ENSEMBLE-CHAIN ADAPTATION

We would like to design ECA algorithms that can make use of parallel resources by updating multiple states in parallel, but ECA is justified as a one-state-at-a-time serial procedure[1]. However, we will show that we are free to perform ECA updates on many states in parallel as long as these updates obey a certain conditional independence requirement.

Figure 1 illustrates the idea. Conceptually, we do standard one-at-a-time ECA, updating each state θ_k conditioned on information from the $k-1$ states $\theta_{1:k-1}$ (the graphical model on the left). But because each update only depends on information from states with a lower index, we have all the information we need to compute the $K-1$ notionally sequential updates for $\theta_{2:K}$ in parallel. Discarding the $K-2$ notional interme-

[1]One might be tempted to go "hogwild" (Niu et al., 2011) and update all states in parallel, but this can lead to incorrect results; see Appendix D for a simple example.

diate states, we are left with an update that updates all but one of the states (the graphical model on the right[2]). We can then permute the states and repeat the procedure to ensure that all states are updated.

Sometimes we may want to cut some connections from the general update structure in Figure 1. In this work, we use the "K-fold" update structure illustrated in Figure 2, in which we break the states into K "folds" of N states each, compute parameters based on each fold, and share those parameters across all updates in the neighboring fold[3]. We skip one fold's update each iteration to maintain the correct conditional independence structure. After K iterations, we randomly reshuffle the states into K new folds.

This scheme has two main advantages over the denser graphical model in Figure 1: it ensures that the parameters ϕ are always computed based on the same number of states N, which facilitates efficient batching, and it ensures that no parameters are estimated from less than N states, which might lead to undesirable behavior (e.g., momentum-negating rejections in GHMC, which are worse than standing still). Its main downside is that it skips updates for $1/K$ of the states each iteration, possibly reducing utilization of parallel resources. This can be mitigated by increasing K, at the expense of possibly increasing the variance of the kernel parameters ϕ. We use $K = 4$ as a compromise, which incurs at most a 25% slowdown; in Appendix A we empirically explore the effect of adjusting K.

4 ECA-FRIENDLY HEURISTICS FOR GHMC

In this section, we derive ECA-compatible heuristics for automatically tuning parameters of GHMC: a diagonal preconditioning matrix, a step size ϵ, a damping coefficient α, and a slice-drift coefficient δ. Putting these pieces together yields Algorithm 3, which we call Maximum-Eigenvalue Adaptation of Damping and Step-size (MEADS).

4.1 Diagonal preconditioning

(G)HMC algorithms can benefit from preconditioning, and one common practice is to scale each dimension by the inverse of an estimate of the posterior standard deviation in that dimension (e.g.; Carpenter

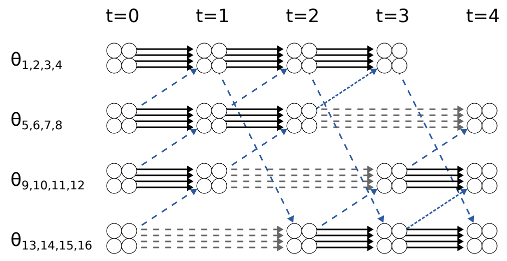

Figure 2: Graphical model illustrating K-fold ECA. The states are split into K folds of N states each, and each fold k is updated using parameters computed from its neighbor fold $k + 1 \mod K$. Each iteration we skip the update for a different fold. Solid black lines denote MCMC updates, dashed blue lines denote dependence through kernel parameters, dashed gray lines denote skipping an update.

et al., 2017; Langmore et al., 2021). This can be implemented by giving the leapfrog integrator a vector of per-dimension step sizes scaled by that dimension's standard deviation (Neal, 2011). In MEADS, we simply compute an estimate $\hat{\sigma}_d$ of the marginal standard deviations σ_d from each fold's states and multiply the neighboring fold's step size accordingly.

In the following sections, we use the transformed variables and gradients $\bar{\theta}_d \triangleq \theta_d/\hat{\sigma}_d$ and $\bar{g}_d \triangleq \nabla_{\hat{\theta}_d} \log p(\theta) = \nabla_{\theta_d} \log p(\theta)\hat{\sigma}_d$ to compute step-size and damping parameters. This yields parameters that are properly adapted to the transformed dynamics.

4.2 Step size

When applied to quadratic potential functions $\frac{1}{2}\theta^\top H\theta$ (corresponding to multivariate-Gaussian target distributions with covariance H^{-1}), the stability of the leapfrog integrator requires that the step size ϵ and largest eigenvalue $\lambda_{\max}$ of H satisfy $\epsilon \leq \frac{2}{\sqrt{\lambda_{\max}}}$ (Leimkuhler and Reich, 2004). More generally, the accuracy of uncorrected underdamped Langevin MCMC likewise depends on keeping the step size inversely proportional to the square root of the largest eigenvalue of the negative Hessian of the log-density (Dalalyan and Riou-Durand, 2020). This suggests trying to set

$$\epsilon := \frac{1}{2}\frac{1}{\sqrt{\lambda_{\max}(-\bar{H})}}; \quad \bar{H} \triangleq \int_\theta p(\theta)\nabla^2 \log p(\theta) d\theta,$$

where $\lambda_{\max}(A)$ is defined as the largest eigenvalue of A. We use the average Hessian $\bar{H}$, since we will estimate $\lambda_{\max}$ from states that may be far from the states we are updating. The $\frac{1}{2}$ is there to give us some margin of error; in Appendix A we show that MEADS is not very sensitive to this factor.

Fortunately, we can estimate $\lambda_{\max}$ from gradients

[2]This graphical model resembles that of inverse autoregressive flows (Kingma et al., 2016), which are also designed to permit efficient parallel sampling.

[3]Leimkuhler et al. (2018) consider a related scheme, updating only one fold at a time conditioned on all other folds, which reduces the number of states that can be updated in parallel by a factor of $K - 1$.

without computing second derivatives. In Appendix E, we show that under mild technical conditions,

$$-\int_\theta p(\theta)\frac{\partial^2 \log p}{\partial\theta\partial\theta^\top}d\theta = \int_\theta p(\theta)(\frac{\partial \log p}{\partial\theta})^\top\frac{\partial \log p}{\partial\theta}d\theta. \quad (4)$$

That is, at stationarity, the expected negative Hessian of the log-density is the expected outer product of the gradient of the log-density. So, given the (transformed) gradients $\bar{g}_{k,1:N}$ for the N (transformed) states $\bar{\theta}_{k,1:N}$ from fold k, we can approximate the average negative Hessian as the outer-product matrix $-\hat{H} \triangleq \frac{1}{N}\sum_n \bar{g}_{k,n}\bar{g}_{k,n}^\top$.

4.3 Damping factor and slice drift

The damping coefficient α controls how many iterations it takes to forget the current value of our momentum m. To see this, note that value of m_{t+i} is roughly (ignoring initial and final leapfrog half-steps)

$$(1-\alpha)^{i/2}m_t + \sum_{j=1}^i \epsilon\nabla\log p(\theta_{t+j}) + \sqrt{\alpha}\xi_j, \quad (5)$$

where $\xi_j \sim \mathcal{N}(0, I)$. m_t's influence on future states m_{t+i} decays exponentially with i.

We want this influence to decay slowly enough that the chain can move far before forgetting its old momentum (since accelerating such motion is why HMC introduced momentum in the first place), but not so slowly that the chain takes too long to forget its previous states (since this forgetting drives mixing). As in standard HMC, we want to forget our momentum once we've had a chance to travel the full length of the highest-variance direction.

When applied to a Gaussian target with covariance Σ, the leapfrog integrator with step size ϵ takes $O(\lambda_{\max}(\Sigma)/\epsilon)$ steps to progress along the least-constrained direction; this suggests that we should choose a damping factor α such that $-\frac{1}{2}\log(1-\alpha) \propto \epsilon/\lambda_{\max}(\Sigma)$ so that the contraction of m after $\lambda_{\max}(\Sigma)/\epsilon$ steps is neither too large nor too small. If α is relatively small (as it should be for difficult problems), then this suggests setting

$$\gamma := 1/\lambda_{\max}(\hat{\Sigma}); \quad \alpha = 1 - e^{-2\epsilon\gamma}, \quad (6)$$

where $\hat{\Sigma}$ is the empirical covariance of a neighboring fold's states $\bar{\theta}$. We use $1 - e^{-x}$ to constrain $\alpha < 1$.

Similar logic applies to the slice-drift parameter δ. We want u to remain stable long enough that we can travel far without rejecting, but not so long that u mixes more slowly than θ and m. We therefore set $\delta := \alpha/2$. This yields a period for u of $4/\alpha$ steps, after which the exponential-decay term in Equation 5 $(1-\alpha)^{2/\alpha} \approx e^{-2} \approx 0.14$, implying that the previous period's momentum has mostly been forgotten.

Algorithm 2 Estimating Largest Eigenvalues

```
1: function MAX_EIG($X$)
2:     $S := XX^\top$
3:     $\lambda_\cdot := \frac{\mathrm{tr}(S)}{N}$.
4:     $\lambda_\cdot^2 := \frac{1}{N(N-1)}\sum_{n,n'\neq n} S_{n,n'}^2$.
5:     return $\lambda_\cdot^2/\lambda_\cdot$.
6: end function
```

For stability, we put a floor on the damping in early iterations to enforce $\gamma_t \geq \frac{1}{t\epsilon_t}$. The logic is that, if the optimal damping would have us forget our momentum after more than t steps, and we have not yet taken t steps, then we have probably not converged to a point where we accurately estimate the appropriate γ. While the chains are far from convergence, the empirical covariance can be large due to differences in how quickly the chains approach the target distribution's typical set; these differences must be damped away before we can safely use a very small damping factor.

4.4 Estimating largest eigenvalues

So far, we have ignored the question of *how* to estimate the largest eigenvalues of the gradient and covariance matrices from Sections 4.2 and 4.3. This turns out to be a bit delicate; the largest eigenvalue of an unbiased estimate $\hat{\Sigma}$ of a matrix Σ can be a highly biased estimate of the largest eigenvalue of Σ.

Instead, we propose an estimator based on the ratio

$$\mathrm{tr}(\Sigma^2)/\mathrm{tr}(\Sigma) = \sum_d \lambda_d^2 / \sum_d \lambda_d, \quad (7)$$

which only relies on our ability to get reasonable estimates of traces, not individual eigenvalues. In the extreme case where all eigenvalues are either 0 or $\lambda_{\max}$, $\frac{\mathrm{tr}(\Sigma^2)}{\mathrm{tr}(\Sigma)} = \lambda_{\max}$. In less-extreme cases, this ratio approximates $\lambda_{\max}$ well unless there are many "small-but-not-tiny" eigenvalues that are large enough to influence the sums, but small enough that they are distinguishable from $\lambda_{\max}$. We explore the properties of this ratio in Appendix C.

The ratio in Equation 7 can be estimated for matrices of the form $X^\top X$ in $O(N^2D)$ operations for $X \in \mathbb{R}^{N\times D}$; Algorithm 2 shows how. The estimator of the trace of $\mathbb{E}[X^\top X]^2$ in line 4 is based on the identity $\mathbb{E}[S_{n,n'}^2] = \mathbb{E}[x_n^\top x_{n'} x_{n'}^\top x_n] = \mathbb{E}[\mathrm{tr}(x_n x_n^\top x_{n'} x_{n'}^\top) = \mathrm{tr}(\mathbb{E}[X^\top X]^2)$ for $n \neq n'$. This cost will be dominated by the cost of computing N gradient evaluations for all but very simple target distributions.

5 EFFICIENCY EXPERIMENTS

In this section, we evaluate MEADS's ability to efficiently estimate posterior expectations. We compare

Algorithm 3 Maximum-Eigenvalue Adaptation of Damping and Step-size (MEADS)

```
Input: $K$ folds of $N$ initial states $\theta_{0,1:K,1:N}$.
Output: Chain of $T$ states $\theta_{1:T,1:K,1:N}$
 1: Initialize $u_{0,1:K,1:N} \sim \text{Uniform}((-1,1))$.
 2: Initialize $m_{0,1:K,1:N} \sim \mathcal{N}(0,I)$.
 3: for $t=1$ to $T$ do
 4:   for $k=1$ to $K$, excluding $k=t \mod K$ do
 5:     Estimate means $\hat{\mu}_{t,k,1:D}$ and standard deviations $\hat{\sigma}_{t,k,1:D}$ from $\theta_{t-1,(k-1) \bmod K,1:D}$.
 6:     Set $\bar{\theta}_{t,k,n,d} := (\theta_{t-1,(k-1) \bmod K,n,d} - \hat{\mu}_{t-1,k,d})/\hat{\sigma}_{t,k,d}$.
 7:     Set $\bar{g}_{t,k,n,d} := \nabla_{\theta_d} \log p(\theta_{t-1,(k-1) \bmod K,n}) \cdot \hat{\sigma}_{t,k,d}$.
 8:     Set $\epsilon_{t,k} := \min\{1, 0.5/\sqrt{\text{MAX\_EIG}(\bar{g}_{t,k})}\}$.
 9:     Set $\gamma_{t,k} := \max\{\frac{1}{t\epsilon_{t,k}}, 1/\sqrt{\text{MAX\_EIG}(\bar{\theta}_{t,(k-1) \bmod K})}\}$.
10:     Set $\alpha_{t,k} := 1 - e^{-2\epsilon_{t,k}\gamma_{t,k}}$, $\delta_{t,k} := \alpha_{t,k}/2$.
11:     Set $\theta_{t,k,n}, m_{t,k,n}, u_{t,k,n} := \text{PERS\_GHMC}(\theta_{t-1,k,n}, m_{t-1,k,n}, u_{t-1,k,n}; \epsilon_{t,k}\hat{\sigma}_{t,k}, \alpha_{t,k}, \delta_{t,k})$.
12:   end for
13: end for
```

MEADS with two baseline HMC algorithms: ChEES-HMC (Hoffman et al., 2021) and NUTS (Hoffman and Gelman, 2011). We implemented MEADS and ChEES-HMC in JAX (Bradbury et al., 2018) on top of the FunMC API (Sountsov et al., 2020), and used TensorFlow Probability's NUTS implementation (Lao and Dillon, 2019). Our MEADS implementation will be open-sourced as part of TensorFlow Probability (Dillon et al., 2017). All experiments were run on TPU v2s with precision set to HIGHEST to avoid bfloat16 matrix multiplication. All algorithms were evaluated on a set of target distributions from the Inference Gym (Sountsov et al., 2020). Table 1 summarizes the target distributions and their dimensionalities.

We let all algorithms adapt per-dimension step sizes. MEADS uses standard-deviation estimates from neighboring folds. For ChEES-HMC and NUTS, we use an exponential moving average of the previous first and second moments with decay rate $\beta = t/(t+8)$.

To adapt step sizes for ChEES-HMC and NUTS, we use Adam (Kingma and Ba, 2015) with a learning rate of 0.05 to tune the average across iterations of the cross-chain harmonic-mean acceptance rate to be approximately 0.8. To adapt trajectory lengths for ChEES-HMC, we use Adam with a learning rate of 0.025 following Hoffman et al. (2021).

For each target distribution, we ran MEADS for 15,000 iterations, thinning the chain by a factor of 10 to save memory. We ran ChEES-HMC and NUTS for 500 iterations, freezing the step-size and (for ChEES-HMC) trajectory-length parameters after 400 iterations. All algorithms were run with 128 chains; MEADS split these 128 chains into four folds. We repeated each experiment 32 times. The initial state $\theta_{0,k,n}$ of all chains of all algorithms is obtained by running 100 iterations of Adam on $-\log p(\theta)$ with learning rate 0.05.

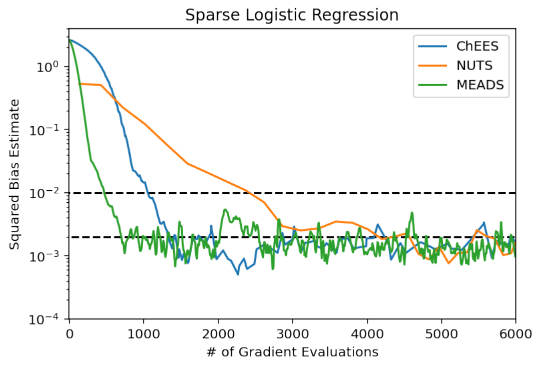

Figure 3: Squared bias estimate versus number of gradients for ChEES, NUTS, and MEADS. Dashed horizontal lines are thresholds of 0.01 and 0.002.

We then estimated the transient bias of each chain as a function of number of iterations; that is, the squared error $(\mathbb{E}[f(\theta_{t,d})] - \mathbb{E}[f(\theta_{\infty,d})])^2$ between the expected value of (a function of) the state of the chain in dimension d after t steps and its expected value at stationarity. To estimate $\mathbb{E}[f(\theta_{t,d})]$, we computed the average across experiments $s \in \{1,\ldots,32\}$ and chains $n \in \{1,\ldots,128\}$ to get $\mathbb{E}[f(\theta_{t,d})] \approx \hat{f}_{t,d} \triangleq \frac{1}{SN}\sum_{s,n} f(\theta_{s,n,t,d})$. Assuming independence between the chains' errors and invoking the central limit theorem, this estimate will have squared error on the order of $\frac{\sigma^2_{f,d}}{4096}$, where $\sigma^2_{f,d}$ is the posterior variance of $f(\theta_d)$. To estimate $\mathbb{E}[f(\theta_{\infty,d})]$, we averaged across the last 100 samples and across all 128 chains and 32 runs of NUTS. We summarize the total bias at iteration t as

$$\text{bias}^2_{f,t} = \max_d (\hat{f}_{t,d} - \mathbb{E}[f(\theta_{\infty,d})])^2/\sigma^2_{f,d}, \tag{8}$$

Full Name	Short Name	Dimensionality
Banana	Banana	2
IllConditionedGaussian	Gaussian	100
GermanCreditNumericLogisticRegression	Logistic	25
GermanCreditNumericSparseLogisticRegression	Sparse	51
RadonContextualEffectsHalfNormalIndiana	Radon	91
BrownianMotionUnknownScalesMissingMiddleObservations	Brownian	32
SyntheticItemResponseTheory	IRT	501
VectorizedStochasticVolatilityLogSP500	Volatility	2519

Table 1: Target distributions used in Section 5.

	Grads to low bias			Grads to very low bias			Grads/ESS		
Target	NUTS	ChEES	MEADS	NUTS	ChEES	MEADS	NUTS	ChEES	MEADS
Banana	220	**164**	290	425	**253**	420	**156**	448	386
Gaussian	9746	**5038**	**5420**	13710	**7180**	8430	3230	1316	**941**
Logistic	40	29	**20**	56	43	**30**	25	**10**	25
Sparse	2616	1068	**510**	4200	1334	**1040**	888	716	**634**
Radon	2229	**1422**	**1470**	3164	**1721**	2040	874	**183**	517
Brownian	2059	500	**310**	2447	682	**380**	580	582	**309**
IRT	1062	**437**	690	1801	**659**	840	**398**	**373**	588
Volatility	4073	**2980**	**2760**	-	-	-	**146**	180	315

Table 2: Number of gradient evaluations needed to achieve $\text{bias}_t^2 \leq 0.01$ ("Grads to low bias"), number of gradient evaluations needed to achieve $\text{bias}_t^2 \leq 0.002$ ("Grads to very low bias"), and number of gradient evaluations divided by effective sample size ("Grads/ESS"). Results within 10% of best across algorithms are in **bold**. "-" entries denote unavailable results on the stochastic volatility target due to the high dimensionality making it difficult to estimate bias.

that is, the maximum bias across dimensions normalized by posterior variance.

Figure 3 shows the results for the sparse logistic regression target; plots for other targets are in Appendix B. The bias decays roughly exponentially until it falls below the level that we can detect with 4096 samples. Table 2 summarizes the number of gradient evaluations it takes each algorithm to reach low (one hundredth of posterior variance) and very low (two thousandths of posterior variance) levels of bias when estimating the second moment (i.e., $\mathbb{E}[\theta_{\infty,d}^2]$, which is sensitive to errors in both mean and variance). These are the levels of bias that will have little impact on estimates based on a total effective sample size of 100 or 500 (respectively). Because it is hard to estimate small biases, the time-to-very-low-bias statistic may be noisy. MEADS generally performs well on these metrics.

We also estimated each algorithm's asymptotic efficiency at generating large effective sample sizes (ESS) per chain using TensorFlow Probability's `tfp.mcmc.effective_sample_size` implementation. ESS was estimated using cross-chain statistics from the last 100 samples. We report the median across runs of the minimum ESS across dimensions and statistics $f(\theta_d) = \theta_d$ and $f(\theta_d) = \theta_d^2$.

Asymptotic ESS efficiency is arguably less important for many-chain Bayesian inference workflows than rapid convergence, since the Monte Carlo error of our estimates scales inversely with both per-chain ESS and number of chains, and there is little point in reducing Monte Carlo error far past the point where it is dominated by posterior uncertainty. Nonetheless, it is interesting to compare the cost of generating larger ESS by running chains for longer to the cost of running more chains in parallel. These costs are generally of the same order—"Grads/ESS" is generally comparable to "Grads to very low bias", although in some cases (e.g., ChEES on Radon), it is substantially lower, likely reflecting poor early preconditioning.

6 DISCUSSION

We have developed MEADS, an ECA tuning scheme that makes turnkey GHMC competitive with existing turnkey HMC algorithms. GHMC is a very flexible algorithm, and we hope that MEADS encourages further research into its applications and extensions. for example, interleaving GHMC with Gibbs steps on discrete latent variables (Neal, 2020) or exploring ways to extend or improve on MEADS, for example using quasi-Newton preconditioning schemes (Zhang and Sutton, 2011; Leimkuhler et al., 2018) or stochastic gradients.

7 ACKNOWLEDGEMENTS

We thank the entire TensorFlow Probability team for many helpful discussions, especially Colin Carroll, Brian Patton, Rif A. Saurous, and Sharad Vikram. We also thank the reviewers for a number of helpful suggestions.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

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).

Cheng, X., Chatterji, N. S., Bartlett, P. L., and Jordan, M. I. (2018). Underdamped Langevin MCMC: A non-asymptotic analysis. In Bubeck, S., Perchet, V., and Rigollet, P., editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 300–323. PMLR.

Dalalyan, A. S. and Riou-Durand, L. (2020). On sampling from a log-concave density using kinetic langevin diffusions. *Bernoulli*, 26(3):1956–1988.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., and Saurous, R. A. (2017). Tensorflow distributions.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222.

Gilks, W. R., Roberts, G. O., and George, E. I. (1994). Adaptive direction sampling. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 43(1):179–189.

Hairer, E., Hochbruck, M., Iserles, A., and Lubich, C. (2006). Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882.

Hoffman, M., Radul, A., and Sountsov, P. (2021). An adaptive-MCMC scheme for setting trajectory lengths in Hamiltonian Monte Carlo. 130:3907–3915.

Hoffman, M. D. and Gelman, A. (2011). The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *The Journal of Machine Learning Research.*

Horowitz, A. M. (1991). A generalized guided Monte Carlo algorithm. *Physics Letters B*, 268(2):247–252.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations.*

Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, (2011):1–8.

Langmore, I., Dikovsky, M., Geraedts, S., Norgaard, P., and Von Behren, R. (2019). A condition number for hamiltonian monte carlo.

Langmore, I., Dikovsky, M., Geraedts, S., Norgaard, P., and von Behren, R. (2021). Hamiltonian monte carlo in inverse problems; ill-conditioning and multimodality.

Lao, J. and Dillon, J. V. (2019). Unrolled implementation of no-U-turn sampler. `https://github.com/tensorflow/probability/blob/main/discussion/technical_note_on_unrolled_nuts.md`.

Leimkuhler, B., Matthews, C., and Weare, J. (2018). Ensemble preconditioning for Markov chain Monte Carlo simulation. *Statistics and Computing*, 28(2):277–290.

Leimkuhler, B. and Reich, S. (2004). *Simulating hamiltonian dynamics*. Number 14. Cambridge university press.

Ma, Y.-A., Chatterji, N. S., Cheng, X., Flammarion, N., Bartlett, P. L., and Jordan, M. I. (2021). Is there an analog of nesterov acceleration for gradient-based MCMC? *Bernoulli*, 27(3):1942–1992.

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(6):1087–1092.

Neal, R. M. (2001). Annealed importance sampling. *Stat. Comput.*, 11(2):125–139.

Neal, R. M. (2003). Slice sampling. *Annals of Statistics*, pages 705–741.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*. CRC Press New York, NY.

Neal, R. M. (2020). Non-reversibly updating a uniform [0, 1] value for Metropolis accept/reject decisions. *arXiv preprint arXiv:2001.11950.*

Niu, F., Recht, B., Re, C., and Wright, S. (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24.

Pasarica, C. and Gelman, A. (2010). Adaptively scaling the Metropolis algorithm using expected squared jumped distance. *Statistica Sinica*, pages 343–364.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

Sohl-Dickstein, J. and Culpepper, B. J. (2012). Hamiltonian annealed importance sampling for partition function estimation. *arXiv preprint arXiv:1205.1925*.

Sountsov, P., Radul, A., and contributors (2020). Inference gym. `https://pypi.org/project/inference_gym`.

Wenzel, F., Roth, K., Veeling, B. S., Swiatkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*.

Zhang, Y. and Sutton, C. (2011). Quasi-Newton methods for Markov chain Monte Carlo. *Advances in Neural Information Processing Systems*, 24:2393–2401.

Supplemental Material for "Tuning-Free Generalized Hamiltonian Monte Carlo"

A ABLATIONS

MEADS has three user-selectable parameters:

- A step-size multiplier, set to 0.5 in our main experiments.
- A damping-slowdown parameter, set to 1.0 in our main experiments.
- A number-of-folds parameter K, set to 4 in our main experiments.

In this section, we examine the effect of different values of these choices, and find that the default values perform well across all target distributions we consider.

A.1 Step-size multiplier

Figure 4 shows the number of steps needed to achieve low-bias estimators when using MEADS with step-size multipliers between 0.2 and 0.7 and the default four folds and damping slowdown factor of 1. The recommended default step-size multiplier of 0.5 is reasonable across problems, although it is occasionally a little conservative (for example, on the IRT and stochastic volatility targets).

A.2 Damping slowdown

Figure 5 shows the number of steps needed to achieve low-bias estimators when using MEADS with damping slowdown factors 0.5, 1, 2, and 5 and the default four folds and step-size multiplier of 0.5. The recommended default damping slowdown factor of 1 is reasonable across problems, although the stochastic volatility model would seem to prefer a more conservative ramping-up of momentum.

A.3 Number of folds

Figure 6 shows the number of steps needed to achieve low-bias estimators when using MEADS with 2, 4, and 8 folds and the default step-size multiplier of 0.5 and damping slowdown factor of 1. Four folds is consistently better than two, since when only using two folds half of the chains must sit idle (we conservatively assign the same cost to idle chains as updating chains on the assumption that enough parallel resources are available that we could have computed gradients for

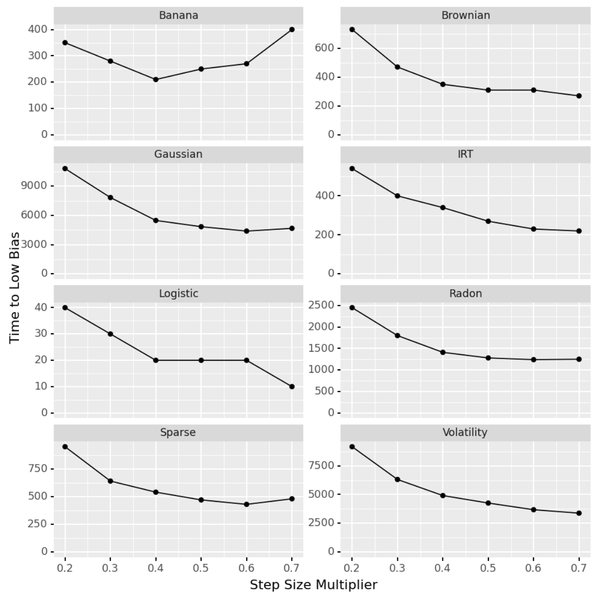

Figure 4: Number of gradient evaluations needed to achieve squared bias less than 0.01 as a function of step-size multiplier for MEADS. Lower is better.

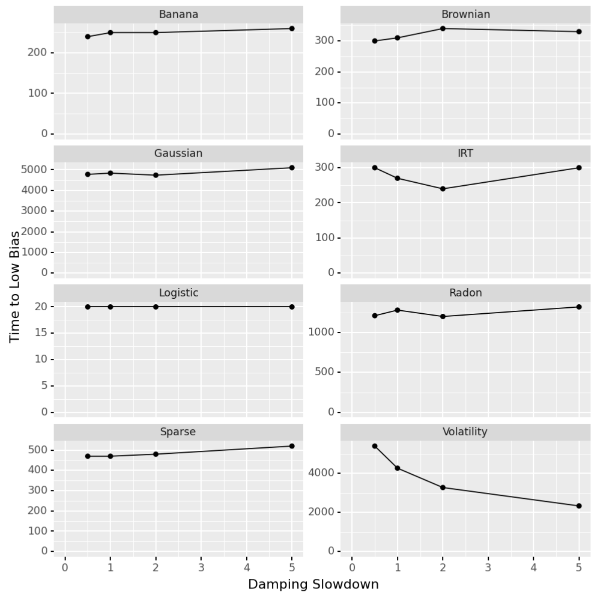

Figure 5: Number of gradient evaluations needed to achieve squared bias less than 0.01 as a function of damping slowdown factor for MEADS. Lower is better.

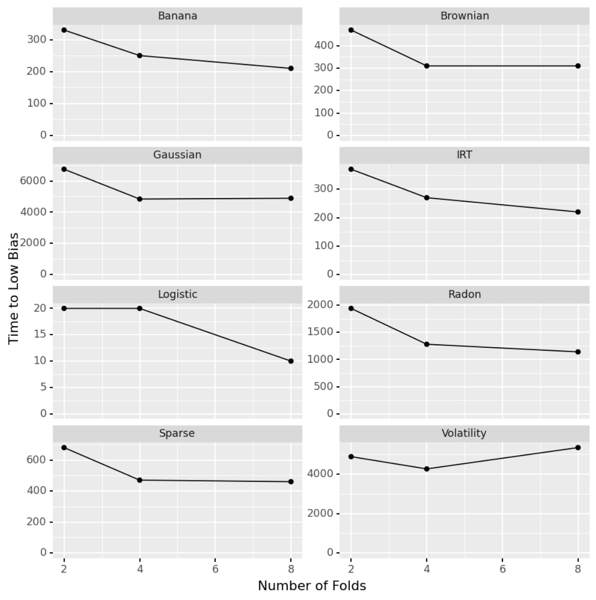

Figure 6: Number of gradient evaluations needed to achieve squared bias less than 0.01 as a function of number of folds for MEADS. Lower is better.

the idle chains at no cost in wallclock time). Increasing the number of folds to eight generally has little effect. (The effect looks large for the logistic-regression target, but this is a quantization artifact due to thinning the MEADS chain by a factor of 10.)

B BIAS PLOTS

Figure 7 shows how the transient biases of ChEES, NUTS, and MEADS evolve as a function of number of gradient steps. As the true bias gets very small, the squared error of the estimator becomes dominated by variance and we are unable to get a good estimate of the true bias.

C EIGENVALUE ESTIMATOR

In this section we consider the maximum-eigenvalue estimator described in Section 4.4 and Algorithm 2.

Given a N-by-D matrix X such that

$$\mathbb{E}[\frac{1}{N}X^\top X] = \Sigma, \tag{9}$$

we are interested in estimating the largest eigenvalue of Σ.

Throughout this section, we will consider the simple case where $X_{n,d} \sim \mathcal{N}(0, \sigma_d)$, so that $\Sigma = \mathrm{diag}(\sigma^2)$ and the largest eigenvalue is $\max_d \sigma_d^2$. We will examine four specific values for σ, each of which has maximum eigenvalue 1 and $D = 100$ dimensions:

1. **Identity:** $\sigma_d = 1$ for all d.
2. **Log-spaced:** Logarithmically spaced σ_d with $\sigma_1 = 0.1$ and $\sigma_D = 1$.
3. **Linear-spaced:** Linearly spaced σ_d with $\sigma_1 = 0.1$ and $\sigma_D = 1$.
4. **Bimodal:** $\sigma_d = 0$ for $d <= D/2$, $\sigma_d = 1$ for $d > D/2$.

Figure 8 shows the result of directly computing the largest eigenvalue of $X^\top X$ ("Naive Estimator") and instead computing the ratio $\frac{\mathrm{tr}(X^\top X X^\top X)}{\mathrm{tr}(X^\top X)}$ ("Ratio Estimator") as the number of observations N goes up. For small sample sizes, the estimates are biased upwards; this bias decays faster for the ratio estimator than for the naive estimator. This faster decay comes at the price of a small asymptotic bias for the log-spaced and linear-spaced spectra; as $N \to \infty$ the ratio estimator converges to $(\sum_d \sigma_d^4)/(\sum_d \sigma_d^2)$, which is a bit lower than $\sigma^2_{\max}$ when there exist eigenvalues less than $\sigma^2_{\max}$ but not so small that they make only a small contribution to the trace $\sum_d \sigma_d^2$.

D AN EXAMPLE OF THE DANGERS OF HOGWILD ADAPTATION

In this section we provide a simple example of an ensemble-chain adaptation (ECA) MCMC kernel which gives correct results when applied to one state at a time, but catastrophically wrong results when applied "hogwild" to all states in parallel.

There are two independent, uniformly distributed states $x_0 \in \{0,1\}$ and $x_1 \in \{0,1\}$, so that $p(x) = \frac{1}{4}$. Our MCMC kernel to update one of the states given the other is the XOR function: $x'_i = \mathrm{XOR}(x_i, x_{1-i})$. If we update x_i holding x_{1-i} fixed, this update satisfies detailed balance, since the two possible updates ($x'_i = \mathrm{XOR}(x_i, 0) = x_i$ and $x'_i = \mathrm{XOR}(x_i, 1) = 1 - x_i$) are both reversible with respect to the uniform distribution on $\{0,1\}$. So if we alternate between updating x_1 holding x_0 fixed and x_0 holding x_1 fixed, then the chain will correctly leave the uniform distribution invariant.

However, if we go "hogwild" and apply the update simultaneously to both states, then $x'_0 = \mathrm{XOR}(x_0, x_1)$ and $x'_1 = \mathrm{XOR}(x_1, x_0) = \mathrm{XOR}(x_0, x_1) = x'_0$. So after one iteration, the two states will be identical. Once the states are identical, another iteration will ensure that they are both zero, since $\mathrm{XOR}(0,0) = \mathrm{XOR}(1,1) = 0$. So after two iterations the chain will be stuck in the absorbing state $x_0 = x_1 = 0$, rather than leaving the uniform distribution invariant.

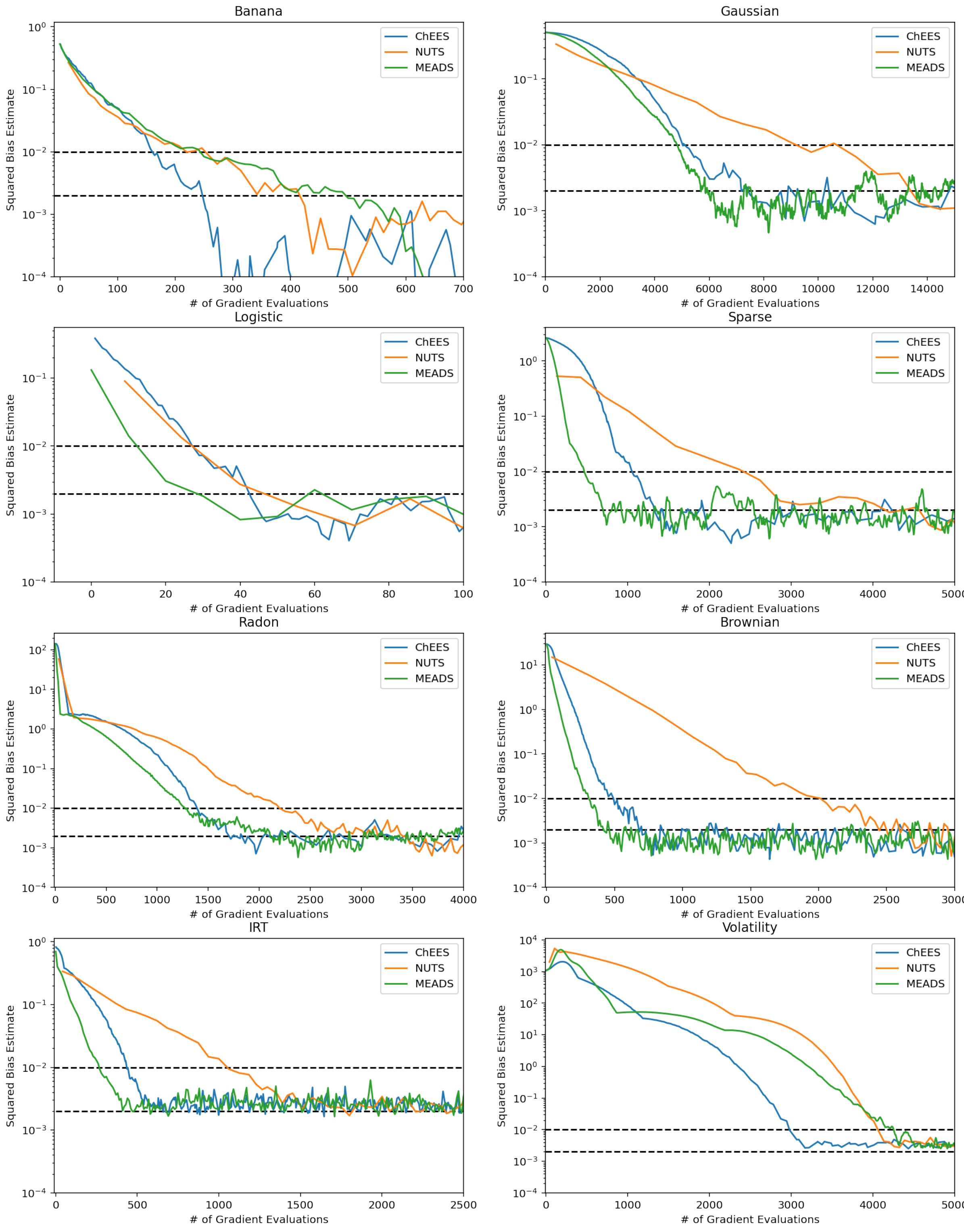

Figure 7: Squared bias estimate versus number of gradients for ChEES, NUTS, and MEADS. Dashed horizontal lines are thresholds of 0.01 and 0.002.

The one-at-a-time version of this chain is neither ergodic nor aperiodic, but this can be fixed by occasionally (say, with probability 0.01) instead applying a kernel that resamples x_0 and x_1 from their uniform target distribution. Since the uniform resampling is applied relatively rarely, the stationary distribution of

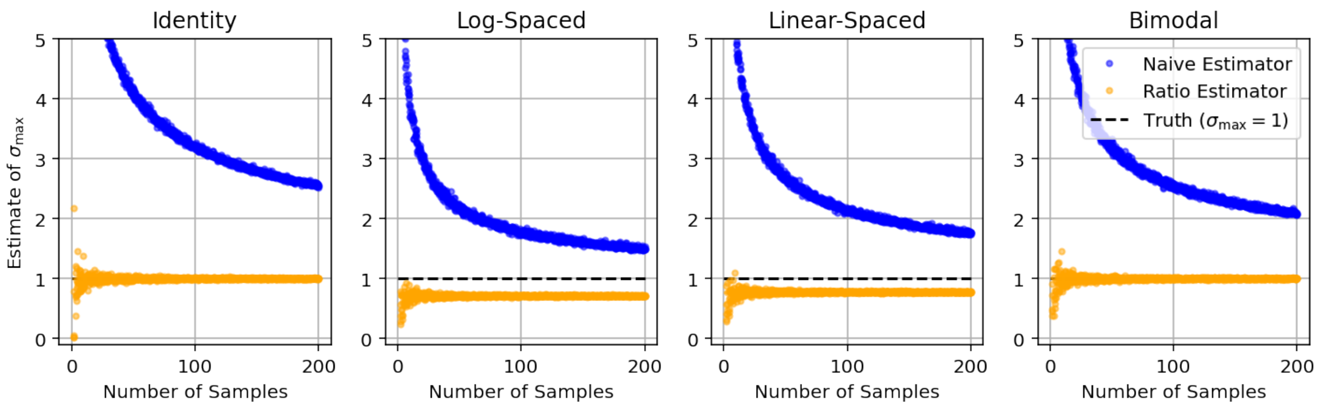

Figure 8: Naive estimates of the square root of the largest eigenvalue of $\mathbb{E}[\frac{1}{N}X^\top X]$ as a function of the number of samples N. For each value of N we computed estimates based on five draws of X. "Naive Estimator" denotes directly computing the largest eigenvalue of $\frac{1}{N}X^\top X$; "Ratio Estimator" denotes computing the ratio $\frac{\mathrm{tr}(X^\top XX^\top X)}{\mathrm{tr}(X^\top X)} = \frac{\mathrm{tr}(XX^\top XX^\top)}{\mathrm{tr}(XX^\top)}$. The orange line denotes $(\sum_d \sigma_d^4)/(\sum_d \sigma_d^2)$, the asymptotic value of the ratio estimator as $N \to \infty$.

the hogwild version of this kernel will still be highly biased towards the state $x_0 = x_1 = 0$.

E PROOF OF EQUATION 4

Our goal is to show that

$$-\int_\theta p(\theta)\nabla^2 \log p(\theta) d\theta = \int_\theta p(\theta) \left(\frac{\partial \log p}{\partial \theta}\right)^\top \frac{\partial \log p}{\partial \theta} d\theta. \tag{10}$$

We note that

$$\begin{aligned}\nabla^2 p(\theta) &= \frac{\partial}{\partial \theta}(p(\theta)\nabla \log p(\theta)) \\ &= p(\theta)\left(\left(\frac{\partial \log p}{\partial \theta}\right)^\top \frac{\partial \log p}{\partial \theta} + \nabla^2 \log p(\theta)\right).\end{aligned} \tag{11}$$

So if, for all indices i and j, $\int_\theta \nabla^2_{\theta_i,\theta_j} p(\theta) d\theta = 0$, then $\mathbb{E}_p[(\frac{\partial \log p}{\partial \theta})^\top \frac{\partial \log p}{\partial \theta}] = \mathbb{E}_p[-\nabla^2 \log p(\theta)]$. We assume that p is twice differentiable almost everywhere and continuous, and that for all $i \in \{1, \ldots, D\}$, $p(\theta_i \mid \theta_{\setminus i}) \to 0$ as $|\theta_i| \to \infty$. By the fundamental theorem of calculus, $\int_{-\infty}^{\infty} \nabla_{\theta_i} p(\theta_i \mid \theta_{\setminus i}) d\theta_i = 0$ and, since the derivatives of $p(\theta_i \mid \theta_{\setminus i})$ also go to 0 as $|\theta_i| \to \infty$, $\int_{-\infty}^{\infty} \nabla_{\theta_i}\nabla_{\theta_i} p(\theta_i \mid \theta_{\setminus i}) d\theta_i = 0$. First, we show that $\int_\theta \nabla^2_{\theta_i,\theta_j} p(\theta) d\theta = 0$ when $i \neq j$:

$$\begin{aligned}\int_\theta \nabla^2_{\theta_i,\theta_j} p(\theta) d\theta &= \int_\theta \nabla_{\theta_i}\nabla_{\theta_j} p(\theta) d\theta \\ &= \int_{\theta_{\setminus i,j}} p(\theta_{\setminus i,j}) \int_{\theta_i} \nabla_{\theta_i} p(\theta_i \mid \theta_{\setminus i,j}) \\ &\quad \times \int_{-\infty}^{\infty} \nabla_{\theta_j} p(\theta_j \mid \theta_{\setminus j}) d\theta_j d\theta_{\setminus j} = 0,\end{aligned} \tag{12}$$

since $\int_{-\infty}^{\infty} \nabla_{\theta_i} p(\theta_i \mid \theta_{\setminus i}) d\theta_i = 0$. Likewise,

$$\begin{aligned}&\int_\theta \nabla_{\theta_i}\nabla_{\theta_i} p(\theta) d\theta \\ &= \int_{\theta_{\setminus i}} p(\theta_{\setminus i}) \int_{-\infty}^{\infty} \nabla_{\theta_i}\nabla_{\theta_i} p(\theta_i \mid \theta_{\setminus i}) d\theta_i d\theta_{\setminus i} = 0. \quad \square\end{aligned} \tag{13}$$