Journal of Machine Learning Research 23 (2022) 1-25 Submitted 1/21; Revised 5/22; Published 9/22

Atlas Sampler: Adapting Trajectory Lengths And Step size

Chirag Modi
ONE@STAT.WASHINGTON.EDU
Department of Statistics
University of Washington
Seattle, WA 98195-4322, USA

Editor: My editor

Abstract

Hamiltonian Monte Carlo (HMC) has three main hyperparameters- step size for the leapfrog integration, trajectory length and the mass matrix. We present a new strategy to locally adapt the step size of HMC throughout the MCMC chain by evaluating a low-rank approximation of the local Hessian at every iteration, and estimating its largest eigenvalue. We then combine it with strategies to similarly adapt the trajectory length, and implement it in a delayed rejection framework for computational efficiency. The resulting sampler is names ATLAS- *Adapting trajectory length and step size*. With a suite of synthetic and real world examples, we show that ATLAS is- i) able to accurately sample difficult distributions with varying curvature that HMC struggles with, and ii) competitive to state-of-the-art samplers like NUTS. iii) All the hyperparameters of ATLAS can be tuned easily in a warmup phase, and iv) its performance is more robust to this tuning than for NUTS. We provide an implementation of Atlas at https://github.com/modichirag/AtlasSampler.

Keywords: keyword one, keyword two, keyword three

1 Introduction

Hamiltonian Monte Carlo (HMC) is one of the most popular Markov Chain Monte Carlo (MCMC) algorithm for Bayesian inference in high dimensions Duane et al. (1987); Neal et al. (2011); Betancourt (2017). However the performance of HMC is known to be very sensitive to tuning the algorithm parameters such as the step size, trajectory length, and the mass matrix (pre-conditioner) citations: . Thus development of auto-tuned variants like the No-U-Turn Sampler (NUTS) Hoffman et al. (2014) that locally adapts trajectory length was important for its widespread usage as a default sampler in many probabilistic programming languages.

While a lot of work has focused on adapting trajectory lengths (Hoffman et al., 2021; Sherlock et al., 2021; Bou-Rabee et al., 2024), there are no widely used samplers that locally adapt the step size of MCMC algorithms. Most implementations of HMC and its variants automatically tune a step size in the warm-up phase by targeting an acceptance rate using dual averaging schemes (Hoffman et al., 2014), or evaluate the step size using the stability criterion of the leapfrog integrator by approximating the covariance matrix of the distribution (Hoffman and Sountsov, 2022). This step size is then kept *fixed* throughout the sampling phase. However MCMC algorithms with constant step size and mass matrix struggle to accurately sample distributions with strongly varying curvature. These distributions require small step sizes in the regions with high curvature for numerical stability, while requiring large step sizes in flat regions for efficient sampling. Canonical examples of such distributions are Neal's Funnel which is highly multiscale, and Rosenbrock distribution. Such geometries also regularly arise in hierarchical models (Betancourt and Girolami, 2015; Pourzanjani and Petzold, 2019; Modi et al., 2023).

One strategy for adapting step size was recently explored by Biron-Lattes et al. (2024) that halves or doubles the step size at every iteration to maintain the ratio of probability densities for leapfrog proposals in a bounded range. While this approach is competitive to NUTS for distributions with varying local geometry, it can be an order-of-magnitude slower than NUTS for simple distributions. An alternate strategy for robustly sampling multi-scale distributions is to use implicit symplectic integrators instead of leapfrog integrator (Pourzanjani and Petzold, 2019). However these present challenges for efficient and stable line search, and thus can also be computationally expensive.

Modi et al. (2023) laid out a different approach for step size adaptation by introducing a delayed rejection (DR) approach to HMC- DRHMC. In this approach, upon a rejection in the MCMC chain, delayed proposals are made by successively reducing the step size until the proposal is accepted, or a preset maximum number of proposals are made (Tierney and Mira, 1999; Green and Mira, 2001). This was further extended in Turok et al. (2024) to generalized HMC and the resulting algorithm outperforms DRHMC. However both these extensions introduce two new hyperparameters in the algorithm which now need to be tuned- reduction factor of the step size for every proposal and the maximum number of delayed proposals to be made. Furthermore, the adapted step sizes were limited to only these pre-selected sequence of step sizes, and did not make use of the information provided by the previously rejected trajectory, thus not maximizing computational efficiency.

In this work, we develop a new strategy to automatically and continuously adapt step size using the local geometry of the distribution. The main idea is to use the positions and gradients along an MCMC trajectory to estimate the local Hessian and its maximum eigenvalue, which for quadratic potential gives an estimate of the maximum stable step size for leapfrog integration in the local neighborhood (Leimkuhler and Reich, 2004; Hoffman and Sountsov, 2022). This is then used to construct a distribution of the feasible step sizes from which we sample the new step size for the next MCMC trajectory. As we will show, this strategy is able to accurately explore multiscale distributions, but it can also be computationally expensive. Hence to make this strategy computationally efficient, we combine this with the delayed rejection framework as developed in Modi et al. (2023) to adapt step size only when necessary.

Finally, we combine this approach of locally adapting step size with strategies for adapting trajectory length to develop a fully adaptive sampler, ATLAS, which stands for *adapting trajectory length and step size*. The goal is to meet the following desiderata:

- accuracy, i.e., the sampler is able to correctly sample distributions with complex geometries, such as distributions with widely varying curvature.
- computationally efficiency, i.e., for simpler distributions, the sampler should be competitive to the state of the art samplers like NUTS.
- automatic parameter tuning, i.e., all parameters can be tuned in the warmup stage if necessary without any manual intervention.

- robustness, i.e., the converged distribution is robust to the choices of hyperparameters and the parameters tuned during the warmup. While the sub-optimal choice of hyperparameters can make the algorithm more expensive, the results should still be accurate.

With these goals in mind, the paper is organized as follows. We begin by reviewing the basics of MCMC and HMC in section 2. We describe the strategy for adapting the step size and combine it with delayed rejection approach for computational efficiency in section 3.2 Then in section 4, we combine this trajectory length adaptation in a delayed rejection framework to develop Atlas sampler. We present numerical evaluations in section 5 and conclude with a discussion in **??**.

2 Background

We seek to generate samples from a differentiable and (generally) unnormalized probability density function $\pi(\theta)$ with parameters $\theta \in \mathbb{R}^D$. In Bayesian inference, the target density π is often a Bayesian posterior.

Notation: Throughout this paper, we will use θ and ρ to denote parameters and momentum respectively. We will use x to denote the state in the phase space for HMC i.e. $x = (\theta, \rho)$. $\mathcal{F}$ is the flip operator that switches the direction of the momentum, so $\mathcal{F}x = (\theta, -\rho)$. $\mathcal{L}_\epsilon^n$ is the operator denoting leapfrog trajectory simulated with step size ϵ for n steps. For HMC, we will use $\tilde{\pi}$ to denote the Gibbs density which is the combined density of (θ, ρ). We use g to refer the gradient of the target $g = \nabla_\theta \pi(\theta)$. We will use $\tilde{}$ to denote list of points sampled on a HMC trajectory i.e. $\tilde{\theta}$ and $\tilde{g}$ is a list of θ and g sampled in an HMC trajectory. i^{th} element of this list will be denoted as $\theta^{(i)}$.

2.1 Hamiltonian Monte Carlo (HMC)

HMC generates samples from the target by simulating Hamiltonian dynamics. Specifically, it interprets the parameters θ as position vector with the associated potential energy $U(\theta) = -\log \pi(\theta)$, and introduces an auxiliary momentum vector $\rho \in \mathbb{R}^D$ for a fictitious particle with mass matrix M. Together, these define a Hamiltonian system with the total energy $H(\theta, \rho) = \frac{1}{2}\rho^T M^{-1}\rho - \log \pi(\theta)$ and the corresponding Gibbs density over the extended phase space: $x = (\theta, \rho)$ is

$$\tilde{\pi}(\theta, \rho) = Z^{-1} \exp(-H(\theta, \rho)) \propto \pi(\theta)\text{normal}(\rho \mid 0, M) \tag{1}$$

for some normalizing constant Z^{-1} and normal scale parameter M.

HMC simulates a Markov chain in two steps to generate samples from this distribution. At any position, it first resamples the momentum $\rho \sim \text{normal}(0, M)$ as a Gibbs update to get the current state $x = (\theta, \rho)$. Then it performs a Metropolis-Hastings (MH) update by simulating the Hamiltonian dynamics for a certain time T, followed by a momentum flip (negation) to reach the proposal point $x' = (\theta', -\rho') = \mathcal{F}\mathcal{L}_\epsilon^n x$. In practice, this Hamiltonian dynamics is simulated using a leapfrog integrator for n steps with step size ϵ, corresponding to a trajectory length $T = n\epsilon$. This mapping has two nice properties- 1) it is volume preserving, and 2) it is an involution i.e. it is time reversible $((\mathcal{F}\mathcal{L}_\epsilon^n)^{-1} = \mathcal{F}\mathcal{L}_\epsilon^n)$.

The proposal made in the Metropolis-Hastings step is accepted or rejected to maintain detailed balance which is a way of ensuring that asymptotically, the samples are generated from the correct target distribution. Mathematically, detailed balance condition requires

$$\tilde{\pi}(x)q(x,x')\alpha(x,x') = \tilde{\pi}(x')q(x',x)\alpha(x',x) \tag{2}$$

where $q(x,x')$ is the proposal distribution defining the transition kernel i.e. the probability of proposing the state x' from x, and $\alpha(x,x')$ is the probability of accepting the proposed point. Since the Hamiltonian flow simulated with a leapfrog integrator is a volume preserving involution, $q(x,x') = q(x',x)$ and the acceptance probability

$$\alpha = 1 \wedge \frac{\tilde{\pi}(x')}{\tilde{\pi}(x)} = 1 \wedge \frac{\pi(\theta') \cdot \mathrm{normal}(\rho' \mid 0, M)}{\pi(\theta) \cdot \mathrm{normal}(\rho \mid 0, M)} \tag{3}$$

The proposed state is accepted with this probability, otherwise it is rejected and we remain at the current state (θ, ρ).

2.2 Delayed rejection approaches

Delayed rejection will form an integral component of the algorithms that we will develop below, hence we also begin by presenting a brief summary of this approach here. We will discuss it in the context of random walk Metropolis-Hastings algorithm (Tierney and Mira, 1999; Green and Mira, 2001) since the extension to HMC and its variants results in similar accept/reject criterion, even though the derivation is different (Modi et al., 2023; Turok et al., 2024).

The main idea behind the delayed rejection approaches is that if the first proposal made from the proposal distribution $q_1(\theta, \theta')$ is rejected, then before falling back to the current state θ, we make a second (delayed) proposal with a different kernel $q_2(\theta, \theta'')$. The acceptance probability for the first proposal stays the same as the standard MH acceptance probability, i.e.,

$$\alpha_1(\theta, \theta') = 1 \wedge \frac{\pi(\theta')q_1(\theta', \theta)}{\pi(\theta)q_1(\theta, \theta')} \tag{4}$$

However the detailed balance condition for the second proposal needs to account for the fact that we have made, and rejected, a first proposal. In this case, the acceptance probability is

$$\alpha_2(\theta, \theta'') = 1 \wedge \left(\frac{\pi(\theta'')q_2(\theta'', \theta')}{\pi(\theta)q_2(\theta, \theta'')} \, \frac{q_1(\theta'', \theta^g)[1 - \alpha_1(\theta'', \theta^g)]}{q_1(\theta, \theta')[1 - \alpha_1(\theta, \theta')]} \right) \tag{5}$$

where θ^g is the *ghost* proposal which is the first proposal that a reversed MCMC chain starting from θ'' would have made and then rejected. The second fraction in the brackets in eq. (5) balances the probability of making and rejecting the first proposals in either directions.

3 Adapting Step Size

In this section, we will develop our strategy for locally adapting step size in the HMC chain.

3.1 Acceptance Probability

Before diving into the details of the step size adaptation, we begin by outlining the broad strategy with a focus on how this will modify the acceptance criterion for proposals in the MCMC chain.

To make any proposal in the MCMC chain, we will construct a distribution of feasible step sizes at the current point $q(\epsilon|x)$ where $x = (\theta, \rho)$, and draw a step size (ϵ_x) from it with which to simulate the next trajectory. Given that the Hamiltonian flow mapping $\mathcal{F} = \mathcal{L}_{\epsilon_x}^N$ is deterministic once we choose a step size and the number of leapfrog steps, the proposal distribution is simply the step size distribution, i.e., $q(x, x') = q(\epsilon|x)$. Thus, the acceptance probability for HMC with adaptive step size is

$$\alpha(x, x') = 1 \wedge \frac{\tilde{\pi}(x')\, q(\epsilon_x|x')}{\tilde{\pi}(x)\, q(\epsilon_x|x)} \tag{6}$$

Hence we need to construct a second step size distribution $q(\epsilon|x')$ at the end of the trajectory and balance by the probability of drawing the exact same step size (ϵ_x) from this second distribution.

Note that we have approached this adaptation here as constructing a local proposal distribution for the MH proposals and identifying the corresponding Hastings correction factor for the asymmetric proposals as the step size distribution. However the same acceptance criterion can be derived in GIST framework which was recently proposed in Bou-Rabee et al. (2024). Here the step size is elevated to an auxiliary parameter which is sampled from its distribution $q(\epsilon|x)$ in a Gibbs step, similar to the momentum ρ in HMC.

3.2 Step size distribution

Now that we have derived the modified acceptance probability for HMC with step size adaptation, all that remains is to construct the appropriate distribution of feasible step sizes $q(\epsilon|\theta, \rho)$. Ideally we wish to use the largest stable step size for the leapfrog integrator to simulate any trajectory. For quadratic potentials $V = \frac{1}{2}\theta^T H \theta$, i.e. multivariate Gaussian targets, this depends on the largest eigenvalue $\lambda_{\max}$ of the Hessian H with the criterion being $\epsilon \leq \frac{2}{\sqrt{\lambda_{\max}}}$ (Leimkuhler and Reich, 2004). Motivated with this, we will approximate the Hessian in the neighborhood of the initial state (θ, ρ) of the HMC trajectory for every proposal, and use its maximum eigenvalue as the basis of constructing our step size distribution. In this subsection, we describe the different components of this procedure and summarize the algorithm in algorithm 2.

3.2.1 Hessian approximation

From any state (θ, ρ), we begin with a default step size ϵ_0 to simulate a small HMC trajectory of length N_{hess}, and use these N_{hess} positions and gradients to approximate the Hessian. If we are in the difficult regions where this baseline step size is unstable and we are unable to simulate this trajectory, we re-try by reducing the step size by a factor of 2 until N_{hess} points are explored, or a maximum number of N_{attempts} are madex. In each attempt, we simulate N_{hess} steps, thereby reducing the trajectory length to keep the computational cost fixed. We fix hyperparameter N_{attempts} to 10.

Algorithm 1: Leapfrog steps

Input : Current state x, step size ϵ, number of steps n, target π, mass matrix M
Output: State after n leapfrog steps

```
1 $\theta^{(0)}, \rho^{(0)} = x$
2 for $i$ in $1 \ldots n$ do
3     $\rho' \leftarrow \rho^{(i)} + \frac{\epsilon}{2} \nabla \log \pi(\theta)|_{\theta^{(i)}}$
4     $\theta^{(i+1)} \leftarrow \theta^{(i)} + \epsilon M^{-1} \rho'$
5     $\rho^{(i+1)} \leftarrow \rho' + \frac{\epsilon}{2} \nabla \log \pi(\theta)|_{\theta^{(i+1)}}$
6 end
7 $x^{(n)} = (\theta^{(n)}, \rho^{(n)})$
8 return $x^{(n)}$
```

We consider two different approximations- 1) BFGS approximation that forms the basis of BFGS and LBFGS optimization citations: bfgs paper, and 2) BaM approximation that was introduced in citations: GSM, BAM. Both of these combine the position and gradient information to obtain a low-rank approximation of Hessian using only vector products, thus achieving $\mathcal{O}(D^2)$ scaling todo: check scaling for BaM. Thus both of these approaches can be scaled to high dimensions. Another commonly used Hessian approximation is the Gauss Newton (Fisher approximation) Hoffman and Sountsov (2022) citations: mass matrix, NGD. However this requires one to evaluate *expectation* of the outer product of gradients over the entire distribution, resulting in a global approximation while we are interested here in local Hessian approximation. It also does not use the position information and we empirically found its performance to be significantly worse the BFGS and BaM approximation. Hence we do not consider it in the rest of this work.

The next step then is to evaluate the maximum eigenvalue $\lambda_{\max}$ of this matrix. We use power iteration for this, though more sophisticated approaches such as Lanczos iteration can also be used. This gives an approximation for the stable step size $\epsilon_s = \frac{1}{2\sqrt{\lambda_{\max}}}$. We have used the factor $\frac{1}{2}$ instead of 2 to give some margin in constructing the step size function below where ϵ_s will correspond to the mean of the distribution.

3.2.2 Distribution

Finally we have all the ingredients for constructing the step size distribution $q(\epsilon|x)$, where this dependence on x will be included via ϵ_s. There are three factors that we keep in mind- 1) the distribution should be concentrated around ϵ_s for maximum efficiency, 2) the distribution should be skewed towards $\epsilon \leq \epsilon_s$ to improve stability, and 3) the distribution should be broad enough that $q(\epsilon_x|x')$, $q(\epsilon_x|x)$ have enough overlap such that the acceptance probability in eq. (9) remains high.

We consider two different distributions for step size- a scaled beta distribution and a lognormal distribution. We describe these below and fig. 1 shows the corresponding histograms for a few ϵ_s.

Algorithm 2: Step Size Adaptation

Input : Current state x, list of positions $\tilde{\theta}$, list of corresponding gradients $\tilde{g}$, Target distribution π, Mass matrix M, initial step size ϵ_0, maximum reduction in step size $r = 1024$, number of samples to estimate Hessian $N_{\text{hess}} = 10$, number of attempts $N_{\text{attempts}} = 10$

Output: Distribution function of step size and a sample from it: $\epsilon, q(\epsilon|x)$

```
$i = 0$
$\epsilon, \epsilon_{\min} = \epsilon_0, \epsilon_0/r$
$\theta^{(0)}, \rho^{(0)} = x$
$g^{(0)} = \nabla_\theta(\pi)|_{\theta^{(0)}}$
for $i$ in $1 \ldots N_{\text{attempts}}$ do
    if Length of $\tilde{\theta} \geq N_{\text{hess}}$ then
        $\hat{H}$ = HessianFunction($\tilde{\theta}, \tilde{g}$)        // BFGS or BAM approximation
        $\hat{\lambda}_{\max}$ = PowerIteration($\hat{H}$)
        if ($\hat{\lambda}_{\max} > 0$) and ($\hat{\lambda}_{\max} \leq 0.25\epsilon_{\min}^{-2}$) then
            $\epsilon_s = \frac{1}{2\sqrt{\hat{\lambda}_{\max}}}$
            $q(\epsilon|x)$ = StepSizeDistribution($\epsilon_s, \epsilon$)        // Lognormal or Beta
            return $q(\epsilon|x)$
        end
    end
    $\epsilon = \epsilon/2$        // Reduce stepsize & re-try
    for $j$ in $1 \ldots N_{\text{hess}}$ do
        $x^{(j)}$ = LeapFrog($x^{(j-1)}, \epsilon, 1, \pi, M$)
        $\theta^{(j)}, \rho^{(j)} = x^{(j)}$
        $g^{(j)} = \nabla_\theta(\pi)|_{\theta^{(j)}}$
    end
    $\tilde{\theta} = [\theta^{(N_{\text{hess}})}, \theta^{(N_{\text{hess}}-1)}, \ldots, \theta^{(0)}]$        // Reverse order for better approximation
    $\tilde{g} = [g^{(N_{\text{hess}})}, g^{(N_{\text{hess}}-1)}, \ldots, g^{(0)}]$
end
$\epsilon_s = 2\epsilon_{\min}$        // Upon failure, set $\epsilon_s$ close to $\epsilon_{\min}$
$q(\epsilon|x)$ = StepSizeDistribution($\epsilon_s, \epsilon$)
$\epsilon \sim q(\epsilon|x)$
return $\epsilon, q(\epsilon|x)$
```

Scaled beta distribution $\mathcal{B}(\alpha, \beta)$: This gives us flexibility to restrict the domain of step size distribution to enhance robustness. Given the stable step size ϵ_s and the baseline step size ϵ_0, we scale the Beta distribution to lie between $[\epsilon_0/r, \epsilon_0/2]$ where r is a hyperparameter that we fix to 1000. The parameters α and β are fixed such that the mean and mode of the distribution are ϵ_s and $\epsilon_s/2$ respectively. This ensures that the distribution is concentrated near ϵ_s but still has a skew towards smaller step size.

Lognormal distribution $\mathcal{LN}(\mu^*, \sigma^*)$: Unlike $\mathcal{B}$ distribution which depends on the baseline step size ϵ_0 and is more conservative in limiting the maximum step size to $\epsilon_0/2$, this distribution is constructed to only depend on ϵ_s. We set the mean $\mu^* = \epsilon_s$ and consider different values for the width of the distribution, finding $\sigma^* = 1.2$ to empirically perform the best. The corresponding normal distribution to this Lognormal distribution is: $\log \epsilon \sim \mathcal{N}\big(\log \mu^* - \frac{(\log \sigma^*)^2}{2}, \log \sigma^*\big)$.

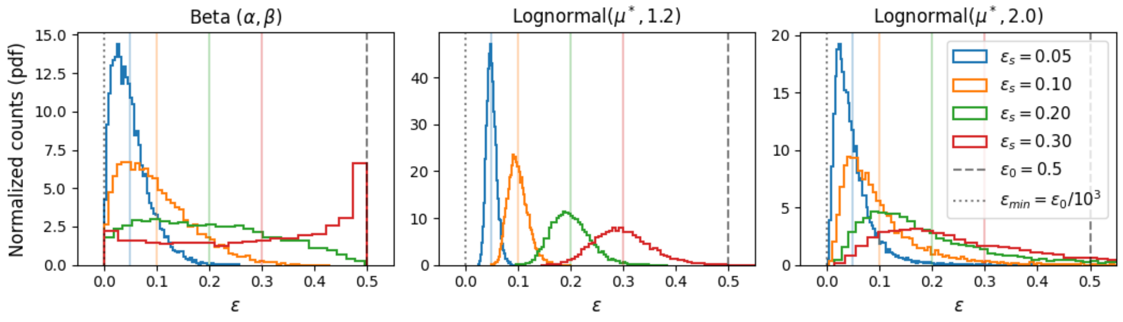


Figure 1: Step size distributions for different values of ϵ_s indicated with thin vertical lines. Black dashed and dotted lines indicate the baseline step size $\epsilon_0 = 0.5$ and ϵ_0/r.

3.3 Step size adaptation with delayed rejection

As we will show in the experiments section, simply combining the aforementioned step size adaptation with HMC can be computationally inefficient due to the extra computation required for evaluating Hessian at every iteration. On the other hand, generally a sensibly chosen baseline step size ϵ_0, can be a good choice for large regions of the phase space. In this case, paying the extra computational cost of adapting step step size can be inefficient. Motivated with these observations, we combine this step size adaptation with a delayed rejection approach Green and Mira (2001); Modi et al. (2023) to improve the computational efficiency.

In the DR framework for step size adaptation, we make the first proposal with a baseline step size (ϵ_0). Typically, this will be the step size fit during the warmup phase to achieve a target acceptance rate. If this proposal is accepted, nothing else needs to be done and we continue with the HMC chain. If this proposal is rejected, we use the positions and gradients from the rejected trajectory to estimate the Hessian. This avoids extra computation for the Hessian when not necessary, and re-uses the previously done computation from the rejected trajectory to make the approach more efficient. The acceptance probability for the delayed proposal follows the same form as eq. (5) with the suitable modifications for HMC and the second proposal kernel given by $q_2(x, x'') = q(\epsilon|x)$ as discussed above.

$$\alpha_2(x, x'') = 1 \wedge \frac{\tilde{\pi}(x'')\, q(\epsilon_x|x'')}{\tilde{\pi}(x)\, q(\epsilon_x|x)} \frac{1 - \alpha_1(x'', x^g)}{1 - \alpha_1(x, x')} \tag{7}$$

where x' is the first proposal and x^g is the ghost proposal made from x'' using the first transition kernel, i.e., HMC with baseline step size ϵ_0.

4 Adapting trajectory length and step size

So far we have discussed only adapting the step size parameter for HMC. The second parameter is the trajectory length or the integration time $T = N\epsilon$. In this section we will develop the strategy to combine the local adaptation of both the parameters.

4.1 Trajectory Length Adaptation

There are several strategies for locally adapting the trajectory lengths, with NUTS (Hoffman et al., 2014) being the most well established algorithm used in probabilistic programming languages. While the approach we develop for the combined adaptation is compatible with NUTS, in this proof of principle work we will use a much simpler trajectory length adaptation- GIST, which was recently introduced in Bou-Rabee et al. (2024)[1].

The main idea behind GIST is the same as NUTS- we want to continue on a trajectory until a U-turn is made. A U-turn is defined as when the distance of the next position in the trajectory from the initial position is less than that of the current position

$$||\theta^{(j+1)} - \theta^{(0)}||_2 < ||\theta^{(j)} - \theta^{(0)}||_2 \tag{8}$$

where $||\cdot||_2$ is the L2 norm of the vector and $x^{(0)} = \theta^{(0)}, \rho^{(0)}$ is the initial state of the trajectory. When this condition is met, GIST stops the trajectory and sets the u-turn trajectory length $n_{\mathrm{ut}} = j$. Then a proposal $\theta^{(n)}$ is chosen from $[\theta^{(0)}, ..., \theta^{(n_{\mathrm{ut}})}]$ according to the proposal distribution which depends only on n_{ut} i.e., $q(n|x^{(0)}) = q(n|n_{\mathrm{ut}})$. To maintain detailed balance, a *ghost* trajectory again needs to be simulated in the reverse direction from $\theta^{(n)}$ to construct a similar distribution for evaluating the probability of choosing $\theta^{(0)}$. Then, the acceptance probability of the proposal $x^{(n)} = (\theta^{(n)}, \rho^{(n)})$ is given by:

$$\alpha(x^{(0)}, x^{(n)}) = 1 \wedge \frac{\tilde{\pi}(x^{(n)})\, q(n|x^{(n)})}{\tilde{\pi}(x^{(0)})\, q(n|x^{(0)})}$$

Bou-Rabee et al. (2024) experiment with uniform and binomial proposal distributions $q(n|n_{\mathrm{ut}})$ for the trajectory length with different parameter values and find that GIST broadly performs competitively with NUTS. In this paper, we fix this distribution to be uniform, parameterized with parameter f_{off} which samples $n \sim \mathcal{U}[\lfloor n_{\mathrm{ut}} f_{\mathrm{off}} \rfloor, n_{\mathrm{ut}}]$ i.e. $\theta^{(n)} \sim \mathcal{U}[\theta^{(\lfloor n f_{\mathrm{off}} \rfloor)}, ..., \theta^{(n)}]$. There is a trade-off in how f_{off} is set. Large value of f_{off} results in larger jumps per acceptance. However it also increases the chance of rejection due to

1. Even though GIST is technically the name of the framework and the referenced paper introduces GIST for path-length tuning as a specific example, we use the name GIST here to refer to strategy of trajectory length tuning and to avoid confusion with NUTS

the possibility that the reverse trajectory from $\theta^{(n)}$ makes a U-turn before $\theta^{(0)}$, and hence $\theta^{(0)}$ cannot be reached in the reverse trajectory. In this case, $q(n|x^{(n)}) = 0$ and $\theta^{(n)}$ gets rejected. We will refer to this scenario as a *sub u-turn* in the trajectory.

In this work, we will randomly vary f_{off} between 0.33 and 0.66 at every iteration to strike a balance. The algorithm for a single iteration of this trajectory adaptation strategy with a few additional bookkeeping is described in algorithm 3.

Algorithm 3: GIST Proposal (No U-Turn)

Input : Current state x, step size ϵ, offset from initial position f_{off}, Target distribution π, Mass matrix M, maximum number of steps $N_{\max} = 1024$

Output: Return a proposal point before U-turn, associated lists and distribution.

```
1  $\theta^{(0)}, \rho^{(0)} = x$
2  $g^{(0)} = \nabla_\theta \pi|_{\theta^{(0)}}$
3  $\tilde{\theta}, \tilde{g} = [\theta^{(0)}], [g^{(0)}]$
4  $d = 0$
5  for $j$ in $1 \ldots N_{\max}$ do
6      $x^{(j)} = \texttt{LeapFrog}(x^{(j-1)}, \epsilon, 1, \pi, M)$
7      $\theta^{(j)}, \rho^{(j)} = x^{(j)}$
8      $g^{(j)} = \nabla_\theta \pi|_{\theta^{(j)}}$
9      $\tilde{\theta}$.prepend($\theta^{(j)}$)
10     $\tilde{g}$.prepend($g^{(j)}$)
11     if $||\theta^{(j)} - \theta^{(0)}||_2 > d$                // Check no u-turn condition
12      then
13         $d = ||\theta^{(j)} - \theta^{(0)}||_2$
14     else
15         break
16     end
17 end
18 $n_{\text{ut}} = j$
19 $q(n|x) = \mathcal{U}(\lfloor n_{\text{ut}} f_{\text{off}} \rfloor, n_{\text{ut}})$      // Distribution of trajectory lengths at $x$
20 $n \sim q(n|x)$
21 $x' = (\theta^{(n)}, \rho^{(n)})$
22 return $x', \tilde{\theta}, \tilde{g}, n, n_{\text{ut}}, q(n|x)$
```

4.2 Atlas: simple adaptive sampler

With algorithm 2 and algorithm 3, we now have all the ingredients to construct a simple adaptive sampler that locally adapts both the step size and the trajectory length at every iteration. Similar to the previous discussion regarding acceptance probability when adapting step size, the proposal distribution is now the product of the step size and the trajectory length distribution, i.e., $q(x, x') = q(\epsilon|x)q(n|x)$. This again follows from the fact that the Hamiltonian flow mapping is deterministic once we choose a step size (ϵ) and the number of leapfrog steps (n). Hence the acceptance probability is

$$\alpha(x, x') = 1 \wedge \frac{\tilde{\pi}(x')\, q(\epsilon|x')\, q(n|x')}{\tilde{\pi}(x)\, q(\epsilon|x)\, q(n|x)} \tag{9}$$

This leads to the following sampler where at every iteration starting from $x = (\theta, \rho)$, which is also summarized in algorithm 4-

1. Construct a step size distribution $q(\epsilon|x)$ with algorithm 2 and sample a step size ϵ.
2. Make a GIST proposal with algorithm 3- simulate a trajectory with this step size ϵ until U-turn. From this, construct a distribution of trajectory lengths $q(n|x)$ and make proposal $x' = x^{(n)}$.
3. Construct the step size distribution at x' with algorithm 2.
4. Simulate a reverse trajectory from x' with the same step size ϵ to construct the trajectory length distribution at the proposed point $q(n|x')$.
5. Evaluate the acceptance probability using eq. (9) and accept/reject the proposed sample.

Algorithm 4: Simple Adaptive Sampler

```
Input  : Current state $x$, maximum number of steps $N_{\mathrm{ut}}$, step size $\epsilon_0$, offset from
         initial position $f_{\mathrm{off}}$ Target distribution $\pi$, Mass matrix $M$,
Output: Distribution function of step size at the current point $(\theta, p)$: $f_\epsilon$

1  $\theta, \rho = x$
2  $g = \nabla_\theta \pi|_\theta$
3  $\epsilon, q(\epsilon|x) =$ StepSizeAdaptation$(x, [\theta], [g], \pi, \epsilon_0)$
4  $x', \tilde{\theta}, \tilde{\rho}, n, n_{\mathrm{ut}}, q(n|x) =$ GISTProposal$(x, \epsilon, f_{\mathrm{off}}, \pi, M)$
5  $\theta', \rho' = x'$
6  $g' = \nabla_\theta \pi|'_\theta$
   /* For clarity, ignore returned values that are not used later with '_'   */
7  $\_, q(\epsilon|x') =$ StepSizeAdaptation$(\mathcal{F}x', [\theta'], [g'], \pi, \epsilon_0)$
8  $\_, \_, \_, \_, \_, q(n|x') =$ GISTProposal$(\mathcal{F}x', \epsilon, f_{\mathrm{off}}, \pi, M)$
9  $\alpha = \frac{\tilde{\pi}(x')}{\tilde{\pi}(x)} \times \frac{q(\epsilon|x')q(n|x')}{q(\epsilon|x)q(n|x)}$        // Acceptance probability
10 $u \sim \mathcal{U}(0, 1)$
11 if $u > \alpha$ then
12     return $\theta$
13 else
14     return $\theta'$
15 end
```

4.3 Atlas: Adaptive sampler in delayed rejection framework

The simple algorithm of the previous section will satisfy three of our four desiderata presented in the introduction- accuracy, robustness and automatic parameter tuning. However as we will show in the next section with empirical evaluations, the sampler turns out to be not competitive to the state of the art samplers like NUTS (see fig. 3). One reason is the same as before, the extra computational cost paid in every iteration to estimate Hessian and adapt step size, which might not be necessary in every region of the phase space[2].

Another contributing factor is that due to the Hastings correction factor of trajectory length adaptation, GIST proposals have a reduced acceptance rate as compared to HMC. Empirically, we find that when the step size is tuned to give $\sim 68\%$ acceptance with HMC, GIST only gives $\sim 50\%$ acceptance. A major contributor to this is sub u-turns in the trajectory, i.e., when the reverse trajectory from $\theta^{(i)}$ does not reach $\theta^{(0)}$ and the proposal is rejected necessarily. If we happen to sample a small step size for such trajectories, corresponding to a large number of leapfrog steps before u-turn, then this results in a lot of wasteful computation.

Motivated with these observations, we again turn to delayed rejection to combine step size and trajectory adaptation in a computationally efficient manner[3]. Along the lines of section 3.3, the first proposal in the DR framework is now made as a GIST proposal with a fixed baseline step size ϵ_0. Say the trajectory length sampled for this proposal is $n_1 \sim q(n|x)$ leapfrog steps, i.e. $T = n_1\epsilon_0$. If this gets accepted, we continue to the next iteration. If this gets rejected, we make a delayed proposal with step size adaptation. However unlike vanilla HMC, this turns out to be less straightforward when we are adapting trajectory length in the first proposal.

4.3.1 Challenge with DR proposals for trajectory adaptation

The Hastings correction for the delayed proposal involves running a ghost trajectory and evaluating $1 - \alpha_1(x'', x^g)$ which is the probability of rejecting the first ghost proposal made from this second proposal point x''. While there is a single ghost proposal in case of vanilla HMC with a fixed trajectory length, in case of trajectory adaptation, there are many possible ghost states depending on the value n sampled from the distribution $q(n|x'')$. Hence evaluating $1 - \alpha_1(x'', x^g)$ will require marginalizing over all of these possible ghost states which is computationally expensive.

A simple workaround is that instead of adapting the trajectory length for the second proposal, we make it a monotonic, deterministic function of the trajectory length for the first proposal i.e. $x'' = \mathcal{F}\mathcal{L}_\epsilon^{f(n_1)}x$. Then for the ghost trajectory, we only need to evaluate the rejection probability for the same trajectory length $n_1 \sim q(n|x'')$ because for any other sampled value of the number of leapfrog steps, the reverse transition $x'' \to x$ will not be possible.

2. We do not include the extra computation of reverse trajectory of GIST sampler in this argument since GIST by itself is competitive with NUTS as shown in Bou-Rabee et al. (2024)
3. Whenever we refer to 'Atlas', we refer to this adaptation of trajectory lengths and step size in DR framework, and not the simple adaptation of algorithm 4. When there is a possibility of confusion, we will refer to them as Atlas-DR and Atlas-simple respectively.

While this solves the issue of marginalizing over a large number of trajectory lengths, it creates another problem. Making the secondary proposal trajectory length same as (or dependent) on the first trajectory length requires the first trajectory length to be reliable. In the regions of very high curvature where the baseline step size ϵ_0 might be unstable, the trajectory length of the first proposal n_1 might be 0 or unreliable. We will refer to this scenario as 'DR upon failure' i.e. when the GIST step fails in generating a useful trajectory that can inform our delayed proposal. These cases need to be treated separately. In the following, we identify these as $n_{\text{ut}} < n_{\text{min}}$, i.e., the u-turn length returned by the GIST step is less than a pre-defined minimum number of leapfrog steps, say $n_{\text{min}} = 3$. In these cases, we will sample the number of leapfrog steps for the second proposal from a global trajectory length distribution $q_g(n)$ that is constructed during the warm-up phase.

4.3.2 Atlas Algorithm

Based on the previous discussion, we develop the following algorithm for adapting trajectory length and step size in a delayed rejection framework. We assume that we have tuned a baseline step size ϵ_0 and a global distribution of trajectory lengths $q_g(n)$ during a warmup phase (we discuss on how to do this in the next section). Then the strategy for every iteration of the algorithm is as follows:

Make a GIST proposal $x \rightarrow x'$ with ϵ_0 step size. Let the u-turn length be n_{ut} and proposed sample be n_1 leapfrog steps away i.e. $x' = \mathcal{L}_{\epsilon_0}^{n_1} x$.

1. **If** $n_{\text{ut}} > n_{\text{min}}$: evaluate MH criterion and accept or reject the proposal x'. If accepted, move to next iteration. In case of rejection consider delayed proposals as follows:
   (a) **If** the rejection occurred due a *sub u-turn* in the ghost trajectory, accept the current state and move to the next iteration without making delayed proposal.
   (b) **Else** make a standard delayed proposal. Use algorithm 2 to adapt the step size ϵ_2, and set the number of leapfrog steps to be the same as the first proposal $n_2 = n_1$ (or same trajectory length i.e. $n_2 = \lfloor \epsilon_0 n_1 / \epsilon_2 \rfloor$). Make a proposal $x'' = \mathcal{L}_{\epsilon_2}^{n_2}(x)$. For detailed balance, simulate a ghost trajectory from x'' with ϵ_0 to ensure the ghost proposal x^g is rejected with the same condition, i.e., i) $n_{\text{ut}}^g > n_{\text{min}}$, ii) there is no sub u-turn in the ghost trajectory, and iii) n_1 is a valid trajectory length $n_1 \in [f_{\text{off}} n_{\text{ut}}^g, n_{\text{ut}}^g]$. Evaluate MH criterion and accept or reject the proposal x''. In case of rejection, fall back to x.

2. **Else-If** $n_{\text{ut}} \leq n_{\text{min}}$: make a 'delayed proposal upon failure'. Use algorithm 2 to adapt the step size ϵ_2, sample a trajectory length from the empirical distribution, $n_2 \sim q_g(n)$ and make a proposal $x'' = \mathcal{L}_{\epsilon_2}^{n_2}(x)$. For detailed balance, simulate a ghost trajectory from x'' with ϵ_0 to ensure the ghost proposal x^g is rejected for the same condition i.e. $n_{\text{ut}}^g \leq n_{\text{min}}$. Evaluate MH criterion and accept or reject the proposal x''. In case of rejection, fall back to x.

The pseudocode of the full algorithm, including the setting up the acceptance probabilities for the detailed balance criterion is presented in algorithm 5

Note that it is not necessary to split the the DR step in scenario (1) above into two steps and we can make the delayed proposal irrespective of the reason for rejecting the first trajectory as long as $n_{\mathrm{ut}} > n_{\min}$. However adapting the step size is likely not going to resolve the underlying geometry that leads to sub u-turn rejection. Empirically, we find that not making delayed proposals in this case reduces the computational cost without affecting the accuracy and robustness of the sampler. This is can be seen as a variant of the *probabilistic* delayed rejection approach introduced in Modi et al. (2023).

4.4 Warmup

The final missing component in our algorithm is the strategy to tune the baseline step size ϵ_0 and the global distribution of trajectory lengths $q_g(n)$ in the warmup stage.

Step size: We adapt the step size using dual averaging adaptation (Hoffman et al., 2014) by simulating HMC trajectories to target an acceptance rate, typically $60 - 65\%$. The acceptance rate for HMC is generally insensitive to the trajectory length (Neal et al., 2011)[4], so we can simulate these trajectories with any sensible length, e.g., 20 leapfrog steps.

Trajectory length: With this baseline step size, we run a short chain with GIST proposals and save the U-turn trajectory lengths for every proposal. In the warmup stage, we do not need to maintain detailed balance since we discard the warmup proposals. Hence as long as there are no divergences, i.e. the energy fluctuations remain below a certain threshold, we simply continue the onward trajectory from GIST proposals to explore as much region of the distribution as we can. Finally we use this collection of trajectory lengths to construct an disribution of the trajectory lengths $q_g(n)$. If the warmup duration is small, we recommend constructing a uniform distribution between the smallest and the largest U-turn lengths (or between 10th and 90th percentile lengths for more robustness). If the warmup duration is longer or if we combine the U-turn lengths from multiple chains, we can construct a discrete, empirical distribution of these trajectory lengths. Note that the trajectories from this distribution will only be used in the delayed proposal stage upon failure, so the overall algorithm is not very sensitive to this tuning.

5 Empirical Evaluations

In this section we evaluate the performance of different samplers that we have developed in the previous two sections and compare their performance to the state-of-the-art No U-Turn Sampler (NUTS) as implemented in Stan (Hoffman et al., 2014; Carpenter et al., 2017). Our focus will be two-fold- 1) to justify the various choices we have made in developing Atlas and 2) on showing that we meet the desiderata laid out in the introduction.

5.1 Setup

We begin by describing the setup for all the experiments.

4. assuming the curvature does not change rapidly throughout the distribution.

Algorithm 5: Atlas

Input : Current state x, Offset f_{off}, Baseline step size ϵ_0, Distribution of trajectory lengths $q_g(n)$, Minimum number of leapfrog steps $n_{\min}$, Target distribution π, Mass matrix M

Output: Next state in the Atlas trajectory

```
$x', \tilde{\theta}, \tilde{\rho}, n, n_{\text{ut}}, q(n|x)$ = GISTProposal($x, \epsilon_0, f_{\text{off}}, \pi, M$)    // First proposal
if $n_{\text{ut}} > n_{\min}$                                                           // No failure of GIST trajectory
 then
    $\_, \_, \_, n', n'_{\text{ut}}, q(n|x')$ = GISTProposal($\mathcal{F}x', \epsilon_0, f_{\text{off}}, \pi, M$)
    $\alpha_{\text{accept}} = \frac{\tilde{\pi}(x')q(n|x')}{\tilde{\pi}(x)q(n|x)}$
    $u \sim \mathcal{U}(0, 1)$
    if $u \leq \alpha_{\text{accept}}$ then
        return $x'$
    else
        if $n \notin [n'_{\text{ut}} f_{\text{off}}, n'_{\text{ut}}]$ then
            return $x$            // No delayed proposal if rejected due to sub U-turn
        else
            $\epsilon$, $q(\epsilon|x)$ = StepSizeAdaptation($x, \tilde{\theta}, \tilde{\rho}, \pi, \epsilon_0$)
            $x''$ = LeapFrog($x, \epsilon, n, \pi, M$)                              // Delayed proposal
            /* Ghost trajectory and marginalize over the ghost position at $n$ steps  */
            $\_, \tilde{\theta}'', \tilde{\rho}'', n'', n''_{\text{ut}}, q(n|x'')$ = GISTProposal($\mathcal{F}x'', \epsilon_0, f_{\text{off}}, \pi, M$)
            $x^g = \tilde{\theta}''^{(n)}, \tilde{\rho}''^{(n)}$
            $\_, \_, \_, n^g, n^g_{\text{ut}}, q(n|x^g)$ = GISTProposal($\mathcal{F}x^g, \epsilon_0, f_{\text{off}}, \pi, M$)
            /* $n$ has to be valid trajectory length, and ghost cannot be rejected due
               to sub U-turn, or failure of GIST proposal                              */
            if $n''_{\text{ut}} < n_{\min}$ or $n \notin [n''_{\text{ut}} f_{\text{off}}, n''_{\text{ut}}]$ or $n \notin [n^g_{\text{ut}} f_{\text{off}}, n^g_{\text{ut}}]$ then
                return $x$
            else
                $\_, q(\epsilon|x'')$ = StepSizeAdaptation($\mathcal{F}x'', \tilde{\theta}'', \tilde{\rho}'', \pi, \epsilon_0$)
                $\alpha^g_{\text{accept}} = \frac{\tilde{\pi}(x^g)q(n|x^g)}{\tilde{\pi}(x'')q(n|x'')}$
                $\alpha^{\text{DR}}_{\text{accept}} = \frac{\tilde{\pi}(x'')q(\epsilon|x'')(1-\alpha^g_{\text{accept}})}{\tilde{\pi}(x)q(\epsilon|x)(1-\alpha_{\text{accept}})}$
                $u \sim \mathcal{U}(0, 1)$
                if $u \leq \alpha^{\text{DR}}_{\text{accept}}$ then
                    return $x''$
                else
                    return $x$
                end
            end
        end
    end
else
    return DelayedStepUponFailure($x, f_{\text{off}}, \epsilon_0, q_g(n), n_{\min}, \pi, M$)
end
```

Algorithm 6: Atlas- Delayed Step Upon Failure

Input : Current state x, Offset f_{off}, Baseline step size ϵ_0, Distribution of trajectory lengths $q_g(n)$, Minimum number of leapfrog steps n_{min}, Target distribution π, Mass matrix M

Output: Next state in the Atlas trajectory

```
θ, ρ = x                                                        
$g = \nabla\pi|_\theta$
$\epsilon,\ q(\epsilon|x) =$ StepSizeAdaptation($x, [\theta], [g], \pi, \epsilon_0$)
$n \sim q_g(n)$                                     // Alternately make a GIST proposal
$x' =$ LeapFrog($x, \epsilon, \lfloor n\epsilon_0/\epsilon \rfloor, \pi, M$)
$x^g, \tilde{\theta}', \tilde{\rho}', n', n'_{\text{ut}}, p'_n =$ GISTProposal($\mathcal{F}x, \epsilon_0, f_{\text{off}}, \pi, M$)        // Ghost
if $n'_{\text{ut}} > n_{\text{min}}$ then
    return $x$                                      // Ensure ghost rejected due to failure
else
    $\epsilon',\ q(\epsilon|x') =$ StepSizeAdaptation($x', [\theta], [g], \pi, \epsilon_0$)
    $\alpha_{\text{accept}} = \frac{\tilde{\pi}(x')q(\epsilon|x')}{\tilde{\pi}(x)q(\epsilon|x)}$
    $u \sim \mathcal{U}(0,1)$
    if $u \leq \alpha_{\text{accept}}$ then
        return $\theta'$
    else
        return $\theta$
    end
end
```

5.1.1 Models Evaluated

We will show results for a suite of models. To illustrate the necessity of step size adaptation in robustly sampling distributions with complex geometry, we will use- a 10+1 and 50+1 dimensional Neal's funnel (`funnel-11, funnel-51`), 10 independent copies of 9+1 dimensional Neal's funnel (`multifunnel-100`), a 2 dimensional Rosenbrock (`rosenrbock-2`)[5], and a more challenging 3 dimensional hybrid Rosenbrock taken from **?** (`rosenrbockhy3-3`). We have chosen only those synthetic models here for which we can analytically generate true samples from the underlying distributions.

In addition, we also compare our algorithms against NUTS on distributions where the latter is known to perform very well. We will refer to these as baseline models and with these, our goal is to validate that in absence of challenging geometry, Atlas is competitive with NUTS. For this, we will use- a 100 dimensional correlated Gaussian with correlation coefficient $r = 0.95$ (`corr_normal95-100`), a 100 dimensional ill-conditioned Gaussian (`ill_normal-100`), 4 dimensional hidden Markov model (`hmm`), 8 dimensional Lotka-Volterra population dynamics (`lotka_volterra`), 7 dimensional autoregressive time series (`arK`), a 45 dimensional Gaussian linear model (`glmm-poisson`, a 143 dimensional item-response theory model (`irt_2pl`), and a 503 dimensional stochastic volatility model (`stochastic_volatility`).

5.1.2 Sampling Configuration

We will compare the performance of our algorithms against NUTS as implemented in `cmdstanpy` citations: cmdstan. All the models are written in `Stan` (Carpenter et al., 2017) while our algorithms are written in `Python`. We use `bridgestan` citations: Rouldes to access the log-probability and its gradients from `Stan` models in `Python`.

We run 32 chains for 2000 samples each for all examples except the variants of Rosenbrock and Neal's funnel, for which we generate 50000 samples. The step size for NUTS is tuned for a 1000 samples to target the acceptance rate of 80% before sampling. For consistency, we will use this same step size as the baseline step size ϵ_0 for our algorithms unless mentioned otherwise. However note that due to additional enhancements in `cmdstanpy` such as multinomial sampling along the trajectory for better acceptance, the standard HMC algorithm with this same step size gives only $60 - 70\%$ acceptance rate. Finally, we use the identity mass matrix for all our experiments.

For the variants of Rosenbrock and Neal's funnel, we analytically generate reference (true) samples from the distribution. For other examples, we generate reference samples (θ_{ref}) by running 16 chains of NUTS for 100,000 samples each. The step size is tuned to target 95% acceptance rate, and we adapt a dense metric in the warmup phase that lasts for 5000 iterations.

5.1.3 Default Atlas Configuration

In the main text, we will present results for the default configuration of Atlas. This consists of following choices. For step size distribution, we use lognormal distribution with $\sigma^* =$

5. We find that as we increase the dimensions, i.e., add multiple copies of 2D Rosnebrock, the default adaptation of step size for NUTS in `cmdstanpy` adapts an increasingly better step size and is thus able to robustly sample the distribution in high dimensions.

1.2. We use BFGS approximation for the Hessian. For delayed proposals, we scale the number of leapfrog steps to keep the trajectory length constant, i.e., $n_2 = \lfloor \epsilon_0 n_1 / \epsilon_2 \rfloor$. The warmup phase for constructing the global distribution of trajectory lengths $q_g(n)$ lasts for 100 proposals. $q_g(n)$ uniformly samples between the 10th and the 90th percentile of the U-turn lengths explored during this warmup. We do not combine information between different chains to keep them independent. The other hyperparameters are set to their default values, which are mentioned in the Inputs of corresponding algorithms (pseudocode). We will investigate the impact of some of these choices in the ablation studies done in the Appendix, but broadly we find their impact to be minimal.

5.1.4 Metric for comparison

To compare different algorithms, we evaluate normalized root mean square error in the parameters (θ) and squared-parameters (θ^2) for every chain.

$$\text{zRMSE}_\theta = \frac{1}{D}\sum_{d=1}^{D} \frac{\left(\frac{1}{N}\sum_i \theta_d^{(i)} - \mu(\theta_{\text{ref},d})\right)^2}{\sigma^2(\theta_{\text{ref},d})} \tag{10}$$

$$\text{zRMSE}_{\theta^2} = \frac{1}{D}\sum_{d=1}^{D} \frac{\left(\frac{1}{N}\sum_i (\theta_d^{(i)})^2 - \mu(\theta^2_{\text{ref},d})\right)^2}{\sigma^2(\theta^2_{\text{ref},d})} \tag{11}$$

where $\mu(\theta_{\text{ref},d})$ and $\sigma^2(\theta_{\text{ref},d})$ are the mean and variance evaluated using reference samples for parameter θ_d (and similarly for parameters squared θ_d^2). We will show these zRMSE for every chain as a boxplot for all the algorithms.

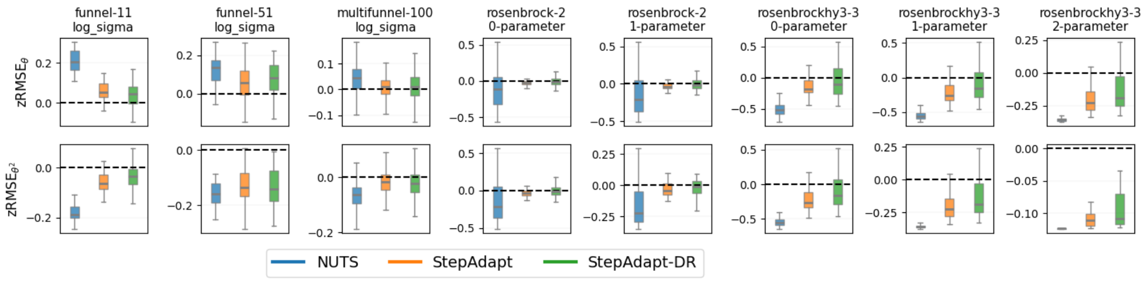
For variants of Rosenbrock and funnel, we evaluate the error for individual parameters. Hence for Neal's Funnel, we will separately evaluate the error for the $\log_\sigma$ (hierarchical) and the latent parameters. For Rosenbrock, we separately evaluate the error for the first, second and the third parameter (in `rosenbrockhy3`) which have very different marginal distributions.

Finally to evaluate the cost of algorithms, we will compare the number of gradient evaluations normalized with the mean number of NUTS gradient evaluations over 32 chains.

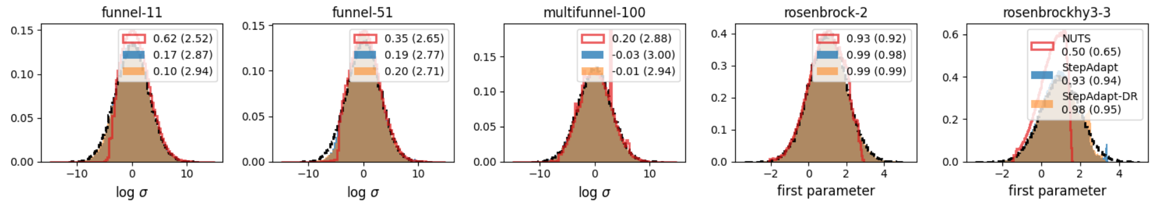
5.2 Validating step size adaptation

We begin by validating the step size adaptation strategy described in section 3.2. We compare the standard HMC with step size adaptation (StepAdapt), and the HMC with delayed rejection (DR) with step size adaptation (StepAdapt-DR). In the former, we run a HMC chain where for every iteration, we construct a stepsize distribution and sample a step size using algorithm 2. In StepAdapt-DR, we run HMC with delayed proposal where the first proposal is made with the baseline step size, and if this gets rejected, the second proposal is made with adapted step size. The number of leapfrog steps in both cases is sampled from the trajectory length distribution $q_g(n)$ that is constructed during warmup as described in section 4.4.

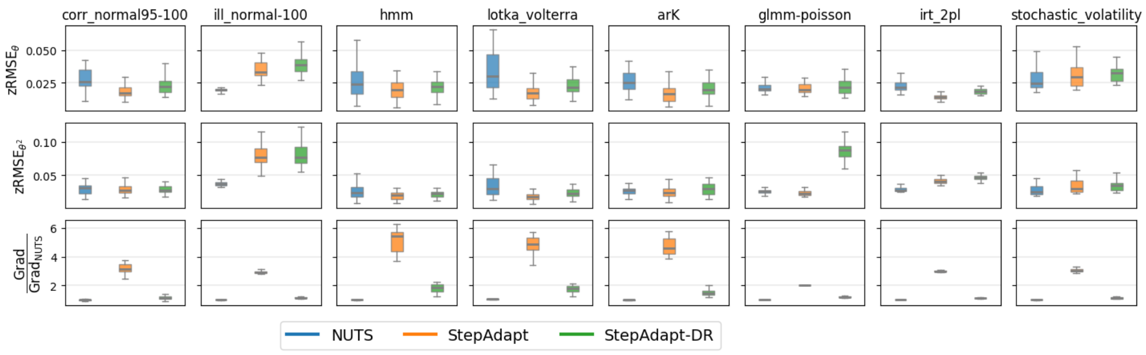
We show the results for these in fig. 2. As can be seen in fig. 2a and fig. 2b, NUTS is not able to robustly sample distributions like Neal's funnel and Rosenbrock which have strongly varying curvature. However our implementation of HMC and HMC + DR with



(a) Models with varying curvature: variants of Neal's funnel and Rosenbrock distribution. NUTS (in blue) fails to robustly sample these distributions, but algorithms adapting step size (orange and green) are able to generate correct samples.



(b) Histogram of the $\log \sigma$ parameter for funnel and the first parameter in Rosenbrock distribution. Their true distribution is shown in black dashed lines, and is $\mathcal{N}(0, 3)$ and $\mathcal{N}(1, 1)$ respectively. Legends show the inferred mean (standard deviation) for different algorithms.



(c) Other baseline models without strongly varying curvature where NUTS is reliably able to sample the distribution.

Figure 2: Adapting step size: We show the normalized RMSE for the parameters and parameter-squared for algorithms adapting step size with standard HMC (`StepAdapt`, in orange) and its more computationally efficient version with delayed rejection (`StepAdapt-DR`, in blue), compared against NUTS (in blue). For baseline models, we also show the cost in terms of number of gradient evaluations normalized against NUTS.

step size adaptation is able to sample these distributions robustly. The histograms of the $\log \sigma$ parameter of Neal's funnel and the first parameter for Rosenbrock further indicate that the inference with the ensemble of chains is better than what the boxplots, which measure the variability across chains, would indicate.

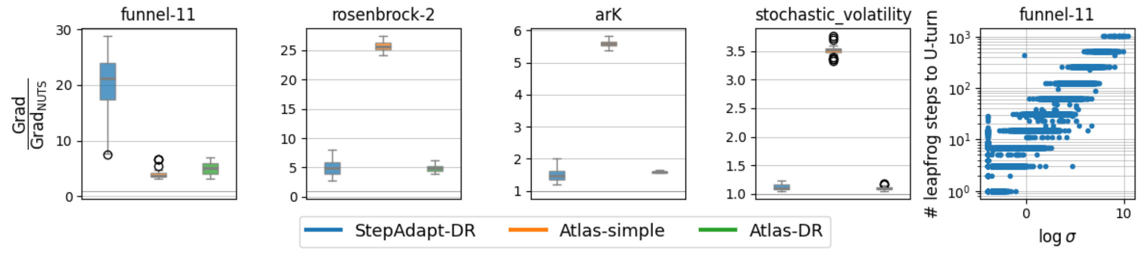


Figure 3: The first four panels compare the cost of step size adaptation with HMC+DR, simple implementation of Atlas (algorithm 4) and Atlas with DR (algorithm 5). The last panel shows that the number of leapfrog steps taken by NUTS before U-turn can vary by orders of magnitude as a function of $\log \sigma$.

In fig. 2c we compare the performance with NUTS on other baseline models which do not have strongly varying curvature. Broadly, all three algorithms perform comparably for these models. However as can be seen from the last row that compares the cost of these algorithms relative to NUTS in terms of number of gradient evaluations, step size adaptation in DR framework is clearly much more computationally efficient than without by a factor of 3-5x, and in itself is comparable to NUTS.

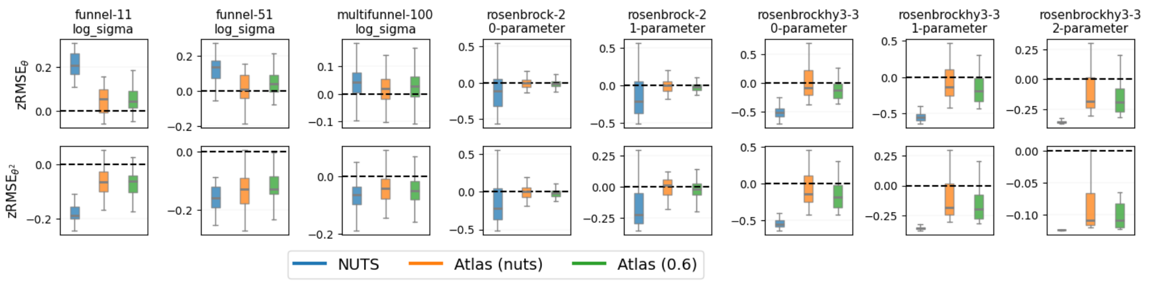
5.3 Motivating Atlas

Figure 2 might lead one to question the necessity of adapting trajectory lengths since Stepadapt-DR sampler appears to be both robust, and computationally efficient. Hence we begin by motivating the necessity for this with fig. 3. The first panel shows that for sampling highly multiscale distributions like Neal's funnel, Stepadapt-DR sampler can be 10x more expensive than Atlas, which adapts both trajectory lengths and step size. The reason for this is shown in the last panel of the same figure, which shows the number of leapfrog steps taken by NUTS before making a U-turn as a function of $\log \sigma$ parameter. At constant step size, there is three orders of magnitude of difference in the number of steps taken, and hence the trajectory length, between the neck and the mouth of the funnel. In such a case, sampling randomly from a global distribution of trajectory lengths $q_g n$ as is done in StepAdapt and StepAdapt-DR can be extremely computationally inefficient. Hence adapting trajectory lengths locally is necessary, not only aesthetically desirable.

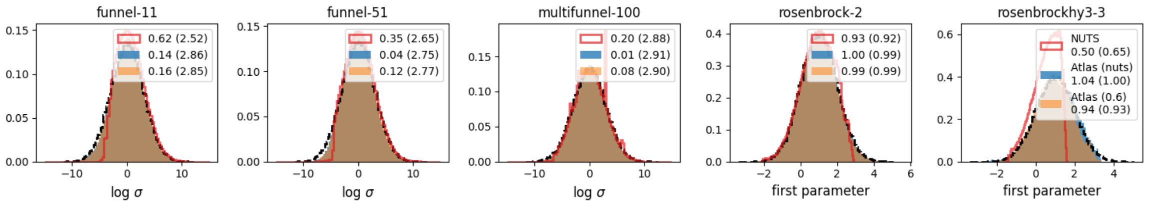
In the panels 2-4 of the fig. 3, we also compare the cost of simple implementation of Atlas (algorithm 4) and Atlas with DR (algorithm 5). As was was discussed in section 4.3, the simple implementation can be 3-5x more expensive than Atlas+DR (though it is surprisingly the same cost for Neal's funnel). As a result, we will focus only on Atlas+DR in the rest of the work, and henceforth refer to it as Atlas.

5.4 Performance of Atlas

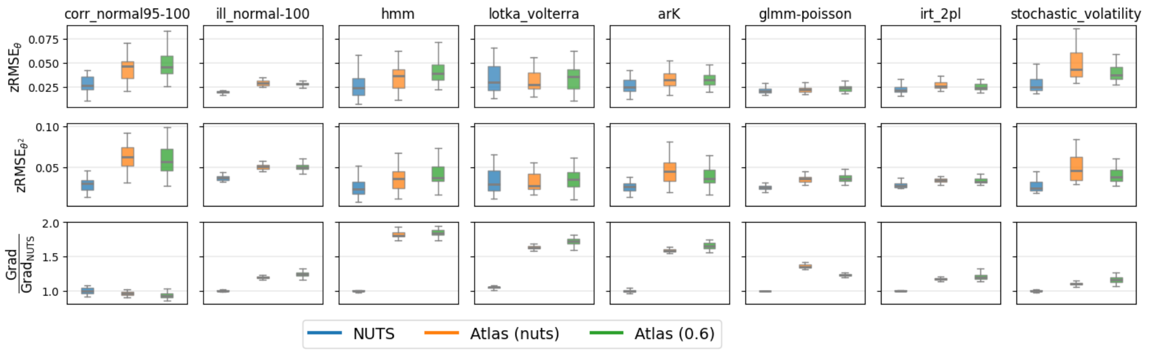
Finally in fig. 4, we present the results for Atlas and show it fulfills our desiderata. We present results for two cases- when we use the same baseline step size ϵ_0 for the first proposal as NUTS (Atlas-(nuts) in orange), and when the baseline step size is tuned using dual



(a) Models with varying curvature: variants of Neal's funnel and Rosenbrock distribution. NUTS (in blue) fails to robustly sample these distributions, but Atlas (orange and green) is able to.



(b) Same histogram of the $\log \sigma$ parameter for funnel and the first parameter in Rosenbrock distribution as fig. 2b. True distribution (black dashed lines) is $\mathcal{N}(0,3)$ and $\mathcal{N}(1,1)$ respectively. Legends show the inferred mean (standard deviation). NUTS results are shown in red line.



(c) Other baseline models without strongly varying curvature.

Figure 4: Atlas: We show the normalized RMSE for the parameters and parameter-squared for NUTS (blue), Atlas with the same baseline step size as NUTS (Atlas (nuts) in orange), and Atlas with baseline step size tuned to target acceptance rate of 0.6 (Atlas (0.6) in green). For baseline models, we also show the cost in terms of number of gradient evaluations normalized against NUTS and find Atlas to be competitive with NUTS.

averaging to target an acceptance rate of 0.6 for the first 100 iterations (Atlas-(0.6) in green). In both cases, we also use 100 iterations to construct the global distribution of trajectory lengths $q_g(n)$ that is used in the 'DR upon failure' case of Atlas. Atlas-(0.6) demonstrates that both the hyperparameters- baseline step size and the distribution of trajectories can be tuned easily with a short warmup period, fulfilling point 3. of desiderata.

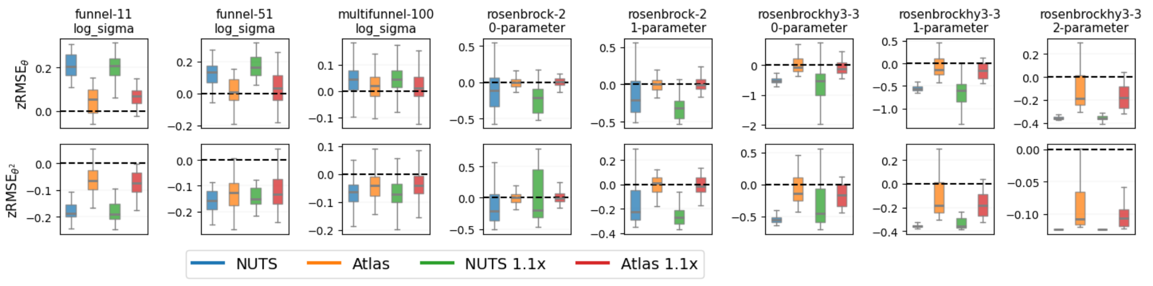
Similar to StepAdapt and StepAdapt-DR, in fig. 4a, we show that Atlas is robustly able to sample complex distributions with strongly varying curvature where NUTS with constant step size fails. We note that in these boxplots, we have not shown outliers to maintain clarity. Hence even though it seems like NUTS is able to sample from `multifunnel-100`, it has many outlier chains while Atlas has almost none. This is also visible in the histogram in fig. 4b where the histogram for NUTS (red line) for this model shows spikes and worse mean than Atlas. Finally, the average computational cost of these models as compared to NUTS is approximately 5x, 2x, 1.5x, 5x, 10x for `funnel-11`, `funnel-51`, `multifunnel-100`, `rosenbrock-2`, `rosenbrockhy3-3` respectively. For context, this is approximately the same cost as if running NUTS with a smaller step size, tuned to target acceptance rate of 0.95 (instead of default 0.8) in `cmdstanpy`. However even with this small step size, NUTS is unable to robustly sample all these distributions (except `multifunnel-100`). Hence based on this, we conclude that Atlas satisfies point 1. of our desiderata.

Finally in fig. 4c, we compare the performance of Atlas and NUTS on baseline models where the latter is able to sample the distribution correctly. In all cases, the accuracy of Atlas is similar to that of NUTS. Computationally, for low dimensional models like `hmm`, `lotka_volterra`, and `arK`, Atlas is 1.5x-1.8x more expensive than NUTS, though these models are relatively cheap and not computationally expensive themselves. For larger (and slower) models like `glmm-poisson` (45 dim), `irt_2pl` (143 dim), and `stochastic_volatility` (503 dim), Atlas is 1.1x-1.25x more expensive. Hence for large dimensions which do not have complex geometries, Atlas is not (much) more expensive than NUTS, thus fulfilling point 2. of desiderata.

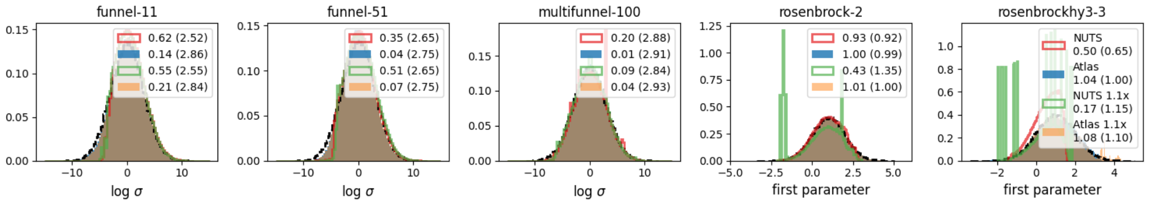
5.5 Robustness to initial step size

Finally we turn to the last desiderata, robustness to the tuning of hyperparameters. In fig. 5 we investigate this robustness for tuning the baseline step size. We show the results for default configuration of NUTS (in blue), i.e., when the step size is tuned to target 80% acceptance rate, and when this step size is increased by 10% (in green). As expected, both these cases fail to correctly sample the variants of Neal's funnel and Rosenbrock. However as shown in the fig. 5c, NUTS with larger step size is also unable to sample from the baseline models which do not have strongly varying curvature. This indicates the sensitivity of NUTS, and more generally MCMC, to hyperparameters like step size citations: sam's paper?. While `cmdstanpy` is very good at tuning step size, it can fail in a few scenarios, e.g., when the warmup phase is not long enough, initialization is poor or the sampler is unable to explore some regions of distribution due to complex geometry.

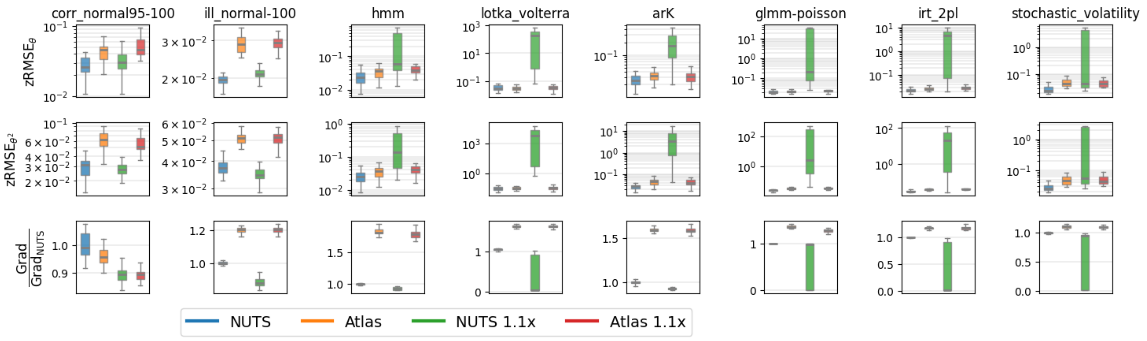
The performance of Atlas is not as sensitive to tuning this step size which is the baseline step size ϵ_0 for the first proposal. In all cases, the accuracy of the sampler broadly stays the same. For the multiscale distributions like funnel and Rosenbrock, the computational cost increases by a factor of 1.2x on average, while for the baseline models, even this



(a) Models with varying curvature: variants of Neal's funnel and Rosenbrock distribution. NUTS (blue and green) fails to robustly sample these distributions, but Atlas (orange and red) is able to.



(b) Same histogram of the $\log \sigma$ parameter for funnel and the first parameter in Rosenbrock distribution as fig. 2b. True distribution (black dashed lines) is $\mathcal{N}(0, 3)$ and $\mathcal{N}(1, 1)$ respectively. Legends show the inferred mean (standard deviation). NUTS results are shown in red line.



(c) Other baseline models without strongly varying curvature.

Figure 5: Sensitivity to step size: We show the normalized RMSE for the parameters and parameter-squared for NUTS (blue), and when the step size is increased by 10% (NUTS 1.1x in green), as well as Atlas for the same two step sizes (orange and red respectively). For baseline models, we also show the cost in terms of number of gradient evaluations normalized against NUTS. NUTS with 10% larger step sizes fails, but Atlas is still able to robustly sample all the distributions.

largely remains unchanged. In the appendix, we show similar lack of sensitivity to other hyperparameter- the global distribution of trajectory lengths $q_g(n)$ from which we sample the number of leapfrog steps in the case of 'DR upon failure'. Together these results indicate that Atlas fulfills the fourth desiderata of robustness to tuning hyperparameters over reasonable ranges.

6 Discussion and conclusions

Appendix A. Hessian approximations

We provide the algorithm for the two Hessian approximations- BFGS and BAM approximations here.

Appendix B. Ablations

B.1 Lognormal vs Beta function for step size

B.2 Uniform vs Empirical Trajectory distribution

B.3 BFGS vs BaM Hessian approximation

Appendix C. Models Evaluated

References

Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

Michael Betancourt and Mark Girolami. Hamiltonian Monte Carlo for hierarchical models. *Current trends in Bayesian methodology with applications*, 79(30):2–4, 2015.

Miguel Biron-Lattes, Nikola Surjanovic, Saifuddin Syed, Trevor Campbell, and Alexandre Bouchard-Cote. autoMALA: Locally adaptive Metropolis-adjusted Langevin algorithm. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 4600–4608. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/biron-lattes24a.html.

Nawaf Bou-Rabee, Bob Carpenter, and Milo Marsden. GIST: Gibbs self-tuning for locally adaptive Hamiltonian Monte Carlo. *arXiv e-prints*, art. arXiv:2404.15253, April 2024. doi: 10.48550/arXiv.2404.15253.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76, 2017.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Peter J Green and Antonietta Mira. Delayed rejection in reversible jump Metropolis–Hastings. *Biometrika*, 88(4):1035–1053, 2001.

Matthew Hoffman, Alexey Radul, and Pavel Sountsov. An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3907–3915. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/hoffman21a.html.

Matthew D. Hoffman and Pavel Sountsov. Tuning-free generalized hamiltonian monte carlo. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 7799–7813. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/hoffman22a.html.

Matthew D Hoffman, Andrew Gelman, et al. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. 2004.

Chirag Modi, Alex Barnett, and Bob Carpenter. Delayed rejection Hamiltonian Monte Carlo for sampling multiscale distributions. *Bayesian Analysis*, 1(1), January 2023. ISSN 1936-0975. doi: 10.1214/23-ba1360. URL http://dx.doi.org/10.1214/23-BA1360.

Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Arya A Pourzanjani and Linda R Petzold. Implicit Hamiltonian Monte Carlo for sampling multiscale distributions. *arXiv preprint arXiv:1911.05754*, 2019.

Chris Sherlock, Szymon Urbas, and Matthew Ludkin. The Apogee to Apogee Path Sampler. *arXiv e-prints*, art. arXiv:2112.08187, December 2021. doi: 10.48550/arXiv.2112.08187.

Luke Tierney and Antonietta Mira. Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in medicine*, 18(17-18):2507–2515, 1999.

Gilad Turok, Chirag Modi, and Bob Carpenter. Sampling From Multiscale Densities With Delayed Rejection Generalized Hamiltonian Monte Carlo. *arXiv e-prints*, art. arXiv:2406.02741, June 2024. doi: 10.48550/arXiv.2406.02741.