If Only My Posterior Were Normal: Adaptive HMC with Fisher Divergence

Adrian Seyboldt
adrian.seyboldt@gmail.com
PyMC-Labs

1. November 2024

Abstract

TODO mostly chatpgt for now...

Hamiltonian Monte Carlo (HMC) has become a crucial tool in Bayesian inference, offering efficient exploration of complex, high-dimensional parameter spaces. However, 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. In this paper, we propose an adaptive framework for HMC that uses Fisher divergence to guide transformations of the parameter space, generalizing the concept of a mass matrix to arbitrary diffeomorphisms. By aligning Fisher scores of the transformed posterior with those of a standard normal distribution, our method identifies optimal transformations that adapt to the posterior's scale, shape, and orientation. We develop theoretical foundations for these adaptive transformations, provide efficient implementation strategies, and demonstrate significant sampling improvements over conventional methods. Additionally, we introduce and evaluate nutpie, an implementation of our method in PyMC, comparing it with existing samplers across a suite of models. Our results show that nutpie delivers enhanced sampling efficiency, particularly for challenging posteriors, positioning it as a powerful tool for Bayesian computation.

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

1 Introduction

TODO (part is from ChatGPT, remove some of the fluff)

Hamiltonian Monte Carlo (HMC) is a powerful Markov Chain Monte Carlo (MCMC) method widely used in Bayesian inference for exploring complex posterior distributions. HMC can explore high-dimensional parameter spaces more efficiently than traditional MCMC techniques, making it popular in probabilistic programming libraries like Stan and PyMC. However, the performance of HMC depends critically on the parameterization of the posterior space, often framed in terms of

a “mass matrix” that defines an inner product on this space. Properly tuning this parameterization is crucial for efficient sampling, yet achieving optimal performance remains a challenging task, particularly for posterior distributions with complex geometries, such as those that exhibit strong correlations or funnel-like structures.

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. While effective in many cases, this method has limitations when the posterior deviates significantly from normality. In cases where the posterior distribution has complex geometries, relying on a fixed mass matrix can lead to inefficient sampling, with trajectories that fail to capture the underlying structure of the parameter space. This inefficiency is especially pronounced when dealing with correlated parameters or posteriors that exhibit challenging topologies, such as those commonly encountered in hierarchical models and large-scale datasets.

To address these limitations, we propose an adaptive HMC framework that extends beyond the traditional concept of a mass matrix, using arbitrary diffeomorphisms to dynamically transform the parameter space. In this work, we utilize the Fisher divergence as a guiding criterion for these transformations, allowing us to adapt the geometry of the posterior space in a way that optimizes HMC’s efficiency. By aligning the Fisher scores (derivatives of the log-density) of the transformed posterior with those of a standard normal distribution, we approximate an idealized parameterization that promotes efficient exploration.

Our approach is grounded in the observation that the Fisher divergence provides a meaningful metric for quantifying the discrepancy between the transformed posterior and a normal distribution. By minimizing this divergence, we can identify transformations that reduce the complexity of the posterior geometry, facilitating more effective trajectories and improved convergence. This adaptive framework generalizes the mass matrix concept, enabling transformations that adapt not only to scale but also to shape, orientation, and other geometric characteristics of the posterior.

In this paper, we develop and analyze the application of Fisher divergence to HMC adaptation, presenting both theoretical insights and practical implementation strategies. Through numerical experiments, we demonstrate that our method can significantly improve sampling efficiency compared to traditional approaches, particularly in cases where the posterior deviates substantially from normality. This adaptive transformation framework offers a flexible and scalable alternative for MCMC sampling, with potential applications across a wide range of complex Bayesian models.

This introduction outlines the problem, motivation, proposed approach, and contributions in a way that provides a clear roadmap for readers unfamiliar with the specific technical concepts. It establishes why the Fisher divergence and adaptive transformations are relevant and what readers can expect from the paper.

The performance of Hamiltonian MCMC samplers like PyMC or Stan depends critically on the choice of mass matrix. The mass matrix represents a choice of inner product in the parameter space.

(I talk about HMC, but everything applies to variations of this, like NUTS or ... todo refs)

The most common choice is to use the inverse of the posterior covariance as mass matrix. Most commonly, the mass matrix is constrained to be a diagonal matrix.

There are previous proposals to use the covariance of the fisher scores instead, or to minimize a KL-divergence between a multivariate normal distribution and the posterior, and to use the resulting precision matrix of that normal distribution.

1.1 Current challenges in HMC

- Slow warmup phase (chicken-egg problem). (mention long trajectories in early phase)
- Bad performance with correlated posteriors. (related to eigenvalues as shown later. Especially if datasets get large)
- Bad geometry in posterior (funnels etc)

2 Adaptive Transformations in HMC: Motivation, Theory, and Examples

2.1 Motivation: Example with independent gaussian posterior

When we run HMC, we compute the derivatives of the posterior log density (the fisher scores). They contain a lot of information about the posterior, but when we compute the mass matrix, we ignore this information.

To illustrate how useful the fisher scores can be, let's start very simple, and assume our posterior is a one dimensional normal distribution $N(\mu, \sigma^2)$ with density $d\mu$, and let's assume we have two posterior draws x_1 and x_2, together with the covector (row-vector) of fisher scores $\alpha_i = \frac{\partial}{\partial x_i} \log d\mu(x_i) = \sigma^{-2}(\mu - x_i)$. Based on this information we can directly compute μ and σ, and so identify the exact posterior, by solving for μ and σ. We get $\mu = \bar{x} + \sigma^2\bar{\alpha}$ and $\sigma^2 = \mathrm{Var}[x_i]^{\frac{1}{2}}\mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$, where $\bar{x}$ and $\bar{\alpha}$ are the sample means of x_i and α_i respectively. So we can compute the exact posterior and ideal mass matrix with no sample variance based on just two points!

This generalizes directly to multivariate normal posteriors $N(\mu, \Sigma)$. Let's assume we have $N+1$ linearly independent draws x_i, and the fisher scores $\alpha_i = \Sigma^{-1}(x_i - \mu)$. The mean of these equations gives us $\mu = \bar{x} - \Sigma\bar{\alpha}$. It follows that $S = \Sigma^{-1}X$, where the i-th column of S is $\alpha_i - \bar{\alpha}$, and the i-th column of X is $x_i - \bar{x}$. And finally we get $SS^T = \mathrm{Cov}[\alpha_i] = \Sigma^{-1}XX^T\Sigma^{-1} = \Sigma^{-1}\,\mathrm{Cov}[x_i]\Sigma^{-1}$. We can recognize Σ as the geometric mean of the positive symmetric matrices $\mathrm{Cov}[x_i]$ and $\mathrm{Cov}[s_i]^{-1}$, so $\Sigma = \exp(\log(\mathrm{Cov}[x_i])/2 - \log(\mathrm{Cov}[\alpha_i]/2)$ or some expression with lots of matrix square roots (todo).

Given the fisher scores, we can compute the parameters of a normal distribution exactly. Most posterior distributions are not multivariate normal distributions, or we would not have to run MCMC in the first place. It is quite common that they approximate normal distributions reasonably well, so this should indicate that the fisher scores contain useful information we are currently neglecting.

2.2 Pullbacks of fisher scores

Let μ be a probability measure on some space X with density $d\mu(x)$, and $F : Y \to X$ a diffeomorphism. (Note that we define the diffeomorphism as going from the transformed space to the original space X, not the other way round).

The pullback of μ defines a measure $F^*\mu$ on Y with density $d(F^*\mu)(y) = d\mu(F(y))\left|\det \frac{\partial}{\partial y} F(y)\right|$, where we use the usual change-of-variable formula.

We define $\hat{F^*}\alpha_x$ for a fisher score α_x as the fisher score of the transformed distribution on Y, so

$$\hat{F^*}\alpha_x = \frac{\partial}{\partial y} \log d(F^*\mu)(y)\Big|_{y=F^{-1}(x)} \tag{1}$$

We can simplify this a bit by defining $\hat{F} : Y \to X \times \mathbb{R}$, where $\hat{F}(y) = \left(F(y), \log\left|\det \frac{\partial F}{\partial y}\right|\right)$. Given a fisher score α_x on the space X, we can show that

$$\hat{F^*}\alpha_x = \hat{F}^*(\alpha_x, 1) \tag{2}$$

This allows us to implement the fisher score pullback using autodiff systems, for instance in Jax (although there are more efficient implementations in special cases):

```
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_fisher_score(x, s_x, logp):
    y = F_inv(x)
    (_, logdet), pullback_fn = jax.vjp(F_and_logdet, y)
    s_y = pullback_fn(s_x)
    return y, s_y, logp - logdet
```

2.3 Intrinsic metric tensor on distributions

The fisher metric defines an intrinsic metric on parameter spaces of distribuitions. This is a setting that often appears in frequentist statistic, where no distribution is defined on the parameter space directly, but only on the dataset, though the likelihood.

When we sample Bayesian models, we would like to have a similarly intrinsic metric, but not with respect to the likelihood, but with respect to the posterior distribution.

To construct such a metric, we start with a standard normal distribution. Here, I think we can argue that we already know what the metric should be: Since the normal distribution is defined by $d\mu(x) \propto \exp(- < x, x\frac{\geq}{2})$, and as such directly in terms of a metric, we can see this metric as the intrinic metric of the normal distribution.

We would now like to transfer this metric to other distributions though diffeomorphisms.

Given an absolute continuous distribution on $X = \mathbb{R}^n$ with sufficiently smooth density, there is a (not necessarily unique) diffeomorphism $F : Y \to X$, where Y is $\mathbb{R}^n$ with the standard inner product, such that $D[F] = \int \left\|y + F^* s(F^{-1}(y))\right\|^2 dF^*\mu(y) = 0$. Even though the diffeomorphism is not unique, the induced inner product on X is unique.

Assume F_1 and F_2 are such that $D[F_1] = D[F_2] = 0$. Then the pullback inner products to X are identical. proof...

This defines an intrinsic metric on X.

2.4 Transformed HMC

Given a posterior μ on X and a diffeomorphism $F : Y \to X$, we can run HMC on the transformed space Y:

```
def hmc_step(rng, x, d_mu, F, step_size, n_leapfrog):
    logp_x, s_x = gradient_and_value(d_mu)(x)
    logp_y, s_y = F.inverse.hat(logp, s_x)

    velocity = rng.normal(size=len(x))
    for i in range(n_leapfrog):
        # TODO
        pass
```

If F is an affine transformation, this simplifies to the usual mass-matrix based HMC. For instance, if we choose $F(x) = \sigma \odot x$, this corresponds to the mass matrix $\text{diag}(\sigma^{-2})$.

HMC efficiency is famously dependent on the parametrization, so we know that this is much more or less efficient for some choices of F than for others. It is however not obvious what criterion we could use to decide how good a particular choice of F is. We need a loss function D that maps the diffeomorphism to a measure of difficulty for HMC.

This hard to quantify in general, but we can notice that the efficiency of HMC largely depends on the trajectory, and this trajectory does not depend on the density directly, but only the fisher scores. We also know that HMC is efficient if the posterior is a standard normal distribution. So a reasonable loss function can ask how different the fisher scores on the transformed space are from the fisher scores of a standard normal distribution. If they match well, we will use the same trajectory we would use for a standard normal distribution, which we know to be a good trajectory. And because the standard normal distribution is defined in terms of an inner product, we already have a well-defined norm on the fisher scores that we can use to evaluate their difference. So we define

$$\begin{aligned} D[F] &= \int \left\| \frac{\partial}{\partial y} \log d(F^*\mu)(y) - \frac{\partial}{\partial y} \log N(y|\ 0, 1) \right\|^2 d(F^*\mu)(y) \\ &= \int \left\| \frac{\partial}{\partial y} \log d(F^*\mu)(y) + y \right\|^2 d(F^*\mu)(y) \\ &= \int \left\| F^{\hat{*}} \frac{\partial}{\partial x} \log(d\mu(x)) + y \right\|^2 d(F^*\mu)(y) \end{aligned} \tag{3}$$

This is a special case of the fisher divergence between the transformed posterior and a standard normal distribution.

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

$$\hat{D}_F = \frac{1}{N} \sum_i \left\| F^{\hat{*}} s_i + F^{-1}(x_i) \right\|^2 \tag{4}$$

Or in code:

```
def log_loss(F_pullback_fisher_scores, draw_data):
    draws, fisher_scores, logp_vals = draw_data
    pullback = vectorize(F_pullback_fisher_scores)
```

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

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.

2.5 Specific choices for the diffeomorphism F

For particular families of diffeomorphisms F, we can get more specific results.

2.5.1 Diagonal mass matrix

If we choose $F_{\sigma,\mu} : Y \to X$ as $x \mapsto y \odot \sigma + \mu$, we get the same effect as diagonal mass matrix estimation.

In this case, the fisher divergence reduces to

$$\hat{D}_{\sigma,\mu} = \frac{1}{N} \sum_i \left\| \sigma \odot \alpha_i + \sigma^{-1} \odot (x_i - \mu) \right\|^2 \tag{5}$$

This is a special case of the affine transformation in Section 6.1 and minimal if $\sigma^2 = \mathrm{Var}[x_i]^{\frac{1}{2}} \mathrm{Var}[\alpha_i]^{-\frac{1}{2}}$ and $\mu = \bar{x}_i + \sigma^2 \bar{s}_i$. So we recover the same result we got in Section 2.1. It is very easy to compute (also using an online algorithm for the variance to avoid having to store the x_i and α_i values), and therefore the default in nutpie. In a Section 5 we will show some benchmarks to compare its performance.

Some theoretical results for normal posteriors

If the posterior is $N(\mu, \Sigma)$, then the minimizers $\hat{\mu}$ and $\hat{\sigma}$ of $\hat{D}_F$) 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 $\mathrm{Cov}[\alpha_i] \to \Sigma^{-1}$.

D_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 $\mathrm{diag}(\mathbb{E}(\alpha\alpha^T))$ we effectively minimize $\sum \lambda_i^{-1}$ and only penalize small eigenvalues. But based on some theoretical work for multivariate normal distributions, we know that both large and small eigenvalues make HMC less efficient. (todo ref)

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.

2.5.2 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 $\hat{D}_F$ only depends on AA^T and μ, we can restrict A to be symmetric positive definite.

We get

$$\hat{D}[F] = \frac{1}{N} \sum \left\| A^T s_i + A^{-1}(x_i - \mu) \right\|^2 \tag{6}$$

which is minimal if $AA^T \operatorname{Cov}[x_i] AA^T = \operatorname{Cov}[\alpha_i]$ (for a proof, see Section 6.1), 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 $\operatorname{Cov}[x_i]$ and $\operatorname{Cov}[s_i]$.

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 fisher scores into the span of x_i and α_i, compute the regularized mean in this subspace and use the mass matrix .... 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. todo expand paragraph

2.5.3 Model specific diffeomorphisms

Sometimes we can suggest useful families of diffeomorphism based on the model itself. A common reparametrization for instance is the non-centered parametrization, where we change a model from the centered parametrization

$$x \sim N(0, \sigma^2) \tag{7}$$

into the non-centered parametrization

$$\begin{aligned} z &\sim N(0, 1) \\ x &= z \odot \sigma \end{aligned} \tag{8}$$

We can generalize this for any choice of $0 \leq t \leq 1$ to (todo add ref, ask Aki)

$$\begin{aligned} z &\sim N(0, \sigma^{2t}) \\ x &= z \odot \sigma^{1-t} \end{aligned} \tag{9}$$

This suggests $F_{t(\sigma,z)} = (\sigma, \sigma^{1-t} z)$...

2.5.4 Normalizing flows

Define F_η as a normalizing flow with parameters η. Use adam or similar to minimize $D[F]$.

todo

3 Adaptation schema to learn the diffeomorphism

Whether we adapt a mass matrix using the posterior variance as Stan does, or if we use any of the bijections based on the fisher divergence defined above, we always have the same problem: In order to generate posterior draws we need the mass matrix (or the bijection), but to estimate the mass-matrix/ bijection we need posterior draws.

There is a well known way out of this conundrum: We start sampling with same 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 remained largely unchanged for a number of years. PyMC, numpyro and blackjax all mostly use the same details as Stan, with at most minor modifications.

There are howerver I think a couple of minor changes that seem to improve the effiencey of this schma significantly. Except for the last section, where I discuss adaptation schemas for normalizing flows, I will assume that we adapt a mass matrix (or equivalently an affine bijection).

3.1 Choice of initial position

Stan draws initial points independently for each chain from the interval $(-1, 1)$ on the unconstrained space. I don't have any clear data to suggest so, but for some time PyMC has initialized using draws from the prior 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 sampler do not run vanilla HMC, but often NUTS, where the length of the trajectory is choosen 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 (in the 100s or typically up to 1000). Very early on during sampling we have a large intcentive to use available information obout the posterior as quickly as possible, to avoid these long trajectories. By default Stan starts adaptation with a step-size adaptation window, (50 draws todo check), where we do not change the mass matrix at all. It is then followed by a mass matrix adaptation window (100 draws? todo), where we generate draws for the next mass matrix update, but still use the initial mass matrix for sampling.

It is not uncommon that trajectories are very long during these first 150(todo) draws, and drop significantly after the first update. And because the trajectory lengths can easily vary by a factor of 10 or 100 between these phases, the draws before the first mass matrix change can take a sizable percentage of the total sampling time.

```
class MassMatrixEstimator:
    def update(self, position, fisher_score):
        ...
    def current(self) -> MassMatrix:
        ...
    def num_points(self) -> int:
        ...

class StepSizeAdapt:
    def reset(self):
        ...
    def update(self, accept_statistic):
        ...
    def current_warmup(self) -> float:
        ...
    def current_best(self) -> float:
        ...

def warmup(num_warmup, num_early, num_late, early_switch_freq, late_switch_freq):
    position, fisher_score = draw_from_prior()
```

```
foreground_window = MassMatrixEstimator()
foreground_window.update(position, fisher_score)
background_window = MassMatrixEstimator()
step_size_estimator = StepSizeAdapt(position, fisher_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, fisher_score,
      diverging, steps_from_init
    ) = hmc_step(mass_matrix, step_size, position, fisher_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, fisher_score)
        background_window.update(position, fisher_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
```

Constant step size adaptation with dual averaging. Overlapping windows, so that we can switch to a better estimate quickly.

Intuition: We want to update quickly, and not use an old mass matrix estimate at a point when we have more information and could already compute a better estimate. We could just use a tailing window and update in each step with the previous k draws. This is computationally inefficient (unless the logp function is very expensive), and can not easily be implemented as a streaming estimator (see below for more details). But if we use several overlapping estimation windows, we can compromise between optimal information usage and computational cost, and still use streaming mass matrix estimators.

Three phases:

- Initial phase with small window size to find the typical set. Discard duplicate draws.
- Main phase with longer windows

- Final phase with constant mass matrix: Only step size is adapted. Use a symmetric estimate of the acceptance statistic (todo ref stan discourse).

This scheme can be used with arbitrary mass matrix estimators. If the estimator allows a streaming (ref) implementation, we do not need to store the draws within each window.

4 Implementation in nutpie

todo

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. todo

6 Appendix

6.1 Minimize Fisher divergence for affine transformations

$D[F]$ for $F(y) = Ay + \mu$ is minimal if $\Sigma \operatorname{Cov}[\alpha] \Sigma = \operatorname{Cov}[x]$ and $\mu = \bar{x} + \Sigma\bar{\alpha}$, where $\Sigma = AA^T$:

We collect all α_i in the columns of G, and all x_i in the columns of X. Let e we the vector containing only ones. And let $\Sigma = AA^T$.

$$D = \frac{1}{N} \left\| A^T G + A^{-1}(X - \mu e^T) \right\|_F^2 \tag{10}$$

6.1.1 Find $\hat{\mu}$

Take the differential with respect to only μ:

$$\begin{aligned} dD &= -\frac{2}{N} \operatorname{tr}\left[\left(A^T G + A^{-1}(X - \mu e^T)\right)^T A^{-1} d\mu e^T \right] \\ &= -\frac{2}{N} \operatorname{tr}\left[e^T \left(A^T G + A^{-1}(X - \mu e^T)\right)^T A^{-1} d\mu \right] \end{aligned} \tag{11}$$

At a minimum of D this has to be zero for all $d\mu$, which is the case iff

$$e^T \left(A^T G + A^{-1}(X - \mu e^T)\right)^T A^{-1} = 0 \tag{12}$$

It follows that

$$\mu = \bar{x} + \Sigma\bar{\alpha} \tag{13}$$

6.1.2 Find $\hat{A}$

Take the differential of D with respect to A

$$dD = \frac{2}{N} \operatorname{tr}\left[\left(A^T G + A^{-1}(X - \mu e^T)\right)^T \left(dA^T G - A^{-1} dA A^{-1}(X - \mu e^T)\right) \right] \tag{14}$$

Using the result for μ we get $X - \mu e^T = \tilde{X} - \Sigma\bar{\alpha} e^T$. This, and using cyclic and transpose properties of the trace gives us

$$
\begin{aligned}
dD = &\frac{2}{N}\operatorname{tr}\left[\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}e^T\right)\right)G^T dA\right] \\
&+\frac{2}{N}\operatorname{tr}\left[A^{-1}\left(\Sigma\bar{\alpha}e^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}e^T\right)\right)^T A^{-1} dA\right]
\end{aligned}
\tag{15}
$$

This is zero for all dA iff

$$
\begin{aligned}
0 &= \left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}e^T\right)\right)G^T + A^{-1}\left(\Sigma\bar{\alpha}e^T - \tilde{X}\right)\left(A^T G + A^{-1}\left(\tilde{X} - \Sigma\bar{\alpha}e^T\right)\right)^T A^{-1} \\
&= A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T + \left(A^T\bar{\alpha}e^T - A^{-1}\tilde{X}\right)\left(\tilde{G}^T + \tilde{X}^T\Sigma^{-1}\right),
\end{aligned}
\tag{16}
$$

where $\tilde{X} = X - \bar{x}e^T$, the matrix with centered x_i in the columns, and $\tilde{G} = G - \bar{\alpha}e^T$. Because $e^T\tilde{X}^T = e^T\tilde{G}^T = 0$ we get

$$
A^T\tilde{G}G^T + A^{-1}\tilde{X}G^T - A^{-1}\tilde{X}\tilde{G}^T - A^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} = 0 \tag{17}
$$

Or

$$
\begin{aligned}
&\left(\tilde{G} + \Sigma^{-1}\tilde{X}\right)G^T - \Sigma^{-1}\tilde{X}\tilde{G}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \\
&= \tilde{G}G^T + \Sigma^{-1}\tilde{X}e\bar{\alpha}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1} \\
&= \tilde{G}\tilde{G}^T - \Sigma^{-1}\tilde{X}\tilde{X}^T\Sigma^{-1}
\end{aligned}
\tag{18}
$$