Part-II Report on "Hamiltonian Neural Network and Hamiltonian Monte Carlo" Topic

Yuxi Cai *caiyuxi@connect.hku.hk*

February 3, 2025

Contents

1 Algorithms and problem formulation

This section introduces how the algorithms in Dhulipala et al. (2023) are adapted for the pseudo-marginal HMC in Alenlöv et al. (2021) as well as the sampling problem formulation for the generalized linear mixed model. To be consistent, we follow the notations in Alenlöv et al. (2021).

1.1 Hamiltonian neural network training

Similar to Dhulipala et al. (2023), we can design neural networks to learn the gradients of pseudo-marginal Hamiltonian. To be more general, we can extend the Hamiltonian in Eq.(10) of Alenlöv et al. (2021) by considering a non-identity covariance matrix for $\boldsymbol{\rho}$, i.e., $\boldsymbol{\rho} \sim N(\mathbf{0}_d, \boldsymbol{M})$, which gives

$$H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}) = -\log p(\boldsymbol{\theta}) - \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u}) + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{M}^{-1}\boldsymbol{\rho} + \frac{1}{2}(\boldsymbol{u}^\top \boldsymbol{u} + \boldsymbol{p}^\top \boldsymbol{p}). \tag{1}$$

Given the training data $\{\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}\}$, we first calculate the numerical gradients

$$\begin{pmatrix} \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} \\ \frac{\mathrm{d}\boldsymbol{\rho}}{\mathrm{d}t} \\ \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \\ \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}^{-1}\boldsymbol{\rho} \\ \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u}) \\ \boldsymbol{p} \\ -\boldsymbol{u} + \nabla_{\boldsymbol{u}} \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u}) \end{pmatrix}. \tag{2}$$

Then for some neural network with parameters $\boldsymbol{\omega}$, it can be trained to minimize the loss function:

$$L = \left\| \frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\theta}} + \frac{\mathrm{d}\boldsymbol{\rho}}{\mathrm{d}t} \right\|_2 + \left\| \frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\rho}} - \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} \right\|_2 + \left\| \frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{u}} + \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} \right\|_2 + \left\| \frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{p}} - \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \right\|_2. \tag{3}$$

The algorithm for network training can be found in Algorithm 1.

Algorithm 1 Hamiltonian neural network training

```
1: HNN parameters: $\boldsymbol{\omega}$; Training data: $\{\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}\}$
2: Evaluate gradients of the training data $\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t}$, $\frac{\mathrm{d}\boldsymbol{\rho}}{\mathrm{d}t}$, $\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t}$, and $\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t}$
3: Initialize HNN parameters $\boldsymbol{\omega}$
4: Compute HNN Hamiltonian $H_{\boldsymbol{\omega}}$
5: Compute gradients of $H_{\boldsymbol{\omega}}$, $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\theta}}$, $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\rho}}$, $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{u}}$, and $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{p}}$
6: Compute loss $L$ as in Eq.(3)
7: Minimize $L$ with respect to $\boldsymbol{\omega}$
```

1.2 Integration with network gradients

Alenlöv et al. (2021) gives their integrator in the special case of $\boldsymbol{M} = \boldsymbol{I}_d$, while the more general case has been derived in Osmundsen et al. (2021). Specifically, the one-step integrator with step size h gives the updating scheme from $\boldsymbol{\Theta}[l] = (\boldsymbol{\theta}[l], \boldsymbol{\rho}[l], \boldsymbol{u}[l], \boldsymbol{p}[l])$ to $\boldsymbol{\Theta}[l+1] = (\boldsymbol{\theta}[l+1], \boldsymbol{\rho}[l+1], \boldsymbol{u}[l+1], \boldsymbol{p}[l+1])$ by the following equations:

$$
\begin{aligned}
\boldsymbol{\theta}[l+0.5] &= \boldsymbol{\theta}[l] + \frac{h}{2}\boldsymbol{M}^{-1}\boldsymbol{\rho}[l] \\
\boldsymbol{u}[l+0.5] &= \cos(\frac{h}{2})\boldsymbol{u}[l] + \sin(\frac{h}{2})\boldsymbol{p}[l] \\
\boldsymbol{p}^{*} &= \cos(\frac{h}{2})\boldsymbol{p}[l] - \sin(\frac{h}{2})\boldsymbol{u}[l] \\
\boldsymbol{p}^{**} &= \boldsymbol{p}^{*} + h\nabla_{\boldsymbol{u}}\log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}[l+0.5], \boldsymbol{u}[l+0.5]) \\
\boldsymbol{\rho}[l+1] &= \boldsymbol{\rho}[l] + h\nabla_{\boldsymbol{\theta}}\{\log p(\boldsymbol{\theta}[l+0.5]) + \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}[l+0.5], \boldsymbol{u}[l+0.5])\} \\
\boldsymbol{\theta}[l+1] &= \boldsymbol{\theta}[l+0.5] + \frac{h}{2}\boldsymbol{M}^{-1}\boldsymbol{\rho}[l] \\
\boldsymbol{u}[l+1] &= \cos(\frac{h}{2})\boldsymbol{u}[l+0.5] + \sin(\frac{h}{2})\boldsymbol{p}^{**} \\
\boldsymbol{p}[l+1] &= \cos(\frac{h}{2})\boldsymbol{p}^{**} - \sin(\frac{h}{2})\boldsymbol{u}[l+0.5].
\end{aligned}
\tag{4}
$$

Note that when $\boldsymbol{M} = \boldsymbol{I}_d$, the equations above are equivalent to those in Appendix A of Alenlöv et al. (2021). To replace numerical gradients, we substitue the terms $\nabla_{\boldsymbol{u}} \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}[l+0.5], \boldsymbol{u}[l+0.5])$ and $\nabla_{\boldsymbol{\theta}}\{\log p(\boldsymbol{\theta}[l+0.5]) + \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}[l+0.5], \boldsymbol{u}[l+0.5])\}$ with $-\partial H_{\boldsymbol{\omega}}/\partial \boldsymbol{u} + \boldsymbol{u}|_{\boldsymbol{u}=\boldsymbol{u}[l+0.5]}$ and $-\partial H_{\boldsymbol{\omega}}/\partial \boldsymbol{\theta}|_{\boldsymbol{\theta}=\boldsymbol{\theta}[l+0.5]}$, respectively. The complete integration with HNN gradients can be found in Algorithm 2.

Algorithm 2 Integration with gradients of Hamiltonian neural networks

```
1:  Trained neural networks with parameters $\boldsymbol{\omega}$; Initial values: $\boldsymbol{\Theta}(0) = \{\boldsymbol{\theta}(0), \boldsymbol{\rho}(0), \boldsymbol{u}(0), \boldsymbol{p}(0)\}$; Step size: $h$. Steps: $L$; Covariance matrix: $\boldsymbol{M}$
2:  for $l = 0$ to $L - 1$ do
3:      $\boldsymbol{\theta}[l+0.5] = \boldsymbol{\theta}[l] + \frac{h}{2}\boldsymbol{M}^{-1}\boldsymbol{\rho}[l]$
4:      $\boldsymbol{u}[l+0.5] = \cos(\frac{h}{2})\boldsymbol{u}[l] + \sin(\frac{h}{2})\boldsymbol{p}[l]$
5:      $\boldsymbol{p}^* = \cos(\frac{h}{2})\boldsymbol{p}[l] - \sin(\frac{h}{2})\boldsymbol{u}[l]$
6:      Compute HNN output gradients $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\theta}[l+0.5]}$ and $\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{u}[l+0.5]}$
7:      $\boldsymbol{p}^{**} = \boldsymbol{p}^* - h \cdot (\frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{u}[l+0.5]} - \boldsymbol{u}[l+0.5])$
8:      $\boldsymbol{\rho}[l+1] = \boldsymbol{\rho}[l] - h \cdot \frac{\partial H_{\boldsymbol{\omega}}}{\partial \boldsymbol{\theta}[l+0.5]}$
9:      $\boldsymbol{\theta}[l+1] = \boldsymbol{\theta}[l+0.5] + \frac{h}{2}\boldsymbol{M}^{-1}\boldsymbol{\rho}[l]$
10:     $\boldsymbol{u}[l+1] = \cos(\frac{h}{2})\boldsymbol{u}[l+0.5] + \sin(\frac{h}{2})\boldsymbol{p}^{**}$
11:     $\boldsymbol{p}[l+1] = \cos(\frac{h}{2})\boldsymbol{p}^{**} - \sin(\frac{h}{2})\boldsymbol{u}[l+0.5]$
12: end for
```

1.3 Pseudo-marginal HMC with NUTS and HNNs

This subsection illustrates how to adapt HNNs and efficient NUTS with online error monitoring in Dhulipala et al. (2023) for the pseudo-marginal HMC in Alenlöv et al. (2021). The main differences are: (i) the Hamiltonian should be changed to the one in Eq.(1); (ii) four variables $\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}$ instead of the two should be considered; (iii) different integrators are used; and (iv) the stop criterion needs to be modified. In particular, the stop criterion is to check whether the states $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-$, and $\boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+$, which are associated with the leftmost and rightmost leaves of some subtree make the U-turn. To align with Eq.(19) in Dhulipala et al. (2023), we can let $\tilde{\boldsymbol{q}} := (\boldsymbol{\theta}^\top, \boldsymbol{u}^\top)^\top$ and $\tilde{\boldsymbol{p}} := (\boldsymbol{\rho}^\top, \boldsymbol{p}^\top)^\top$, where $\tilde{\boldsymbol{q}}$ and $\tilde{\boldsymbol{p}}$ are the position and momentum variables in Dhulipala et al. (2023). Note that such parameterization is assumed to be reasonable as it does not break the original properties of pseudo-marginal HMC. Therefore, to check whether $(\tilde{\boldsymbol{q}}^+ - \tilde{\boldsymbol{q}}^-)^\top \tilde{\boldsymbol{p}}^- < 0$ and$(\tilde{\boldsymbol{q}}^+ - \tilde{\boldsymbol{q}}^-)^\top \tilde{\boldsymbol{p}}^+ < 0$, it is equivalent to see whether

$$(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^+ + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^+ < 0 \quad \text{and} \quad (\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^- + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^- < 0.$$

The details of pseudo-marginal HMC with efficient NUTS and HNNs are included in Algorithms 3 and 4.

1.4 Problem formulation for generalized linear mixed model

As described in Section 4.3 of Alenlöv et al. (2021), the generalized linear mixed model (GLMM) that we want to make inference is defined as:

$$\boldsymbol{Y}_{ij} \sim \text{Bernoulli}(p_{ij}),\ \text{logit}(p_{ij}) = X_i + \boldsymbol{Z}_{ij}^\top \boldsymbol{\beta},\ X_i \overset{\text{i.i.d.}}{\sim} wN(\mu_1, \lambda_1^{-1}) + (1-w)N(\mu_2, \lambda_2^{-1}),$$

for $1 \le i \le T$, $1 \le j \le n$ and $\boldsymbol{Z}_{ij}, \boldsymbol{\beta} \in \mathbb{R}^p$. The target parameters are $\boldsymbol{\theta} = (\boldsymbol{\beta}^\top, \mu_1, \mu_2, \log(\lambda_1), \log(\lambda_2), \text{logit}(w))^\top$ and we assume $N(0, 100)$ as the prior for each component of $\boldsymbol{\theta}$. Therefore, the Hamiltonian can be written as

$$\begin{aligned} H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}) &= -\log p(\boldsymbol{\theta}) - \log \hat{p}(\boldsymbol{Y}|\boldsymbol{\theta}, \boldsymbol{u}) + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{M}^{-1}\boldsymbol{\rho} + \frac{1}{2}(\boldsymbol{u}^\top \boldsymbol{u} + \boldsymbol{p}^\top \boldsymbol{p}) \\ &= \underbrace{\frac{d}{2}\log(2\pi) + \frac{1}{2}\log(\det(\boldsymbol{\Sigma})) + \frac{1}{2}\boldsymbol{\theta}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\theta}}_{-\log p(\boldsymbol{\theta})} \underbrace{- \sum_{i=1}^{T}\sum_{j=1}^{n} \log\left\{\frac{1}{N}\sum_{l=1}^{N} \frac{g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) f_{\boldsymbol{\theta}}(X_{il})}{q_{\boldsymbol{\theta}}(X_{il})}\right\}}_{-\log \hat{p}(\boldsymbol{Y}|\boldsymbol{\theta},\boldsymbol{u})} \\ &\quad + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{M}^{-1}\boldsymbol{\rho} + \frac{1}{2}(\boldsymbol{u}^\top \boldsymbol{u} + \boldsymbol{p}^\top \boldsymbol{p}), \end{aligned}$$

where $\boldsymbol{\Sigma} = 100\boldsymbol{I}_d$, $g_{\boldsymbol{\theta}}(Y_{ij}|X_{il})$ is the probability density function (p.d.f) of Bernoulli distribution, $f_{\boldsymbol{\theta}}(X_{il})$ is the p.d.f of the Gaussian mixture distribution, and $q_{\boldsymbol{\theta}}(X_{il})$ is the p.d.f of $N(0, 3^2)$. Given that $\boldsymbol{u} = \{u_{il}\}_{1 \le i \le T, 1 \le l \le N}$ and $u_{il} \sim N(0, 1)$, we can construct X_{il} by $X_{il} = 3u_{il}$. Since $T = 500, n = 6, p = 8, N = 128$, we have $\boldsymbol{\theta}, \boldsymbol{\rho} \in \mathbb{R}^{13}$ and $\boldsymbol{u}, \boldsymbol{p} \in \mathbb{R}^{64000}$.

Algorithm 3 Pseudo-marginal HMC with NUTS and HNN (main loop)

```
 1: Hamiltonian: $H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p})$ as in (1); Samples: $K$; Starting sample: $\{\boldsymbol{\theta}^0, \boldsymbol{\rho}^0, \boldsymbol{u}^0, \boldsymbol{p}^0\}$; Step size: $h$; Threshold for numerical gradients: $\Delta_{\max}^{\text{num}}$; Threshold for HNNs: $\Delta_{\max}^{\text{hnn}}$; Number of samples from numerical gradients: $N_{\text{ng}}$; Covariance matrix: $\boldsymbol{M}$
 2: Initialize $\mathbb{1}_{\text{num}} = 0$, $n_{\text{num}} = 0$
 3: for $i = 1$ to $K$ do
 4:     $\boldsymbol{\rho}(0) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{M})$, $\boldsymbol{p}(0) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_D)$
 5:     $\boldsymbol{\theta}(0) = \boldsymbol{\theta}^{i-1}$, $\boldsymbol{u}(0) = \boldsymbol{u}^{i-1}$
 6:     $u \sim \text{Uniform}\{0, \exp\{-H(\boldsymbol{\theta}(0), \boldsymbol{\rho}(0), \boldsymbol{u}(0), \boldsymbol{p}(0))\}\}$
 7:     Initialize $\boldsymbol{\theta}^- = \boldsymbol{\theta}(0)$, $\boldsymbol{\theta}^+ = \boldsymbol{\theta}(0)$, $\boldsymbol{\rho}^- = \boldsymbol{\rho}(0)$, $\boldsymbol{\rho}^+ = \boldsymbol{\rho}(0)$, $\boldsymbol{u}^- = \boldsymbol{u}(0)$, $\boldsymbol{u}^+ = \boldsymbol{u}(0)$, $\boldsymbol{p}^- = \boldsymbol{p}(0)$, $\boldsymbol{p}^+ = \boldsymbol{p}(0)$, $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$, $\boldsymbol{u}^i = \boldsymbol{u}^{i-1}$, $j = 0$, $n = 1$, $s = 1$
 8:     if $\mathbb{1}_{\text{num}} = 1$ then
 9:         $n_{\text{num}} \leftarrow n_{\text{num}} + 1$
10:     end if
11:     if $n_{\text{num}} = N_{\text{num}}$ then
12:         $\mathbb{1}_{\text{num}} = 0$, $n_{\text{num}} = 0$
13:     end if
14:     while $s = 1$ do
15:         Choose direction $\nu_j \sim \text{Uniform}(\{-1, 1\})$
16:         if $v_j = -1$ then
17:             $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, \ldots, \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', n', s', \mathbb{1}_{\text{num}} = \textbf{BuildTree}(\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, u, \nu_j, j, h, \mathbb{1}_{\text{num}})$
18:         else
19:             $\ldots, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', n', s', \mathbb{1}_{\text{num}} = \textbf{BuildTree}(\boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, u, \nu_j, j, h, \mathbb{1}_{\text{num}})$
20:         end if
21:         if $s' = 1$ then
22:             With probability $\min\{1, \frac{n'}{n}\}$, set $\{\boldsymbol{\theta}^i, \boldsymbol{u}^i\} \leftarrow \{\boldsymbol{\theta}', \boldsymbol{u}'\}$
23:         end if
24:         $n \leftarrow n + n'$
25:         $s \leftarrow s' \mathbb{1}\{(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^+ + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^+ \geq 0\} \mathbb{1}\{(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^- + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^- \geq 0\}$
26:         $j \leftarrow j + 1$
27:     end while
28: end for
```

Algorithm 4 Pseudo-marginal HMC with NUTS and HNN (build tree function)

```
1: function BuildTree($\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}, u, \nu, j, h, \mathbb{1}_{\text{num}}$)
2:     if $j = 0$ then
3:         Base case taking one integration step
4:         $\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}' \leftarrow$ Algorithm 2 with initial conditions: $\boldsymbol{\Theta}(0) = \{\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}\}$, Steps: $\nu$, Step size: $h$
5:         $\mathbb{1}_{\text{num}} \leftarrow \mathbb{1}_{\text{num}}$ or $\mathbb{1}\{H(\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}') + \ln u > \Delta_{\max}^{\text{hnn}}\}$
6:         $s' \leftarrow \mathbb{1}\{H(\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}') + \ln u \leq \Delta_{\max}^{\text{hnn}}\}$
7:         if $\mathbb{1}_{\text{num}} = 1$ then
8:             $\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}' \leftarrow$ Numerical integration with initial conditions: $\boldsymbol{\Theta}(0) = \{\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}\}$, Steps: $v$, Step size: $h$
9:             $s' \leftarrow \mathbb{1}\{H(\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}') + \ln u \leq \Delta_{\max}^{\text{num}}\}$
10:        end if
11:        $n' \leftarrow \mathbb{1}\{u \leq \exp\{-H(\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}')\}\}$
12:        return $\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', n', s', \mathbb{1}_{\text{num}}$
13:    else
14:        Recursion to build left and right sub-trees
15:        $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', n', s', \mathbb{1}_{\text{num}}$ = BuildTree($\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}, u, \nu_j, j - 1, h, \mathbb{1}_{\text{num}}$)
16:        if $s' = 1$ then
17:            if $v_j = -1$ then
18:                $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, \ldots, \boldsymbol{\theta}'', \boldsymbol{\rho}'', \boldsymbol{u}'', \boldsymbol{p}'', n'', s'', \mathbb{1}_{\text{num}}$ = BuildTree($\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, u, \nu_j, j - 1, h, \mathbb{1}_{\text{num}}$)
19:            else
20:                $\ldots, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, \boldsymbol{\theta}'', \boldsymbol{\rho}'', \boldsymbol{u}'', \boldsymbol{p}'', n'', s'', \mathbb{1}_{\text{num}}$ = BuildTree($\boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, u, \nu_j, j - 1, h, \mathbb{1}_{\text{num}}$)
21:            end if
22:            With probability $\frac{n''}{n'+n''}$, set $\{\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}'\} \leftarrow \{\boldsymbol{\theta}'', \boldsymbol{\rho}'', \boldsymbol{u}'', \boldsymbol{p}''\}$
23:            $s' \leftarrow s''\mathbb{1}\{(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^+ + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^+ \geq 0\}\mathbb{1}\{(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)^\top \boldsymbol{\rho}^- + (\boldsymbol{u}^+ - \boldsymbol{u}^-)^\top \boldsymbol{p}^- \geq 0\}$
24:            $n' \leftarrow n' + n''$
25:        end if
26:        return $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{u}^-, \boldsymbol{p}^-, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{u}^+, \boldsymbol{p}^+, \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{u}', \boldsymbol{p}', n', s', \mathbb{1}_{\text{num}}$
27:    end if
28: end function
```

2 Training difficulties and attempted solutions

This section delves into the challenges faced during the training and sampling process while replicating the results of the GLMM as described in Section 4.3 of Alenlöv et al. (2021). Moreover, it also outlines the strategies and solutions attempted to overcome these difficulties.

2.1 Implementation of Hamiltonian and its gradients

Problems Initially, I used `tfp.distributions` and `tfp.math.value_and_gradient` to calculate the p.d.f of probability distributions and the numerical gradients of Hamiltonian, respectively. However, I noticed that when generating relatively long trajectories, even with a small step size and different initial samples, the `nan` values would arise almost surely in the Hamiltonian value and its gradients. Moreover, many of them were caused by the overflow errors in exponentiation. This hinders the generation of long trajectories for Hamiltonian neural network training, and also brings problems to sampling with NUTS as the Hamiltonian becomes `nan`.

Attempted solutions Several solutions have been tried sequentially to solve this numerical problem and the `nan` values have been successfully avoided. First, I try to maintain most of the calculations in log scale and use the trick to improve the calculations of the so-called "log-sum-exp":

$$\begin{aligned}\text{log-sum-exp}(u, v) &:= \log(\exp(u) + \exp(v)) \\ &= \max(u, v) + \log(\exp(u - \max(u, v)) + \exp(v - \max(u, v))). \end{aligned} \tag{5}$$

In the GLMM inference problem, the trick is most useful for deriving $\log \hat{p}(\boldsymbol{Y}|\boldsymbol{\theta}, \boldsymbol{u})$, which renders

$$\log \left\{ \frac{1}{N} \sum_{l=1}^{N} \frac{g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) f_{\boldsymbol{\theta}}(X_{il})}{q_{\boldsymbol{\theta}}(X_{il})} \right\}$$

$$
\begin{aligned}
&= \log\left\{\sum_{l=1}^{N} \exp\left\{\log g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) + \log f_{\boldsymbol{\theta}}(X_{il}) - \log q_{\boldsymbol{\theta}}(X_{il})\right\}\right\} - \log N \\
&= \log\left\{\sum_{l=1}^{N} \exp\{\log g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) + \log f_{\boldsymbol{\theta}}(X_{il}) - \log q_{\boldsymbol{\theta}}(X_{il}) - C_{\max}\}\right\} + C_{\max} - \log N,
\end{aligned}
$$

and

$$C_{\max} = \max_{l}\left\{\log g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) + \log f_{\boldsymbol{\theta}}(X_{il}) - \log q_{\boldsymbol{\theta}}(X_{il})\right\}.$$

Similarly, the trick is also applied to derive the log p.d.f of Gaussian mixture:

$$
\begin{aligned}
\log f_{\boldsymbol{\theta}}(X_{il}) &= \log\left\{w\phi(X_{il};\mu_1,\lambda_1^{-1}) + (1-w)\phi(X_{il};\mu_2,\lambda_2^{-1})\right\} \\
&= \log\left\{\exp\left\{\log w + \log\phi(X_{il};\mu_1,\lambda_1^{-1}) - C'_{\max}\right\}\right. \\
&\qquad\qquad \left. + \exp\left\{\log(1-w) + \log\phi(X_{il};\mu_2,\lambda_2^{-1}) - C'_{\max}\right\}\right\} + C'_{\max},
\end{aligned}
$$

where $C'_{\max} = \max\{\log w + \log\phi(X_{il};\mu_1,\lambda_1^{-1}), \log(1-w) + \log\phi(X_{il};\mu_2,\lambda_2^{-1})\}$ and $\phi(X;\mu,\lambda^{-1})$ denotes the p.d.f of $N(\mu,\lambda^{-1})$ at X.

Secondly, I replace the auto gradient calculation of `tfp.math.value_and_gradient` with my own gradient derivation. The implementation may cost slightly more time, but it indeed improves the numerical stability of gradient calculation. Specifically, from Eq.(14) and (15) of Alenlöv et al. (2021), we have

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log \hat{p}(\boldsymbol{Y}|\boldsymbol{\theta},\boldsymbol{u}) &= \sum_{i=1}^{T}\sum_{j=1}^{n}\sum_{l=1}^{N} \frac{\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il})}{\sum_{l=1}^{N}\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il})} \nabla_{\boldsymbol{\theta}} \log \omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}) \\
\nabla_{u_{il}} \log \hat{p}(\boldsymbol{Y}|\boldsymbol{\theta},\boldsymbol{u}) &= \frac{\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il})}{\sum_{l=1}^{N}\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il})} \nabla_{u_{il}} \log \omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}).
\end{aligned}
$$

Further, since $\nabla_{\boldsymbol{\theta}} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}) = (\nabla_{\boldsymbol{\beta}} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}), \nabla_{\mu_1} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}), \nabla_{\mu_2} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}),$ $\nabla_{\log(\lambda_1)} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}), \nabla_{\log(\lambda_2)} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}), \nabla_{\mathrm{logit}(w)} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}))$, each component can be derived separately: for $k = 1,2$,

$$\nabla_{\boldsymbol{\beta}} \log\omega_{\boldsymbol{\theta}}(Y_{ij},u_{il}) = (Y_{ij}(1-p_{ij}) - (1-Y_{ij})p_{ij})\boldsymbol{Z_{ij}}$$

$$\nabla_{\mu_k} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il}) = \frac{w_k \lambda_k^{1/2}/\sqrt{2\pi} \cdot \exp\{-\lambda_k (X_{il} - \mu_k)^2/2\} \cdot \lambda_k (X_{il} - \mu_k)}{f_{\boldsymbol{\theta}}(X_{il})}$$

$$\nabla_{\log(\lambda_k)} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il}) = \frac{w_k \lambda_k^{1/2}/\sqrt{2\pi} \cdot \exp\{-\lambda_k (X_{il} - \mu_k)^2/2\} \cdot (0.5 - 0.5\lambda_k (X_{il} - \mu_k)^2)}{f_{\boldsymbol{\theta}}(X_{il})}$$

$$\nabla_{\text{logit}(w)} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il}) = \frac{\lambda_1^{1/2}/\sqrt{2\pi} \cdot \exp\{-\lambda_1 (X_{il} - \mu_1)^2/2\} - \lambda_2^{1/2}/\sqrt{2\pi} \cdot \exp\{-\lambda_2 (X_{il} - \mu_2)^2/2\}}{f_{\boldsymbol{\theta}}(X_{il})} \times w(1-w).$$

Moreover,

$$\nabla_{u_{il}} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il}) = 3(Y_{ij}(1-p_{ij}) - (1-Y_{ij})p_{ij}) - 3(\nabla_{\mu_1} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il}) + \nabla_{\mu_2} \log \omega_{\boldsymbol{\theta}}(Y_{ij}, u_{il})) + u_{il}.$$

Thirdly, a numerically stable way is adopted to calculate $\log(1+\exp(x))$ based on Mächler (2015). Specifically,

$$\log(1+\exp(x)) := \begin{cases} \exp(x) & x \leq -37, \\ \log(1+\exp(x)) & -37 < x \leq x_1 := 18, \\ x + \exp(-x) & x_1 < 18 \leq x_2 := 33.3, \\ x & x > x_2. \end{cases}$$

The method can be used when calculating the log pdf of the Bernoulli distribution:

$$\begin{aligned} \log g_{\boldsymbol{\theta}}(Y_{ij}|X_{il}) &= Y_{ij} \log p_{ijl} + (1-Y_{ij}) \log(1-p_{ijl}) \\ &= -Y_{ij} \log(1+\exp(-(X_{il} + \boldsymbol{Z}_{ij}^\top \boldsymbol{\beta}))) - (1-Y_{ij}) \log(1+\exp(X_{il} + \boldsymbol{Z}_{ij}^\top \boldsymbol{\beta})). \end{aligned}$$

2.2 Design and training of Hamiltonian neural networks

Network architecture The network structure proposed by Dhulipala et al. (2023) is multilayer perceptron (MLP):

$$\boldsymbol{x}_p = \phi(\boldsymbol{W}_p \boldsymbol{x}_{p-1} + \boldsymbol{b}_p), \quad \text{for } 1 \leq p \leq P-1,$$

$$\boldsymbol{\lambda} = \boldsymbol{W}_P \boldsymbol{x}_{P-1} + \boldsymbol{b}_P, \quad H_{\boldsymbol{\omega}} = \sum_{i=1}^{d'} \lambda_i,$$

where $\boldsymbol{x}_0 = (\boldsymbol{\theta}^\top, \boldsymbol{\rho}^\top, \boldsymbol{u}^\top, \boldsymbol{p}^\top)^\top$, $\boldsymbol{\lambda} \in \mathbb{R}^{d'}$, $\boldsymbol{W}$'s and $\boldsymbol{b}$'s are weights and biases, respectively. In practice, Dhulipala et al. (2023) used a three-layer MLP and chose d' as the dimensionality of the uncertainty space. However, in the GLMM example, the dimension of network inputs is $2d + 2TN = 128,026$ and the dimension of the uncertainty space d' is now $d + TN = 64,013$, both substantially larger than a few dozens in the examples of Dhulipala et al. (2023). Therefore, when setting the hidden size of MLP to 100, the total number of parameters in HNN will be around 2.03×10^7, easily leading to the out-of-memory problem and training difficulties such as a non-decreasing training error.

To reduce the number of parameters to a proper level and enable the efficient learning of the network, I have attempted to (i) adjust the dimension of $\boldsymbol{\lambda}$, (ii) impose l_1-regularization on weight matrices of MLPs, (iii) construct convolution-based architectures, and (iv) make the network focus on learning the gradients of target density. The details are as follows:

(i) This method is straightforward and d' is fixed to 26 across all the architecture designs, which is twice the dimension of $\boldsymbol{\theta}$.

(ii) In literature of statistics, the Lasso regularization (Tibshirani, 1996) is widely adopted for dimension reduction. In my attempt, a l_1-regularization is imposed on $\boldsymbol{W}_1 \in \mathbb{R}^{100\times128026}$, the weight matrix of the first layer, as it contributes significantly to the total number of parameters.

(iii) The convolutional layer is known to be a parameter-efficient design (Goodfellow et al., 2016), because its parameters are shared across different locations of inputs and thus do not necessarily increase the total counts with the input dimension. Since the input of HNN is a sequence, I apply multiple 1D convolutional layers to learn the calculation of Hamiltonian. Specifically, five 1D convolutional layers are used with hyperparameters (kernel size, stride size, number of filters) set as (10, 4, 10), (10, 4, 10), (7, 4, 10), (10, 4, 10), and (10, 4, 1), respectively, and the variable dimension is reduced to 122 only. The 1D zero paddings are applied whenever necessary. Then a MLP is adopted with 100 hidden neurons to get the outputs. The total number of parameters for the resulting network is 18,223, significantly smaller than that of the original design.

(iv) Recall the numerical gradients in (2), the terms $\mathrm{d}\boldsymbol{\theta}/\mathrm{d}t$ and $\mathrm{d}\boldsymbol{u}/\mathrm{d}t$ can be derived easily from the input data without evaluating the complicated target density. Nonetheless, the network may find it difficult to learn such a simple relationship as it is not explicit when only receiving the inputs and the corresponding Hamiltonian. Hence I directly set the last entry of $\boldsymbol{\lambda}$ as $\lambda_{d'} = \boldsymbol{\rho}^\top \boldsymbol{M}^{-1}\boldsymbol{\rho}/2 + (\boldsymbol{u}^\top \boldsymbol{u} + \boldsymbol{p}^\top \boldsymbol{p})/2$ and only use $(\boldsymbol{\theta}^\top, \boldsymbol{u}^\top)^\top \in \mathbb{R}^{64013}$ as the input of MLPs or convolutional layers. This guarantees the correctness of the partial derivatives $\partial H_{\boldsymbol{\omega}}/\partial \boldsymbol{\rho}$ and $\partial H_{\boldsymbol{\omega}}/\partial \boldsymbol{p}$ as well as reducing the number of network parameters with a smaller input size.

Training scheme The original training scheme of Dhulipala et al. (2023) uses a fixed learning rate and total number of epochs, which could pose challenges on hyperparameter tuning. To improve the training efficiency, I adopt a more flexible training scheme by letting the learning rate decrease by half if the validation error increases for three consecutive epochs. The training will be terminated if the learning rate is smaller than 10^{-6} or the maximum number of epochs is reached. In addition, the original implementation of Dhulipala et al. (2023) loads the entire dataset into memory. This is impractical for the GLMM problem as the input dimension is particularly large, resulting in the out-of-memory problem. Therefore, I split the whole generated dataset into multiple files and dynamically load them with `tf.data.Dataset` whenever the data are needed. Shuffling is applied to training data at every epoch.

Model comparison Table 1 presents the squared errors of architectures (i) - (iv) on the test set, where errors are averaged across samples and variable dimensions. Notably, the cumulative dimension of all variables amounts to 128,026, a substantial figure. Despite the marginal differences in average errors in Table 1, the actual test losses exhibit significant variations. The convolution-based architecture demonstrates superior performance compared to other designs, showcasing the efficacy of parameter reduction. In practice, enforcing the network to learn only the gradients of the target density appears to yield counterproductive results, potentially stemming from overly stringent constraints. Additionally, during training, I observed that MLP-based networks, even with sparsity regularization, do not consistently reduce loss, underscoring the challenges posed by an excessive number of parameters. Given these observations, I opt for the convolution-based neural network for subsequent pseudo-marginal HMC sampling.

Table 1: Test mean squared errors (MSE) of four different architectures, where "CNN" refers to the aforementioned convolution-based architecture in (iii) and "InfoCNN" stands for the convolution-based architecture in (iv) that focues on learning the gradients of target density only.

	MLP	MLP with l_1 regularization	CNN	InfoCNN
Test MSE	18.11	18.12	17.70	17.97

2.3 Pseudo-marginal HMC sampling

Problems There are several issues about the pseudo-marginal HMC sampling with NUTS and HNNs if applying the original implementations from the code provided by Dhulipala et al. (2023) without any modification. One is the monitored error, $\epsilon \equiv H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}) + \ln u$, could become `-inf` when u is directly sampled from $\text{Uniform}(0, \exp\{-H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p})\})$. This makes the error always smaller than the predescribed thresholds $\Delta^{\text{hnn}}_{\text{max}}$ or $\Delta^{\text{num}}_{\text{max}}$, and thus the algorithm will not stop even if the integration error is very large.

The other issue is the pseudo-marginal HMC for GLMM always gets stuck at extreme values of λ_1 or λ_2, which is also reported in Section 4.3 of Alenlöv et al. (2021). Two scenarios have been observed:

- λ_1 or λ_2 becomes very large without tendency of returning to the normal range. This

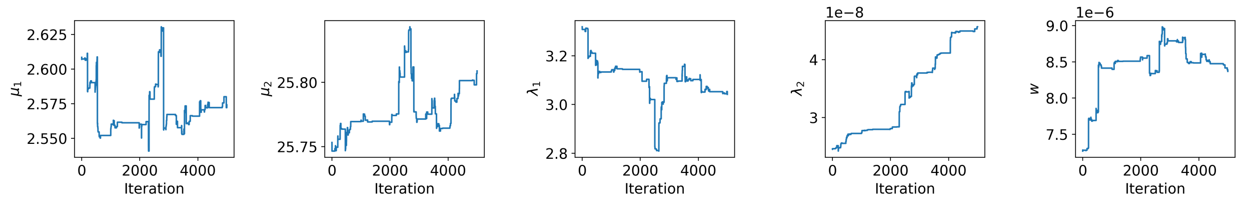

Figure 1: Samples generated from pseudo-marginal HMC with NUTS and the convolution-based network. The total number of samples is set to 7000, whereas the figure plots the last 5000. The parameters corresponding to plots from left to right are μ_1, μ_2, λ_1, λ_2, and w.

may correspond to the well-known problem that the density of the binary mixture model grows without bound as λ_1 or $\lambda_2 \to \infty$ and μ_1 or $\mu_2 \to Y_{ij}$ for some i and j, i.e., one of the mixture components concentrate all of its mass around a single data item Y_{ij} (Stan Development Team, 2023).

- One of the λ's, say λ_2, approaches 0, and the corresponding component weight goes to 1. This phenomenon can be attributed to the collapse of components in the Gaussian mixture model, where one component accumulates all the weight and expands its variance infinitely to encompass all the data points.

Since λ's remain stagnant at the extreme values, the resulting samples exhibit pronounced autocorrelation, as depicted in Figure 1.

Attempted solutions As there are multiple confounding factors, including step size, initialization, NUTS, and HNNs, present in the problem, I have chosen to systematically eliminate their influences one by one in order to pinpoint the bugs. I have tried: (i) using different initializations and step size for NUTS, (ii) checking NUTS without HNNs, (iii) adaptively tuning the step size for NUTS on top of (ii), (iv) using a non-identity mass matrix for $\boldsymbol{\rho}$, and (v) examining the pseudo-marginal HMC without NUTS and HNNs.

(i) In the beginning, the initializations for $\boldsymbol{\theta}$ provided in Appendix G.2 of Alenlöv et al. (2021) are used. For step size, the values $\{0.05, 0.02, 0.01, 0.007, 0.005, 0.002\}$ have

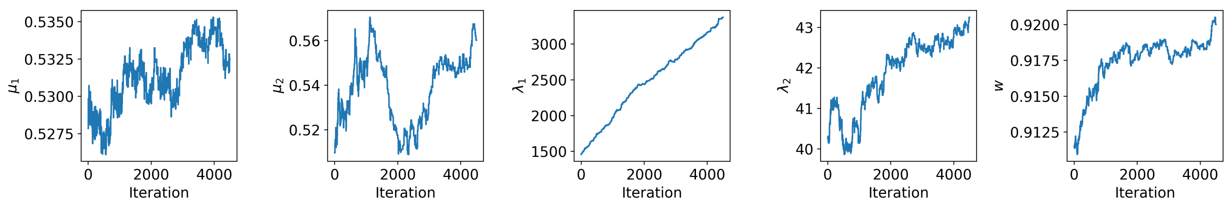

Figure 2: Samples drawn by pseudo-marginal HMC with NUTS and numerical gradients. The step size is set as 0.002. The panels from left to right are plots for the last 4500 samples of μ_1, μ_2, λ_1, λ_2 and w.

been considered. It can be observed that λ_1 or λ_2 goes to a significantly large value more slowly as the step size decreases. However, the extreme behavior is inevitable as more samples are drawn by the algorithm. Changing the initializations for $\boldsymbol{\theta}$ to be randomly drawn from normal distribution does not alleviate the problem either.

(ii) Due to the challenging training process of the HNN with a high input dimension, utilizing gradients from the HNN often leads to integration errors, resulting in inaccurate Hamiltonian calculations. Consequently, I decide to elminate the influence of HNNs and rely solely on numerical gradients for debugging purpose, despite the increased computational time required for these calculations. Nevertheless, the issue persistently arises as λ_1 or λ_2 tend to converge to excessively large values across various initialization methods and step sizes considered in (i). Figure 2 illustrates the samples drawn with a step size of 0.002, where λ_1 continuously grows towards large values.

(iii) In HMC or NUTS, the step size is critical to the sampling behavior (Neal et al., 2011). Therefore, I further consider adaptively tuning the step size of NUTS by the dual averaging method proposed by Hoffman and Gelman (2014). The heuristic method outlined in Algorithm 4 of Hoffman and Gelman (2014) first chooses the initial value of the step size as around 0.007 such that the acceptance probability of the Langevin proposal crosses 0.5. Then Algorithm 6 of Hoffman and Gelman (2014) with the target acceptance probability $\delta \in \{0.65, 0.75, 0.8, 0.85, 0.9\}$ and the number of adaptive iterations M^{adapt} set to 1000. Note that Hoffman and Gelman (2014) recommends $\delta = 0.65$ as the optimal value, while the default setting in Stan's NUTS implementation (Stan Development Team, 2023) employs $\delta = 0.8$. A larger value of δ will result in smaller step sizes, which will improve sampling efficiency albeit at the expense of longer iteration times. The evolution of the learned step size and the samples drawn at every adaptive iteration with $\delta = 0.65$ are displayed in Figure 3. It can be seen from Figure 3(f) that the dual averaging algorithm converges to a very small step size which is at the order of 10^{-6}, making the movement of samples μ_1, μ_2, λ_1 and w in Figure 3(a)-(c) and (e) very small. Nonetheless, the value of λ_2 still keeps growing to large magnitudes with the increasing iteration, as evident in Figure 3(d).

(iv) The mass or covariance matrix $\boldsymbol{M}$ of the momentum variable $\boldsymbol{\rho}$ also plays an important role in the performance of HMC process (Neal et al., 2011). As suggested by Tran and Kleppe (2024), the diagonal elements of $\boldsymbol{M}$ serve as parameters reflecting the speed at which the Hamiltonian dynamics explore the coordinate space or the time between two consecutive events where each coordinate crosses its corresponding median. Therefore, it may be advantageous to use a non-identity $\boldsymbol{M}$. Adapting to this non-identity scenario necessitates adjustments to the Hamiltonian function, gradient calculations, and the numerical integrator. The details of the implementations have been elaborated in Section 1.1. Since λ_1, λ_2 and w change more rapidly compared to the other parameters, I have tried setting $\boldsymbol{M}$ as $\{1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, x, x, x, x, x\}$ where $x \in \{1, \ln 10, 10, 100\}$. It can be observed that the value of λ_1 or λ_2 tends to a large value more rapidly as x becomes smaller. However, the problem has not been addressed eventually.

(v) Given that the experiments using NUTS in (ii)-(iv) fail to address the core issue, I proceed by removing the influence of NUTS and reverting to the original pseudo-marginal HMC method as introduced in Alenlöv et al. (2021). The generated samples are de-

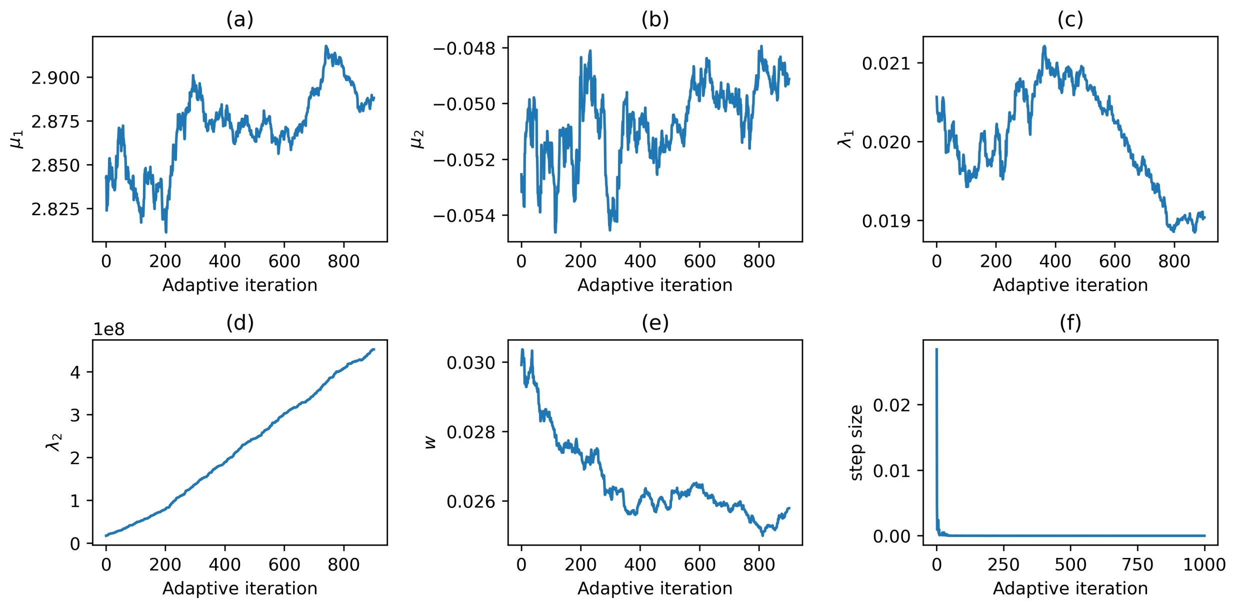

Figure 3: At every adaptive iteration with $\delta = 0.65$: samples of (a) μ_1, (b) μ_2, (c) λ_1, (d) λ_2, (e) w, and (f) the learned step size.

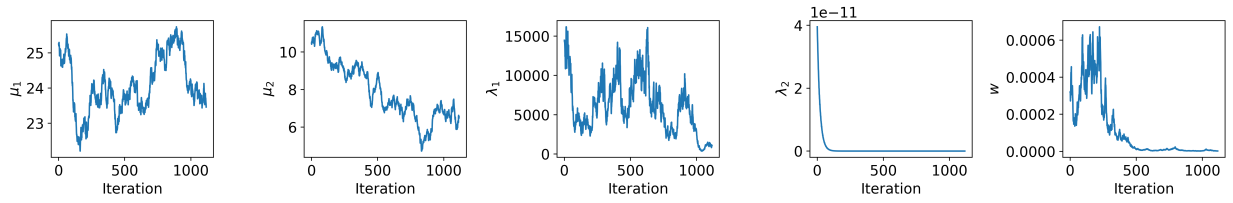

Figure 4: Samples drawn by pseudo-marginal HMC with numerical gradients. The step size is set as 0.002. The panels from left to right are plots of μ_1, μ_2, λ_1, λ_2 and w.

picted in Figure 4. In contrast to the outcomes portrayed in Alenlöv et al. (2021), the parameters λ_2 and w converge towards zero without leaving the region. The abnormal sampling pattern is consistent with the observations from previous experiments.

3 Future work

This section discusses alternative methods from the existing literature that potentially offer solutions to the current challenges faced in pseudo-marginal HMC sampling. The directions on how to adapt these methods and the potential challenges are also outlined here. Due to the unsolved difficulties in implementations and the time constraints, I decide to leave them as future work.

3.1 Modify the mass matrix

Girolami and Calderhead (2011) proposed to adopt a position-specific mass matrix for the momentum variable, which gets rid of the burdensome tuning of the mass matrix. More importantly, such design of the mass matrix can improve the overall mixing of the chain by considering the local structure of the target density when proposing moves to different probability regions (Girolami and Calderhead, 2011). Therefore, the method may also be beneficial in the context of pseudo-marginal HMC process for GLMM, overcoming the problem that the parameters λ's stagnate at the extreme values.

Denote $\boldsymbol{\theta}$ as the target parameter and $\boldsymbol{\rho}$ as its corresponding momentum variable. The proposed Hamiltonian is defined as

$$H(\boldsymbol{\theta}, \boldsymbol{\rho}) = -\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{y}|\boldsymbol{\theta}) + \frac{1}{2}\log\{(2\pi)^d |\boldsymbol{G}(\boldsymbol{\theta})|\} + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{G}(\boldsymbol{\theta})^{-1}\boldsymbol{\rho}, \tag{6}$$

where

$$\boldsymbol{G}(\boldsymbol{\theta}) = \mathrm{cov}\left[\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right] = -\mathbb{E}_{\boldsymbol{y}|\boldsymbol{\theta}}\left[\frac{\partial^2 \log p(\boldsymbol{y}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}\right]$$

is the expected Fisher information matrix, d is the variable dimension of $\boldsymbol{\theta}$, and $|\cdot|$ denotes the determinant of a matrix. The Fisher information matrix defines a metric that enables the computation of distances between parameterized probability measures, imbuing the space of parameterized probability measures with a Riemann geometry. Thus, this technique proposed by Girolami and Calderhead (2011) is referred to as Riemann manifold HMC (RMHMC). To generate trajectories, one has to solve the following Hamiltonian equations:

$$\begin{aligned}
\frac{\mathrm{d}\theta_i}{\mathrm{d}t} &= [\boldsymbol{G}(\boldsymbol{\theta})^{-1}\boldsymbol{\rho}]_i, \\
\frac{\mathrm{d}\rho_i}{\mathrm{d}t} &= \frac{\partial p(\boldsymbol{y}|\boldsymbol{\theta})}{\partial \theta_i} - \frac{1}{2}\mathrm{tr}\left\{\boldsymbol{G}(\boldsymbol{\theta})^{-1}\frac{\partial \boldsymbol{G}(\boldsymbol{\theta})}{\partial \theta_i}\right\} + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{G}(\boldsymbol{\theta})^{-1}\frac{\partial \boldsymbol{G}(\boldsymbol{\theta})}{\partial \theta_i}\boldsymbol{G}(\boldsymbol{\theta})^{-1}\boldsymbol{\rho}.
\end{aligned}$$

It is worth noting that the standard leapfrog integrator with these gradients does not satisfy detailed balance. Thus, Girolami and Calderhead (2011) uses a generalized leapfrog integrator which needs to be solved with fixed point iterations; please refer to Eq.(16) - (18) in Girolami and Calderhead (2011) for the details of the generalized leapfrog integrator.

Adapting the pseudo-marginal HMC to operate within the Riemann manifold poses several challenges, such as determining the appropriate formulation of the Hamiltonian and refining the numerical integrator to satisfy detailed balance in this setting. An intuitive idea is to make the Hamiltonian become

$$\begin{aligned}&H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p})\\ =& -\log p(\boldsymbol{\theta}) - \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u}) + \frac{1}{2}\log\{(2\pi)^d |\boldsymbol{G}(\boldsymbol{\theta}, \boldsymbol{u})|\} + \frac{1}{2}\boldsymbol{\rho}^\top \boldsymbol{G}(\boldsymbol{\theta}, \boldsymbol{u})^{-1}\boldsymbol{\rho} + \frac{1}{2}(\boldsymbol{u}^\top\boldsymbol{u} + \boldsymbol{p}^\top\boldsymbol{p}),\end{aligned}$$

with

$$\boldsymbol{G}(\boldsymbol{\theta}, \boldsymbol{u}) = \mathrm{cov}\left[\frac{\partial \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u})}{\partial \boldsymbol{\theta}}\right] = -\mathbb{E}_{\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u}}\left[\frac{\partial^2 \log \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{u})}{\partial \boldsymbol{\theta}^2}\right].$$

It should be noted that $\boldsymbol{G}(\boldsymbol{\theta}, \boldsymbol{u})$ is one of the diagonal entries of the expected Fisher information matrix. Deciding on whether to leverage the complete information matrix and devising strategies to accomplish this, such as adjusting the mass matrix of $\boldsymbol{p}$ to be position-specific, require further investigation. Besides, the error of the generalized leapfrog integrator may still depend on the dimension of system as the standard leapfrog version does, whereas the dimension in the GLMM problem is considerably large (which is 64013 in our current implementation). Therefore, in order to utilize the RMHMC, one has to develop a generalized version of the Strang-splitting integrator that preserves detailed balance. The feasibility of such construction also requires additional theoretical examination. Last but not least, compared to HMC, RMHMC requires a substantially increased computational time, arising from the fixed point iterations (Burda and Maheu, 2012). It could be especially computationally expensive for the high-dimensional setting that we consider in GLMM. Therefore, the efficiency of the method and whether the neural networks can contribute to saving time are also worth further consideration.

3.2 Modify the importance density

Following Alenlöv et al. (2021), the current importance density for GLMM problem is set as $N(0, 3^2)$, which is easy to implement but may not be efficient. It does not take into account the information about the location and scale of the latent variables X_i's in the data likelihood, and thus may not be well suited. A more well-designed importance density may address the current sampling issue.

Osmundsen et al. (2021) proposed to let the importance density $q_{\boldsymbol{\theta}}(\boldsymbol{X})$ equal to the conditional posterior of the latent variables $p(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{\theta}) \propto p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{X}|\boldsymbol{\theta})$ and set the number of samples $N = 1$ for importance sampling. The first modification makes the target density become $\pi(\boldsymbol{\theta}, \boldsymbol{u}|\boldsymbol{Y}) \propto N(\boldsymbol{u}|\mathbf{0}_D, \boldsymbol{I}_D)p(\boldsymbol{\theta}|\boldsymbol{Y})$, allowing $\boldsymbol{\theta}$ and latent variables to be decoupled and more suited for HMC sampling; while the second modification reduces the computational complexities. In practice, the posterior $p(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{\theta})$ usually does not have an analytically tractable form. To sufficiently accurately approximate it, Osmundsen et al. (2021) suggested to construct the importance density $q_{\boldsymbol{\theta}}(\boldsymbol{X})$ from a local Gaussian Laplace approximation to $p(\boldsymbol{X}|\boldsymbol{Y}, \boldsymbol{\theta})$.

Specifically, they set $q_{\boldsymbol{\theta}}(\boldsymbol{X}) = N(\boldsymbol{X}|\boldsymbol{h}_{\boldsymbol{\theta}}, \boldsymbol{G}_{\boldsymbol{\theta}}^{-1})$ with

$$\begin{aligned}
\boldsymbol{h}_{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{X}} \log\{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{X}|\boldsymbol{\theta})\} \\
\boldsymbol{G}_{\boldsymbol{\theta}} &= -\nabla^2_{\boldsymbol{X}} \log\{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{X}|\boldsymbol{\theta})\}_{\boldsymbol{X}=\boldsymbol{h}_{\boldsymbol{\theta}}},
\end{aligned}$$

Here $\boldsymbol{h}_{\boldsymbol{\theta}}$ needs to be solved by Newton's method, and the updating formula is $\boldsymbol{h}_{\boldsymbol{\theta}}^{(j)} = \boldsymbol{h}_{\boldsymbol{\theta}}^{(j-1)} + [\boldsymbol{G}_{\boldsymbol{\theta}}^{(j-1)}]^{-1}\nabla_{\boldsymbol{X}} \log\{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{X}|\boldsymbol{\theta})\}_{\boldsymbol{X}=\boldsymbol{h}_{\boldsymbol{\theta}}^{(j-1)}}$ and $\boldsymbol{G}_{\boldsymbol{\theta}}^{(j)} = -\nabla^2_{\boldsymbol{X}} \log\{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{X}|\boldsymbol{\theta})\}_{\boldsymbol{X}=\boldsymbol{h}_{\boldsymbol{\theta}}^{(j)}}$, for j from one to some predetermined value J. The map $\gamma_{\boldsymbol{\theta}}(\cdot)$ that transforms $\boldsymbol{u}$ to $\boldsymbol{X}$ is defined as

$$\gamma_{\boldsymbol{\theta}}(\boldsymbol{u}) = \boldsymbol{h}_{\boldsymbol{\theta}}^{(J)} + (\boldsymbol{L}_{\boldsymbol{\theta}}^{(J)})^{-1}\boldsymbol{u},$$

where $\boldsymbol{L}_{\boldsymbol{\theta}}^{(J)}$ is the lower triangular Cholesky factor of $\boldsymbol{G}_{\boldsymbol{\theta}}^{(J)}$.

We can apply the aforementioned importance density to the GLMM setting, which may help overcome the current problem as it utilizes the geometric information of latent variables' posterior density. A potential concern is the computational speed slowed down by the Netwon's method at each step of numerical integration. Furthermore, the importance density obtained from the Gaussian Laplace approximation will introduce additional complexities for gradient calculations, which poses more challenges to HNN learning.

4 Application to financial models

This section explores two possible applications of the methods in Dhulipala et al. (2023) and Alenlöv et al. (2021) to the financial models: one is the stochastic volatility model and the other is the BEKK GARCH model.

4.1 Stochastic volatility model

Literature review The stochastic volatility (SV) models are widely used in literature, such as those on option pricing (Hull and White, 1987), to model the variance of returns on assets. Since the variance tends to change over time, it is appropriate to assume it to follow some latent stochastic process (Kim et al., 1998). A standard version of the SV model (Takaishi, 2009) is given by

$$y_t = \sigma_t \epsilon_t = \exp\{h_t/2\}\epsilon_t, \quad h_{t+1} = \mu + \phi(h_t - \mu) + \sigma_\eta \eta_{t+1} \text{ for } t \geq 1 \text{ and } h_1 \sim N(\mu, \frac{\sigma_\eta^2}{1-\phi^2}). \tag{7}$$

Here $\{y_t\}_{t=1}^n$ is the mean corrected return on holding the asset at time t, and $h_t \equiv \log \sigma_t^2$ is the log volatility at time t which is assumed to follow a stationary autoregressive process ($|\phi| < 1$) with h_1 drawn from a normal distribution. The error terms ϵ_t and η_t are uncorrelated standard normal white noise. The parameter ϕ can be thought to model the persistence in the volatility, and σ_η depicts the volatility of the log volatility. Hence for this model, the parameters to be determined are $\boldsymbol{\theta} = (\mu,\, \phi,\, \sigma_\eta^2)$. The likelihood function $L(\boldsymbol{\theta})$ can be written as

$$L(\boldsymbol{\theta}) = \int \prod_{t=1}^{n} f(y_t|h_t) \cdot \prod_{t=2}^{n} f(h_t|h_{t-1}, \boldsymbol{\theta}) \cdot f(h_1|\boldsymbol{\theta}) \mathrm{d}h_1 \mathrm{d}h_2 \ldots \mathrm{d}h_n, \tag{8}$$

$$\text{where} \quad f(y_t|h_t) = \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp\left(-\frac{y_t^2}{2\sigma_t^2}\right),$$

$$f(h_t|h_{t-1}, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma_\eta^2}} \exp\left(-\frac{[h_t - \mu - \phi(h_{t-1} - \mu)]^2}{2\sigma_\eta^2}\right) \quad \text{for } t \geq 2,$$

$$f(h_1|\boldsymbol{\theta}) = \sqrt{\frac{1-\phi^2}{2\pi\sigma_\eta^2}} \exp\left(-\frac{(h_1 - \mu)^2}{2\sigma_\eta^2/(1-\phi^2)}\right).$$

The Markov chain Monte Carlo (MCMC) methods have been explored to conduct inference on the SV models (Jacquier et al., 1994; Kim et al., 1998). Importantly, in addition to $\boldsymbol{\theta}$, the log volatilities h_t's also need updates as they have to be integrated out in the likelihood function $L(\boldsymbol{\theta})$. Takaishi (2009) first applied the HMC algorithm to the SV models. In this paper, the parameters μ, ϕ, σ_η, and h_t are sampled, respectively; and HMC is used for sampling h_t's. Specifically, Takaishi (2009) sets the prior distribution as $\pi(\sigma_\eta^2) \sim (\sigma_\eta^2)^{-1}$ and $\pi(\mu) = \pi(\phi) =$ constant. Then the posterior distribution of σ_η^2 becomes an inverse gamma distribution which is given by

$$\pi(\sigma_\eta^2|\boldsymbol{\theta}) \sim (\sigma_\eta^2)^{-n/2-1} \exp\left(-\frac{A}{\sigma_\eta^2}\right)$$

where $A = \{(1-\phi^2)(h_1 - \mu)^2 + \sum_{t=2}^{n}[h_t - \mu - \phi(h_{t-1} - \mu)^2]\}/2$. The posterior distribution of μ becomes a Gaussian distribution which is of the form

$$\pi(\mu|\boldsymbol{\theta}) \sim \exp\left\{-\frac{B}{2\sigma_\eta^2}(\mu - \frac{C}{B})^2\right\}.$$

Here $B = (1-\phi^2)+(n-1)(1-\phi)^2$ and $C = (1-\phi^2)h_1+(1-\phi)\sum_{t=2}^{n}(h_t-\phi h_{t-1})$. The sampling of ϕ can be done with the Metropolis-Hastings algorithm. The posterior distribution is given by

$$\pi(\phi|\boldsymbol{\theta}) \sim \underbrace{(1-\phi^2)^{1/2}}_{\pi_1(\phi|\boldsymbol{\theta})} \underbrace{\exp\{-\frac{D}{2\sigma_\eta^2}(\phi - \frac{E}{D})^2\}}_{\pi_2(\phi|\boldsymbol{\theta})},$$

where $D = -(h_1-\mu)^2+\sum_{t=2}^{n}(h_{t-1}-\mu)^2$ and $E = \sum_{t=1}^{n}(h_t-\mu)(h_{t-1}-\mu)$. One may first draw a sample, denoted as ϕ_{new}, from the Gaussian distribution $\pi_2(\phi|\boldsymbol{\theta})$, and then correct it to follow the desired distribution by accepting the sample with probability $\min\left\{1, \frac{\pi(\phi_{\text{new}}|\boldsymbol{\theta})\pi_2(\phi|\boldsymbol{\theta})}{\pi(\phi|\boldsymbol{\theta})\pi_2(\phi_{\text{new}}|\boldsymbol{\theta})}\right\}$. Lastly, the joint posterior distribution of $(h_1, h_2, \ldots, h_n)$ can be written as

$$\begin{aligned}&\pi(h_1, h_2, \ldots, h_n|\boldsymbol{\theta})\\ &= \exp\left\{-\sum_{t=1}^{n}\left(\frac{h_t}{2}+\frac{\epsilon_t^2}{2}\exp(-h_t)\right)-\frac{(h_1-\mu)^2}{2\phi_\eta^2/(1-\phi^2)}-\sum_{t=2}^{n}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}\right\}.\end{aligned}$$

Denote all the volatility variables h_t's as a vector $\boldsymbol{h} = (h_1, h_2, \ldots, h_n) \in \mathbb{R}^n$ and its corresponding momentum variable as $\boldsymbol{p} \sim N(\boldsymbol{0}_n, \boldsymbol{I}_n)$. Then the Hamiltonian can be defined as

$$\begin{aligned}H(\boldsymbol{h},\boldsymbol{p}) &= -\log\pi(h_1, h_2, \ldots, h_n|\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{\rho}^\top\boldsymbol{\rho}\\ &= \sum_{t=1}^{n}\left(\frac{h_t}{2}+\frac{\epsilon_t^2}{2}\exp(-h_t)\right)+\frac{(h_1-\mu)^2}{2\phi_\eta^2/(1-\phi^2)}+\sum_{t=2}^{n}\frac{(h_t-\mu-\phi(h_{t-1}-\mu))^2}{2\sigma_\eta^2}+\frac{1}{2}\boldsymbol{\rho}^\top\boldsymbol{\rho}.\end{aligned} \tag{9}$$

As a result, we may use the leapfrog integrator or other symplectic integrators to perform sampling.

Apply the method of Dhulipala et al. (2023) With the Hamiltonian defined in Eq.(9), one possible way of applying the efficient NUTS with HNNs is to let the HNNs learn the gradients $\partial H/\partial\boldsymbol{h}$ and $\partial H/\partial\boldsymbol{p}$ and use them to replace the numerical gradients in the integrator. Besides, the original HMC sampling procedure can be substituted by the efficient NUTS with online error monitoring as in Algorithm 5 of Dhulipala et al. (2023). Compared to the original HMC method for the SV model in Takaishi (2009), the time can be saved in gradient computation especially when the sample size of time series n is large. Furthermore, the NUTS helps get rid of the troubles brought by tuning the length of trajectories, which is critical to the sampling quality.

Apply the pseudo-marginal HMC of Alenlöv et al. (2021) The SV model is a typical latent variable model, as Eq.(7) can be rewritten as

$$y_t \sim N(0, e^{h_t}), \quad h_{t+1} \sim N(\mu + \phi(h_t - \mu), \sigma_\eta^2) \text{ for } t \geq 1, \quad h_1 \sim N(\mu, \frac{\sigma_\eta^2}{1-\phi}),$$

and h_t's are the latent variables. Then following Eq.(13) of Alenlöv et al. (2021), we may use a surrogate $\hat{p}(y_{1:n}|\boldsymbol{\theta}, \boldsymbol{u})$ from the importance sampling to approximate the intractable likelihood $L(\boldsymbol{\theta})$ in Eq.(8), where $\hat{p}(y_{1:n}|\boldsymbol{\theta}, \boldsymbol{u})$ is defined as

$$\hat{p}(y_{1:n}|\boldsymbol{\theta}, \boldsymbol{u}) = \frac{1}{N} \sum_{l=1}^{N} \frac{\prod_{t=1}^{n} f(y_t|h_t^{(l)}) \prod_{t=2}^{n} f(h_t^{(l)}|h_t^{(l)}, \boldsymbol{\theta}) f(h_1^{(l)}|\boldsymbol{\theta})}{q(\boldsymbol{h}^{(l)}|\boldsymbol{\theta})}.$$

Here N is the number of samples in importance sampling, $\boldsymbol{u} = \{u_{tl}\}_{1 \leq t \leq n, 1 \leq l \leq N}$ are the auxiliary variables and each $u_{il} \sim N(0, 1)$. The vector $\boldsymbol{h}^{(l)} = (h_1^{(l)}, \ldots, h_n^{(l)})$ denotes the l-th sample of the latent variable $\boldsymbol{h}$, and $q(\cdot|\boldsymbol{\theta})$ is the importance density for $\boldsymbol{h}$ which we assume can be simulated by $\boldsymbol{h}^{(l)} = \gamma(\boldsymbol{\theta}, \{u_{tl}\}_{1 \leq t \leq n})$. The map $\gamma : \boldsymbol{\Theta} \times \mathbb{R}^n \mapsto \mathbb{R}^n$ is predetermined. Then we may define the Hamiltonian as

$$H(\boldsymbol{\theta}, \boldsymbol{\rho}, \boldsymbol{u}, \boldsymbol{p}) = -\log \pi(\boldsymbol{\theta}) - \log \hat{p}(y_{1:n}|\boldsymbol{\theta}, \boldsymbol{u}) + \frac{1}{2} \boldsymbol{\rho}^\top \boldsymbol{M}^{-1} \boldsymbol{\rho} + \frac{1}{2}(\boldsymbol{u}^\top \boldsymbol{u} + \boldsymbol{p}^\top \boldsymbol{p}), \tag{10}$$

where $\pi(\boldsymbol{\theta})$ is the prior distribution of $\boldsymbol{\theta}$, $\boldsymbol{\rho}$ and $\boldsymbol{p}$ are the corresponding momentum variables of $\boldsymbol{\theta}$ and $\boldsymbol{u}$ with $\boldsymbol{\rho} \sim N(\boldsymbol{0}_3, \boldsymbol{M})$ and $\boldsymbol{p} \sim N(\boldsymbol{0}_{nN}, \boldsymbol{I}_{nN})$. Both Algorithm 1 of Alenlöv et al. (2021) and the HNN-augmented pseudo-marginal HMC in Algorithm 3 can be applied for sampling. Compared to the original HMC method in Takaishi (2009), the pseudo-marginal HMC method may converge much more quickly as $\boldsymbol{h}_t$ and $\boldsymbol{\theta}$ are strongly correlated and the posterior is multimodal. Besides, joint sampling of $\boldsymbol{h}$ and $\boldsymbol{\theta}$ may be more efficient than the alternating sampling scheme.

4.2 BEKK GARCH model

Literature review Another popular method to model the changing volatility is the Generalized Autoregressive Conditional Heteroskedastic (GARCH) model. Especially, the Multivariate GARCH model (MGARCH) is a useful tool to analyze the co-movements of financial returns, such as the dynamics of asset returns in a portfolio. Various formulations of MGARCH models have been proposed, among which the BEKK model (Engle and Kroner, 1995) is one of the most flexible ones. Let $\boldsymbol{r}_t \in \mathbb{R}^N$ be a vector of asset returns with $t = 1, \cdots, T$ and denote the information set as $\mathcal{F}_{t-1} = \{\boldsymbol{r}_1, \cdots, \boldsymbol{r}_{t-1}\}$. The BEKK model assumes

$$\begin{aligned} &\boldsymbol{r}_t | \mathcal{F}_{t-1} = N(\mathbf{0}_N, \boldsymbol{H}_t) \\ \text{and} \quad &\boldsymbol{H}_t = \boldsymbol{C}_0 \boldsymbol{C}_0^\top + \sum_{q=1}^{Q} \sum_{i=1}^{l} \boldsymbol{A}_{qi} \boldsymbol{r}_{t-i} \boldsymbol{r}_{t-i}^\top \boldsymbol{A}_{qi}^\top + \sum_{q=1}^{Q} \sum_{j=1}^{m} \boldsymbol{B}_{qj} \boldsymbol{H}_{t-j} \boldsymbol{B}_{qj}^\top, \end{aligned}$$

where $\boldsymbol{C}$, $\boldsymbol{A}_{qi}$, and $\boldsymbol{B}_{qj} \in \mathbb{R}^{N \times N}$ are parameter matrices, and $\boldsymbol{C}$ is constrained to be lower triangular. Due to the difficulties in estimating a substantial amount of parameters, it is often assumed $l = m = Q = 1$ and directly uses a positive definite matrix $\boldsymbol{C}$ to replace $\boldsymbol{C}_0 \boldsymbol{C}_0^\top$ (Livingston and Nur, 2023), which gives the form:

$$\boldsymbol{H}_t = \boldsymbol{C} + \boldsymbol{A} \boldsymbol{r}_{t-1} \boldsymbol{r}_{t-1}^\top \boldsymbol{A}^\top + \boldsymbol{B} \boldsymbol{H}_{t-1} \boldsymbol{B}^\top.$$

The parameters involved here are $\boldsymbol{\theta} = (\text{vech}(\boldsymbol{C})^\top, \text{vec}(\boldsymbol{A})^\top, \text{vec}(\boldsymbol{B})^\top)^\top \in \mathbb{R}^{2N^2 + N(N+1)/2}$, which is of large dimension. It is more suitable to use HMC rather than the random walk style sampling as the latter could result in sluggish exploration of the parameter space with high autocorrelations and thus unsatisfactory mixing of the chain. Therefore, it is possible to apply the method of Dhulipala et al. (2023), which can enhance the computational efficiency by using the HNNs to replace numerical gradients. However, the pseudo-marginal HMC may not be necessary to be applied here, as the likelihood function is not intractable.

Apply the method of Dhulipala et al. (2023) Following Livingston and Nur (2023), we may set the prior for $\boldsymbol{\theta}$ as $\pi(\boldsymbol{\theta}) \propto \mathbb{1}_{\mathcal{A}}(\boldsymbol{\theta})$, where the indicator function will take the value of one when the conditions for covariance stationarity are satisfied and zero otherwise. Note that the BEKK model will be covariance stationary if and only if the eigenvalues of the matrix $\boldsymbol{A} \otimes \boldsymbol{A} + \boldsymbol{B} \otimes \boldsymbol{B}$ are less than one in modulus, where $\otimes$ denotes the Kronecker product. The conditional likelihood function can be written as

$$p(\boldsymbol{r}_{1 \leq t \leq T} | \boldsymbol{\theta}) = (2\pi)^{-N(T-1)/2} \left(\prod_{t=2}^{T} |\boldsymbol{H}_t| \right)^{-1/2} \exp\left\{ -\frac{1}{2} \sum_{t=2}^{T} \boldsymbol{r}_t^\top \boldsymbol{H}_t^{-1} \boldsymbol{r}_t \right\}.$$

Therefore, the Hamiltonian can be defined as

$$H(\boldsymbol{\theta}, \boldsymbol{\rho}) = -\log \pi(\boldsymbol{\theta}) - \log p(\boldsymbol{r}_{1 \leq t \leq T} | \boldsymbol{\theta}) + \frac{1}{2} \boldsymbol{\rho}^\top \boldsymbol{\rho},$$

where $\boldsymbol{\rho} \sim N(\mathbf{0}_{2N^2+N(N+1)/2}, \boldsymbol{I}_{2N^2+N(N+1)/2})$ is the corresponding momentum variable. With the Hamiltonian defined, one can apply Algorithms 1 and 4 of Dhulipala et al. (2023).

References

Alenlöv, J., Doucet, A., and Lindsten, F. (2021). Pseudo-marginal hamiltonian monte carlo. *Journal of Machine Learning Research*, 22(141):1–45.

Burda, M. and Maheu, J. M. (2012). Bayesian adaptively updated hamiltonian monte carlo with an application to high-dimensional bekk garch models. Working Paper series 46_12, Rimini Centre for Economic Analysis.

Dhulipala, S. L., Che, Y., and Shields, M. D. (2023). Efficient bayesian inference with latent hamiltonian neural networks in no-u-turn sampling. *Journal of Computational Physics*, 492:12425.

Engle, R. F. and Kroner, K. F. (1995). Multivariate simultaneous generalized arch. *Econometric Theory*, 11(1):122–150.

Girolami, M. and Calderhead, B. (2011). Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 73(2):123–214.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(47):1593–1623.

Hull, J. and White, A. (1987). The pricing of options on assets with stochastic volatilities. *The Journal of Finance*, 42(2):281–300.

Jacquier, E., Polson, N. G., and Rossi, P. E. (1994). Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics*, 12(4):371–389.

Kim, S., Shephard, N., and Chib, S. (1998). Stochastic volatility: Likelihood inference and comparison with arch models. *The Review of Economic Studies*, 65(3):361–393.

Livingston, G. C. and Nur, D. (2023). Bayesian inference of multivariate-garch-bekk models. *Statistical Papers*, 64(5):1749–1774.

Mächler, M. (2015). Accurately computing log(1 -exp(-|a|)) assessed by the rmpfr package.

Neal, R. M., Brooks, S., Meng, X.-L., Gelman, A., Jones, G., Gelman, A., Jones, G., Meng, X.-L., and Brooks, S. (2011). Mcmc using hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman and Hall/CRC, United Kingdom.

Osmundsen, K. K., Kleppe, T. S., and Liesenfeld, R. (2021). Importance sampling-based transport map hamiltonian monte carlo for bayesian hierarchical models. *Journal of Computational and Graphical Statistics*, 30(4):906–919.

Stan Development Team (2023). Stan modeling language users guide and reference manual, version 2.32.

Takaishi, T. (2009). Bayesian inference of stochastic volatility model by hybrid monte carlo. *Journal of Circuits, Systems and Computers*, 18(08):1381–1396.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

Tran, J. H. and Kleppe, T. S. (2024). Tuning diagonal scale matrices for hmc. *Statistics and Computing*, 34(6):196.