If Only My Posterior Were Normal: Introducing Fisher HMC

Adrian Seyboldt
adrian.seyboldt@gmail.com
PyMC Labs

Eliot Carlson
eliot.carlson@pymc-labs.com
PyMC Labs

Bob Carpenter
bcarpenter@flatironinstitute.org
Flatiron Institute

2025-05-04

Abstract

Hamiltonian Monte Carlo (HMC) is a powerful tool for Bayesian inference, as it can explore complex and high-dimensional parameter spaces. But HMC's performance is highly sensitive to the geometry of the posterior distribution, which is often poorly approximated by traditional mass matrix adaptations, especially in cases of non-normal or correlated posteriors. We propose Fisher HMC, an adaptive framework that uses the Fisher divergence to guide transformations of the parameter space. It generalizes mass matrix adaptation from affine functions to arbitrary diffeomorphisms. By aligning the score function of the transformed posterior with those of a standard normal distribution, our method identifies transformations that adapt to the posterior's scale and shape. We develop theoretical foundations efficient implementation strategies, and demonstrate significant sampling improvements. Our implementation, nutpie, integrates with PyMC and Stan and delivers better efficiency compared to existing samplers.

Keywords Bayesian Inference · Hamiltonian Monte Carlo · Mass Matrix Adaptation · Normalizing Flows · Fisher Divergence

1 Introduction

Hamiltonian Monte Carlo (HMC) is a powerful Markov Chain Monte Carlo (MCMC) method widely used in Bayesian inference for sampling from complex posterior distributions. HMC can explore high-dimensional parameter spaces more efficiently than traditional MCMC techniques, which makes it popular in probabilistic programming libraries like Stan (Carpenter et al. 2017) and PyMC (Salvatier, Wiecki, and Fonnesbeck 2015). However, the performance of HMC depends critically on the parameterization of the posterior space. Modern samplers automate a part of these reparametrizations by adapting a "mass matrix" in the warmup phase of sampling. A common approach

in HMC is to estimate a mass matrix based on the inverse of the posterior covariance, typically in a diagonalized form, to adjust for differences in scale across dimensions. We can think of this as a reparametrization that simply rescales the parameters such that they have a posterior variance of one. It is not obvious, however, that this is the best rescaling that can be done. Morever, even a well-tuned mass matrix can not do much to help us when sampling from more challenging posterior distributions such as those with strong correlations in high dimensions, or with funnel-like pathologies. For researchers working with multilevel hierarchical models with correlated group-level parameters, manually rescaling and rotating the parameter space to improve sampling efficiency requires deep statistical expertise and can be time-consuming. In many cases, good reparametrizations are also data-dependent, which makes it difficult to write a model once, and apply it to a wide range of individual datasets.

To address these limitations, we propose an adaptive HMC framework that extends beyond the traditional concept of a mass matrix: instead of just rescaling variables, we allow for arbitrary diffeomorphisms that dynamically transform the parameter space. We use the Fisher divergence as a criterion to choose between different transformations, which allows us to adapt the geometry of the posterior space in a way that optimizes HMC's efficiency. By aligning the scores (derivatives of the log-density) of the transformed posterior with those of a standard normal distribution, we approximate an idealized parameterization that facilitates efficient sampling. In the first section, we motivate why the scores are useful for for Hamiltonian dynamics. We then present the Fisher divergence as a metric by which we can assess the transformations of target distributions, deriving closed-form solutions for optimal diffeomorphisms in the affine case, i.e. mass matrices. Finally, we suggest additional modifications to the adaptation schedule shared by the major software implementations, which complement gradient-based adaptation.

2 Fisher HMC: Motivation and Theory

2.1 Motivation: Example with Normal Posterior

HMC is a gradient-based method, meaning that the algorithm computes the derivatives of the log posterior density (the scores). While these gradients contain significant information about the target density, traditional methods of mass matrix adaptation ignore them. To illustrate how useful the scores can be, consider a standard normal posterior $N(\mu, \sigma^2)$ with density $p(x) \propto \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$. Let's assume we have two posterior draws x_1 and x_2, together with the covector of scores

$$\alpha_i = \frac{\partial}{\partial x_i} \log p(x_i) = \sigma^{-2}(\mu - x_i). \tag{1}$$

Based on this information alone, we can directly compute μ and σ to identify the exact posterior. Solving for μ and σ, we get

$$\mu = \bar{x} + \sigma^2 \bar{\alpha} \quad \text{and} \quad \sigma^2 = \mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}, \tag{2}$$

where $\bar{x}$ and $\bar{\alpha}$ are the sample means of x_i and α_i, respectively. If we take advantage of the scores, we can compute the exact posterior and thus an optimal mass matrix with no sample variance, based on just two draws!

This generalizes directly to multivariate normal posteriors $N(\mu, \Sigma)$, where we can leverage the elegant fact that the scores are normally distributed with covariance Σ^{-1}. Assume we have $N+1$ linearly independent draws $x_i \in \mathbb{R}^N$ with scores $\alpha_i = \Sigma^{-1}(x_i - \mu)$. The mean of these equations gives us $\mu = \bar{x} - \Sigma\bar{\alpha}$. It follows that $\Sigma^{-1}X = S$, where the i-th column of S is $\alpha_i - \bar{\alpha}$, and the i-th column of X is $x_i - \bar{x}$. Finally, we have

$$SS^T = \mathrm{Cov}[\alpha_i] = \Sigma^{-1}XX^T\Sigma^{-1} = \Sigma^{-1}\,\mathrm{Cov}[x_i]\Sigma^{-1} \tag{3}$$

and we can recover Σ as the geometric mean of the positive symmetric matrices $\mathrm{Cov}[x_i]$ and $\mathrm{Cov}[s_i]^{-1}$:

$$\Sigma = \mathrm{Cov}[x_i]^{-\frac{1}{2}}\Big(\mathrm{Cov}[x_i]^{\frac{1}{2}}\,\mathrm{Cov}[\alpha_i]\mathrm{Cov}[x_i]^{\frac{1}{2}}\Big)^{\frac{1}{2}}\mathrm{Cov}[x_i]^{-\frac{1}{2}} \tag{4}$$

In this way we can compute the parameters of the normal distribution exactly. Of course, most posterior distributions of interest are not multivariate normal, and if they were, we would not have to run MCMC in the first place. But it is common in Bayesian inference for the posterior to approximate a normal distribution reasonably well, which suggests that the scores contain useful information that is ignored in standard methods.

2.2 Transformed HMC

When we manually reparameterize a model to make HMC more efficient, we try to find a transformation of our posterior such that HMC performs better on it. Formally, if our posterior μ is defined on a space M, we try to find a diffeomorphism $f : N \to M$ such that the transformed posterior $f^*\mu$ is well-behaved with respect to some property. Note that we define the transformation as a function *from* the transformed space *to* the original posterior, in keeping consistent with the Normalizing Flow literature.[1] $f^*\mu$ refers to the pullback of the posterior (which we can interpret as a volume form), i.e. we *pull it back* to the space N along the transformation f. If f is an affine transformation, this simplifies to mass matrix-based HMC, wherein choosing $f(x) = \Sigma^{\frac{1}{2}}x + \mu$ corresponds to the mass matrix Σ^{-1}, as described in more detail in Neal (2012). In standard implementations, Σ^{-1} appears in full in the leapfrog integrator:

[1] Since the transformation is a bijection, we can choose any direction we want, as long as we stay consistent with our choice.

LEAPFROG$(\theta,\ \rho,\ L,\ \varepsilon,\ \Sigma)$:

$\theta^0 \leftarrow \theta, \rho^0 \leftarrow \rho$

for i from 0 **to** L:

$\quad \rho^{(i+\frac{1}{2})} \leftarrow \rho^{(i)} - \frac{\varepsilon}{2}\nabla U\big(\theta^{(i)}\big)$ // half-step momentum

$\quad \theta^{(i+1)} \leftarrow \theta^{(i)} + \varepsilon\Sigma^{-1}\rho^{(i+\frac{1}{2})}$ // full-step position

$\quad \rho^{(i+1)} \leftarrow \rho^{(i+\frac{1}{2})} - \frac{\varepsilon}{2}\nabla U\big(\theta^{(i+1)}\big)$ // half-step momentum

return $\big(\theta^{(L)}, \rho^{(L)}\big)$

In a transformed setting, however, draws in the posterior space M are pulled back along f to the more forgiving space N. Leapfrog requires computing the log densities and scores in the transformed space. In practice, we work with densities p and q relative to Lebesgue measures λ_N, λ_M: $\mu = p\lambda_M, f^*\mu = q\lambda_N$. So our transformed score is $\nabla \log\big(f^*\frac{\mu}{\lambda_N}\big)$. Note, however, that the computation of this transformed score requires a push-forward. Using the change-of-variables $f^*\lambda_M = |\det(df)|\lambda_N$, we have

$$\nabla \log\left(f^*\frac{\mu}{\lambda_N}\right) = \nabla \log\left(f^*\frac{p}{\lambda_M}\frac{f^*\lambda_M}{\lambda_N}\right) = f^*\nabla \log(p) + \nabla \log|\det(df)| = \hat{f}^*(\nabla \log(p), 1) \quad (5)$$

where $\hat{f} : N \to M \times \mathbb{R}, x \mapsto (f(x), \log|\det(f)|)$. On N, the Hamiltonian is simulated using an identity mass matrix, meaning we no longer distinguish between momentum and velocity.

NF-LEAPFROG$(\theta,\ v,\ L,\ \varepsilon,\ f)$:

$\theta^0 \leftarrow \theta, v^0 \leftarrow v$

$y \leftarrow f^{-1}(\theta^0)$

$\delta \leftarrow \nabla \log\big(f^*\frac{\mu}{\lambda_N}\big)(y)$ // evaluate score on N

for i from 1 **to** L:

$\quad v^{(i+\frac{1}{2})} \leftarrow v^{(i)} - \frac{\varepsilon}{2}\delta$ // half-step velocity

$\quad y \leftarrow y + \varepsilon v^{(i+\frac{1}{2})}$ // full-step position ($\Sigma = I$)

$\quad \theta^{(i)} \leftarrow f(y)$ // recover corresponding draw on M

$\quad \delta \leftarrow \nabla \log\big(f^*\frac{\mu}{\lambda_N}\big)(y)$

$\quad v^{(i+1)} \leftarrow v^{(i+\frac{1}{2})} - \frac{\varepsilon}{2}\delta$ // half-step velocity

return θ

NF-LEAPFROG$(\theta,\ v,\ L,\ \varepsilon,\ f)$:
$\theta^0 \leftarrow \theta, v^0 \leftarrow v$
$y, \delta \leftarrow \text{PULLBACK}(f, \theta^0)$
for i from 1 **to** L:
    $v^{(i+\frac{1}{2})} \leftarrow v^{(i)} - \frac{\varepsilon}{2}\delta$ // half-step velocity
    $y \leftarrow y + \varepsilon v^{(i+\frac{1}{2})}$ // full-step position ($\Sigma = I$)
    $\theta^{(i)} \leftarrow f(y)$ // recover corresponding draw on M
    $y, \delta \leftarrow \text{PULLBACK}\big(f, \theta^{(i)}\big)$
    $v^{(i+1)} \leftarrow v^{(i+\frac{1}{2})} - \frac{\varepsilon}{2}\delta$ // half-step velocity
return $\theta^{(L)}$

PULLBACK$(f,\ \theta)$:
$y \leftarrow f^{-1}(\theta)$ // pull back θ to N
$\delta \leftarrow \nabla \log\Big(f^* \frac{\mu}{\lambda_N}\Big)(y)$ // evaluate score on N
return (y, δ)

2.3 Fisher Divergence

HMC efficiency is notoriously dependent on the parametrization, so it is to be expected that transformed HMC be much more efficient for some choices of f than for others. It is not, however, obvious what criterion should be used to evaluate a particular choice of f, in order to guide the learning of a transformation. We need a loss function that maps the diffeomorphism to a measure of difficulty for HMC. This is hard to quantify in general, but we can observe that HMC efficiency largely depends on the trajectory, which in fact does not depend on the density directly, but rather only on the scores. Therefore, a reasonable loss function might assess how well the transformed space's *scores* align with those of our desired transformed posterior. We choose the standard normal distribution as the ideal transformed posterior, since we know that HMC is efficient in this case, given the nice Gaussian properties such as constant curvature. This still leaves open the choice of a specific norm for comparing the scores of the standard normal with those of the transformed posterior. But since the standard normal distribution is defined in terms of an inner product, we already have a well-defined norm on the scores that allows us to evaluate their difference. This directly motivates the following definition of the Fisher divergence.

Let (N, g) be a Riemannian manifold with probability volume forms ω_1 and ω_2. We define the Fisher divergence of ω_1 and ω_2 as

$$\mathcal{F}_g(\omega_1, \omega_2) = \int \left\| \nabla \log\left(\frac{\omega_2}{\omega_1}\right) \right\|_g^2 d\omega_1. \tag{6}$$

Note that $\mathcal{F}$ requires more structure on N than KL-divergence $\int \log\left(\frac{\omega_2}{\omega_1}\right) d\omega_1$, as the norm depends on the metric tensor g. Given a second (non-Riemannian) manifold M with a probability volume form μ and a diffeomorphism $f : N \to M$, we can define the divergence between μ and ω_2 by pulling back μ to N, i.e. $\mathcal{F}_g(f^*\mu, \omega_2)$. We can also compute this Fisher divergence directly on M, by pushing forward the metric tensor:

$$\mathcal{F}_g(f^*\mu, \omega_2) = \mathcal{F}_{(f^{-1})^* g}\left(\mu, (f^{-1})^* \omega_2\right) \tag{7}$$

In this case, μ is our posterior, M is the space on which it is originally defined, and ω_2 is the standard normal distribution.

2.4 Affine choices for the diffeomorphism

We focus on three families of affine diffeomorphisms F, for which derive specific results.

2.4.1 Diagonal mass matrix

If we choose $f_{\sigma,\mu} : Y \to X$ as $x \mapsto y \odot \sigma + \mu$, we are doing diagonal mass matrix estimation. In this case, the sample Fisher divergence reduces to

$$\hat{\mathcal{F}}_{\sigma,\mu}(f^*X, N(0, I_d)) = \frac{1}{N} \sum_i \left\| \sigma \odot \alpha_i + \sigma^{-1} \odot (x_i - \mu) \right\|^2 \tag{8}$$

This is a special case of affine transformation and minimal at $\sigma^* = \mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$ and $\mu^* = \bar{x}_i + \sigma^2 \bar{s}_i$, corresponding to the result in Section 2.1. This solution is very computationally inexpensive, and is hence the default in nutpie. Using Welford's algorithm to keep online estimates of the draw and score variances during sampling (thereby avoiding the need to explicity store scores), the mass matrix is set to a diagonal matrix with the i'th entry on the diagonal equal to $\mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$.

If our target density is $N(\mu, \Sigma)$, then the minimizers μ^* and σ^* of $\hat{\mathcal{F}}$ derived above converge to μ and $\exp\left(\frac{1}{2}\log \mathrm{diag}(\Sigma) - \frac{1}{2}\log \mathrm{diag}(\Sigma^{-1})\right)$, respectively. This is a direct consequence of the fact that $\mathrm{Cov}[x_i] \to \Sigma$ and $\mathrm{Cov}[\alpha_i] \to \Sigma^{-1}$. The divergence $\hat{\mathcal{F}}$ converges to $\sum_i \lambda_i + \lambda_i^{-1}$, where λ_i are the generalized eigenvalues of Σ with respect to $\mathrm{diag}(\hat{\sigma}^2)$. So large and small eigenvalues are penalized. Choosing $\mathrm{diag}(\Sigma)$ as our mass matrix effectively minimizes $\sum_i \lambda_i$, only penalizing large eigenvalues. If we choose $\mathrm{diag}(\mathbb{E}(\alpha\alpha^T))$, as proposed in Tran and Kleppe (2024), we effectively minimize $\sum \lambda_i^{-1}$ and only penalize small eigenvalues. But based on theoretical results for multivariate normal posteriors in Langmore et al. (2020), we know that both large and small eigenvalues make HMC less efficient. We can use this result to evaluate the different diagonal mass matrix choices on various gaussian posteriors, with different numbers of observations. Figure todo shows the resulting condition numbers of the posterior as seen by the sampler in the transformed space.

2.4.2 Full mass matrix

The full affine diffeomorphism $f_{A,\mu}(y) = Ay + \mu$ corresponds to a mass matrix $M = (AA^T)^{-1}$. The Fisher divergence in this case is

$$\hat{\mathcal{F}}\left[f_{A,\mu}\right] = \frac{1}{N} \sum \left\| A^T s_i + A^{-1}(x_i - \mu) \right\|^2 \tag{9}$$

which is minimized when $AA^T \operatorname{Cov}[x_i] AA^T = \operatorname{Cov}[\alpha_i]$ (proof in Appendix A), corresponding again to the derivation in Section 2.1. Because $\hat{\mathcal{F}}$ only depends on AA^T and μ, we restrict A to be symmetric positive definite such that there is a unique solution for A. If the two covariance matrices are full rank, we get a unique minimum at the geometric mean of $\operatorname{Cov}[x_i]$ and $\operatorname{Cov}[s_i]$.

2.4.3 Diagonal plus low-rank

Either to save computation in high dimensional settings, or if the number of dimensions is larger than the number of available draws at the time of adaptation, we can also approximate the full mass matrix with a "diagonal plus low-rank" matrix. This low rank matrix is parametrized by a cutoff c, determining a critical distance from one at which point we ignore eigenvectors, returning a truncated set of eigenvectors U_c and corresponding eigenvalues Λ_c of Σ. In fact, we implement this as the composition of two affine transformations, the first one being the element-wise (diagonal) affine transformation defined earlier, and the second a low-rank approximation to the geometric mean of the draw and gradient empirical covariance matrices. The corresponding normalizing flow is $f = f_{A,\mu} \circ f_{\sigma,\mu}$, and the mass matrix is

$$\Sigma = D^{\frac{1}{2}}(QU_c(\Lambda_c - 1)QU_c^T + I)D^{\frac{1}{2}} \tag{10}$$

where Q is an orthonormal basis for the shared subspace, which we'd like to optimize over (D, U, Λ). To do this, we do a greedy optimization where we first apply the optimal element-wise rescaling factor $\operatorname{Var}[x_i]^{\frac{1}{2}} \operatorname{Var}[\alpha_i]^{-\frac{1}{2}}$, "pulling back" μ to an intermediate space, from which we then optimize a low-rank transformation to the final transformed posterior. For this second leg of optimization, we project x_i and α_i into their joint span, compute the geometric mean in this subspace as in Section 2.4.2, and then decompose the resulting Σ. We can avoid $O(n^2)$ storage by keeping only Λ_c and U_c, which form the reduced A, and the diagonal components from the element-wise transformation. The algorithm is as follows:

LOW-RANK-ADAPT$(X,\ S,\ c,\ \gamma)$:

$X \leftarrow \left(X - \bar{X}\right) \odot \hat{\sigma}_X^{-1}\hat{\sigma}_S$ // apply diagonal transform

$S \leftarrow \left(S - \bar{S}\right) \odot \hat{\sigma}_X^{-1}\hat{\sigma}_S$

$U^X \leftarrow \text{SVD}(X), U^S \leftarrow \text{SVD}(S)$

$Q, \_ \leftarrow \text{QR-THIN}(\left[U^X\ U^S\right])$ // Get jointly-spanned orthonormal basis

$P^X \leftarrow Q^T X, P^S \leftarrow Q^T S$ // Project onto shared subspace

$C^X \leftarrow P^X\left(P^X\right)^T + \gamma I$ // Get empirical covariances, regularize

$C^S \leftarrow P^S\left(P^S\right)^T + \gamma I$

$\Sigma \leftarrow \text{SPDM}\left(C^X, C^S\right)$ // Solve $\Sigma C^S \Sigma = C^X$ for Σ

$U\Lambda U^{-1} \leftarrow \text{EIGENDECOMPOSE}(\Sigma)$ // Extract eigenvalues to subset

$U_c \leftarrow \left\{U_i : i \in \left\{i : \lambda_i \geq c \textbf{ or } \leq \frac{1}{c}\right\}\right\}$

$\Lambda_c \leftarrow \left\{\lambda_i : i \in \left\{i : \lambda_i \geq c \textbf{ or } \leq \frac{1}{c}\right\}\right\}$ // Full matrix $\Sigma = QU_c(\Lambda_c - 1)QU_c^T + I$

return QU_c, Λ_c

3 Adaptation Schema

Whether we adapt a mass matrix using the posterior variance as Stan does, or if we use a bijection based on the Fisher divergence, we invariably have the same challenge: in order to generate suitable posterior draws, we need a good mass matrix (or bijection), but to estimate a good mass-matrix, we need posterior draws. There is a well-known way out of this "chicken and egg" conundrum: we start sampling with an initial transformation, and collect a number of draws; based on those draws, we estimate a better transformation, and repeat. This adaptation-window approach has long been used in the major implementations of HMC, and has remained largely unchanged for a number of years. PyMC, Numpyro, and Blackjax all use the same details as Stan, with at most minor modifications. There are, however, a couple of small changes that improve the efficiency of this schema significantly.

Stan's HMC sampler begins warmup using an identity mass matrix. We instead initialize with $M = \text{diag}(\alpha_0^T \alpha_0)$, which in expectation is equal to the Fisher information. This also makes the initialization independent of variable scaling.

3.1 Accelerated Window-Based Adaptation

Most widely used HMC implementations do not run vanilla HMC, but variants, most notably the No-U-Turn Sampler (NUTS) (Hoffman and Gelman 2011), where the length of the Hamiltonian trajectory is chosen dynamically (see Appendix B). Such schemas

can make it extremely costly to generate draws with a poor mass matrix, because in these cases the algorithm can take a huge number of HMC steps for each draw (typically up to 1000). Thus very early on during sampling, we have a big incentive to use available information about the posterior as quickly as possible, to avoid these scenarios. By default, Stan starts adaptation with a step-size adaptation window of 75 draws, where the mass matrix is untouched. This is followed by a mass matrix adaptation window consisting of a series of "memoryless" intervals of increasing length, the first of which (25 draws) still uses the initial mass matrix for sampling. These 100 draws before the first mass matrix change can constitute a sizable percentage of the total sampling time.

Intuition might suggest that we could just use a tailing window and update the matrix at each iteration to an estimate based on the previous k draws via Welford's algorithm for online variance estimation. However, removing the influence of early draws when they are no longer in the window requires rewinding the algorithm, which is unnecessarily inefficient and not easily implemented. Using two overlapping estimation windows - a "foreground" and a "background" - we can accomplish the goal of using recent information immediately, while avoiding the computational cost of pure streaming estimation. At each iteration in the warmup phase, the transformation used is taken from the "foreground" estimator (which itself is updated at each iteration). The "background" estimator, while using the same update logic, maintains a fresher estimate based on only the previous n draws, periodically handing off this fresher estimate to the foreground, then resetting itself. The foreground estimator then builds on this estimate until recieving a new one. In this way, the estimate used for the transformation at any given iteration is informed by at most $2n$ draws, where n is the update frequency. This scheme is illustrated in Figure 1.

We split the matrix adaptation in the warmup phase into two regimes, the first with a very quick update frequency (10 draws) and the second with a much longer one (80 draws).

WARMUP(N, N_{early}, N_{late}, $\nu_{\text{early}} : 10$, $\nu_{\text{late}} : 80$):

$\theta_0, \alpha_0 \sim p(\theta)$ // initial draw from prior

$F =$ MASSMATRIXESTIMATOR() // foreground estimator

UPDATE-ESTIMATES(F, θ_0, α_0)

$B =$ MASSMATRIXESTIMATOR() // background estimator

$\varepsilon =$ STEPSIZEADAPT(θ_0, α_0)

$\mathbb{I}_{\text{init}} \leftarrow 1$ // indicator for initial mass matrix

for i **in** 1 **to** N:

    early $= i < N_{\text{early}}$ // indicator for early regime

    late $= N - i < N_{\text{late}}$ // indicator for late regime

    $f \leftarrow$ GET-TRANSFORM(F)

    $\theta^{(i)}, \alpha^{(i)} =$ HMC-STEP$\left(\theta^{(i-1)}, \varepsilon, f\right)$ // simulate Hamiltonian

    if not early:

        UPDATE-ESTIMATES$\left(F, \theta^{(i)}, \alpha^{(i)}\right)$ // update both windows

        UPDATE-ESTIMATES$\left(B, \theta^{(i)}, \alpha^{(i)}\right)$

    if late:

        UPDATE(ε)

        continue

    UPDATE(ε)

    $\nu \leftarrow \nu_{\text{early}}$ **if** early **else** ν_{late}

    $N_{\text{remain}} \leftarrow N - i - N_{\text{late}}$

    if $N_{\text{remain}} > \nu_{\text{late}}$ **and** N-DRAWS(B) $> \nu$:

        $F \leftarrow G$ // dump background into foreground

        $B \leftarrow$ MASSMATRIXESTIMATOR() // reset background estimator

        if $\mathbb{I}_{\text{init}}$:

            RESET(ε)

            $\mathbb{I}_{\text{init}} \leftarrow 0$

    return

MassMatrixEstimator():

def update-estimates(self, θ, α):

$n \leftarrow$ self.n

$\bar{\theta}, \hat{\sigma}^2_\theta \leftarrow \text{update}\big(n, \hat{\sigma}^2_\theta, \bar{\theta}, \theta\big)$ // Welford update for draws

$\bar{\alpha}, \hat{\sigma}^2_\alpha \leftarrow \text{update}(n, \hat{\sigma}^2_\alpha, \bar{\alpha}, \alpha)$ // update gradients

self.$n \leftarrow n + 1$

return

def current():

return $\big(\bar{\theta}, \bar{\alpha}, \hat{\sigma}^2_\theta, \hat{\sigma}^2_\alpha\big)$

def n-points():

return self.n

update$(n,\ \hat{\sigma}^2_{n-1},\ \bar{x}_{n-1},\ x_n)$:

$\bar{x}_n \leftarrow \bar{x}_{n-1} + \frac{1}{n}(x_n - \bar{x}_{n-1})$

$\hat{\sigma}^2_n \leftarrow \big(\frac{1}{n-1}\big)\hat{\sigma}^2_{n-1} + (x_n - \bar{x}_{n-1})^2$

return $(\bar{x}_n, \hat{\sigma}^2_n)$

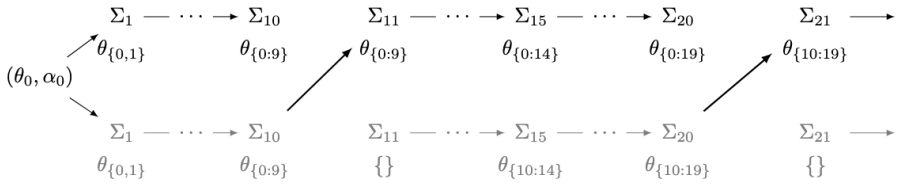

Figure 1: Example mass matrix adaptation scheme with background (below) and foreground (above) variance estimators, with a switch/flush frequency of 10 draws. Labels beneath states $\Sigma_{\{\text{iter}\}}$ indicate the draws (and their gradients) on which that estimate is based. The transformation used for the Hamiltonian at each iteration is informed by the estimate stored in the foreground's state.

4 Implementation in nutpie

The core algorithms presented here are implemented in the rust language, with all array operations abstracted away to allow users of the rust API to provide GPU implementations. Nutpie accepts both PyMC and Stan models, the latter accessed through the bridgestan library, which compiles C libraries that nutpie can load dynamically to call the logp function gradient with little overhead. PyMC models can be sampled either through the numba backend, which also allows evaluating the density and its gradient with little overhead. Alternatively, it can use the PyMC Jax backend. This incurs a higher per-call overhead, but allows evaluating the density on the

GPU, which can significantly speed up sampling for larger models. Traces are returned as ArviZ inference data objects, allowing for easy posterior analysis and convergence checks. Code: https://github.com/pymc-devs/nutpie

5 Experimental evaluation of nutpie

We run nutpie and cmdstan on posteriordb to compare performance in terms of effective sample size per gradient evaluation and in terms of effective sample size per time…

Bibliography

[1] B. Carpenter *et al.*, "Stan: A Probabilistic Programming Language," *Journal of Statistical Software*, vol. 76, no. 1, pp. 1–32, 2017, doi: 10.18637/jss.v076.i01.

[2] J. Salvatier, T. Wiecki, and C. Fonnesbeck, "Probabilistic Programming in Python using PyMC." [Online]. Available: https://arxiv.org/abs/1507.08050

[3] R. M. Neal, "MCMC using Hamiltonian dynamics." Accessed: Nov. 26, 2024. [Online]. Available: http://arxiv.org/abs/1206.1901

[4] J. H. Tran and T. S. Kleppe, "Tuning diagonal scale matrices for HMC." Accessed: Nov. 26, 2024. [Online]. Available: http://arxiv.org/abs/2403.07495

[5] I. Langmore, M. Dikovsky, S. Geraedts, P. Norgaard, and R. Von Behren, "A Condition Number for Hamiltonian Monte Carlo." Accessed: Oct. 16, 2022. [Online]. Available: http://arxiv.org/abs/1905.09813

[6] M. D. Hoffman and A. Gelman, "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." [Online]. Available: https://arxiv.org/abs/1111.4246

A Minimization of Fisher divergence for affine transformations

Here we prove that $\hat{F}$ for $F(y) = Ay + \mu$ is minimal when $\Sigma\,\mathrm{Cov}[\alpha]\Sigma = \mathrm{Cov}[x]$ and $\mu = \bar{x} + \Sigma\bar{\alpha}$, where $\Sigma = AA^T$:

Let G be the matrix of scores, with α_i as the ith column, and similarly let X be the draws matrix, consisting of x_i as the i'th column, and $\Sigma = AA^T$. The Fisher divergence between some p and $N(0, I_d)$ is

$$E_p[\|\nabla \log p(x) + X\|^2] \tag{11}$$

and as such the estimated divergence is

$$\hat{F} = \frac{1}{N}\|G + X\|^2 \tag{12}$$

Now, for some transformed $y = A^{-1}(x - \mu)$, we have

$$\hat{F}_y = \frac{1}{N}\left\|A^T G + Y\right\|^2 = \frac{1}{N}\left\|A^T G + A^{-1}(X - \mu 1^T)\right\|_F^2 \tag{13}$$

Differentiating with respect to μ, we have:

$$\frac{d\hat{F}}{d\mu} = -\frac{2}{N}\,\mathrm{tr}\Big[\mathbf{1}^T\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T A^{-1}\Big] \tag{14}$$

Setting this to zero,

$$\mathbf{1}^T\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T A^{-1} = 0 \tag{15}$$

It follows that

$$\mu^* = \bar{x} + \Sigma\bar{\alpha} \tag{16}$$

Differentiating with respect to A:

$$d\hat{F} = \frac{2}{N}\,\mathrm{tr}\Big[\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T\left(dA^T G - A^{-1}dAA^{-1}(X - \mu\mathbf{1}^T)\right)\Big] \tag{17}$$

Plugging in the result for μ^* and using the cyclic- and transpose-invariance properties of the trace gives us

$$\begin{aligned} d\hat{F} = &\frac{2}{N}\,\mathrm{tr}\big[\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)G^T dA\big] \\ &+\frac{2}{N}\,\mathrm{tr}\Big[A^{-1}\left(\Sigma\bar{\alpha}\mathbf{1}^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)^T A^{-1}dA\Big] \end{aligned} \tag{18}$$

where $\tilde{X} = X - \bar{x}\mathbf{1}^T$, the matrix with centered x_i as the columns. This is zero for all dA iff

$$\begin{aligned} 0 &= \left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)G^T + A^{-1}\left(\Sigma\bar{\alpha}\mathbf{1}^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)^T A^{-1} \\ &= A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T + \left(A^T\bar{\alpha}\mathbf{1}^T - A^{-1}\tilde{X}\right)\left(\tilde{G}^T + \tilde{X}^T\Sigma^{-1}\right), \end{aligned} \tag{19}$$

Where similarly $\tilde{G} = G - \bar{\alpha}\mathbf{1}^T$. Because $\mathbf{1}^T\tilde{X}^T = \mathbf{1}^T\tilde{G}^T = 0$, this expands to

$$\begin{aligned} 0 &= A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T - A^{-1}\tilde{X}\tilde{G}^T - A^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \\ &= \left(\tilde{G} + \Sigma^{-1}\tilde{X}\right)G^T - \Sigma^{-1}\tilde{X}\tilde{G}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \end{aligned} \tag{20}$$

$$\begin{aligned} &= \tilde{G}G^T + \Sigma^{-1}\tilde{X}\mathbf{1}\bar{\alpha}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \\ &= \tilde{G}\tilde{G}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \end{aligned} \tag{21}$$

B The No-U-Turn Sampler

NUTS(θ : position, ε : step-size, T : max tree depth):
$\rho \sim N(0, I_{d\times d})$ // refresh momentum
$B \sim \text{Unif}(\{0,1\}^T)$ // resample Bernoulli process
$(a, b, \_) \leftarrow$ ORBIT-SELECT $(\theta, \rho, B, \varepsilon)$
$(\theta^*, \_, \_) \leftarrow$ INDEX-SELECT $(\theta, \rho, a, b, \varepsilon)$
return θ^*

ORBIT-SELECT(θ, ρ, B, ε):
$a, b \leftarrow 0$
for i from 0 **to** T:
    $\tilde{a} \leftarrow a + (-1)^{B_i} 2^{i-1}, \tilde{b} \leftarrow b + (-1)^{B_i} 2^{i-1}$ // tree doubling
    $\mathbb{I}_{\text{U-Turn}} \leftarrow$ U-TURN $(a, b, \theta, \rho, \varepsilon)$
    $\mathbb{I}_{\text{Sub-U-Turn}} \leftarrow$ SUB-U-TURN $\left(\tilde{a}, \tilde{b}, \theta, \rho, \varepsilon\right)$ // recursive U-turn checks
    if $\max(\mathbb{I}_{\text{U-Turn}}, \mathbb{I}_{\text{Sub-U-Turn}}) = 0$:
        $a \leftarrow \min(a, \tilde{a}), b \leftarrow \max\left(b, \tilde{b}\right)$
    else:
        break
return $a, b, \nabla H$

INDEX-SELECT(θ, ρ, a, b, ε):
$a, b \leftarrow 0$
for i from 0 **to** $\text{len}(B)$:
    $\tilde{a} \leftarrow a + (-1)^{B_i} 2^{i-1}, \tilde{b} \leftarrow b + (-1)^{B_i} 2^{i-1}$
    $\mathbb{I}_{\text{U-Turn}} \leftarrow$ U-TURN $(a, b, \theta, \rho, \varepsilon)$
    $\mathbb{I}_{\text{Sub-U-Turn}} \leftarrow$ SUB-U-TURN $\left(\tilde{a}, \tilde{b}, \theta, \rho, \varepsilon\right)$
    if $\max(\mathbb{I}_{\text{U-Turn}}, \mathbb{I}_{\text{Sub-U-Turn}}) = 0$:
        $a \leftarrow \min(a, \tilde{a}), b \leftarrow \max\left(b, \tilde{b}\right)$
    else:
        break
return $a, b, \nabla H$

U-TURN$(\theta,\ \rho,\ a,\ b,\ \varepsilon)$:

$\theta^-, \rho^-, H^+, H^- \leftarrow$ LEAPFROG$(\theta, \rho, \varepsilon\alpha, \varepsilon)$

$\theta^+, \rho^+, \tilde{H}^+, \tilde{H}^- \leftarrow$ LEAPFROG$(\theta, \rho, \varepsilon\beta, \varepsilon)$

$H^+ = \max\big(\tilde{H}^+, H^+\big), H^- = \min\big(\tilde{H}^-, H^-\big)$

$\mathbb{I}_{\text{U-Turn}} = \rho^+ \cdot (\theta^+ - \theta^-) < 0$ **or** $\rho^- \cdot (\theta^+ - \theta^-) < 0$

return $\mathbb{I}_{\text{U-Turn}}, H^+, H^-$

SUB-U-TURN$(\theta,\ \rho,\ a,\ b,\ \varepsilon)$:

if $a = b$:

    return 0

$m \leftarrow \left\lfloor \frac{a+b}{2} \right\rfloor$

full = U-TURN$(a, b, \theta, \rho, \varepsilon)$

left = SUB-U-TURN$(a, m, \theta, \rho, \varepsilon)$

right = SUB-U-TURN$(m+1, b, \theta, \rho, \varepsilon)$

return max(left, right, full)

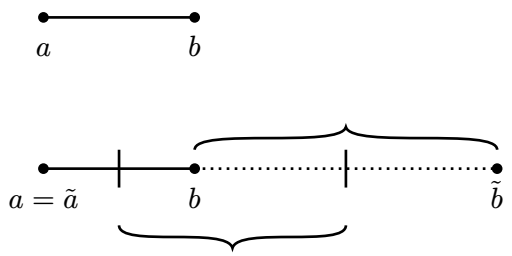

Figure 2: Modified NUTS orbit checks

```
def warmup(num_warmup, num_early, num_late, early_switch_freq,
late_switch_freq):
    position, score = draw_from_prior()
    foreground_window = MassMatrixEstimator()
    foreground_window.update(position, score)
    background_window = MassMatrixEstimator()
    step_size_estimator = StepSizeAdapt(position, score)
    first_mass_matrix = True

    for draw in range(num_warmup):
        is_early = draw < num_early
        is_late = num_warmup - draw < num_late

        mass_matrix = foreground_window.current()
        step_size = step_size_estimator.current_warmup()
        (
          accept_stat, accept_stat_sym, position, score,
          diverging, steps_from_init
        ) = hmc_step(mass_matrix, step_size, position, score)

        # Early on we ignore diverging draws that did not move
        # several steps. They probably just used a terrible step size
        ok = (not is_early) or (not diverging) or (steps_from_init > 4)
        if ok:
            foreground_window.update(position, score)
            background_window.update(position, score)

        if is_late:
            step_size_estimator.update(accept_stat_sym)
            continue

        step_size_estimator.update(accept_stat)

        switch_freq = early_switch_freq if is_early else late_switch_freq
        remaining = num_warmup - draw - num_late
        if (remaining > late_switch_freq
            and background_window.num_points() > switch_freq
        ):

            foreground_window = background_window
            background_window = MassMatrixEstimator()
            if first_mass_matrix:
                step_size_estimator.reset()
            first_mass_matrix = False
```