If Only My Posterior Were Normal: Introducing Fisher HMC

Adrian Seyboldt
adrian.seyboldt@gmail.com
PyMC Labs

2025-04-05

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 and PyMC. 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. In most cases, these problems can be overcome by careful, manual reparametrization of models, but this requires expertise and time. 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.

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 $\alpha_i = \Sigma^{-1}(x_i -$

μ). 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 (or diffeomorphism) of our posterior such that HMC works better. 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. Since the transformation is a bijection, we can choose any direction we want, as long as we stay consistent with our choice. $f^*\mu$ refers to the pullback of the posterior (which we interpret as a volume form), i.e. we *pull it back* to the space N along the transformation f. We show later how this is done in practice. If f is an affine transformation, this simplifies to mass-matrix based HMC, which we discuss now. For example, choosing $f(x) = \Sigma^{\frac{1}{2}}x + \mu$ corresponds to the mass matrix Σ^{-1}. This is described in more detail in (Neal 2012).

HMC efficiency is notoriously dependent on the parametrization, so it's 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 an automatic learning of the 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.

2.3 Fisher divergence

Let (N, g) be a Riemannian manifold with probability volume forms ω_1 and ω_2. We can define a scalar function z on N by $\omega_2 = z\omega_1$, or equivalently we also write this as $z = \frac{\omega_2}{\omega_1}$.

We define the Fisher divergence of ω_1 and ω_2 as

$$\mathcal{F}_g(\omega_1, \omega_2) = \int \|\nabla \log(z)\|_g^2 d\omega_1. \tag{5}$$

Note that $\mathcal{F}$ requires more structure on N than KL-divergence $\int \log(z) d\omega_1$, as the norm depends on the metric tensor. 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_1 by pulling back μ to N, i.e. $\mathcal{F}_g(f^*\mu, \omega_1)$.

We can also compute this Fisher divergence directly on M, by pushing the metric tensor to M:

$$\mathcal{F}_g(f^*\mu, \omega_1) = \mathcal{F}_{(f^{-1})^* g}\big(\mu, (f^{-1})^* \omega_1\big) \tag{6}$$

In this case, μ is our posterior, M is the space on which it is originally defined, and ω_1 is the standard normal distribution.

2.4 Affine choices for the diffeomorphism F

We focus on three families of 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 effectively doing diagonal mass matrix estimation. In this case, the sample Fisher divergence reduces to

$$\hat{F}_{\sigma,\mu}(f^*Y, Z) = \frac{1}{N} \sum_i \left\| \sigma \odot \alpha_i + \sigma^{-1} \odot (x_i - \mu) \right\|^2 \tag{7}$$

This is a special case of the affine transformation in Appendix A and minimal if $\sigma^2 = \mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$ and $\mu = \bar{x}_i + \sigma^2 \bar{s}_i$; the same result from the solvable case 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 $\mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$.

Some theoretical results for normal posteriors

If the posterior is $N(\mu, \Sigma)$, then the minimizers $\hat{\mu}$ and $\hat{\sigma}$ of $\hat{\mathcal{F}}$ converge to μ and $\exp(\frac{1}{2} \log \mathrm{diag}(\Sigma) - \frac{1}{2} \log \mathrm{diag}(\Sigma^{-1})$ respectively. This is a direct consequence of $\mathrm{Cov}[x_i] \to \Sigma$ and $\mathrm{Cov}[\alpha_i] \to \Sigma^{-1}$.

$\hat{\mathcal{F}}$ 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. When we choose $\mathrm{diag}(\Sigma)$ as mass matrix, we effectively minimize $\sum_i \lambda_i$, and only penalize large eigenvalues. If we choose

$\text{diag}(\mathbb{E}(\alpha\alpha^T))$ (as proposed, for instance, in Tran and Kleppe (2024)) we effectively minimize $\sum \lambda_i^{-1}$ and only penalize small eigenvalues. But based on theoretical results for multivarite normal posteriors in Langmore et al. (2020), we know that both large and small eigenvalues make HMC less efficient. To this effect, we

We can use the result in (todo ref) 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

We choose $f_{A,\mu}(y) = Ay + \mu$. This corresponds to a mass matrix $M = (AA^T)^{-1}$. Because as we will see $\hat{\mathcal{F}}$ only depends on AA^T and μ, we can restrict A to be symmetric positive definite. We get

$$\hat{\mathcal{F}}[f] = \frac{1}{N} \sum \left\| A^T s_i + A^{-1}(x_i - \mu) \right\|^2 \tag{8}$$

which is minimized when $AA^T \,\text{Cov}[x_i] AA^T = \text{Cov}[\alpha_i]$ (proof in Appendix A), and as such corresponds again to our earlier derivation in Section 2.1. If the two covariance matrices are full rank, we get a unique minimum at the geometric mean of $\text{Cov}[x_i]$ and $\text{Cov}[s_i]$.

2.4.3 Diagonal plus low-rank

If the number of dimensions is larger than the number of draws, we can add regularization terms. And to avoid $O(n^3)$ computation costs, we can project draws and scores into the span of x_i and α_i and compute the regularized mean in this subspace. If we only store the components, we can avoid $O(n^2)$ storage, and still do all operations we need for HMC quickly. To further reduce computational cost, we can ignore eigenvalues that are close to one with a cutoff parameter c. The algorithm is as follows:

function LOW-RANK-ADAPT(D, G, c)
    ▷ Combine bases
    $U^D \leftarrow$ **SVD** (D)
    $U^G \leftarrow$ **SVD** (G)
    $S \leftarrow [U^D U^G]$

    ▷ Get jointly-spanned orthonormal basis
    $Q, \_ \leftarrow$ **QR_thin** (S)
    $P^D \leftarrow Q^T D$
    $P^G \leftarrow Q^T G$

    $C^D \leftarrow P^D (P^D)^T + \gamma I$
    $C^G \leftarrow P^D (P^D)^T + \gamma I$

    $\Sigma \leftarrow$ **spdm** (C^D, C^G)

    $U \Lambda U^{-1} \leftarrow$ **eigendecompose** (Σ)

18: $U_c \leftarrow \{U_i : i \in \{i : \lambda_i \geq c\}\}$
19: $M \leftarrow QU_c(\Lambda_c - 1)U_c^T + I$
20: **return** M

By subtracting the identity matrix from the diagonal matrix of eigenvalues, we obtain a matrix whose eigenvalues include all those of Σ which are sufficiently far from 1, with the remaining eigenvalues equal to 1. This is then a "diagonal plus low-rank" matrix.

2.5 Normalizing flows

Normalizing flows provide a large family of diffeomorphisms that we can use to transform our posterior. We have rather strong requirements for the flows, however: we need forward and inverse transformations, and we need to be able to compute the log determinant of the jacobian of the transformation efficiently. A well studied family of normalizing flows that provides all of those is RealNVP (Dinh, Sohl-Dickstein, and Bengio (2017)). For our experiments, we used the library flowjax that implements RealNVP and other normalizing flows in jax. I slightly changed the adaptation schema from the usual nutpie algorithm (described in more detail in the next section), so that for the first couple of windows only diagonal mass matrix adaptation is used, and only after 150 draws do we start to fit a normalizing flow. We then repeatedly run an Adam optimizer on a window of draws, and use the updated normalizing flow to sample.

Especially for larger models the size of our training data set seems to be very important, so I included the full trajectory of the HMC sampler as training data, not just the draws themselves. I think this might be useful to do even if the amount of training data is not a limiting factor, as we would like that the transformed posterior matches the normal distribution everywhere the HMC sampler evaluates it, not just at those points it accepts as draws. So far I have not tested this systematically, but my experiments suggest that this can help us to sample a wide range of posterior distributions a lot more efficiently, or also to sample many distributions that were previously not possible to sample without extensive manual reparametrizations. It also comes with a large computational cost however, as the optimization itself can take a long time. Luckily, it seems to run relatively efficiently on GPUs, so that even if the logp function itself does not lend itself to evaluation on GPUs, we can still spend most of the computational time running scalable code. Often, the number of gradient evaluations that are necessary to sample even complicated posterior distributions decrease a lot.

code: https://github.com/pymc-devs/nutpie

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 always have the same problem: 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.

3.1 Choice of initial position

Stan draws initial points independently for each chain from the interval $(-2, 2)$ on the unconstrained space. I don't have any clear data to suggest so, but for some time PyMC has initialized using draws around the prior mode instead, and it seems to me that this tends to be more robust. This is of course only possible for model definition languages that allow prior sampling, and would for instance be difficult to implement in Stan.

3.2 Choice of initial diffeomorphism

Stan starts the first adaptation with an identity mass matrix. The very first HMC trajectories seem to be overall much more reasonable if we use $M = \text{diag}(\alpha_0^T \alpha_0)$ instead. This also makes the initialization independent of variable scaling.

3.3 Accelerated Window-Based Adaptation

Stan and other samplers do not run vanilla HMC, but often the No-U-Turn Sampler (NUTS), where the Hamiltonian trajectory length is chosen automatically. This can make it very costly to generate draws if the mass matrix is not adapted well, because in those cases we often use a large number of HMC steps for each draw (typically up to 1000). Thus very early on during sampling, we have a large incentive to use available information about the posterior as quickly as possible, to avoid these long trajectories. 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 at each step based on the previous k draws. However, this is computationally inefficient (unless the logp function is very expensive), and is not easily implemented as a streaming estimator (see below for more details). Using several overlapping estimation windows, though, we can compromise between optimal information usage and computational cost, while still using streaming estimators.

4 Implementation in nutpie

nutpie implements:

- New adaptation schema
- Diagonal mass matrix adaptation based on the Fisher divergence
- Low rank mass matrix adaptation as described here
- Explicit transformation with normalizing flows

The core algorithms are implemented in rust, which all array operations abstracted away, to allow users of the rust API to provide GPU implementations.

It can take PyMC or Stan models.

Stan is used through bridgestan, 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 of pymc, which also allows evaluating the density and its gradient with little overhead. Alternatively, it can use the pymc jax backend. This incures a higher per-call overhead, but allows evaluating the density on the GPU, which can significantly speed up sampling for larger models.

nutpie returns sampling traces as arviz datasets, to allow easy posterior analysis and convergence checks.

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

[2] 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

[3] 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

[4] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP." Accessed: Mar. 21, 2025. [Online]. Available: http://arxiv.org/abs/1605.08803

A Minimize 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

$$E_p[\|\nabla \log p(x) + X\|^2] \tag{9}$$

and then the estimated divergence is

$$\hat{F} = \frac{1}{N}\|G + X\|^2 \tag{10}$$

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

$$\hat{F}_y = \frac{1}{N}\left\|A^T G + Y\right\|^2 = \frac{1}{N}\left\|A^T G + A^{-1}(X - \mu 1^T)\right\|_F^2 \tag{11}$$

Differentiating with respect to μ, we have:

$$\frac{d\hat{F}}{d\mu} = -\frac{2}{N}\,\mathrm{tr}\Big[\mathbf{1}^T\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T A^{-1}\Big] \tag{12}$$

Setting this to zero,

$$\mathbf{1}^T\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T A^{-1} = 0 \tag{13}$$

It follows that

$$\mu^* = \bar{x} + \Sigma\bar{\alpha} \tag{14}$$

Differentiating with respect to A:

$$d\hat{F} = \frac{2}{N}\,\mathrm{tr}\Big[\left(A^T G + A^{-1}(X - \mu\mathbf{1}^T)\right)^T\left(dA^T G - A^{-1}dAA^{-1}(X - \mu\mathbf{1}^T)\right)\Big] \tag{15}$$

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

$$\begin{aligned} d\hat{F} = &\frac{2}{N}\,\mathrm{tr}\big[\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)G^T dA\big] \\ &+\frac{2}{N}\,\mathrm{tr}\Big[A^{-1}\left(\Sigma\bar{\alpha}\mathbf{1}^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)^T A^{-1}dA\Big] \end{aligned} \tag{16}$$

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

$$\begin{aligned} 0 &= \left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)G^T + A^{-1}\left(\Sigma\bar{\alpha}\mathbf{1}^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}\mathbf{1}^T\right)\right)^T A^{-1} \\ &= A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T + \left(A^T\bar{\alpha}\mathbf{1}^T - A^{-1}\tilde{X}\right)\left(\tilde{G}^T + \tilde{X}^T\Sigma^{-1}\right), \end{aligned} \tag{17}$$

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

$$0 = A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T - A^{-1}\tilde{X}\tilde{G}^T - A^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \tag{18}$$

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

We can explicitly incorporate this transformation into HMC (or variations of it like NUTS):

```
def hmc_proposal(rng, x, mu_density, F, step_size, n_leapfrog):
  # Compute the log density and score (differential of the log density)
  # at the initial point.
  logp_x, score_x = score_and_value(mu_density)(x)
  # Compute the log density and score in the transformed space.
  # We will see later how to do this.
  y, score_y, logp_y = f_pullback_score(x, score_x, logp_x)

  # Sample a new velocity for our trajectory.
  # In the transformed space we assume an identity mass matrix,
  # so we don't have to distinguish between momentum and velocity.
  velocity = rng.normal(size=len(x))

  # Several leapfrog steps
  for i in range(n_leapfrog):
    # velocity half-step
    velocity += step_size / 2 * score_y
    # position step
    y += step_size * velocity

    # Transform back and evaluate density
    x = f_transform(y)
    logp_x, score_x = score_and_value(mu_density)(x)
    y, score_y, logp_y = f_pullback_score(x, score_x, logp_x)

    # second velocity half-step
    velocity += step_size / 2 * score_y

  return x
```

AA Sobolev divergence

The Fisher divergence uses the information in the scores of our posterior. But it still ignores some additional information we have. Specifically, for each pair of points where we evaluate the posterior, we know the ratio of the densities at those points. If our transformed posterior has density $p(x)$, then we'd like that $\frac{p(x)}{p(y)} \approx \frac{q(x)}{q(y)}$, where q is the standard normal density. (We look only at pairwise density ratios, because we don't know the normalization factor of our posterior, and thus can't compute the ratios $\frac{p(x)}{q(x)}$ directly.)

This leads us to the additional loss term

$$\begin{aligned} &\int\int ((\log(p(x)) - \log(p(y))) - (\log(q(x)) - \log(q(y)))^2)p(x)p(y)dxdy \\ &= 2\,\mathrm{Var}_{p(x)}[\log(p(x)) - \log(q(x))] \end{aligned} \tag{20}$$

We extend the Fisher divergence to a new divergence, the Sobolev divergence, by adding this second term:

$$\mathcal{S}_{g(\omega_1,\omega_2)} = \mathcal{F}_{g(\omega_1,\omega_2)} + \int \log(z)^2 \omega_1 - \left(\int \log(z)\omega_1\right)^2 = \mathcal{F}_{g(\omega_1,\omega_2)} + \mathrm{Var}_{\omega_1}[\log(z)] \quad (21)$$

The name is due to the similarity to the Sobolev norm $\|f\|^2_{H^1} = \int |f|^2 + |\nabla f|^2$.

AB Transformations of scores

In practice, we don't work with the measure μ directly, but with the densities relative to Lebesgue measures λ_N and λ_M, so $\mu = p\lambda_M$ and $\omega = q\lambda_N$. The change-of-variable tells us that $f^*\lambda_M = |\det(df)|\lambda_N$. This allows us to compute the transformed scores $d\log\left(f^*\frac{p}{\lambda_N}\right)$:

$$d\log\left(f^*\frac{\mu}{\lambda_N}\right) = d\log\left(f^*\frac{p}{\lambda_M}\frac{f^*\lambda_M}{\lambda_N}\right) = f^* d\log(p) + d\log|\det(df)| = \hat{f}^*(d\log(p), 1) \quad (22)$$

where $\hat{f} : N \to M \times \mathbb{R}, x \mapsto (f(x), \log|\det(f)|)$

This allows us to implement the score pullback function in autodiff systems, for instance in Jax:

```
def f_and_logdet(y):
  """Compute the transformation F and its log determininant jacobian."""
  ...

def f_inv(x):
  """Compute the inverse of F."""
  ...

def f_pullback_score(x, s_x, logp):
  """Compute the transformed position, score and logp."""
  y = F_inv(x)
  (_, log_det), pullback_fn = jax.vjp(f_and_logdet, y)
  s_y = pullback_fn(s_x, jnp.ones(()))
  return y, s_y, logp + log_det
```

We can further simplify the Fisher divergence if we set ω to a standard normal distribution, and use the standard euclidean inner product as the metric tensor:

$$\begin{aligned}\mathcal{F}(f^*\mu, \omega) &= \int \left\|\nabla \log \frac{f^*\mu}{\omega}\right\|^2_g f^*\mu \\ &= \int \left\|\hat{f}^*(d\log p, 1) - d\log(q)\right\|^2_{g^{-1}} f^*\mu \\ &= \int \left\|\hat{f}^*(d\log p_x, 1) + x\right\|^2 f^*\mu(x)\end{aligned} \quad (23)$$

Given posterior draws x_i and corresponding scores α_i in the original posterior space X we can approximate this expectation as

$$\hat{\mathcal{F}} = \frac{1}{N}\sum_i \left\|\hat{f}^*(s_i, 1) + f^{-1}(x_i)\right\|^2 \quad (24)$$

Or in code:

```
def log_loss(f_pullback_scores, draw_data):
  draws, scores, logp_vals = draw_data
  pullback = vectorize(f_pullback_scores)
```

```
    draws_y, scores_y, _ = pullback(draws, scores, logp_vals)
    return log((draws_y + scores).sum(0).mean())

def log_sobolev_loss(f_pullback_scores, draw_data):
    draws, scores, logp_vals = draw_data
    pullback = vectorize(f_pullback_scores)
    draws_y, scores_y, logp_y = pullback(draws, scores, logp_vals)

    fisher_loss = (draws_y + scores).sum(0).mean()
    std_normal_logp = - draws_y @ draws_y / 2
    var_loss = (logp_y - std_normal_logp).var()
    return log(fisher_loss + var_loss)
```

Note: Some previous literature (todo ref) proposed to minimize $\mathbb{E}[\alpha_x^T \alpha_x]$, which is similar, but does not solve the issue of choosing a well-defined inner product. But finding a good inner product is the whole point of mass matrix adaptation. If we pull pack the inner product of the standard normal distribution to X and use the corresponding inner product on the dual space of 1-forms, we end up with an equivalent definition for the loss function defined above.