MCMC sampling for latent factor models

David Frich Hansen
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
s144242@student.dtu.dk

Tommy Sonne Alstrm
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
tsal@dtu.dk

Mikkel Nørgaard Schmidt
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
mnsc@dtu.dk

***Abstract*—Markov Chain Monte Carlo sampling methods have provided a significant framework for sampling complex distributions such as posterior distributions in Bayesian modeling.**
In this paper, we show that the No U-Turn sampler augmenting the well-known Hamiltonian Monte Carlo might be useful when extracting underlying spectra in Raman spectroscopy when supplied with certain smoothing priors on the spectra.
We provide an implementation in native Python with the use of Numpy.

I. Introduction

One of the two major paradigms in inference in Bayesian models is sampling with Markov Chain Monte Carlo (MCMC) methods - the idea being that getting enough samples from the posterior distribution will give a good picture of the distribution itself.
The MCMC methods is generally built up around being able to sample a probability distribution where the probability density function (PDF) is known only up to a normalizing constant.
In this project, we develop and implement the MCMC sampler, *No U-Turn Sampler* (NUTS) [1], which is an extension of the famous *Hamiltonian Monte Carlo* (or *Hybrid Monte Carlo*, HMC) [2] that automatically tunes hyperparameters in the sampler. In the following we introduce the theory of these samplers and evaluate the samplers on a latent factor model set in a Bayesian framework - namely the Non-negative matrix factorization with Gaussian Process Priors developed in [3].
We implement the sampler in native Python.
The code for this project is available in my Github repository, https://github.com/davidfrichhansen/NUTS_sampler

A. Previous work

In 2011, Radford M. Neal proposed using Hamiltonian dynamics to get proposals in a Metropolis-like setting giving rise to the Hamilitonian (or hybrid) Monte Carlo algorithm, HMC published in the Handbook of Markov Chain Monte Carlo [4]. We will refer to the relevant section directly with [2].
Finally, Hoffmann and Gelman introduced the No-U-Turn Sampler in 2014 - a proposed automatic tuning of parameters in the HMC algorithm [1].
NUTS uses some ideas from Slice sampling, which is also developed by R. Neal [5].
We test the implemented sampler on the NMF-GPP model developed by M. N. Schmidt and H. Laurberg [3] which is essentially the original Non-negative matrix factorization model by Lee and Seung [6], but in a Bayesian framework.
In this project, we focus on HMC and NUTS.

II. Theory

We discuss the theory of the samplers and apply to a model in this section.

A. Hamiltonian Monte Carlo

The general idea of Hamiltonian Monte Carlo (HMC) is to simulate Hamiltonian dynamics to build a Markov Chain leaving the posterior distribution invariant.

In general, we work with two vectors, *position*, θ and *momentum*, r, where we will interpret position as the current value in the parameter of the posterior distribution and momentum will be sampled from a standard Gaussian distribution.
Hamiltonian dynamics is defined from a function, the *Hamilitonian*, $H(\theta, r)$ and the motion equations that it satisfies,

$$\frac{dr_i}{dm} = -\frac{\partial H}{\partial \theta_i}$$
$$\frac{d\theta_i}{dm} = \frac{\partial H}{\partial r_i}$$

where m denotes the time. Hamiltonian dynamics may be understood as a generalization of the conservation of mechanical energy in a system, as we may define H as,

$$H(\theta, r) = \mathcal{L}(\theta) + K(r) \tag{1}$$

such that $\mathcal{L}$ denotes 'potential energy' and K denotes 'kinetic energy' in the system. Namely, the motion equations ensure that the Hamiltonian is conserved over time, as shown by R.M Neal in [2].
Neal also shows the following, important properties of Hamiltonian dynamics:

1) The mapping T_s that maps (θ_m, r_m) to any other time (θ_{m+s}, r_{m+s}) is reversible, which ensures that the relevant distribution is invariant.
2) It preserves volume under the mapping T_s (T_s has unit Jacobian determinant).

If we choose $\mathcal{L}(\theta) = -\log[p(X|\theta)p(\theta)]$ then one can relatively easily show that the expression,

$$p(\theta, r) \propto \exp[-H(\theta, r)] \propto \exp\left[-\mathcal{L}(\theta) - K(r)\right] \tag{2}$$

is the so-called *canonical distribution* for this form of the Hamiltonian. As H is preserved under these dynamics, the (exact) simulation of those will let θ and r move on a hypersurface of constant probability density wrt. the joint, canonical distribution $p(\theta, r)$.
We assume that θ and r are independent, so choosing $\mathcal{L}$ as the log-probability of θ, we have that the samples of $p(\theta, r)$ can be used as samples of $p(\theta)$ which is the desired posterior.

It is not feasible to simulate the dynamics exactly so we introduce the *Leapfrog integrator* to solve the PDE's numerically. The discrete numerical updates is then as such:

$$r_{m+\varepsilon/2} = r_m - (\varepsilon/2)\nabla_\theta \mathcal{L}(\theta_m) \tag{3a}$$

$$\theta_{m+\varepsilon} = \theta_m + \varepsilon r_{m+\varepsilon/2} \tag{3b}$$

$$r_{m+\varepsilon} = r_{m+\varepsilon/2} - (\varepsilon/2)\nabla_\theta \mathcal{L}(\theta_{m+\varepsilon}) \tag{3c}$$

That is, we update the momentum r by half a step, then update the position θ with this value of the momentum, and then take another half a step of the momentum. Neal argues that this does indeed preserve volume and is indeed reversible.
The method is highly sensitive to the choice of ε and Neal [2] provides examples of how bad this can actually be.
In practice, to simulate the dynamics for some time s we require s/ε iterations of the Leapfrog integrator.

The HMC sampling algorithm may now proceed as follows, as described by Neal [2]. First we draw a new momentum variable from the correct distribution. If we choose a quadratic energy as proposed here, the components of the momentum vector r is iid. standard Gaussians, ie. $r \sim \mathcal{N}(0, I)$. This update leaves the canonical joint distribution invariant, as r is drawn from the correct conditional distribution given θ.
Then, we use the Hamiltonian dynamics (simulate with the Leapfrog integrator) to propose a new state and accept using a Metropolis acceptance based on the change of the Hamiltonian (ie. mechanical energy) of the two states.
The way the Hamilitonian was introduced, we stated that it was invariant wrt. the Hamiltonian dynamics. Had we not resampled the momentum r in the first step, we would simply stay on a hypersurface of constant joint probability, $p(\theta, r)$. This resampling ensures that we can move, as it may be the case that $p(\theta, r) \neq p(\theta, r^*)$.

Now, we pick a value for ε and a natural number L (similar to s in the description of the Leapfrog integrator) for the Leapfrog integrator, and take L Leapfrog steps of length ε, moving a distance of $L\varepsilon$ through the space of the joint distribution $p(\theta, r)$.
This provides us with a proposed set of position-momentum values, eg. $(\theta, r) \rightarrow (\theta^*, r^*)$ where r is the resampled momentum.
We then accept (θ^*, r^*) as the next state of the Markov chain based on the Metropolis acceptance rate,

$$\alpha = \min\left\{1, \exp(-H(\theta^*, r^*) + H(\theta, r))\right\} \tag{4}$$

For the proposal to be symmetric (we require this for simplicity such that the Hastings correction is not necessary) we technically need to negate the momentum at the end of the simulation, but with the quadratic kinetic energy, we have that $K(r) = K(-r)$ so this is not necessary in practice.

Left is to show that this procedure indeed does leave the canonical joint distribution invariant, such that we sample from the right distribution. We do this in appendix A

We now address the problem of choosing the number of steps, L and the step length, ε such that convergence is achieved quickly. Furthermore we eliminate the need to hand tune these parameters in the algorithm. This is exactly what NUTS [1] attempts to do.

B. The No-U-Turn Sampler

The *No-U-Turn sampler* (NUTS) is an extension of the HMC framework developed by M. D. Hoffmann and A. Gelman [1] to eliminate the need for handtuning L and ε in HMC.
We change up the notation slightly here. We change the Hamiltonian such that $H(\theta, r) = \mathcal{L}(\theta) - K(r)$ and as such the sign of the step in (3a) and (3c) also changes. Furthermore the acceptance probability in (4) changes such that it becomes,

$$\alpha = \min\left\{1, \exp[H(\theta^*, r^*) - H(\theta, r)]\right\}. \tag{5}$$

Thus, in the notation of [1], we have that the posterior is proportional to $\mathcal{L}$. The canonical joint distribution becomes

$$p(\theta, r) \propto \exp[\mathcal{L}(\theta) - \tfrac{1}{2} r^T r]. \tag{6}$$

We start by looking at how to select the number of steps, L. Ideally, the HMC trajectory (ie. the simulated path) should end up as far as possible from the beginning position as possible to explore as much of the state space as possible and avoid random walk behaviour. Thus, a reasonable heuristic would be to stop taking leapfrog steps once the distance, $\|\theta^* - \theta\|$ no longer increases. Doing this crudely would violate the time reversibility of the algorithm and thus the canonical distribution would no longer (guaranteed) be invariant.
This heuristic, however, gives a good stopping criterion, as we have that the derivative of the half squared distance is exactly $(\theta^* - \theta) r^*$ which is easily computable. That is, we stop when simulating an infinitesemal amount of time on, would result in a shorter distance if the derivative becomes negative - that is, the trajectory starts to make a 'U-turn', hence the name of the algorithm.

As in [1], we will initially derive a simple NUTS and then revise that to a more memory efficient NUTS.

1) Simple NUTS: We start by introducing a 'slice variable', u similar to the method in [5], with a uniform conditional distribution - that is,

$$u|\theta, r \sim \mathcal{U}(0, \exp[\mathcal{L}(\theta) - \tfrac{1}{2} r^T r]),$$

such that the 'reverse conditional', $p(\theta, r|u)$ is also uniform as long as $\exp[\mathcal{L}(\theta) - \frac{1}{2}r^T r] \geq u$.
This gives a joint probability over θ, r and u,

$$p(\theta, r, u) \propto \mathbb{I}(u \in [0, 0, \exp(\mathcal{L}(\theta) - \tfrac{1}{2}r^T r)]), \tag{7}$$

and integrating over u (ie. computing the marginal distribution) we get the canonical joint, (6).

The general idea is now that we resample $u|\theta, r$ and trace out Leapfrog steps forward or backwards (chosen randomly) - doubling the amount of steps until the trajectory makes a 'U-turn' - that the aforementioned stopping criteria is violated. That is, we take 1 Leapfrog step either forwards or backwards (in time) and then 2 steps forwards or backwards and then 4 etc.

Abstractly this builds a balanced binary tree where the rightmost node from the top corresponds to the latest doubling of leapfrog steps.
We now introduce a set, $\mathcal{B}$ consisting of all the position-momentum pairs traced out during this doubling process. We also introduce $\mathcal{C} \subseteq \mathcal{B}$ consisting of all position-momentum pairs that are candidates for the next sample of $p(\theta, r)$ - that is, the subset of $\mathcal{B}$ that does not violate detailed balance, see (13) in appendix A.
So if we are able to build $\mathcal{C}$ deterministically from $\mathcal{B}$ we are able to randomly select a position-momentum pair from $\mathcal{C}$ as a candidate for the Metropolis update with acceptance probability (5).

So how do we construct $\mathcal{B}$ and $\mathcal{C}$? We define a distribution, $p(\mathcal{B}, \mathcal{C}|\theta, r, u)$ that satisifies the following properties,

1) Any transformation mapping (θ, r) to (θ^*, r^*) such that $(\theta^*, r^*) \in \mathcal{C}$ must preserve volume.
2) The current state (ie. from where we start the doubling) must be in $\mathcal{C}$.
3) All $(\theta, r) \in \mathcal{C}$ must be on the slice defined by u.
4) For $z, z^* \in \mathcal{C}$, then $p(\mathcal{B}, \mathcal{C}|z, u) = p(\mathcal{B}, \mathcal{C}|z^*, u)$ for any $\mathcal{B}$ This ensures reversibility.

Hoffmann and Gelman argue that if we are able to construct such a distribution, we are able to create a procedure that samples position and momentum values from their correct distribution.
In practice, we create a procedure that leaves the joint distribution $p(\theta, r, u, \mathcal{B}, \mathcal{C})$ invariant. This procedure is outlined and discussed in Appendix B.
This requires us to construct a transition kernel T such that the invariant distribution, $p(\theta, r|\mathcal{B}, \mathcal{C}, u)$ is uniform over the elements of $\mathcal{C}$. So left is to design the distribution, $p(\mathcal{B}, \mathcal{C}|\theta, r, u)$ and the transition kernel T.

We construct $\mathcal{B}$ by the doubling process described in the beginning of this section. We let j denote the height of a balanced tree whose leaf nodes consist of $\mathcal{B}$ - ie. the trajectory, initialized as $j = 0$ for the initial state, (θ_0, r_0). The idea is then that the leaf nodes of the balanced tree should consist of the simulated trajectory where the leftmost node corresponds to the position-momentum pair furthest back in time and the rightmost node corresponds to the pair furthest forward in time.
We then increase j by one and sample $v_j \sim$ Discrete uniform$(\{-1, 1\})$ and take 2^j Leapfrog steps of size $v_j\varepsilon$ such that $v_j = 1$ correspond to forward in time and $v_j = -1$ correspond to backwards in time.
This process doubles the amount of Leapfrog steps (L in HMC) every time j increases and thus doubles the length of the total trajectory in this iteration. As the momentum is not resampled within a single iteration but simply simulated with the Leapfrog integrator, the only random part of this procedure is the direction v_j creating 2^j possible trees for each j given (θ_0, r_0) and a step length ε all with the same probability.
So when should we stop this doubling procedure? As we want to explore as much of the state space as possible during a single iteration, we will allow the doubling to continue until the trajectory curls back on itself, thus making a 'U-turn' in the state space. Let (θ^+, r^+) denote the rightmost leaf node of the binary tree (ie. the pair 'furthest ahead' in time) and let (θ^-, r^-) be the leftmost leaf node (ie. the pair 'furhtest back' in time) for any j in the iteration. Then, as mentioned in the beginning of this section, if

$$(\theta^+ - \theta^-)^T r^- < 0 \tag{8}$$

holds, then moving infinitesemally further backwards in time would result in the trajectory moving closer to initial point. Similarly, we have that if

$$(\theta^+ - \theta^-)^T r^+ < 0, \tag{9}$$

we are in the same situation, only forward in time. Another stopping criteria is if,

$$\mathcal{L}(\theta) - \frac{1}{2}r^T r - \log u < -\Delta_{\text{max}} \tag{10}$$

is satisfied. If we set Δ_{max} is large enough (on the log-scale) and the stopping criterion above is satisfied, then the resulting position-momentum pair is likely to have very low probability, and thus we stop the simulation.
With these stopping criteria there is a chance that some of the intermediate position values in a doubling curls back on itself, but ends up going further. This is not too bad, however, as the idea is that the entire trajectory should explore as much of the state space as possible. In practice, the binary tree is not explicitly built - rather it is built by a recursion that takes the initial state and j as well as the direction v_j and takes 2^j Leapfrog steps of length $v_j\varepsilon$ until either of the stopping criteria is met. Details can be found in alogrithm 2 in [1].

All the above defines a distribution on $\mathcal{B}$ given θ, r, u - but we still need to describe how to select $\mathcal{C}$ deterministically from $\mathcal{B}$.
Putting the conditions 1)-4) on the distribution $p(\mathcal{B}, \mathcal{C}|\theta, r, u)$ ensures that the distribution $p(\theta, r|u, \mathcal{B}, \mathcal{C})$ is uniform on the elements of $\mathcal{C}$ as outlined in Appendix B.

The procedure to choose the elements from $\mathcal{B}$ is laid out in Appendix C. This means that we can simply pick T as picking a new sample of (θ, r) uniformly from $\mathcal{C}$ after this has been built.

The method is thus as such:

1) Sample a slice variable u s.t $u|\theta, r \sim \mathcal{U}(0, \exp[\mathcal{L}(\theta) - \frac{1}{2}r^T r])$
2) Simulate Hamiltonian dynamics with the Leapfrog integrator as in HMC and do the doubling process described above, yielding a set of position-momentum pairs $\mathcal{B}$.
3) Select candidate samples deterministically from $\mathcal{B}$, thus building $\mathcal{C}$ by the process described above.
4) Pick the next sample of (θ, r) uniformly from $\mathcal{C}$.
5) Repeat the process until the desired amount of samples have been computed.

This entire algorithm leaves the canonical joint distribution, (6), invariant and thus allows us to sample new values from the correct distribution, while eliminating the need to hand tune the number of Leapfrog steps to take, L, but requires that we store all visited states during a given trajectory - that is, we have to store $\mathcal{O}(2^j)$ position-momentum vectors during any iteration of NUTS, which may be unacceptable depending on application.
The reason for this is the naive transition kernel - simply sampling uniformly from $\mathcal{C}$. But any transition kernel leaving the uniform distribution $p(\theta, r|u, \mathcal{B}, \mathcal{C})$ invariant. A proposal to solve this is denoted *efficient NUTS* by Hoffmann and Gelman [1].

2) Efficient NUTS: We will not discuss this as much in depth as the simple NUTS algorithm, as the basic principles are much the same.
Efficient NUTS involves defining a more sophisticated transition kernel for sampling (θ, r) from $\mathcal{C}$ while still leaving the uniform distribution invariant. How to choose this more efficient kernel is shown in Appendix D. The details and arguments that this more efficient kernel does indeed leave the uniform distribution on the elements of $\mathcal{C}$ invariant is found in Hoffmann's and Gelman's original NUTS paper, [1].

3) Tuning the step length, ε: The last part to discuss is how to choose the step length ε. Until now, we have only discussed how many steps to take, but we still need to address how long the steps should be.
Hoffmann and Gelman suggest using *Dual Averaging* - a stochastic optimization algorithm, that can be used to tune any hyperparameter in any MCMC algorithm as long as the following is satisfied:

1) There exist an easily computeable statistic H_m that measures some statistic of the MCMC algorithm (eg. acceptance rate) as a function of the parameter we want to tune, say ε.
2) Its expectation $h = \mathbb{E}(H_m)$ is non-decreasing and may be considered a function of ε.

Then by the Robbins-Monro algorithm, we can adapt ε with the update scheme,

$$\log \varepsilon_{m+1} = \log \varepsilon_m - \gamma_m H_m \tag{11}$$

as long as γ_m satisfies,

$$\sum_{i=1}^{\infty} \gamma_m = \infty$$
$$\sum_{i=1}^{\infty} \gamma_m^2 < \infty$$

then $H_m \to 0$ guaranteed. So we can adapt this algorithm if we can choose H_m to some relevant statistic in NUTS that measures how well the sampler mixes as a function of ε. In practice, we do this adaptation for a fixed amount of iterations, say M_{adapt} and then fix it at the final value. Then the initial M_{adapt} iterations will serve as burn in for the sampler.

Hoffmann and Gelman takes this further, however. They argue that the Robbins-Monro scheme puts to much weight on early iterations, but this should be the other way around as the adaptation should reflect the ideal value when the sampler has already reached an equilibrium (ie. stationarity).
So they suggest combining the Robbins-Monro scheme with the Dual Averaging framework developed by Y. Nesterov [7]. They suggest using a different update instead, namely

$$\log \varepsilon_{m+1} = \mu - \frac{\sqrt{m}}{\tau} \frac{1}{m + m_0} \sum_{i=1}^{m} H_i$$
$$\log \overline{\varepsilon}_{m+1} = \gamma_m \log \varepsilon_{m+1} + (1 - \gamma_m) \log \overline{\varepsilon}_m$$

where μ is a value that ε is shrunk towards, τ is a parameter that controls the rate with which the shrinkage occurs and m_0 ensures stability for small values of m.
Hoffmann and Gelman suggest setting $\gamma_m = m^{-\kappa}$ for some $\kappa \in (0.5, 1]$ as this ensures convergence.
It is to be understood as such: The first equation adapts ε by the expectation of the statistic H_m with a step size of $\frac{\sqrt{m}}{\tau}$. Then we update $\overline{\varepsilon}$ by a weighted average of the previous value of $\overline{\varepsilon}$ and the adapted values of ε. As $\gamma_m \to 0$ the adaptation weighs in less as $m \to \infty$ which ensures convergence of $\overline{\varepsilon}$.
Left is to choose the statistic H_m for NUTS. Hoffmann and Gelman use the following:

$$\widetilde{H}_m = \frac{1}{|\mathcal{B}_m^{\text{final}}|} \sum_{z \in \mathcal{B}_m^{\text{final}}} \min\left[1, \frac{p(z)}{p(z_0)}\right] \tag{12}$$

where $z_0 = (\theta_m^0, r_m^0)$ is the initial position momentum pair in this iteration of NUTS - ie. the point from which the doubling process started. This can be considered a sort of 'average acceptance rate' over the last doubling process of $\mathcal{B}$ - notice the similarity to the Metropolis acceptance rate, (5). Finally, we choose $H_m = \delta - \widetilde{H}_m$ where δ is the desired average acceptance rate.
Finally we have to discuss how to choose a reasonable initial value of ε from which we can start the adaptation. This is

done in Appendix E.

That concludes the description of the NUTS algorithm. In the following, we will test it on the Non-negative Matrix Factorization with Gaussian Process Priors as introduced by [3].

III. Experimental evaluation

We evaluate the NUTS sampler on the Non-negative matrix factorization with Gaussian process priors introduced by M. N. Schmidt and H. Laurberg in [3]. Details of this model can be found in Appendix F.

We work on simulated Raman spectroscopy data. Raman spectroscopy builds on the (rare) physical phenomenon Raman scattering, in which molecules emit light of different wavelengths when excited by a laser of a specific energy. The resulting spectra can be used to identify the molecule as the spectra depend on molecular structure.

In practice, a plate is prepared in ways that increase the occurence of Raman spectroscopy tremendously before it is covered in a given molecule. This results in 'hotspots' on the where Raman spectroscopy is likely to occur, when measurements is made. All other positions on the plate consists largely of background spectra that can be due to residual molecules, inaccurate measurements etc.

In all experiments below, we simulate Raman spectroscopy where the plate is of size 25×25 and 50 wavenumbers are measured. This gives a data matrix of size 625×50 where each row is a measurement of on the Raman plate and the row consits of the measured spectrum.

In all experiments below, we apply the NMF-GPP model. We let M be equal to the number of spectra (known, as the data is simulated) and as such the matrix H will consist of the constituent spectra and the matrix D will consist on the loading of each spectra - ie. where on the plate is the molecule present. In each experiment, we apply a 1D squared expnential kernel (ie. covariance) on the indeces, on H ie.

$$k(x_i, x_j) = \exp\left[-\frac{(i-j)^2}{\beta^2}\right]$$

and a 2D exponential kernel on the Euclidean distances based on the indeces, on D, ie.

$$k(x_i, x_j) = \exp\left[-\frac{\|x_i - x_j\|}{\beta^2}\right]$$

such that β is the length-scale in both cases, see [8]. The size of β determines the 'smoothness' of the prior. These kernels are chosen from prior experience.[1]. We try the experiment twice. Once where we let $\beta = 10$ on both kernels (denoted 'unsmooth prior') and one where we let $\beta = 20$ for a really smooth prior on both kernels (denoted 'smooth prior').

Furthermore, we apply direct optimization to the log posterior to achieve a MAP estimate as suggested by [3] and start the sampling in the MAP estimate. All experiments consist of 1500 samples with $M_{\text{adapt}} = 1500$. In all experments, it was verified that the dual averaging for the stepsize had indeed converged before the samples were drawn.

A. Experiment 1

In this experiment, we set only 1 constituent spectrum with 2 hotspots. We set $M = 2$ - one for the spectrum and one for the background.

1) Unsmooth prior: Figure 1 shows the MAP estimate of one of the components in H and the true spectrum. Herew we see that it does indeed seem like we find a good estimate - the peak is at the same wavenumber approximately. We also note the difference in amplitude. This doesn't mean much, as this is evened out by the corresponding component in D.

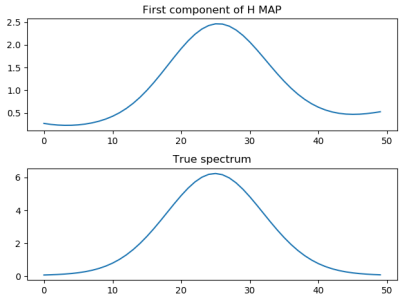

Fig. 1. True spectrum and MAP estimate, unsmooth prior

The MAP estimates of D can be seen in Figure 2, and also here we see that it does indeed seem to find a good estimate, although it has trouble separating the two hotspots. It should be noted, that the two colormaps are not on the same scale.

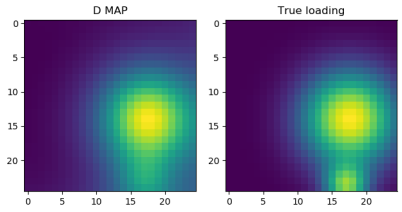

Fig. 2. True loading and MAP estimate reshaped to the shape of the Raman plate, unsmooth prior

So the model does indeed find some good estimates. We now look at some samples, Figure 3. Here we see the MAP estimate and 4 samples evenly spaced between the last 100 samples. We see that although it has some trouble it does generally seem to be alright. It is problematic, however that the shapes of the samples of the first component (left) show some 'pseudopeaks' which may lead to errors in interpretation. Generally there also seem to be some inaccuracies on the second component (corresponding to background).

[1]These kernels gave the best results in the experiments done in 02460 Advanced Machine Learning, spring 2018

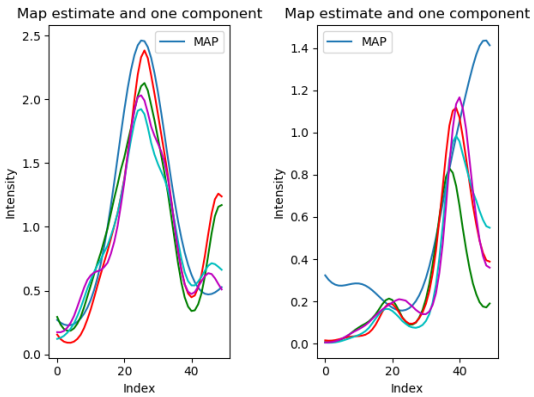

Fig. 3. MAP estimate and 4 samples of H, unsmooth prior

We also look at some samples of D on Figure 4. These show that the samples catches the overall structure, but with some inaccuracies - these samples seem unable to distinguish the two true hotspots - they are close to eachother, so the 2D smoothing might not work in the samplers favor here.

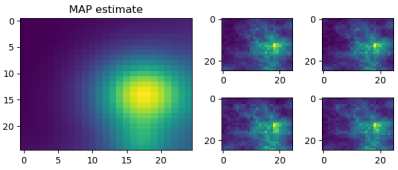

Fig. 4. Map of first component of D and samples, unsmooth prior

2) Smooth prior: We show the corresponding figures here. Figure 5 shows that the MAP estimate is still good for H, although the curve seems wider due to the stronger covariance on the GP prior.

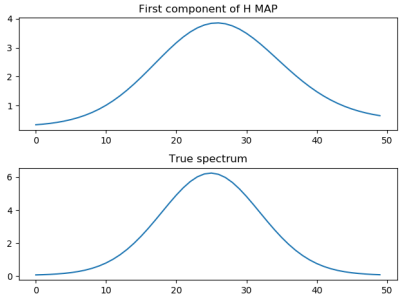

Fig. 5. True spectrum and MAP estimate, smooth prior

On Figure 6 we really see the effect of the stronger smoothing - the two hotspots are not almost virtually one big hotspot instead.

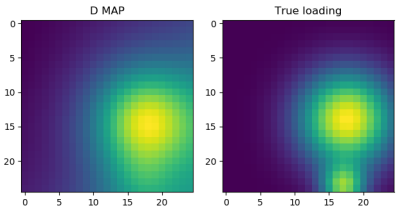

Fig. 6. True spectrum and MAP estimate, smooth prior

We look at some samples like before, see Figure 7 and Figure 8. Here we see that the spectrum is caught much better than before, but this may be due to the harder smoothing - the prior is very strong in this case. The loadings however (Figure 8) are not very good - perhaps due to the strong smoothing - so we might ask ourselves, how much of this was actually controlled by the data.

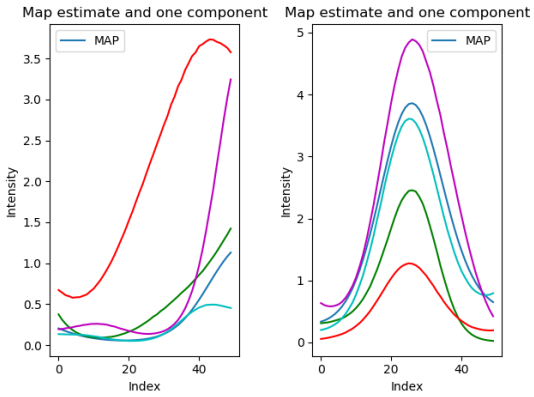

Fig. 7. MAP estimate and 4 samples of H, smooth prior

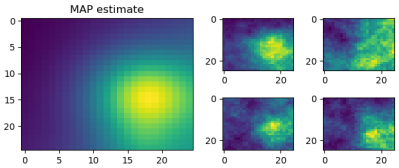

Fig. 8. Map of first component of D and samples, smooth prior

B. Experiment 2

This experiment is almost identical to the one above, but the true spectrum is now made up of 2 Voigt curves (theoretical shape of the peaks), almost with the same peak. This is not uncommon in a real life setting, so we try our sampler here.

Choosing both $M = 2$ and $M = 3$ here was tried, but for $M = 3$ the model failed to recognize the two distinct peaks - but with $M = 2$ worked quite well. We show the results for $M = 2$ below.

1) Unsmooth prior: We plot the exact same things as in experiment 1.

We see on Figure 9 and Figure 10 that in the unsmooth case, we are indeed able to find something that looks like the true spectrum and true loading. Especially on the MAP estimate of H it seems that the peak at wavenumber app. 15 is almost fully recovered.

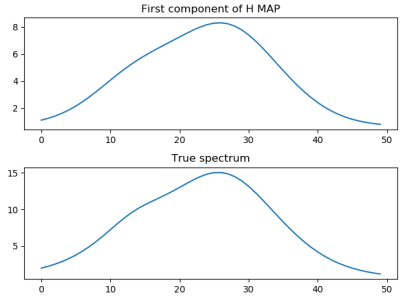

Fig. 9. True spectrum and MAP estimate, unsmooth prior

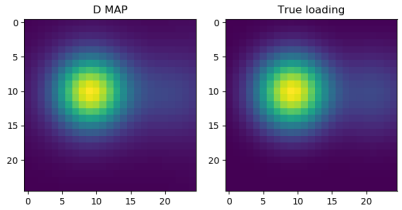

Fig. 10. True loading and MAP estimate reshaped to the shape of the Raman plate, unsmooth prior

We look at some samples, and see on Figure 11 that the samples are actually quite good at recovering the two distinct peaks, which is quite good. Unfortunately, however, it seems that it mixes some energy into the second component from the 'weak' peak. This could maybe be fixed by additional regularization.

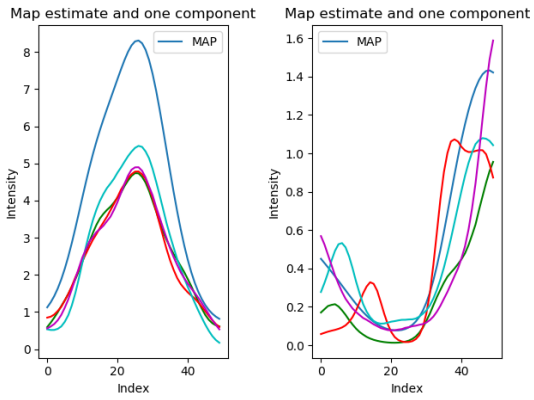

Fig. 11. MAP estimate and 4 samples of H, unsmooth prior

Some samples of one component of D is seen on Figure 12. These are not much different from before, as we expect - the only difference here is that the shape of the true spectrum has changed, so we expect no difference in D in essence (ie. the placement is different, but structurewise it's the same).

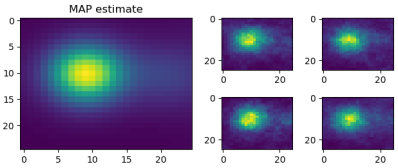

Fig. 12. Map of first component of D and samples, unsmooth prior

2) Smooth prior: We once more show the MAP estimates alongside the true spectra and loadings and then we show examples of samples.

Figure 13 shows that the MAP estimate is not able to recover the second peak at wavenumber app. 15. This is almost certainly due to the strong prior.

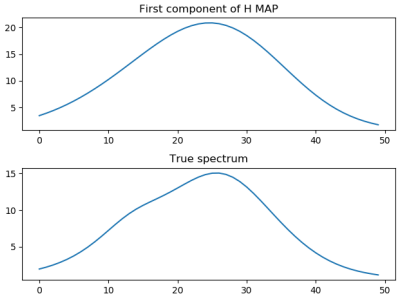

Fig. 13. True spectrum and MAP estimate, smooth prior

On Figure 14 we see that we are able recover one hotspot pretty well. There is a second hotspot there (just right of the

strong one), but it is much weaker than the first one, which is why it is not visible on this plot.

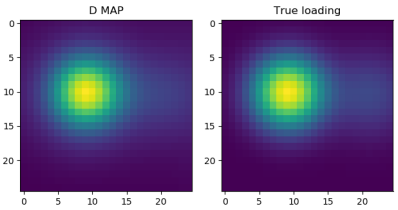

Fig. 14. True spectrum and MAP estimate, smooth prior

Figure 15 shows that the strong smoothing of the samples completely removes the peak at wavenumber app. 15 - but otherwise recovers the dominant peak.

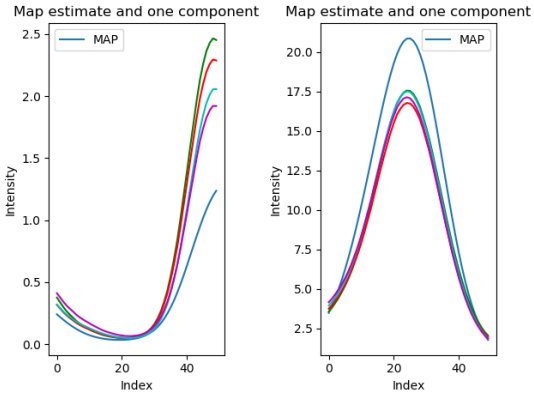

Fig. 15. MAP estimate and 4 samples of H, smooth prior

Figure 16 is not much different from experiment one in essence. This is to be expected, as the only thing that has been changed is the spectra which should be found in H.

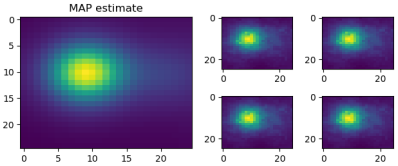

Fig. 16. Map of second component of D and samples, smooth prior

IV. Discussion

We notice a few things in our experiments.
The sampler seems sensitive to the length-scale β of the covariances - ie. the how smooth the Gaussian procceses should be. This means that we should tune this parameter carefully. Furhtermore, we have seen that too much smoothing eliminates details in the spectra, which may distort information.
Overall, a strategy could be to learn the covariances from the data - but this would probably require large amounts of it.

It is possible that during the exploration of the state space, the components switch around - and I have seen examples hereof. This can happen as the model is invariant wrt. the order of components. This makes it difficult to say anything in general about the posterior distribution - one of the major reasons to use MCMC sampling in the first place - so one should take care to look out for this. It may be possible to regularize the components so it is unlikely to happen, but this may be to hard of a prior to put on the matrix factors. This would also require knowledge of the constituent spectra prior to inference - we would have to assign one or more components to 'spectra' components, but how many?
That opens up a totally new discussion. How to choose the model order, M? It could be learned from the data but would require extensive computations and once more, prior knowledge, which may not be unrealistic depending on application.

The easiest way to solve the problem with sensitivity to the length-scale, β could be to include this in the sampler, perhaps as a Metropolis step and thus infer this from the data.

Finally, we see that what the sampler does is not the same as the direct optimization of the log-posterior. The direct optimization puts much more weight on the priors, as this may be done without decreasing the log-probability very much. The sampler finds a tradeoff between the likelihood and the priors and stays within this region - practically all samples have the same probability under the model, but have lower probability than the MAP estimate. This gives an indication, that there might be a quite narrow 'peak' of probability mass around the MAP estimate, that the sampler doesn't reach easily. As such, the MAP estimate may not be bad as a point estimate in this model.

V. Conclusion

The goal of this project was to describe and implement the NUTS algorithm for sampling the posterior distribution in the NMF-GPP model. This has been done, and we have discussed the results. To summarize, we see that when sampling, the model is very sensitive to the choice of covariance - especially the smoothness of the covariance. This sensitivity could be made less influent in future work.
We described the theory behind NUTS and HMC and was able to get meaningful samples for interpretation, but was unable to say anything in general about the posterior due to lack of regularization, such that the components of the NMF-GPP factors could switch around, making general comments impossible.

Overall, we conclude that there is indeed a potential in using MCMC for NMF-GPP, but it requires some work to be truly useful.

REFERENCES

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

[2] Radford M. Neal. Mcmc using hamiltonian dynamics. 2012.

[3] Mikkel Nø rgaard Schmidt and Hans Laurberg. Non-negative matrix factorization with gaussian process priors. *Computational Intelligence and Neuroscience*, 2008:361705, 2008.

[4] *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC, 2011.

[5] RM Neal. Slice sampling. *Annals of Statistics*, 31(3):705–741, 2003.

[6] DD Lee and HS Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[7] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1):221–259, 2009.

[8] Carl Edward. Rasmussen and Christopher K.I. Williams. *Gaussian processes for machine learning*. MIT,, 2006.

[9] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer,, 2006.

APPENDIX A
HMC LEAVES CANONINCAL JOINT DISTRIBUTION INVARIANT

As noted by C. Bishop [9, p. 540] a sufficient criterion for the canonical joint distributon to be invariant is if the Markov chain satisfies *detailed balanced* wrt. the joint distribution. This is defined as such. Let $x = (\theta, r)$ and $x' = (\theta', r')$ be different states. Then detailed balance is satisfied if,

$$p(x)T(x|x') = p(x')T(x'|x)$$

where $T(z|y)$ is the probability of transitioning to state z from state y. So we have to show that the Metropolis step with proposal from simulated Hamiltonian dynamics satisfies detailed balance wrt. the canoncial distribution. Neal [2] does this as such:
Divide the (θ, r)-space into different regions A_k with equal (but small) volume V. Let B_k be the to A_k associated region through L Leapfrog integrator steps (ie. B_k is the image of A_k under the 'Leapfrog mapping', T). Then, B_k also has volume V as the Leapfrog integrator preserves volume and partitions the space as well.
Then, detailed balance is satisfied if for all i and j:

$$p(A_i)T(B_j|A_i) = p(B_j)T(A_i|B_j), \tag{13}$$

which is equivalent to 'pointwise detailed balance' as the Leapfrog mapping is deterministic.
By definition of the regions B_k we have that $T(A_i|B_j) = T(B_j|A_i) = 0$ for all $i \neq j$ so (13) is obviously satisfied in this case.
For the case of i and j is equal to some common value, say k, Neal argues that the Hamiltonian is continuous almost everywhere, so if the volume V is small enough, $H(\theta, r)$ effectively becomes constant within the regions.
We define $H(A_k)$ as the constant value of $H(\theta, r)$ for $(\theta, r) \in A_k$. Then using (13) and (4) we can write the detailed balance criterion as (up to a normalizing constant),

$$\begin{aligned}\exp(-H(A_k))\min\{1, \exp(-H(B_k) + H(A_k))\} = \\ \exp(-H(B_k))\min\{1, \exp(-H(A_k) + H(B_k))\}\end{aligned} \tag{14}$$

Now three posibilities exist: $H(A_k) = H(B_k)$, $H(A_k) < H(B_k)$ or $H(A_k) > H(B_k)$. The last two are analogous by the symmetry of (14).
If $H(A_k) = H(B_k)$, (14) simplifies to

$$\exp(-H(A_k)) = \exp(-H(B_k))$$

which is obviously true.
If $H(A_k) < H(B_k)$, the left hand side of (14) simplifies to $\exp(-H(B_k))$ as $\exp(-H(B_k) + H(A_k)) < 1$. The right hand side is also $\exp(-H(B_k))$ as $\exp(-H(A_k) + H(B_k)) > 1$, rendering (14) true.
The case of $H(B_k) < H(A_k)$ is very similar due to symmetry of detailed balance.

Thus detailed balance is satisfied for this scheme. The proof that detailed balance does indeed leave the distribution invariant is found in [2], [9].

Thus we have shown that simulating Hamiltonian dynamics, where we construct the dynamics in a special way does indeed allow us to sample any probability distribution for which the gradient of the log-probability can be found.
Neal argues that HMC has better convergence properties than the random-walk behaviour often exhibited by other Metropolis-like algorithms (such as Metropolis-Hastings).

APPENDIX B
PROCEDURE TO SAMPLE $p(\theta, r, u, \mathcal{B}, \mathcal{C})$ IN SIMPLE NUTS

The procedure is in itself simple enough and is originally described by Hoffmann and Gelman, [1] - we start by sampling r,u,$\mathcal{B}$ and $\mathcal{C}$ from their correct conditional distributions, thus making a Gibbs sweep of these variables, ie.

1) Sample $r \sim \mathcal{N}(0, I)$, as described in the HMC section.
2) Sample $u \sim \mathcal{U}(0, \exp[\mathcal{L}(\theta_m) - \frac{1}{2}r^T r])$ where θ_m denotes the m'th step - ie. the current value of θ.
3) Sample $\mathcal{B}$ and $\mathcal{C}$ from their conditional as described in subsubsection II-B1.

These are all sampled from their correct conditionals and thus the three steps constitute a Gibbs step.
Finally we sample,

$$\theta_{m+1}, r \sim T(\theta_m, r, \mathcal{C})$$

where T is a transition kernel. T could in principle be any kernel that leaves a uniform distribution over $\mathcal{C}$ invariant wrt. the states in $\mathcal{C}$. That is,

$$\frac{1}{|\mathcal{C}|}\sum_{(\theta,r)\in\mathcal{C}} T(\theta^*, r^*|\theta, r, \mathcal{C}) = \frac{1}{|\mathcal{C}|}\mathbb{I}((\theta^*, r^*) \in \mathcal{C}),$$

that is, the distribution of position-momentum pairs in $\mathcal{C}$ is uniform over $\mathcal{C}$ with the relevant transition kernel, T.
We do this, because this procedure allows us to sample $p(\theta, r|u, \mathcal{B}, \mathcal{C})$ such that we can use the sample - because putting exactly the conditions 1)-4) in subsubsection II-B1 on $p(\mathcal{B}, \mathcal{C}|r, u, \theta)$ ensures that the conditional distribution $p(\theta, r|\mathcal{B}, \mathcal{C}, u)$ uniform over the elements of $\mathcal{C}$.
This is because we have,

$$\begin{aligned}p(\theta, r|u, \mathcal{B}, \mathcal{C}) &\propto p(\mathcal{B}, \mathcal{C}|\theta, r, u)p(\theta, r|u) \\ &\propto p(\mathcal{B}, \mathcal{C}|\theta, r, u)\mathbb{I}(u \leq \exp[\mathcal{L}(\theta) - \tfrac{1}{2}r^T r]) \\ &\propto \mathbb{I}((\theta, r) \in \mathcal{C})\end{aligned} \tag{15}$$

The following description refers to the points 1)-4) in subsubsection II-B1.
The first equation is due to the chain rule of probability. The second is due to 1) - the adding of (θ, r) to $\mathcal{C}$ must preserve volume so we can treat $p(\theta, r|u)$ as an unnormalized probability and the conditional uniform distribution, $p(\theta, r|u)$. From 4), we have that given, θ and r in $\mathcal{C}$, the distribution $p(\mathcal{B}, \mathcal{C}|\theta, r, u)$ is constant - otherwise the identity wouldn't

hold for all pairs of position-momentum values in $\mathcal{C}$. This gives that,

$$p(\mathcal{B}, \mathcal{C}|\theta, r, u) \propto \mathbb{I}((\theta, r) \in \mathcal{C})$$

From 3), we have that if $(\theta, r) \in \mathcal{C}$ then (θ, r) lies on the slice defined by u, thus $\mathbb{I}(u \leq \exp[\mathcal{L}(\theta) - \frac{1}{2}r^T r]) = 1$ for all $(\theta, r) \in \mathcal{C}$, which is required by 2).
Thus we have argued that the procedure described results in a conditional distribution $p(\theta, r|u, \mathcal{B}, \mathcal{C})$ that is uniform over the elements of $\mathcal{C}$. Thus we can sample θ, r from *any* transition kernel T that leaves this distribution invariant.

Appendix C
Choosing $\mathcal{C}$ from $\mathcal{B}$ so conditions 1)-4) is satisfied in NUTS

We describe a procedure for choosing $\mathcal{C}$ from $\mathcal{B}$ such that conditions 1)-4) are satisfied.
1) is satisfied for all elements since we use the Leapfrog integrator. 2) is satisfied if we choose $(\theta_0, r_0) \in \mathcal{C}$. 3) is satisfied by discarding any pairs from $\mathcal{C}$ not on the slice defined by u. Left is 4) which is described below.

Recall condition 4),

$$p(\mathcal{B}, \mathcal{C}|\theta, r, u) = p(\mathcal{B}, \mathcal{C}|\theta^*, r^*, u) \tag{16}$$

for any $(\theta^*, r^*) \in \mathcal{C}$ - that is, the binary tree must have equal probability of being built from any of the leaf nodes also in $\mathcal{C}$. As discussed in the main section, the only random part of the doubling procedure is choosing the direction v_j, which means that for any given element in $\mathcal{C}$, there is a probability of 2^{-j} of building any $\mathcal{B}$.
Ensuring this obviously makes the condition hold. So how can we ensure this?
We have to disregard any elements in $\mathcal{C}$ that could not have resulted in $\mathcal{B}$ which is only the case if one of the stopping criteria (8),(9) or (10) are satisfied before a doubling has finished. In this case, the resulting binary tree is no longer balanced, and thus there is no longer a 2^{-j} probability of reconstructing the tree from any of the leaf nodes. Thus, to ensure this property we disregard *all* leaf nodes added in the last doubling. This means that condition 4) is satisfied.

Appendix D
Efficient NUTS

As discussed in [1], there are some inefficiencies in the NUTS algorithm laid out in this report.
One of the main issues is the transition kernel requiring storage of $\mathcal{O}(2^j)$ position-momentum pairs which may be unacceptable. Thus, they suggest an alternative transition kernel.
Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be disjoint partitions of $\mathcal{C}$ such that $\mathcal{C}_1 \cup \mathcal{C}_2 = \mathcal{C}$. Let $(\theta, r), (\theta^*, r^*) \in \mathcal{C}_1$.

$$T((\theta^*, r^*)|(\theta, r), \mathcal{C}) = \begin{cases} p_1 & \text{if } |\mathcal{C}_2| > |\mathcal{C}_1| \\ p_2 & \text{otherwise} \end{cases} \tag{17}$$

where

$$p_1 = \frac{\mathbb{I}((\theta^*, r^*) \in \mathcal{C}_2)}{|\mathcal{C}_2|}$$

and

$$p_2 = \frac{|\mathcal{C}_2|}{|\mathcal{C}_1|} \frac{\mathbb{I}((\theta^*, r^*)) \in \mathcal{C}_2}{|\mathcal{C}_2|} + (1 - \frac{|\mathcal{C}_2|}{|\mathcal{C}_1|} \mathbb{I}((\theta^*, r^*) = (\theta, r)))$$

That is, the kernel proposes a move from $\mathcal{C}_1$ to $\mathcal{C}_2$ with probability proportional to the ratio of sizes between the two subsets. We choose $\mathcal{C}_2$ to be the elements added to $\mathcal{C}$ in the latest doubling of the binary tree. Then we can apply T after every doubling of the binary tree instead of sampling uniformly over the final version of $\mathcal{C}$ - that is, after one of the stopping criteria have been met and according to Hoffmann and Gelman, this will on average take longer steps than the naive kernel.
Hoffmann and Gelman also argue that this transition kernel leaves the uniform distribution $p(\theta, r|u, \mathcal{C})$ invariant, and has an efficient implementation requiring storing only $\mathcal{O}(j)$ position-momentum pairs by sampling from subtrees and cleverly weighting each sample by the amount of nodes in the corresponding subtree.
We will not show the details here, but they are readily available in [1].

Appendix E
Initial value of ε

Choosing an initial epsilon is a heuristically justified method that repeatedly doubles (or halves) the initial value of ε until the value of epsilon when run through the Leapfrog integrator crosses 1/2. That is,

1) Initialize $\varepsilon = 1$
2) Get (θ^*, r^*) with a single Leapfrog step with step length ε.
3) Double or half ε and take new Leapfrog steps until $\frac{p(\theta^*, r^*)}{p(\theta, r)}$ crosses 1/2 and choose this value for ε.

This value ensures that the simulation is reasonably accurate while not giving too extreme values, thus wasting computations.

Appendix F
NMF-GPP

Non-negative Matrix Factorization with Gaussian Process Priors (NMF-GPP) is a Bayesian version of the classical Least Squares NMF, [6] with smoothing priors on the elements of the matrix factors. The model was introduced by Schmidt and Laurberg in [3]
It looks to factor the data matrix $X \in \mathbb{R}^{K \times L}$ into non-negative factors $D \in \mathbb{R}_+^{K \times M}$ and $H \in \mathbb{R}_+^{M \times L}$ such that,

$$X \approx DH.$$

M is called the model order and is to be chosen in modeling. One can think of this in terms of learning a non-negative basis for the data matrix X, and M may then be considered the number of basis vectors.
Schmidt and Laurberg suggest putting smoothing priors on

the matrix factors through Gaussian processes. They do this by introducing vectors, d and h associated with D and H through a link function,

$$h = f_h(\text{vec}(H))$$

where $f_h : \mathbb{R}_+ \to \mathbb{R}$ and vec takes a matrix and reshapes it into a vector. We then let h (and d) follow a multivariate Gaussian (ie. the Gaussian process) with 0 mean and covariance matrices Σ_h and Σ_d, ie.

$$h \sim \mathcal{N}(0, \Sigma_h)$$

and similarly for d. There are not many restrictions on the link function, other than it should be smooth (ie. differentiable) and the inverse should exist and be relatively easy to compute.
We also introduce vectors η and δ associated to d and h through rotation by the Cholesky decomposition of the covariances of d and h such that η and δ are standard Gaussian. This gives the relation,

$$H = \text{vec}^{-1}(f_h^{-1}(C_h^T \eta)) \tag{18}$$

where C_h is the Cholesky decomposition of Σ_h.
The change-of-variables theorem of probability theory now yields that the log-prior is,

$$\mathcal{L}_\eta \propto -\frac{1}{2}\eta^T \eta$$

and similarly for δ and where the proportionality is up to an additive constant.
We also need to define a likelihood for the model, ie. $\mathcal{L}_{X|\delta,\eta}(\delta, \eta)$. In this project we choose the Least Squares likelihood originally introduced by Lee and Cheung in the original NMF paper, [6].
This gives the likelihood,

$$\mathcal{L}_{X|\delta,\eta}(\delta, \eta) \propto -\frac{1}{2\sigma_N^2}\|X - g_d(\delta)g_h(\eta)\|_F^2$$

where g_d and g_h correspond to the inverse link, (18) and σ_N is the standard deviation of the noise in the measurements.
We assume independence between the matrix factors, which gives us the log-posterior through Bayes theorem:

$$\begin{aligned}\mathcal{L}_{\delta,\eta|X}(\delta, \eta) &\propto \mathcal{L}_{X|\delta,\eta}(\delta, \eta) + \mathcal{L}_\delta(\delta) + \mathcal{L}_\eta(\eta) \\ &\propto -\frac{1}{2}(\sigma_N^{-2}\|X - g_d(\delta)g_h(\eta)\|_F^2 + \delta^T\delta + \eta^T\eta)\end{aligned} \tag{19}$$

For NUTS, we also need the gradient of the log-posterior wrt. both η and δ. This is given by,

$$\begin{aligned}&\nabla_\eta \mathcal{L}_{\delta,\eta|X}(\delta, \eta) = \\ &-\sigma_N^{-2}(\text{vec}(D^T(DH - X)) * (f_h^{-1})'(C_h^T\eta))^T C_h - \eta\end{aligned} \tag{20}$$

where $*$ denotes elementwise multiplication. Similarly for δ we have,

$$\begin{aligned}&\nabla_\delta \mathcal{L}_{\delta,\eta|X}(\delta, \eta) = \\ &-\sigma_N^{-2}(\text{vec}(H(DH - X)^T) \odot (f_d^{-1})'(C_d^T\delta))^T C_d - \delta\end{aligned} \tag{21}$$

where D and H once more comes from the relation, (18).
Finally, we state the link function used in this project. We use the Exponential-to-Gaussian link function as described in [3]. It is defined by the inverse link,

$$f_h^{-1}(h_i) = -\frac{1}{\lambda}\log\left(\frac{1}{2} - \frac{1}{2}\Phi\left[\frac{h_i}{\sqrt{2}\sigma_i}\right]\right) \tag{22}$$

where λ is a parameter to be chosen, σ_i is the i'th diagonal element of Σ_h and Φ denotes the error function.
The derivative is given by,

$$(f_h^{-1})'(h_i) = \frac{1}{\sqrt{2\pi}\sigma_i\lambda}\exp\left(\lambda f_h^{-1}(h_i) - \frac{h_i^2}{2\sigma_i^2}\right) \tag{23}$$

We will set up the covariances such that the individual components (M in total) is independent.

In [3], they find the MAP estimate by direct optimization of (19) - we will also do that here to find a starting point for the NUTS sampler.