Question 1 Part 2 Report

Yan Kin Chi
yan_kin.chi@unsw.edu.au

I. Introduction

Hamiltonian Monte Carlo (HMC) is a Markov Chain Monte Carlo (MCMC) method known for its efficiency at exploring complex posterior distributions within the context of Bayesian inference. Formally, if $y \in \gamma$ denotes observed data and $\theta \in \mathbb{R}^d$ denotes parameters of interest, we start by defining the Bayesian inference problem as aiming to infer the posterior distribution:

$$\pi(\theta) = p(\theta|y) \propto p(y|\theta)p(\theta)$$

Where $p(\theta|y)$ is the posterior distribution, $p(\theta)$ is the prior distribution, and $p(y|\theta)$ is the likelihood function. Next, instead of sampling from $\pi(\theta)$ directly, momentum variables $\rho \in \mathbb{R}^d$ are introduced to define an augmented joint distribution over (θ, ρ) such that:

$$\pi(\theta, \rho) \propto e^{-H(\theta,\rho)}$$

With $H(\theta, \rho) = U(\theta) + K(\rho)$ being the Hamiltonian function with potential energy $U(\theta) = -\log p(\theta|y)$ component and kinetic energy component $K(\rho) = \frac{1}{2}\rho^T M^{-1}\rho$, assuming M is the mass matrix and $\rho \sim \mathcal{N}(0, M)$.

A common challenge during HMC is the scenario where the likelihood is intractable. While many methods have been proposed, we notably focus on Pseudo-Marginal Hamiltonian Monte Carlo (PM-HMC) [1]. Specifically, proposes augmenting the parameter space with i.i.d. distributed auxiliary variables. In addition with the implementation of a optimized Strang Splitting numerical estimator, this results in the ability to create an unbiased estimator of the intractable likelihood assuming the following:

1. Given the intractable likelihood $p(y|\theta)$ and auxiliary random variables $\mathbf{u} \sim m(\mathbf{u}), \mathbf{u} \in \mathbb{R}^D$, there exists a non-negative unbiased estimator $\tilde{p}(y|\theta, \mathbf{u})$ such that:

$$p(y|\theta) = \int \hat{p}(y|\theta, \mathbf{u})m(\mathbf{u})du$$

2. Smoothness and normality of auxiliary variables. Specifically, $\hat{p}(y|\theta, \mathbf{u})$ is continuously differentiable and $\mathbf{U} \sim \mathcal{N}(0_D, I_D)$. Additionally the gradient of the log-likelihood estimator is computable at any point $(\theta, \mathbf{u})$ such that: $\nabla_{(\theta,\mathbf{u})} log\ \hat{p}(y|\theta, \mathbf{u})$ exists and is not singular.

Additionally, this setup provides the following propositions:

1. The Markov chain PM-HMC constructs has a stationary distribution with the correct marginal posterior $p(\theta|y)$.
2. The error between the PM-HMC trajectory (using the unbiased estimator) and the ideal HMC trajectory (using the true likelihood) can be bounded in terms of the log-likelihood gradient allowing its independence to the number of importance samples N.
3. Based on the central limit theorem (CLT) the gradient estimation error decreases at the rate of $1/\sqrt{N}$.

However, being in the class of MCMC method, the prohibitive cost of computing likelihoods and gradients especially for high-dimensional problems remains a big challenge [2].

Recent work has explored integrating deep learning with Hamiltonian Monte Carlo (HMC) through latent Hamiltonian neural networks (L-HNNs), which simulate Hamiltonian dynamics while preserving time reversibility and energy conservation. By inferring gradients directly, L-HNNs eliminate the need for explicit numerical integration, significantly reducing computational overhead [3]. However, they are not strict alternatives to traditional numerical methods, as inadequate training data in high-uncertainty regions can lead to large integration errors. To mitigate this, implementations such as [3] use L-HNNs to prioritize inferred gradients while incorporating adaptive monitoring schemes that revert to numerical integration when necessary.

In this report we follow the footsteps of [3] to explore the possibility of integrating Pseudo-Marginal Hamiltonian Monte Carlo with latent Hamiltonian neural networks.

II. Methodology

Traditionally, the HNN class of methods divide the Bayesian inference problem into three stages: generating Hamiltonian gradient samples, training the L-HNN, and sampling the posterior.

A. *Generating Hamiltonian gradient samples*

We start with the extended Hamiltonian defined as

$$H(\theta, \rho, \mathbf{u}, \mathbf{p}) = -\log p(\theta) - \log \hat{p}(y|\theta, \mathbf{u}) + \frac{1}{2}\{\rho^{\mathrm{T}}\rho + \mathbf{u}^{\mathrm{T}}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{p}\}$$

with $\mathbf{p} \in \mathbb{R}^{\boldsymbol{D}}$ being the momentum variable for $\mathbf{u}$. The differential equations of motion with respect to time t for the extended Hamiltonian can be written as:

$$\frac{d}{dt}\begin{pmatrix}\theta \\ \rho \\ \mathbf{u} \\ \mathbf{p}\end{pmatrix} = \begin{pmatrix}\rho \\ \nabla_\theta[\log p(\theta) + \log \hat{p}(y|\theta, \mathbf{u})] \\ \mathbf{p} \\ -\mathbf{u} + \nabla_{\mathbf{u}} \log \hat{p}(y|\theta, \mathbf{u})\end{pmatrix}$$

By specifying the number of time steps, step size, and employing a symplectic numerical integrator, Hamiltonian gradient samples can be obtained by numerically solving Hamilton's equations at different discrete time steps.

Choice of the symplectic numerical integrator

A suitable numerical integrator must satisfy the following three conditions:

1. Symplecticity: preserves the volume of phase and the underlying Hamiltonian structure.
2. Reversibility: integrating forward with step size h and then backwards with step size $-h$ should recover the initial state

3. Second order accurate: the local truncation error must be $O(h^3)$ ensure global error accumulation remains at $O(h^2)$

In [1], the author proposes a preferred numerical integrator known as Strang splitting. Specifically, it decomposes the Hamiltonian:

$$H(\theta, \rho, \mathbf{u}, \mathbf{p}) = A(\rho, \mathbf{u}, \mathbf{p}) + B(\theta, \mathbf{u})$$

into two partial flows

$$A(\rho, \mathbf{u}, \mathbf{p}) = \frac{1}{2}\{\rho^{\mathrm{T}}\rho + \mathbf{u}^{\mathrm{T}}\mathbf{u} + \mathbf{p}^{\mathrm{T}}\mathbf{p}\}$$

$$B(\theta, \mathbf{u}) = \log p(\theta) - \log \hat{p}(y|\theta, \mathbf{u})$$

where the equations of motion of flow A (time-h) are:

$$\frac{\partial H}{\partial \rho} = \rho, \frac{\partial H}{\partial \theta} = \mathbf{0}, \frac{\partial H}{\partial \mathbf{p}} = \mathbf{p}, \frac{\partial H}{\partial \mathbf{u}} = \mathbf{u}$$

and flow B (time-h) are:

$$\frac{\partial H}{\partial \rho} = 0, \frac{\partial H}{\partial \theta} = \nabla_\theta[\log p(\theta) + \log \hat{p}(y|\theta, \mathbf{u})]$$

$$\frac{\partial H}{\partial \mathbf{p}} = 0, \frac{\partial H}{\partial \boldsymbol{u}} = -\mathbf{u} + \nabla_{\mathbf{u}} \log \hat{p}(y|\theta, \mathbf{u})$$

While it is proven that Strang splitting integrators are second-order accurate with their local truncation remaining at $O(h^3)$[1], during our implementation a crucial revelation is that **<u>the magnitude of the realized gradients can be very large</u>**. We illustrate this using the latent variable model in section 2.3 of [1]. Specifically, given $\mathbf{U}$ is the collection of all auxiliary variables, T be the number of subjects, based on equation (14) in [1], we consider the following scenario:

$$X_k \overset{\text{i.i.d.}}{\sim} f_\theta(.), \ \ Y_k|X_k \sim g_\theta(.|X_k),$$

where $(X_k)_{k\geq 1}$ are $\mathbb{R}^n$-valued latent variables and each Y_k takes values from some space $\mathbf{Y}$. Furthermore let latent variable X_k have importance density $q_\theta(x_k|y_k)$ which we assume is simulated using $X_k = \gamma_k(\theta, V)$ where $\gamma_k: \Theta \times \mathbb{R}^p \to \mathbb{R}^n$is a deterministic map and $V \sim \mathcal{N}(0_p, I_p)$. Note the authors assume that the map of $(\theta, \mathbf{u})$ to $\nabla_{\mathbf{u}} \log \omega_\theta(y, \mathbf{u})$is Lipschitz in $\mathbf{u}$ with constant M and that $\omega_\theta(y, \mathbf{u})$ itself is Lipshitz with constant D. If

Next, let $\omega_\theta(y_k, \mathbf{U}_{k,i}) = \frac{g_\theta(y_k|\mathbf{U}_{k,i}) f_\theta(\mathrm{X}_{k,i})}{q_\theta(\mathbf{U}_{k,i}|y_k)}$, we omit the derivation process and focus on the gradients of θ shown in equation (14):

$$\nabla_\theta \log \hat{p}(y_{1:T}|\theta, \mathbf{U}) = \sum_{k=1}^{T}\sum_{i=1}^{N} \frac{\omega_\theta(y_k, \mathbf{U}_{k,i})}{\sum_{j=1}^{N} \omega_\theta(y_k, \mathbf{U}_{k,j})} \nabla_\theta \log \omega_\theta(y_k, \mathbf{U}_{k,i})$$

Notice the gradient involves a double summation that sums a d dimensional vector across all important samples N and sample length T. Recall that θ has dimension d containing all parameters of interest. Therefore, during gradient calculation when accumulating the partial derivatives of the logistic model the magnitude (Euclidean norm) of the gradient error can in the worst-case scale by $\sqrt{d \cdot N \cdot T}$. This assumes the gradient contributions from all summands are uncorrelated and accumulate as a sum of squared terms.

Using Strang splitting, the momentum variables $(\rho, \mathbf{p})$ are updated by a full step:

$$\rho_{new} = \rho - h \cdot \nabla_\theta B(\theta, \mathbf{u}), \ \ \mathbf{p}_{new} = \mathbf{p} - h \cdot \nabla_{\mathbf{u}} B(\theta, \mathbf{u})$$

this means that the full gradient is applied over the entire step h. This setup is not idea because the metropolis acceptance probability is:

$$\alpha = 1 \wedge \exp(-\Delta H)$$

This means that large gradients can deteriorate the acceptance probability. To resolve this issue, we inherit a velocity-Verlet leapfrog style symplectic numerical integrator that is also known to be second order accurate and reversable [3]. Let the extended Hamiltonian have kinetic function:

$$K(\rho, \mathbf{p}) = \frac{1}{2}\{\rho^{\mathrm{T}}\rho + \mathbf{p}^{\mathrm{T}}\mathbf{p}\}$$

and potential function:

$$U(\theta, \mathbf{u}) = -\log p(\theta) - \log \hat{p}(y|\theta, \mathbf{u}) + \frac{\mathbf{u}^{\mathrm{T}}\mathbf{u}}{2}$$

We then split the momentum update into two half-steps:

$$\rho_{new} = \rho - \frac{h}{2} \cdot \nabla_\theta U(\theta, \mathbf{u}) - \frac{h}{2} \cdot \nabla_\theta U(\theta_{new}, \mathbf{u}_{new})$$

$$\mathbf{p}_{new} = \mathbf{p} - \frac{h}{2} \cdot \nabla_{\mathbf{u}} U(\theta, \mathbf{u}) - \frac{h}{2} \cdot \nabla_{\mathbf{u}} U(\theta_{new}, \mathbf{u}_{new})$$

Notably, because the gradients distribute the update over two half-steps, there is a reduction in the effective magnitude of updates. Furthermore, averaging can smooth out large gradients through partial cancellation. Ultimately, this can reduce the amplitude of large single step errors of ΔH and improve the Metropolis acceptance rate.

B. *Training of the L-HNN*

We train the L-HNN using the generated Hamiltonian gradient samples using the following Algorithm 1.

ALGORITHM 1: L-HNN Training

```
1. HNN parameters $\theta$; Training data $\{\theta, \mathbf{u}, \rho, \mathbf{p}\}$
2. Evaluate gradients of training data $\{\frac{d\theta}{dt}, \frac{d\mathbf{u}}{d\boldsymbol{t}}, \frac{d\rho}{dt}, \frac{d\mathbf{p}}{d\boldsymbol{t}}\}$
3. Initialize L-HNN parameters $\theta$
4. Computer L-HNN Hamiltonian $H_\theta$
5. Compute $H_\theta$ gradients $\frac{\partial H}{\partial \rho}, \frac{\partial H}{\partial \mathbf{p}}, \frac{\partial H}{\partial \boldsymbol{\theta}}, \frac{\partial H}{\partial \mathbf{u}}$
6. Compute loss $\mathcal{L} = \left\|\frac{\partial H}{\partial \rho} - \frac{d\theta}{dt}\right\|^2 + \left\|\frac{\partial H}{\partial \mathbf{p}} - \frac{d\mathbf{u}}{d\boldsymbol{t}}\right\|^2 + \left\|\frac{\partial H}{\partial \boldsymbol{\theta}} - \frac{d\rho}{dt}\right\|^2 + \left\|\frac{\partial H}{\partial \mathbf{u}} - \frac{d\mathbf{p}}{d\boldsymbol{t}}\right\|^2$
7. Minimize $\mathcal{L}$ w.r.t. $\theta$ using Adam optimizer
```

C. *Sampling*

During sampling, we continue the use of No-U-Turn Sampling (NUTS) used by [3]. The procedure follows traditional HMC just with the Hamiltonian gradients inferred using the L-HNN based on Algorithm-2.

ALGORITHM 2: L-HNN Velocity-Verlet Integration

```
   Inputs: Steps ($N$), final time ($T$)
1. $\Delta t = \frac{T}{N}$
2. Initialize $[\theta, \mathbf{u}, \rho, \mathbf{p}\ ]$ at $t = 0$
3. For $j = 0$ to $N - 1$:
4.     Compute HNN gradients at current state
5.     Half-step momentum updates:
           $\rho \leftarrow \rho - \frac{\Delta t}{2} \cdot \frac{\partial H}{\partial \theta}, \mathbf{p} \leftarrow \mathbf{p} - \frac{\Delta t}{2} \cdot \frac{\partial H}{\partial \mathbf{u}}$
6.     Full-step position updates:
           $\theta \leftarrow \theta - \Delta t \cdot \rho, \mathbf{u} \leftarrow \mathbf{u} - \Delta t \cdot \mathbf{p}$
7.     Compute HNN gradients at new position state
8.     Final Half-step momentum updates:
           $\rho \leftarrow \rho - \frac{\Delta t}{2} \cdot \frac{\partial H}{\partial \theta}, \mathbf{p} \leftarrow \mathbf{p} - \frac{\Delta t}{2} \cdot \frac{\partial H}{\partial \mathbf{u}}$
9. End For
```

ALGORITHM 3: L-HNN in PM_HMC with NUTS

```
    Inputs: Number of samples ($M$), step size ($\epsilon$), max
    depth ($J_{max}$), Hamiltonian $H$ over $(\theta, \mathbf{u}, \rho, \mathbf{p})$, Failsafe
    threshold (hnn_threshold) before falling back to
    numeric leapfrog
1.  Initialize initial sample $[\theta^0, \mathbf{u^0}]$
2.  For $i = 1$ to $M$:
3.      Sample new momenta: $\rho^0 \sim \mathcal{N}(0, I), \mathbf{p}^0 \sim \mathcal{N}(0, I)$
4.      Compute current Hamiltonian $H(z(0))$ with
         $z(0) = (\theta^{i-1}, \rho^0, \mathbf{u^{i-1}}, \mathbf{p^0})$
5.      Draw slice-sampling threshold $\log r$ where
          $r \sim \text{Uniform}\left[0, \exp\left(-H(z(0))\right)\right]$
6.      Build NUTS tree:
          •  Initialize left/right states $z^- = z^+ = z(0)$,
             depth j=0
          •  While not stopped and $j < J_{max}$:
               o  Double the tree in random direction
                  $v \in \{-1,1\}$
               o  Integrate trajectory using Alg. 2
               o  Fallback to manual leapfrog if
                  $H(z) + \log r > hnn\_threshold$
               o  Stop if NUTS reversal condition met
               o  $j += 1$
7.       Choose candidate $\theta^*$ uniformly from leaf of tree:
8.       Compute Metropolis acceptance probability
           $\alpha = \min\left(1, \exp\left(H(z(0)) - H(z^*)\right)\right)$
           accept with probability $\alpha$, otherwise retain
           $\theta^{i-1}, \mathbf{u^{i-1}}$
9.  End For
```

III. Experiment Setup & Results

In this section we investigate the effectiveness of our solution. For our experiments we start with the setting listed in section 4.3 of [1]. Specifically let $Y_{ij} \sim \text{Bernoulli}(p_{ij})$ be the logistic link function with $\text{logit}(p_{ij}) = X_i + Z_{ij}^{\text{T}}\beta$, where Y_{ij} denotes the j^{th} observation for the i^{th} subject, β is a vector of fixed effects, and X_i to be random effects for subject i. Z_{ij} is the covariate vector associated with the observation j of subject i. We let $X_i \overset{\text{i.i.d.}}{\sim} \sum_{j=1}^{K} \omega_j \mathcal{N}(\mu_j, \lambda_j^{-1})$ and set $K = 2, \mu_1 = 0, \mu_2 = 3, \lambda_1 = 10, \lambda_2 = 3, \omega_1 = 0.8$. Both β and Z_{ij} were generated from standard normal distributions. The parametrization of θ is as the following:

$$\theta = (\beta^{\text{T}}, \mu_1, \mu_2, \log\lambda_1, \log\lambda_2, \text{logit}(\omega_1))^{\text{T}}$$

with each subject embedded in 4 dimensions. Lastly, following [3] we set the hnn_threshold to be 10. All results were benchmarked on an Intel ® Xeon (R) W-2135 @ 3.70GHz with a RTX 2080 Ti @11GB VRAM.

A. *Training of the HNN*

To measure the generalizability of our method for experiments below we train a single L-HNN based on the above setting for 200 samples with a 10% test split. Each consist of a sample length of 30 and a step size h of 0.03. The choice of 0.03 is due to L-HNN being unstable when $h \geq 0.05$ for ≥ 500 training samples during our testing. Investigating the specific training dynamics of h against L-HNN's gradient stability is a promising future research direction but goes beyond the scope of research for this report.

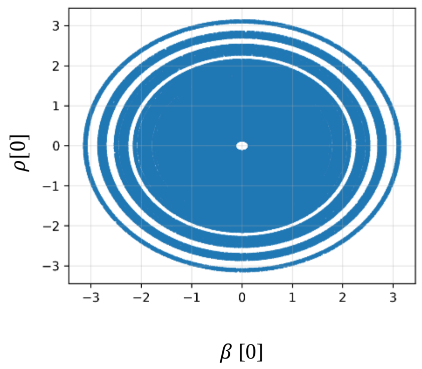

Figure 1: Phase plot for β [0] against ρ[0]

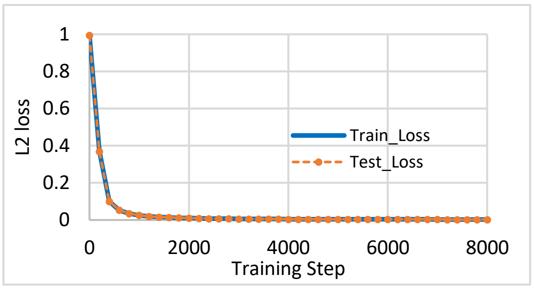

Figure. 2: L-HNN Training Loss

We start by ensuring the HNN is adequately trained. Specifically, Fig. 1 shows the phase plot for the first dimension of fixed effects parameter β governed by a standard normal prior, against the first dimension of the independently sampled ρ (also from a standard normal). It can be seen the generated training samples cover all regions around the mode. Next, Fig. 2 shows the evolution of the training and test loss, after 8000 training steps. Both losses showing a typical exponential decreasing shape indicate convergence with minimal overfitting. This suggests that the L-HNN is trained effectively.

B. *Computational Efficiency*

One of the most important advantages of utilizing HNNs is its ability to reduce computational costs due to repeated numerical integrating. We benchmark this by disabling the hnn_threshold in Algorithm 3 and force the trained L-HNN and numeric velocity-Verlet leapfrog (VVL) to each generate 100 samples at different N (importance sample) values. For each procedure we collect the acceptance probability averaged across each of the 100 iterations, as well as, the total elapsed time recorded using Ubuntu's *time* command.

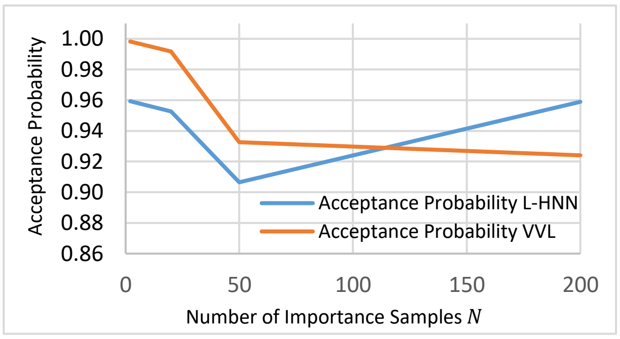

Figure. 3: Acceptance Probability for L-HNN and VVL

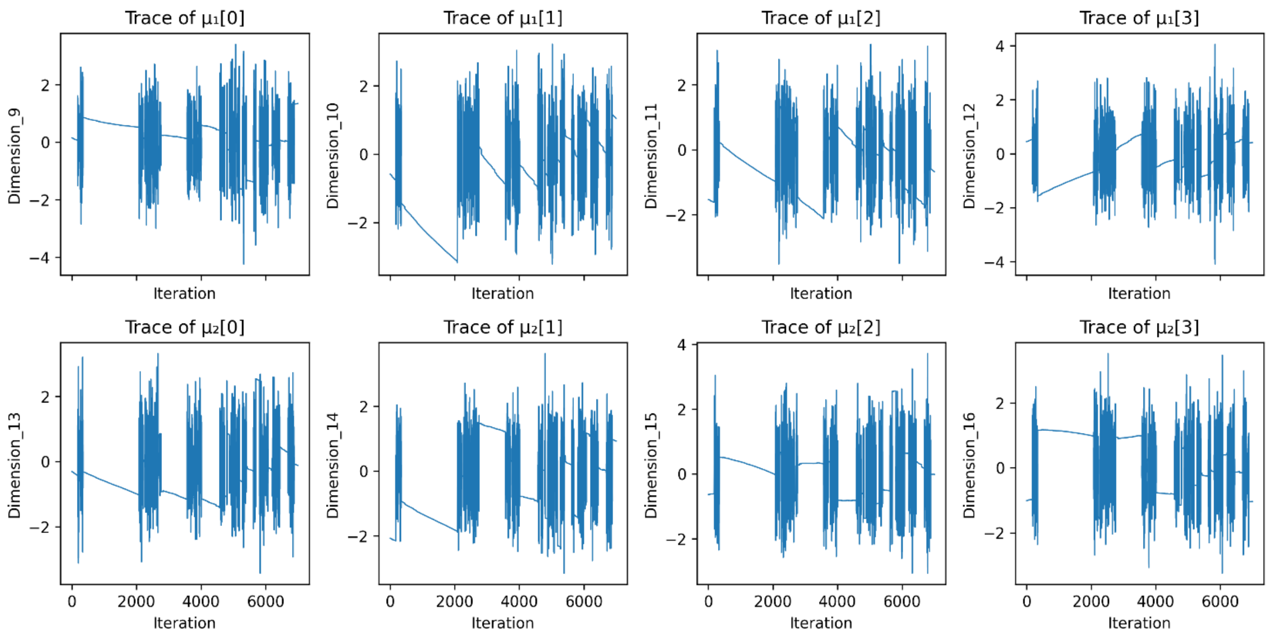

Figure 4: Trace Plots for parameters μ_1 and μ_2

N	Total Elapsed Time (s)	
	L-HNN	VVL
2	28	1499
20	97	6774
50	250	15950
200	544	29650

Table 1[1]: Computation Time for HNN vs VVL

From Figure 3 and Table 1 it can be seen that L-HNN very significantly reduces the computational load by up 98.43%. This is in addition to the fact that L-HNN shows similar acceptance probabilities across all tested values of N. Note that Strang splitting was also tested. However, under the same setting it's acceptance probability was consistently $\leq 10\%$, This prevented the sampling Markov chain from escaping its initialized state and is thus omitted from our analysis.

From Figure 3, there are two additional observations. Firstly as N increases L-HNN's acceptance probability surpasses VVL. We attribute this behavior to the well documented behavior of symplectic numerical integrators [4] where discrete leapfrog steps may not perfectly conserving energy. This causes integration errors to accumulate, leading to energy drift and decreasing the acceptance probability. Fortunately, due to L-HNNs learn the energy function directly rather than following fixed-step numerical updates, it does not suffer from the accumulation of discretization. This is another major advantage of HNN based approaches over symplectic numerical integrators especially when the dimension is large.

Secondly, [1] documents that VVL's acceptance probability degrades significantly for large $N \geq 100$. However, in our similar setup the deterioration is not nearly as significant. We hypothesize that this is due to a much smaller step size of 0.03 compared to 0.35 which once again reduces the accumulation of numerical integration errors.

Lastly, Figure 4 showcases the trace plots for parameters μ_1 and μ_2 for a large 8000 sample task using L-HNN enhanced PM-HMC with hnn_threshold enabled. To minimize the accumulative effects of discretization integration errors caused by VVL, we set N to 2.

Compared to section 4.3 of [1], we observe two distinct trace patterns: regions of smooth trajectories correspond to L-HNN's learned gradients, while rapid, zigzag transitions correspond to VVL's numerical integration. In our testing, the smooth L-HNN updates still allow the chain to move broadly within the parameter space, ensuring that the posterior is adequately explored while reducing reliance on computationally expensive numerical integration. Flat regions, which indicate that the chain is stuck, are briefly observed for iterations > 4000. However, the sampler quickly corrects them by transitioning to numerical integration, preventing long-term stagnation. This dynamic adaptation suggests that L-HNN effectively maintains ergodicity, allowing the Markov chain to explore the target distribution without suffering from persistent local trapping. Thus, we in addition to computational benefits, we conclude that L-HNN successfully enhances sampling efficiency by balancing learned dynamics with robust numerical correction, ensuring stable Hamiltonian trajectory integration while preserving the theoretical guarantees of Hamiltonian Monte Carlo (HMC) for posterior exploration.

[1] For N=200 VVL and L-HNN's acceptance probability is based on **40** samples with elapsed time linearly interpolated due to VVL requiring $\geq$ 12hrs for 100 samples.

References

[1] Alenlov, J., Doucet, A., & Lindsten, F. (2021). Pseudo-Marginal Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *22*(141), 1–45. Retrieved from http://jmlr.org/papers/v22/19-486.html

[2] Osmundsen, K., Kleppe, T., & Liesenfeld, R. (01 2018). Pseudo-Marginal Hamiltonian Monte Carlo with Efficient Importance Sampling. *SSRN Electronic Journal*. doi:10.2139/ssrn.3304077

[3] Dhulipala, S. L. N., Che, Y., & Shields, M. D. (2023). Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling. *Journal of Computational Physics*, *492*, 112425. doi:10.1016/j.jcp.2023.112425

[4] Wu, X., Huang, TY., Zhang, H. *et al.* A note on the algorithm of symplectic integrators. *Astrophysics and Space Science* **283**, 53–65 (2003). doi:10.1023/A:1021268602971