HAMILTONIAN NEURAL NETWORKS AND PSEUDO-MARGINAL HAMILTONIAN MONTE CARLO

CHEN, Hongxiao
Department of Economics
The Chinese University of Hong Kong

March 13, 2025

Part I describes the theorical background of PM-HMC.

Part II describes how I realize traditional PM-HMC. In part II, I presented two methods for evaluating the gradients of target probability. The first one uses TensorFlow auto differentiation to find gradients, and the second one manually calculate the gradients, which serves as a verification to the first one.

Part III describes the construction of Hamiltonian Neural Network.

Part IV describes PM-NUTS with HNN and online monitoring mechanism.

Part V presents results of PM-HMC, PM-NUTS, HNN-HMC, and HNN-PM-NUTS with online monitoring

Part VI presents test plans and results.

Part VII dicusses PM-HMC's application in finance.

Part VIII is the Appendix

Contents

Part I
Background

1 Bayesian inference and Hamiltonian

1.1 Intractable likelihood

The bayesian equation states that

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

For certain families of distributions, the posterior $p(\theta|y)$ has the same form of distribution of the prior. In this case, we only need to sample from $p(\theta|y)$ using HMC by constructing Hamiltonian:

$$H(\theta, \rho) = U(\theta) + K(\rho) = -log\left(p(\theta|y)\right) + \frac{1}{2}\rho^T\rho$$

In general cases, we do not know the explicit form of $p(\theta|y)$. Therefore, we need to use prior and likelihood rather than the posterior to construct the Hamiltonian:

$$H(\theta, \rho) = U(\theta) + K(\rho) = -log\left(p(\theta)\right) - log\left(p(y|\theta)\right) + \frac{1}{2}\rho^T\rho$$

This is not the end of the solution method. In this new case, we still don't know the explicit form of the log-likelihood $log\left(p(y|\theta)\right)$, which makes the Hamiltonian not applicable. To address this issue, a method called Pseudo-Marginal Hamiltonian Monte Carlo is introduced.

2 Pseudo-Marginal Hamiltonian Monte Carlo

The spirit of Pseudo-Marginal Hamiltonian Monte Carlo is to obtain an unbiased estimator of the likelihood $p(y|\theta)$, denoted as $\hat{p}(y|\theta)$. The way of constructing the unbiased estimator is to introduce an auxilary variable called **u**, and assume it also has an accompanied momentum, called **p**. There are mainly two parts in using this PM-HMC method. The first part is on how to construct this unbiased estimator $log\left(\hat{p}(y|\theta)\right)$, and the second part is on how to construct the Hamiltonian that is used to sample θ.

2.1 Unbiased estimator

Suppose we have a set of observations $y = \{y_1, y_2, ..., y_T\}$,then we can decompose the unbiased estimator of likelihood as $\hat{p}(y|\theta) = \prod_{k=1}^{T} \hat{p}(y_k|\theta)$. The next step is on how to find an explicit form of $\hat{p}(y_k|\theta)$. A method called **Importance Sampling**is used. The flow of logic is as below. Let's write an explicit form of $p(y_k|\theta)$ first by introducing a latent variable called X_k with importance density $q_\theta(X_k|y_k)$.

$$\begin{aligned} p(y_k|\theta) &= \int p(y_k|X_k, \theta)p(X_k|\theta)dX_k \\ &= \int \frac{p(y_k|X_k, \theta)p(X_k|\theta)}{q_\theta(X_k|y_k)} q_\theta(X_k|y_k)dX_k \end{aligned}$$

Then, we use $\hat{p}(y_k|\theta)$ to apporximate $p(y_k|\theta)$. Suppose we have N observations of X_k, denoted as $\{X_{k,1}, X_{k,2}, ..., X_{k,N}\}$, then $p(y_k|\theta)$ can be appoximated as

$$\hat{p}(y_k|\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{p(y_k|X_{k,i},\theta)p(X_{k,i}|\theta)}{q_\theta(X_{k,i}|y_k)}$$

Let $p(y_k|X_{k,i},\theta) = g_\theta(y_k|X_{k,i})$, $p(X_{k,i}|\theta) = f_\theta(X_{k,i})$, we can explicitly write

$$\hat{p}(y_k|\theta) = \frac{1}{N}\sum_{i=1}^{N}\omega_\theta(y_k, X_{k,i})$$

where

$$\omega_\theta(y_k, X_{k,i}) = \frac{g_\theta(y_k|X_{k,i})f_\theta(X_{k,i})}{q_\theta(X_{k,i}|y_k)}$$

2.1.1 Importance sampling

Where do the latent variables $\{X_{k,1}, X_{k,2}, ..., X_{k,N}\}$ come from? It can be simulated using $X_k = \gamma_k(\theta, U)$, where γ_k is an explicit deterministic map and $U \sim N(0_p, I_p)$. The explicit form of $\gamma_k(\theta, U)$ is determined by properties of X_k, where $\gamma_k : \Theta \times R^p \to R^n$. A simple example is Chapter 4.1 of the paper Pseudo-Marginal Hamiltonian Monte Carlo. The latent variable X_k follows

$$X_k|\theta \sim N(\theta, \sigma_X^2)$$

which means

$$X_k = \theta + \sigma_X Z$$

where $Z \sim N(0,1)$. To use importance sampling to simulate X_k, PM-HMC introduce the auxilary variable $U_{k,i} \sim N(0,1)$. Therefore, we can represent $\gamma_k(\theta, U_{k,i}) = \theta + \sigma_X U_{k,i}$. In the simple example, $\gamma_k(\theta, U)$ is a linear function. In more complicated cases, it can be in other forms, but we have to follow the principle such that $\gamma_k(\theta, U)$ has same distribution as $p(X_k|\theta)$.

The last step is to explicitly express $g_\theta(y_k|X_{k,i})$, $f_\theta(X_{k,i})$ and $q_\theta(U_{k,i})$, which depends on the specific settings of the model.

2.2 Hamiltonian Monte Carlo

There are four variables, θ, ρ, u, p, which are parameters to be inferred, the momentum, the auxilary variable and the momentum of auxilary variables. The extended Hamiltonian is

$$H(\theta, \rho, u, p) = -logp(\theta) - log\hat{p}(y|\theta, u) + \frac{1}{2}\left\{\frac{\rho^T\rho}{m_\rho} + u^Tu + p^Tp\right\} \tag{1}$$

2.2.1 Theory

Equations of motion asscociated with this extended Hamiltonian is

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

2.2.2 One step numerical Integration,

One step is decomposed into step A and B. A full step is as below:

Half step A:

$$\theta \leftarrow \theta + \frac{h}{2}\frac{\rho}{m_\rho}$$

Rotate (u, p):

$$\mathbf{u} \leftarrow \mathbf{u} cos\left(\frac{h}{2}\right) + \mathbf{p} sin\left(\frac{h}{2}\right)$$
$$\mathbf{p} \leftarrow \mathbf{p} cos\left(\frac{h}{2}\right) - \mathbf{u} sin\left(\frac{h}{2}\right)$$

Full step B:

$$\rho \leftarrow \rho + h\nabla_\theta \left\{ logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\}$$
$$\mathbf{p} \leftarrow \mathbf{p} + h\nabla_{\mathbf{u}} \left\{ logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\}$$

Then again, Half step A:

$$\theta \leftarrow \theta + \frac{h}{2}\frac{\rho}{m_\rho}$$

Rotate (u, p):

$$\mathbf{u} \leftarrow \mathbf{u} cos\left(\frac{h}{2}\right) + \mathbf{p} sin\left(\frac{h}{2}\right)$$
$$\mathbf{p} \leftarrow \mathbf{p} cos\left(\frac{h}{2}\right) - \mathbf{u} sin\left(\frac{h}{2}\right)$$

where h represents size of time step.

Part II
Data and Algorithm

3 Dimensions of All variables

This part explains how dimensions of variables are matched.

There are T numbers of $y = \{y_1, y_2, ..., y_T\}$, within which there are n observations, denoted as $\{y_{i,1}, y_{i,2}, ..., y_{i,n}\}$. Therefore, the dimension of y is $[T, n]$.

Dimension of U is $\{T, N, p\}$.

Dimension of X is $[T, N, d]$.

Suppose $d = 1$. Dimension of X is $[T, N, 1]$. Then broadcast dimension of X to $[T, N, n]$. Broadcast y to $[T, 1, n]$.

logistic / Bernoulli => likelihood => shape $[T, N]$

4 Algorithms

I tried multiple algorithms for PM-HMC.

4.1 General Idea of PM-HMC.

All varibles are in the form of TensorFlow tensors. Initialize θ from the prior. Initialize ρ from $N(0_d, I_d) \times m_\rho$, and $\mathbf{p}$ from $N(0_D, I_D)$, where $D = T \times N \times p$, d is the dimension of θ.

Step 1: For each $y_k \in \{y_1, y_2, ..., y_T\}$, initialize N auxilary variables $U_{k,i}$ by drawing them from $N(0_p, I_p)$. The dimension of all $U_{k,i}$ is $\{T, N, p\}$, because there are T numbers of k, N numbers of i, and each $U_{k,i}$ is with dimension p

Step 2: Calculate $log\hat{p}(y|\theta, \mathbf{u})$ with the following logic:

Generate $X_{k,i} = \gamma_k(\theta, U_{k,i})$. Calculate $\omega_\theta(y_k, X_{k,i})$, and $\hat{p}(y_k|\theta) = \frac{1}{N}\sum_{i=1}^{N} \omega_\theta(y_k, X_{k,i})$. Then, Calculate $log\hat{p}(y|\theta, \mathbf{u}) = \sum_{k=1}^{T} log\hat{p}(y_k|\theta)$.

Step 3: Calculate Hamiltonian using equation (1) as $H(\theta, \rho, u, p)$.

Step 4: Update $\{\theta, \rho, \mathbf{u}, \mathbf{p}\}$ by using the one step numerical integration in 2.2.2. The new variables are defined as $\{\theta', \rho', \mathbf{u}', \mathbf{p}'\}$

Step 5: In each point of $\{T, N\}$, we have one $U_{k,i}$ of dimension p. Use the same method as it is in step 2 and 3 to calculate Hamiltonian using equation (1) as $H(\theta', \rho', u', p')$. Accept $\{\theta', \rho', \mathbf{u}', \mathbf{p}'\}$ with probability $\min\left(1, exp\left(H(\theta, \rho, u, p) - H(\theta', \rho', u', p')\right)\right)$.

Step 6: Repeatedly perform step 2 to step 5, until all L steps are finished.

Step 7: Use the end point of last trajectory to initialize θ. Initialize ρ from $N(0_d, I_d)$, and $\mathbf{p}$ from $N(0_D, I_D)$. Repeat step 1 to step 6, until all samples of θ are collected.

4.2 Numerical improvement for GLMM using auto-differentiation

Inputs are data points Y with dimension $[T, n]$. T is number of individuals, each individual has 6 observations. (Bernoulli 0/1). Parameters to be inferred are $\theta = \{\beta, \mu_1, \mu_2, log(\lambda_1), log(\lambda_2), logit(w_1)\}$, who has a dimension of $p_Z + 5$. Step size is h, number of steps is L. Numbers of iterations (samples collected) is M. The output is $\{\theta^m; m = 1, 2, ..., M\}$.

Data are generated as following:

1. Initialize β which contains 8 elements, each element is initialized from $N(0, 1)$. Initialize Z with dimension $[T, n, 8]$, each element is initialized from $N(0, 1)$.

2. Generate $X \sim 0.8N_1(0, 10^{-1}) + 0.2N_2(3, 3^{-1})$, with dimension $[T]$, and each element is independent with each other.

3. Shape of $Z_{i,j}^{\mathbb{T}}\beta = [T, n]$. Unsqueeze X to shape $[T, 1]$. Broadcasting leads to $X + Z_{i,j}^{\mathbb{T}}\beta$ with shape $[T, n]$

4. Generate $p_{ij} = logit^{-1}(X + Z_{i,j}^{\mathbb{T}}\beta)$ with shape $[T, n]$

5. Generate date $Y \sim Bernoulli(p_{ij})$ with shape $[T, n]$

Parameters to be inferred are $\theta = \{\beta, \mu_1, \mu_2, log(\lambda_1), log(\lambda_2), logit(w_1)\}$, who has a dimension of $p_Z + 5$. Step size is h, number of steps is L. Numbers of iterations (samples collected) is M. The output is $\{\theta^m; m = 1, 2, ..., M\}$.

4.2.1 Step 1

Initialization:

1.1 Initialize θ^0 . β contains 8 elements, each element is initialized from $N(0, 1)$. $\mu_1^0 = 0.0$, $\mu_2^0 = 0.0$, $log\lambda_1^0 = 0.0$, $log\lambda_2^0 = 1 - log(10)$, $logit(w_1^0) = 0$

1.2 Initialize ρ from $N(0_d, I_d)$, and $\mathbf{p}$ from $N(0_D, I_D)$, where D is $[T, N, p]$, d is the dimension of θ.

1.3 Initialize $\mathbf{u}$ from $N(0_D, I_D)$.

4.2.2 Step 2

Preliminary:

$$
\begin{aligned}
log(w_1) &= -log(1 + exp(-logit(w_1))) \\
log(1 - w_1) &= -log(1 + exp(logit(w_1))) \\
\lambda_i &= exp(log(\lambda_i)) \\
logit(w_1) &= log\left(\frac{w_1}{1 - w_1}\right) \\
logit^{-1}(x) &= \frac{1}{1 + exp(-x)} = Sigmoid(x)
\end{aligned}
$$

To calculate $log\hat{p}(y|\theta, \mathbf{u})$, the main idea is

$$log\hat{p}(y|\theta) = \sum_{i=1}^{T} log\hat{p}(y_i|\theta)$$
$$log\hat{p}(y_i|\theta) = log\left(\sum_{k=1}^{N} \omega_\theta(y_i, X_{i,k})\right) - log(N)$$

$$log\left(\sum_{k=1}^{N} \omega_\theta(y_i, X_{i,k})\right) = M + log\left(\sum_{k=1}^{N} \left(exp\left(log\omega_\theta(y_i, X_{i,k}) - M\right)\right)\right)$$
$$M = \max_k log\omega_\theta(y_i, X_{i,k})$$
$$log\omega_\theta(y_i, X_{i,k}) = loggg_\theta(y_i|X_{i,k}) + logf_\theta(X_{i,k}) - logq_\theta(X_{i,k}|y_i)$$

2.1 $X = \gamma_k(\theta, \mathbf{u})$. $\gamma_k : \mathbb{R}^p \to \mathbb{R}^1$. The dimension of X is $[T, N]$. The exact form of $\gamma_k(\theta, \mathbf{u})$ is $X = 3u$

2.2
$$logq_\theta = -\frac{1}{2}log(2\pi) - \frac{1}{2}log(9) - \frac{X^2}{18}$$

2.3
$$loggg_\theta(y_i|X_{i,k}) = \sum_{j=1}^{n} \left[y_{i,j}\left(-softplus\left(-\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right) + (1 - y_{i,j})\left(-softplus\left(\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right)\right]$$

2.4 f_θ is the prior of $X_{i,k}$ and $X_{i,k} \sim w_1 N(\mu_1, \lambda_1^{-1}) + (1 - w_1)N(\mu_2, \lambda_2^{-1})$. Applying log-sum-exp,

$$\begin{aligned}
logf_\theta(X) &= max(z_1, z_2) + log\left[exp(z_1 - maxz) + exp(z_1 - maxz)\right] \\
&---------- \\
&---------- \\
z_1 &= log(w_1) + logN(X|\mu_1, \lambda_1^{-1}) \\
&= -log(1 + exp(-logit(w_1))) + \frac{1}{2}log(\lambda_1) - \frac{1}{2}log(2\pi) - \frac{exp(log(\lambda_1))}{2}(X - \mu_1)^2 \\
&= -softplus(-logit(w_1)) + \frac{1}{2}log(\lambda_1) - \frac{1}{2}log(2\pi) - \frac{exp(log(\lambda_1))}{2}(X - \mu_1)^2 \\
&---------- \\
&---------- \\
z_2 &= log(1 - w_1) + logN(X|\mu_2, \lambda_2^{-1}) \\
&= -log(1 + exp(logit(w_1))) + \frac{1}{2}log(\lambda_2) - \frac{1}{2}log(2\pi) - \frac{exp(log(\lambda_2))}{2}(X - \mu_2)^2 \\
&= -softplus(logit(w_1)) + \frac{1}{2}log(\lambda_2) - \frac{1}{2}log(2\pi) - \frac{exp(log(\lambda_2))}{2}(X - \mu_2)^2 \\
&---------- \\
&----------
\end{aligned}$$
$$z_1 - z_2 = logit(w_1) + \frac{1}{2}log(\lambda_1) - log(\lambda_2) - \frac{exp(log(\lambda_1))}{2}(X - \mu_1)^2 + \frac{exp(log(\lambda_2))}{2}(X - \mu_2)^2$$

2.5
$$log\omega_\theta(y_i, X_{i,k}) = loggg_\theta(y_i|X_{i,k}) + logf_\theta(X_{i,k}) - logq_\theta(X_{i,k}|y_i)$$

2.6 Calculate
$$log\left(\sum_{k=1}^{N} \omega_\theta(y_i, X_{i,k})\right)$$

with log-sum-exp.

2.7

$$log\hat{p}(y_i|\theta) = log\left(\sum_{k=1}^{N}\omega_\theta(y_i, X_{i,k})\right) - log(N)$$

$$log\hat{p}(y|\theta) = \sum_{i=1}^{T} log\hat{p}(y_i|\theta)$$

2.8 The gradients

$$\frac{\partial log\hat{p}(y|\theta)}{\partial \mathbf{u}}, \frac{\partial log\hat{p}(y|\theta)}{\partial \theta}$$

are calculated using auto-differentiation.

4.2.3 Step 3

Construct Hamiltonian according to $H(\theta, \rho, u, p) = -logp(\theta) - log\hat{p}(y|\theta, u) + \frac{1}{2}\left\{\frac{\rho^T\rho}{m^\rho} + u^T u + p^T p\right\}$.

Details are:

3.1 $\theta = \{\beta, \mu_1, \mu_2, log(\lambda_1), log(\lambda_2), logit(w_1)\}$, $logp(\theta) = \sum logp(\theta_i)$, where θ_i in $\{\beta, \mu_1, \mu_2, log(\lambda_1), log(\lambda_2), logit(w_1)\}$. $p(\theta) \sim N(0, 1)$ for each parameter.

3.2 $log\hat{p}(y|\theta, u)$ comes from step 2.

3.3 $\rho^T\rho = \|\rho\|^2$,$u^T u = \|u\|^2$,$p^T p = \|p\|^2$

4.2.4 Step 4

Strang splitting/Leapfrog. Use the function in 2.2.2 that takes $(\theta, \rho, \mathbf{u}, \mathbf{p})$ as input and outputs $(\theta', \rho', \mathbf{u}', \mathbf{p}')$. Repeat the process until all L steps are finished, and output $(\theta', \rho', \mathbf{u}', \mathbf{p}')$ at the end of the trajectory..

4.2.5 Step 5

Metropolis-Hasting acceptance

Calculate

$$\Delta H = H(\theta, \rho, \mathbf{u}, \mathbf{p}) - H(\theta', \rho', \mathbf{u}', \mathbf{p}')$$

accept the new sample with probability

$$\alpha = \min(1, exp(\Delta H))$$

If accepted, $(\theta, \mathbf{u}) \leftarrow (\theta', \mathbf{u}')$, otherwise $(\theta, \mathbf{u})$ stays as old $(\theta, \mathbf{u})$. Do not keep $(\rho', \mathbf{p}')$.

4.2.6 Step 6

Repeat Step 2 to Step 5 for M times. In each repeatance:

-Initialize ρ from $N(0_d, I_d)$, and $\mathbf{p}$ from $N(0_D, I_D)$

-Use previous $(\theta, \mathbf{u})$ as start point

-L steps of Strang splitting for $(\theta', \mathbf{u}')$.

-Metropolis-Hasting acceptance

-New $(\theta, \mathbf{u})$.

4.2.7 Step 7

Burn-in the first m samples.

4.3 Manual Calculation of gradients for GLMM

The method in 4.2 raises several concerns. The concern comes from possibly not being able to fully track gradients when using auto differentiation, because there are terms like $max(z_1, z_2), |z_1 - z_2|, tf.where()$ etc. So, in this attempt, I manually compute the gradients, and compare the results with auto differentiation. Details are put into appendix

After testing, the auto-differentiation version in 4.3 provides gradients that are very closely to the manually computed gradients. In later applications, I choose to use the auto-differentiation version, because this version allows more flexible function forms of the log-likelihood function, and is more clear and easier to maintain. Also, the speeds of two versions are quite close.

Part III
Hamiltonian Neural Networks

5 HNN and Training

In this chapter, I presented a successful method to build and train HNN. In the Appendix, I presented other methods that currently do not have any ideal result yet.

Equation (12) in PM-HMC paper states that

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \rho\\ \mathbf{u}\\ \mathbf{p}\end{pmatrix} = \begin{pmatrix}+\partial H/\partial\rho\\ -\partial H/\partial\theta\\ +\partial H/\partial\mathbf{p}\\ -\partial H/\partial\mathbf{u}\end{pmatrix} = \begin{pmatrix}\rho/m_\rho\\ \nabla_\theta\left\{logp(\theta) + log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}\\ \mathbf{p}\\ -\mathbf{u} + \nabla_{\mathbf{u}}\left\{logp(\theta) + log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}\end{pmatrix}$$

We know that if $log\hat{p}(y|\theta,u)$ converges to $logp(y|\theta)$ in probability, then as N gets larger, $\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\rho$ converges to $\partial H(\theta,\rho)/\partial\rho$, and $\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\theta$ converges to $\partial H(\theta,\rho)/\partial\theta$

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \rho\\ \mathbf{u}\\ \mathbf{p}\end{pmatrix} = \begin{pmatrix}+\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\rho\\ -\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\theta\\ +\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\mathbf{p}\\ -\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\mathbf{u}\end{pmatrix} \to \begin{pmatrix}+\partial H(\theta,\rho)/\partial\rho\\ -\partial H(\theta,\rho)/\partial\theta\\ -\\ -\end{pmatrix}$$

So, I can let HNN output $H(\theta,\rho)$. Now we already have $\nabla_\theta H(\theta,\rho,\mathbf{u},\mathbf{p})$, we can use it to train $\nabla_\theta H(\theta,\rho)$ and $\nabla_\rho H(\theta,\rho)$. Because

$$H(\theta,\rho,u,p) = -logp(\theta) - log\hat{p}(y|\theta,u) + \frac{1}{2}\left\{\frac{\rho^T\rho}{m_\rho} + u^Tu + p^Tp\right\}$$
$$H(\theta,\rho) = -logp(\theta) - logp(y|\theta) + \frac{1}{2}\frac{\rho^T\rho}{m_\rho}$$

The Algorithm is that, the HNN takes input $\{\theta,\rho\}$,output $\{\tilde{H}(\theta,\rho), \nabla_\theta\tilde{H}(\theta,\rho), \nabla_\rho\tilde{H}(\theta,\rho)\}$, then construct the loss function:

$$w_1\left\|\nabla_\theta H(\theta,\rho,\mathbf{u},\mathbf{p}) - \nabla_\theta\tilde{H}(\theta,\rho)\right\|^2 + w_2\left\|\rho/m_\rho - \nabla_\rho\tilde{H}(\theta,\rho)\right\|^2$$

This is the only method that works now, which is not only much easier to be trained but also keeps symplectic structure.

Part IV
No-U-Turn Sampling

6 HNN-PM-NUTS

6.1 PM-NUTS without HNN

The HNN No-U-Turn sampling with online error monitoring shares the same logic as it is in Dhulipala (2023). The main difference comes from the stopping criterior.

In traditional NUTS, the stopping criterior is defined as

$$(\theta^+ - \theta^-) \cdot \rho^- < 0; or; (\theta^+ - \theta^-) \cdot \rho^+ < 0$$

In the case of PM-HMC, the stopping criterior becomes

$$(z^+ - z^-) \cdot p_z^- < 0; or; (z^+ - z^-) \cdot p_z^+ < 0$$

where

$$z = \begin{pmatrix} \theta(13,) \\ \mathbf{u}(500, 128) \end{pmatrix}; p_z = \begin{pmatrix} \rho(13,) \\ \mathbf{p}(500, 128) \end{pmatrix}$$

In application, I calculate the inner production of θ and $\mathbf{u}$ separately and then sum them up. Specifically,

$$(z^+ - z^-) \cdot p_z^- = (\theta^+ - \theta^-) \cdot \rho^- + \sum_{t=1}^{T=500} \sum_{n=1}^{N=128} \left((\mathbf{u}^+ - \mathbf{u}^-) \times \mathbf{p}^- \right)$$

To ensure numerical stability, I also set another stopping criteior, which is

$$|(z^+ - z^-) \cdot p_z^-| > \Delta_{max}^z; or; |(z^+ - z^-) \cdot p_z^+| > \Delta_{max}^z$$

In application, Δ^z is set to 50,000.

6.2 HNN-PM-NUTS with online monitoring

Following Dhulipala (2023), the HNN-PM-NUTS uses HNN to output the gradients, $\nabla_\theta H(\theta, \rho)$ and $\nabla_\rho H(\theta, \rho)$,while at the same time monitoring the error accumulated from using HNN. Once the error exceeds Δ_{max}^{hnn}, HNN is abandoned and traditional standard strang splitting takes over. The traditional standard strang splitting takes at least N_{lf} steps which can be viewed as the number of "cooldown" samples needed to return the sample regions where HNN is sufficiently trained. The stopping criteria for HNN-PM-NUTS is

$$(\theta^+ - \theta^-) \cdot \rho^- < 0; or; (\theta^+ - \theta^-) \cdot \rho^+ < 0$$

When switching to PM-NUTS, I initialize $\mathbf{u} \sim N(0, 1)$ and $\mathbf{p} \sim N(0, 1)$ at current θ and ρ, and then applied the method described in 6.1, with stopping criteria

$$(z^+ - z^-) \cdot p_z^- < 0; or; (z^+ - z^-) \cdot p_z^+ < 0$$

There is another threshold, Δ_{max}^{lf} for PM-NUTS without using HNN. Once the error accumulated in using traditional standard strang splitting, the tree building stops and the sample is not accepted.

Part V
Results

7 HMC and NUTS

Traditional HMC replicates Figure 10 as it is in the paper 'Pseudo-Marginal Hamiltonian Monte Carlo' (Alenlov, 2021). NUTS extends Alenlov (2021)'s method.

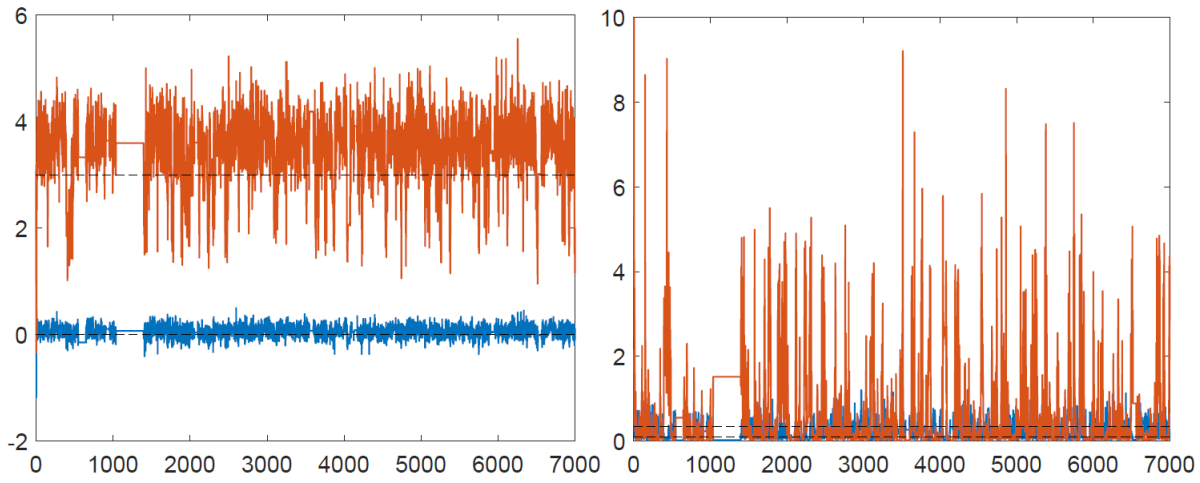
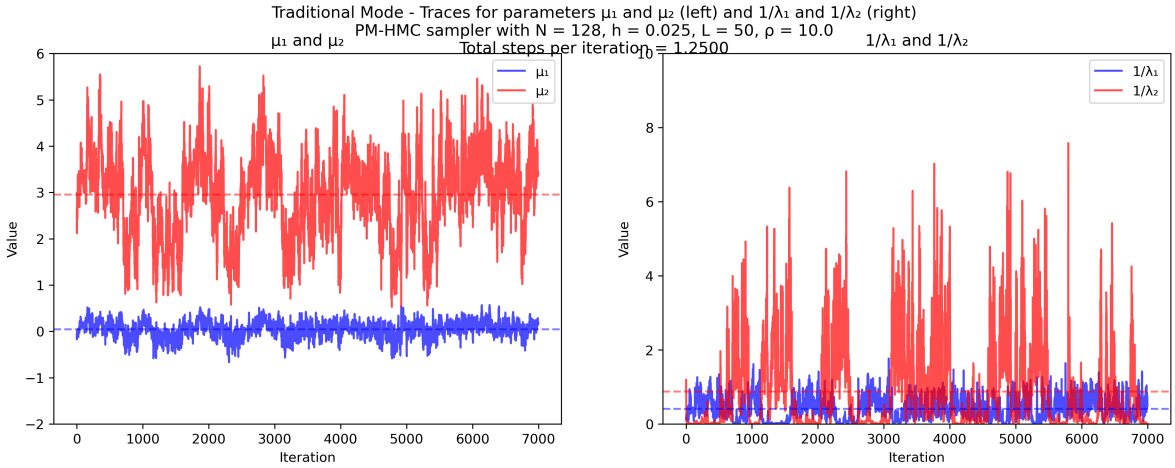


Figure 10: Traces for parameters μ_1 and μ_2 (left) and $1/\lambda_1$ and $1/\lambda_2$ (right) for the **pseudo-marginal HMC sampler** with $N = 128$.

7.1 PM-HMC

This part exhibits results from performing PM-HMC. Relative parameters are presented in the graph. The red and blue dashlines represents the average of all samples.



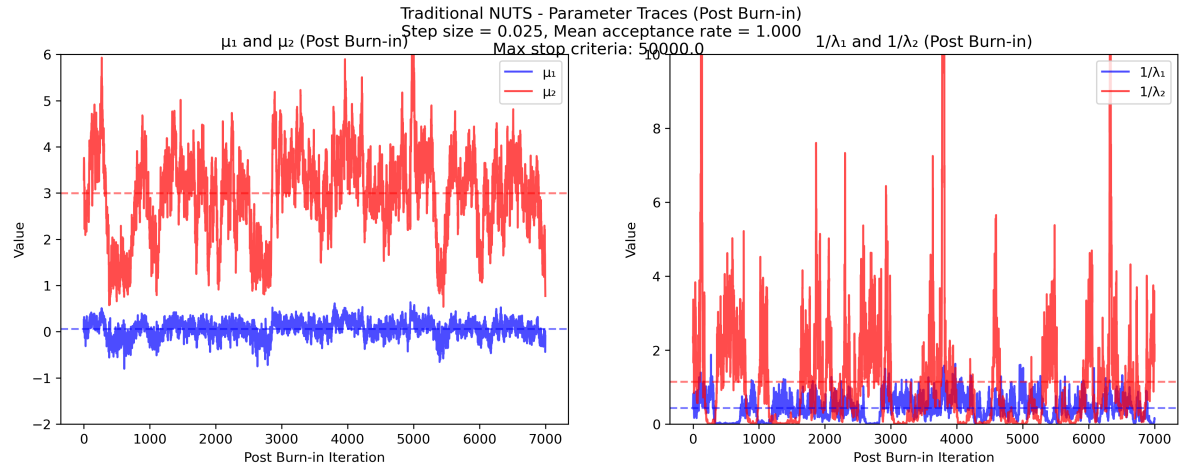
Detailed results are as below:

True β	-0.7565	-0.0685	0.0759	-1.2573	-0.2319	-1.8107	0.0998	-0.5099
Predict β	-0.7316	-0.0300	0.0723	-1.1904	-0.2603	-1.7666	0.1168	-0.4827

True w_1	0.8	True μ_1	0.0	True μ_2	3.0
Predict w_1	0.8083	Predict μ_1	0.0442	Predict μ_2	2.9515

7.2 PM-NUTS

This part exhibits results from running NUTS that is discussed in Chapter 6.1 of this article, without using HNN. Similarly, the dash lines, which represents the average of all samples, are close to true values.

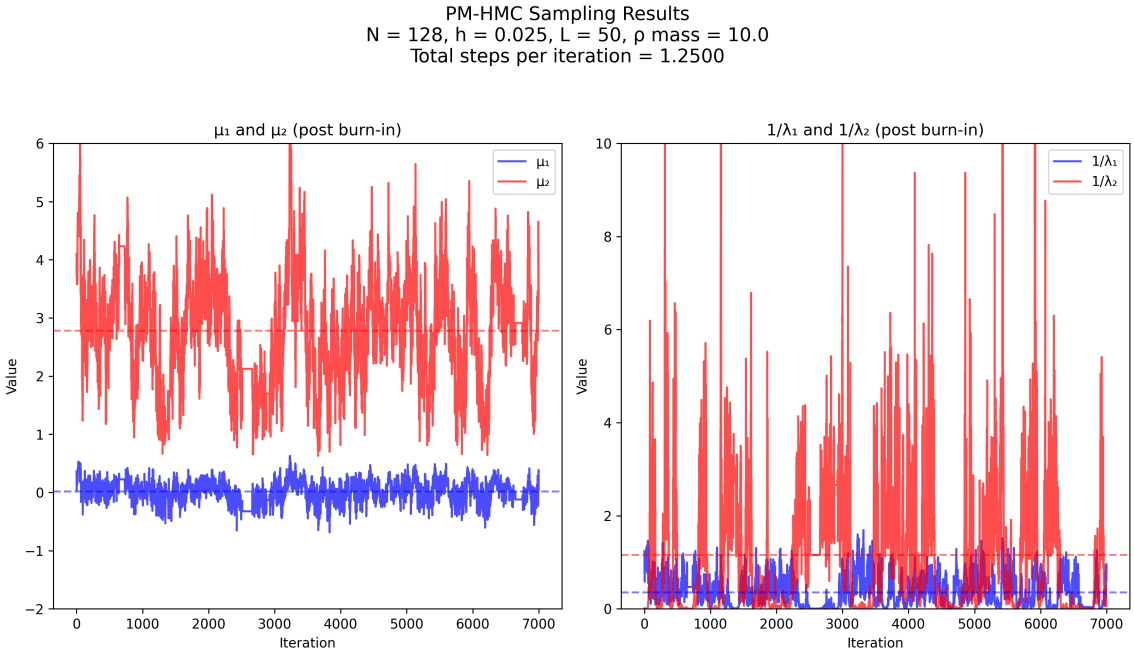


Detailed results are as below:

True β	-0.7565	-0.0685	0.0759	-1.2573	-0.2319	-1.8107	0.0998	-0.5099
Predict β	-0.7315	-0.0299	0.0713	-1.1878	-0.2596	-1.7646	0.1130	-0.4822

True w_1	0.8	True μ_1	0.0	True μ_2	3.0
Predict w_1	0.8271	Predict μ_1	0.0590	Predict μ_2	3.0124

7.3 PM-HMC-TFP



This part illustrates results from performing PM-HMC with TFP Transition Kernel

True β	-0.7565	-0.0685	0.0759	-1.2573	-0.2319	-1.8107	0.0998	-0.5099
Predict β	-0.7317	-0.0316	0.0724	-1.1879	-0.2594	-1.7645	0.1113	-0.4838

True w_1	0.8	True μ_1	0.0	True μ_2	3.0
Predict w_1	0.7872	Predict μ_1	0.0216	Predict μ_2	2.7595

8 HNN-HMC

In this part, I exhibits valid results from performing HNN-HMC, and HNN-PM-NUTS with online monitoring.

8.1 Results

In this case, I train $H(\theta, \rho)$, and use the two-step leapfrog to perform HNN-HMC. To apply the property of convergence in probability, I set number of **u** to be 2048. Compared to PM-HMC, the speed of the sampling is fast, so more samples are available in a short time. But the flaw is similar to Dhulipala (2023), which states that the sampler is unable to rapidly move to regions of high probability density after visiting a region of low proability density.

8.1.1 HNN-HMC

In this part, I present HNN-HMC that uses Hamiltonian Neural Networks and its gradients $\nabla_\theta \tilde{H}(\theta, \rho)$ and $\nabla_\rho \tilde{H}(\theta, \rho)$ to perform HMC. As stated above, one step of leapfrog is performed as:

$$\theta \leftarrow \theta + \frac{h}{2}\nabla_\rho \tilde{H}(\theta, \rho)$$

$$\rho \leftarrow \rho - h\nabla_\theta \tilde{H}(\theta, \rho)$$

$$\theta \leftarrow \theta + \frac{h}{2}\nabla_\rho \tilde{H}(\theta, \rho)$$

To collect a single sample, the above process is repeated for L times. At the begining of each sample collection, initialize $\rho \sim N(0_d, I_d) \times \sqrt{m_\rho}$

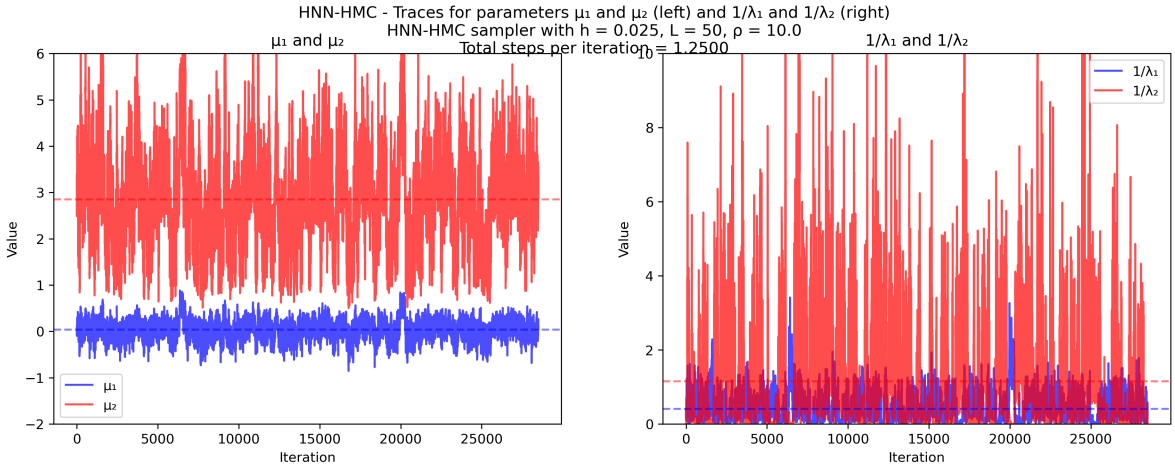
To perform Metropolis-Hasting acceptance, calculate

$$\Delta\tilde{H} = \tilde{H}(\theta, \rho) - \tilde{H}(\theta', \rho')$$

accept the new sample with probability

$$\alpha = \min(1, exp(\Delta\tilde{H}))$$

If accepted, $\theta \leftarrow \theta'$, Do not keep ρ'.



Detailed results are as below:

True β	-0.7565	-0.0685	0.0759	-1.2573	-0.2319	-1.8107	0.0998	-0.5099
Predict β	-0.7293	-0.0306	0.0696	-1.1851	-0.2601	-1.7614	0.1135	-0.4811

True w_1	0.8	True μ_1	0.0	True μ_2	3.0
Predict w_1	0.8048	Predict μ_1	0.0378	Predict μ_2	2.8503

8.1.2 HNN-NUTS

Currently I have not obtained a good results using this method.

8.2 Is there any advantage of this Hamiltonian Neural Network approach versus the numerical integrator?

The answer to this question is, yes, it accelerates the sampling process. But for a one-time task, it is not worth using this method, because the computation cost from generating training samples and training HNN would be much larger.

To make this method of more advantage, we should train $H(\theta, \rho|y, Z)$ that takes not only θ and ρ, but also different y and Z as inputs. Then when there is an update in data, we can use HNN to infer the new parameter.

Part VI
Tests

9 Test plans

This section briefly discusses what unittests are performed. Detailed of the test can be found in the README file on Github site.

9.1 Initial samples

This section tests if the generate_samples.py give correct shapes of θ, y, Z. Test if initialize_theta() in params.py generate the initial θ that is of the same shape, and if it always generate the same initialization.

9.2 Log likelihood

There are two files that compute log-likelihood function and its gradients with respect to θ and $\mathbf{u}$. The first one is log_likelihood_auto.py, which uses importance sampling to compute log-likelihood function and use Tensorflow auto-differentiation to compute the gradients. The second file is log_likelihood_stable.py, which explicitly computes both log-likelihood function and the gradients. Based on this, the test contains two parts:

1. Make sure all functions in log-likelihood function take the correct form of input, and output correct outputs. The correctness lies in outputs' type, shape, and values.

2. Make sure the final log-likelihood, and its gradients with respect to θ and $\mathbf{u}$ in log_likelihood_auto.py is correct. This utilizes log_likelihood_stable.py.

9.3 Strang splitting

This part tests pm_hmc_steps.py, mainly focusing on the correctness of input/output shapes, time reversibility and symplectic property.

9.4 No U turn sampling

This part tests nuts_hnn_olm_complex.py. First, I test if the built-in single_leapfrog_update produces correct shape and finite results. Then, I test the shape of the output of build_tree. Lastly I tried if the NUTS can function by running 3 samples.

9.5 Hamiltonian Neural Networks

This part tests hnn_architectures.py, collect_hamiltonian_trajectories.py, train_hnn_hamiltonian.py. The tests ensure that the HNN architecture, training data collection, and training process function as expected.

9.6 HNN and HMC

This section focuses on the integration of HNN with Hamiltonian Monte Carlo (HMC) in run_hnn_hmc.py. It tests the proper loading and execution of the HNN and verifies the energy conservation property during HMC sampling.

10 Test results

Most tests are passed. But there are several failure worth noticing:

1. Time reversibility does not hold for auxiliary variable **u**. I simulate the trajectories forward 10 steps with step size 0.01, and then simulate the trajectories backward by 10 steps with step size 0.01. The error between the final **u** and initial **u** exceeds the threshold, which is set at 1.0.
2. Hamiltonian conservation does not hold for HNN for large step size, for example, 0.5 at one step.

Part VII
Application in Finance

To apply PM-HMC, we require several conditions

1. We have a dataset that is generated through a known data generating process with noises
2. We need to infer the parameters in the data generating process
3. The likelihood function is intractable, otherwise traditional HMC or NUTS can be applied.

11 Example of Bond's term premium

11.1 Model setting

We want to infer the term premium of zero coupon bonds of different maturities. Specifcally, under real-world probability measure

$$\frac{dP_t^\tau}{P_t^\tau} = (r_t + RP_t^\tau)dt + \sigma_t^\tau dB_t$$

where P_t^τ refers to bond prices of maturity τ at time t. r_t is the short rate, which can be taken as the overnight interbank rate controlled by the Fed. RP_t^τ is the term premium, or risk premium of long-term bond with maturity τ, which arises from investor's risk-averse taste and uncertainty. Below is a specific example on how the term premium is priced, which is based on preferred-habitat model.

The Bond price with maturity τ follows

$$\frac{dP_t^\tau}{P_t^\tau} = \left(r_t + \frac{\gamma}{W_t}\sigma_t^\tau \int X_t^s \sigma_t^s ds\right) dt + \sigma_t^\tau dB_t$$

where X_t^s is the demand of bond at maturity s. W_t is the total wealth of the society which usually can be measured by M2, and γ is the risk-averse parameter. P_t can be think of as a value function that has two state variables $P(r_t, W_t)$. Note that I am not assuming the time to maturity τ as a state variable in that in real application, maturities are discretized and a 30-year bond would still be classified as a 30-year one after ten day. The short rate and wealth follows the following law of motion:

$$dr_t = \kappa\left(r - \bar{r}\right) dt + \sigma_t^r dB_t$$

$$dW_t = \omega dt + \sigma^W dB_t$$

Lastly, we can take log on P_t, so that

$$dlogP_t = \left(r_t + \frac{\gamma}{W_t}\sigma_t^\tau \int X_t^s \sigma_t^s ds - \frac{1}{2}\left(\sigma_t^\tau\right)^2\right) dt + \sigma_t^\tau dB_t$$

11.2 Data structure and parameters to be inferred

Data is time-series data. First, the noise in this model comes from the single Brownian Motion dB_t. Although it is a process noise rather than usual cross-sectional noise, it can still be considered as the noise in DGP because the data is time-series. Denote the data as $y = y_{1:T}$, and y_t is the sample collected at time t. Within each sample, we have observations $\{logP_t^\tau, X_t^\tau, r_t, W_t\}$, where maturities τ are discretized into several intervals and $\tau \in \{3m, 1y, 2y, 5y, 7y, 10y, 15y, 20y, 30y\}$. According to market clearing condition, X_t^τ can be measured by current debt outstanding within the interval of maturity τ. $logP_t$ can be obtained from the market zero-coupon yield curve and computed as

$$logP_t^\tau = -YTM_t^\tau \times \tau$$

If we discretize the law of motions and let $dt = \Delta$, and Δ^s be the length of interval for a specific maturity s, then the data generating process of y_t is given by

$$logP_t^\tau = logP_{t-\Delta}^\tau + \left(r_{t-\Delta} + \frac{\gamma}{W_{t-\Delta}} \sigma^\tau \sum X_t^s \sigma^s \Delta^s - \frac{1}{2} \left(\sigma^\tau\right)^2 \right) dt + \sigma^\tau \sqrt{\Delta} N(0,1)$$

$$r_t = r_{t-\Delta} + \kappa(r_{t-\Delta} - \bar{r})dt + \sigma^r \sqrt{\Delta} N(0,1)$$

$$W_t = W_{t-\Delta} + \omega\Delta + \sigma^W \sqrt{\Delta} N(0,1)$$

I assume within time period $1 : T$, the parameters to be inferred are time-invariant, which are $\theta = \{\gamma, \sigma^s, \kappa, \sigma^r, \omega, \sigma^W, \bar{r}\}$.

11.3 Bayesian inference

Several factors contrubute to the intracatablity of the likelihood:

1. The drift term of $logP_t$ involces the short rate $r_{t-\Delta}$, and a non-linear term $\gamma/W_{t-\Delta}$ multiplied by an integral term $\sigma^\tau \sum X_t^s \sigma^s \Delta^s$ capturing risk exposure across maturities.

2. Even when employing an Euler–Maruyama discretization, the resulting one-step transition densities retain the complicated nonlinearity and integral terms. This complexity renders the overall joint likelihood function, constructed as a product of these one-step densities, analytically intractable and numerically challenging to evaluate precisely.

To overcome the challenges posed by an intractable likelihood, the Pseudo-Marginal Hamiltonian Monte Carlo (PM-HMC) approach is employed. The core idea is to replace the true likelihood with an unbiased estimator while constructing a modified Hamiltonian system for sampling from the posterior.

11.4 Applications

The current yield curve is extremely flat. After inferring these parameters, we can tell whether flat curve is mainly driven by

1. The expectation of lower short rates in the future, measured by κ and $\bar{r}$
2. Low term-premium out of low bond price volatility
3. Too much liquidity or expected future liquidity in the market, measured by ω
4. Low risk-averse parameter, or the animal spirit, measured by γ.

Part VIII
Appendix

12 Manual Computation of Gradients

Data generation is the same. Inputs are data points Y with dimension $[T, n]$. T is number of individuals, each individual has 6 observations. (Bernoulli 0/1). Parameters are of a bit different, to be inferred are $\theta = \{\beta, \mu_1, \mu_2, log(\lambda_1), log(\lambda_2), \eta = logit(w_2) = logit(1 - w_1)\}$

Dimensions are

$$T = 500$$
$$N = 128$$
$$n = 6$$
$$p^z = 8$$

Initial shapes of $Z = [T, n, p^z]$, shape of $\beta = [p^z]$

12.0.1 Step 1

Initialization:

1.1 Initialize θ^0 . β^0 contains p^z elements, each element is initialized from $N(0, 1)$. $\mu_1^0 = 0.0$, $\mu_2^0 = 0.0$, $\lambda_1^0 = 1.0$, $\lambda_2^0 = 0.1$, $w_1^0 = 0.5$,so that

$$\theta^0 = \{\beta^0, \mu_1^0, \mu_2^0, log(\lambda_1^0), log(\lambda_2^0), \eta^0 = logit(1 - w_1^0)\}$$

1.2 Initialize ρ from $N(0_d, I_d)$, and $\mathbf{p}$ from $N(0_D, I_D)$, where D is $[T, N]$, d is the dimension of θ.

1.3 Initialize $\mathbf{u}$ from $N(0_D, I_D)$.

12.0.2 Step 2

There are several simplifications

2.1 $X = \gamma_k(\theta, \mathbf{u})$. $\gamma_k : \mathbb{R}^p \to \mathbb{R}^1$. The dimension of X is $[T, N]$. The exact form of $\gamma_k(\theta, \mathbf{u})$ is:

Generate $u \sim N(0, 1)$, then compute

$$X = 3 \times \mathbf{u}$$

which has shape $[T, N]$

2.2 Unsqueeze X to shape $[T, N, 1]$ and $Z_{i,j}^{\mathbb{T}}\beta$ to $[T, 1, n]$. Broadcasting leads to $X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta$ with shape $[T, N, n]$. Unsqueeze $y_{i,j}$ from $[T, n]$ to $[T, 1, n]$.

$$logg_\theta(y_i|X_{i,k}) = \sum_{j=1}^{n} \left[y_{i,j} \left(-softplus\left(-\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right) + (1 - y_{i,j}) \left(-softplus\left(\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right)\right]$$

with shape $[T, N]$. This shape comes from summing across the last dimension. In real application, I use the exact form instead:

$$logg_\theta(y_i|X_{i,k}) = -\sum_{j=1}^{n} \left[y_{i,j} \times log\left(1 + exp\left(-\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right) + (1 - y_{i,j}) \times log\left(1 + exp\left(\left(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta\right)\right)\right)\right]$$

2.3 $logq_\theta$ is also naive

$$logq_\theta = -\frac{1}{2}log(2\pi) - \frac{1}{2}log(9) - \frac{X^2}{18}$$

with shape $[T, N]$.

2.4 f_θ is the prior of $X_{i,k}$ and $X_{i,k} \sim w_1 N(\mu_1, \lambda_1^{-1}) + (1 - w_1)N(\mu_2, \lambda_2^{-1})$. The algorithm for $logf_\theta$ is as below:

$$w_2 = sigmoid(\eta); w_1 = 1 - w_2; \lambda_1 = exp(log(\lambda_1)); \lambda_2 = exp(log(\lambda_2))$$
$$z_1 = log(w_1) + \left[\frac{1}{2}log(\lambda_1\pi) - \frac{1}{2}log(2) - \frac{exp(log(\lambda_1))}{2}(X - \mu_1)^2\right]$$
$$z_2 = log(w_2) + \left[\frac{1}{2}log(\lambda_2\pi) - \frac{1}{2}log(2) - \frac{exp(log(\lambda_2))}{2}(X - \mu_2)^2\right]$$
$$maxz = max(z_1, z_2)$$
$$logf_\theta = maxz + log\left[exp(z_1 - maxz) + exp(z_1 - maxz)\right]$$

with shape $[T, N]$. Because the shape for z_1 and z_2 are also $[T, N]$.

2.5

$$log\omega_\theta(y_i, X_{i,k}) = loggg_\theta(y_i|X_{i,k}) + logf_\theta(X_{i,k}) - logq_\theta$$

2.6 Next, gradients are computed. First, define a relative weight parameter as

$$RelativeWeight = \begin{pmatrix} exp(z_1 - logf_\theta) \\ exp(z_2 - logf_\theta) \end{pmatrix}$$

which has a shape of $[T, N, 2]$, with $[T, N, 1st]$ determined by $exp(z_1 - logf_\theta)$ and $[T, N, 2nd]$ determined by $exp(z_2 - logf_\theta)$.

2.7 Again, unsqueeze X to shape $[T, N, 1]$. Then, gradients are:

$$\frac{\partial logf_\theta}{\partial \mu} = \begin{pmatrix} \lambda_1(X - \mu_1) \\ \lambda_2(X - \mu_2) \end{pmatrix} \times RelativeWeight$$

which has a shape of $[T, N, 2]$. $[T, N, 1st]$ is $\partial logf_\theta / \partial \mu_1$. $[T, N, 2nd]$ is $\partial logf_\theta / \partial \mu_2$.

$$\frac{\partial logf_\theta}{\partial log\lambda} = \begin{pmatrix} \frac{1}{2} - \frac{1}{2}\lambda_1(X - \mu_1)^2 \\ \frac{1}{2} - \frac{1}{2}\lambda_2(X - \mu_2)^2 \end{pmatrix} \times RelativeWeight$$

which has a shape of $[T, N, 2]$. $[T, N, 1st]$ is $\partial logf_\theta / \partial log\lambda_1$. $[T, N, 2nd]$ is $\partial logf_\theta / \partial log\lambda_2$.

2.8 Then, I need to compute $\partial logf_\theta / \partial \eta$.

$$\frac{\partial logf_\theta}{\partial \eta} = exp\left(z_2 + log(\frac{1}{1 + exp(\eta)}) - logf_\theta\right) - exp\left(z_1 + log(\frac{1}{1 + exp(\eta)}) + \eta - logf_\theta\right)$$

with shape $[T, N]$.

2.9 The gradients with respect to $\mathbf{u}$:

$$\frac{\partial logf_\theta}{\partial X} = -\sum_1^2 \frac{\partial logf_\theta}{\partial \mu}$$

$$\frac{\partial logf_\theta}{\partial \mathbf{u}} = 3 \times \frac{\partial logf_\theta}{\partial X}$$

With respect to the importance sampling $q_\theta(X)$, I use a very naive term:

$$\begin{aligned} logq_\theta &= -\frac{1}{2}log(2\pi) - \frac{1}{2}log(9) - \frac{X^2}{18} \\ &= -\frac{1}{2}log(2\pi) - \frac{1}{2}log(9) - \frac{\mathbf{u}^2}{2} \end{aligned}$$

so that

$$\frac{\partial logq_\theta}{\partial \mathbf{u}} = -\mathbf{u}$$

Lastly, $X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta$ is with shape $[T, N, n]$. Calculate

$$P_{i,j} = Sigmoid(X_{i,k} + Z_{i,j}^{\mathbb{T}}\beta)$$

with shape $[T, N, n]$. Unsqueeze $y_{i,j}$ from $[T, n]$ to $[T, 1, n]$. Let

$$A = (1 - y) \times P_{i,j} - y \times (1 - P_{ij})$$

which has shape $[T, N, n]$. Then,

$$\frac{\partial loggg_\theta}{\partial \mathbf{u}} = -3 \times \sum_{j=1}^{n} A$$

The summation is across the third dimension so the shape is $[T, N]$.

2.10 In this step, I am going to calculate the normalized ω as defined in equation (14) and (15) in the paper. We already know that

$$log\omega_\theta(y_i, X_{i,k}) = loggg_\theta(y_i|X_{i,k}) + logf_\theta(X_{i,k}) - loggq_\theta$$

with shape $[T, N]$. A log-sum-exp method can still be applied here. The steps are below:

$$max\omega = \max_N log\omega_\theta(y_i, X_{i,k})$$

where we find the max term accross the second dimension. So the shape of $max\omega$ is $[T, 1]$.Then,

$$\omega = exp(log\omega_\theta(y_i, X_{i,k}) - max\omega)$$

which has shape $[T, N]$. Lastly, we can obtain the log-likelihood and normalized ω:

$$log\hat{p}(y_i|\theta) = log\left(\sum_{k=1}^{N}\omega\right) + max\omega - log(N)$$

which has shape $[T, 1]$ as the summation is across the second dimension of ω. Lastly,

$$log\hat{p}(y|\theta) = \sum_{i=1}^{T} log\hat{p}(y_i|\theta)$$

The normalized ω is

$$W = \frac{\omega}{\sum_{k=1}^{N}\omega}$$

which has a shape of $[T, N]$.

2.11 What remain unknown are:

$$\frac{\partial log\hat{p}(y|\theta)}{\partial \mathbf{u}}; \frac{\partial log\hat{p}(y|\theta)}{\partial \beta}; \frac{\partial log\hat{p}(y|\theta)}{\partial \mu_1}; \frac{\partial log\hat{p}(y|\theta)}{\partial \mu_2}; \frac{\partial log\hat{p}(y|\theta)}{\partial log\lambda_1}; \frac{\partial log\hat{p}(y|\theta)}{\partial log\lambda_2}; \frac{\partial log\hat{p}(y|\theta)}{\partial \eta}$$

First, the gradients with respect to $\mathbf{u}$ is of shape $[T, N]$ and is calculated by

$$\frac{\partial log\hat{p}(y|\theta)}{\partial \mathbf{u}} = W \times \left(\frac{\partial logf_\theta}{\partial \mathbf{u}} + \frac{\partial loggg_\theta}{\partial \mathbf{u}} - \frac{\partial loggq_\theta}{\partial \mathbf{u}}\right)$$

Since $\theta^{other} = \{\mu_1, \mu_2, log(\lambda_1), log(\lambda_2), \eta\}$ are only contained in f_θ. The gradients can be collectively calculated as

$$\frac{\partial log\hat{p}(y|\theta)}{\partial \theta^{other}} = \sum_{i=1}^{T}\sum_{i=1}^{N}\left(W \times \frac{\partial logf(y|\theta)}{\partial \theta^{other}}\right)$$

where $\frac{\partial logf(y|\theta)}{\partial \theta^{other}}$ are calculated above, and will give a vector of 5 elements.

2.12 The calculation of gradients of β is as below. First, recall that we already have

$$A = (1 - y) \times P_{i,j} - y \times (1 - P_{ij})$$

and Z with shape $[T, n, p^z]$.Reshape Z to $[T, p^z, n]$. For each row in the second dimension, defind as $Z_i = Z[:, i, :]$, I can use the broadcast mechanism to calculate

$$Z_i A = Z_i \times A$$

with shape $[T, N, n]$. Summing up across the third dimension and take a negative term, I have

$$gbeta_i = -\sum_{1}^{n} Z_i A$$

with shape $[T, N]$. Since I have p^z numbers of $Z_i A$, I will have a $gbeta$ with shape $[T, N, p^z]$, where each line of dimension p^z represents one $gbeta_i$. Lastly,

$$\frac{\partial log\hat{p}(y|\theta)}{\partial \beta} = \sum_{i=1}^{T}\sum_{i=1}^{N}(W \times gbeta)$$

which gives a vector of p^z numbers of β.
2.13 Combine $\frac{\partial log\hat{p}(y|\theta)}{\partial \beta}$ and $\frac{\partial log\hat{p}(y|\theta)}{\partial \theta^{other}}$ together into $\frac{\partial log\hat{p}(y|\theta)}{\partial \theta}$ which is a vector with p^z+5 elements.
2.14 Compute prior + loglikelihood

$$logp(\theta) = -\frac{1}{2} \times \left[Length(\theta) \times log(2\pi\sigma^2) + \frac{1}{\sigma^2} \sum \theta^2 \right]$$

$$PriorLogLikelihood = log\hat{p}(y|\theta) + logp(\theta)$$

where $\sigma^2 = 100$
2.15 In the codes, I have a main function that takes input (θ, u, y, Z), and output

$$(PriorLogLikelihood, \frac{\partial log\hat{p}(y|\theta)}{\partial \mathbf{u}}, \frac{\partial log\hat{p}(y|\theta)}{\partial \theta})$$

13 HNN Attempts

13.1 Methods

13.1.1 Method 1: The most general case

In this case, the HNN takes input $\{\theta, \rho, \mathbf{u}, \mathbf{p}\}$, output $\left\{\tilde{H}, \nabla_\theta \tilde{H}, \nabla_\rho \tilde{H}, \nabla_{\mathbf{u}} \tilde{H}, \nabla_{\mathbf{p}} \tilde{H}\right\}$. The total loss incorporates the loss of four gradients. This is an ideal version but is the most difficult to be performed. The dimension of inputs and outputs of the neural network is $26 + 2 \times T \times N$, which is too high. The difficulty lies in two aspects. First, a large neural network is required so that we can have enough parameters that can help approximate the data. Second, because of the high dimension of this data and limited memory resource, we can only have a small number of samples in the training data, which is far less than the number of dimension.

Below I present several simplified versions of HNN.

13.1.2 Method 2: Simpler case

In this case, Hamilton Neural Network (HNN) can be applied to predict $\nabla_\theta \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\}$ and $\nabla_{\mathbf{u}} \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\}$. The HNN takes the following input:

$$\left(batch, \quad \begin{pmatrix} \theta(13,) \\ \mathbf{u}(T,N) \end{pmatrix} \right)$$

and output

$$\left(batch, \quad \begin{pmatrix} \nabla_\theta \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\}(13,) \\ \nabla_{\mathbf{u}} \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\}(T,N) \end{pmatrix} \right)$$

To train this HNN, I minimize the total loss

$$\begin{aligned} & w_1 \left\| \nabla_\theta \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\} - \nabla_\theta \left\{logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\} \right\|^2 \\ & + w_2 \left\| \nabla_{\mathbf{u}} \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\} - \nabla_{\mathbf{u}} \left\{logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\} \right\|^2 \\ & + w_3 \left\| \left\{log\tilde{p}(\theta) + log\tilde{p}\left(y|\theta, \mathbf{u}\right)\right\} - \left\{logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\} \right\|^2 \end{aligned}$$

where w_1, w_2, w_3 are choices of weights for each individual loss. What really matters are the gradients, but adding the posterior to the loss may also be beneficial to the training results and serves as an attempt.

For each $\{y, Z\}$ pair, I record current $\left\{\theta, \mathbf{u}, logp(\theta|y,Z), \nabla_\theta \left\{logp(\theta|y,Z)\right\}, \nabla_{\mathbf{u}} \left\{logp(\theta|y,Z)\right\}\right\}$ in Step B of Strang splitting:

$$\begin{aligned} \rho &\leftarrow \rho + h\nabla_\theta \left\{logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\} \\ \mathbf{p} &\leftarrow \mathbf{p} + h\nabla_{\mathbf{u}} \left\{logp(\theta) + log\hat{p}\left(y|\theta, \mathbf{u}\right)\right\} \end{aligned}$$

The difficulty of training this HNN lies in the high dimension of inputs $\mathbf{u}$ and outputs $\nabla_{\mathbf{u}} \left\{log\tilde{p}(\theta|y)\right\}(T,N)$, which are both of shape $[500, 128]$. So, it faces similar cons as described in 5.1, and the training performs poorly.

13.1.3 Method 3: Case with property of convergence in probability

We know

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \rho\\ \mathbf{u}\\ \mathbf{p}\end{pmatrix}=\begin{pmatrix}+\partial H/\partial\rho\\ -\partial H/\partial\theta\\ +\partial H/\partial\mathbf{p}\\ -\partial H/\partial\mathbf{u}\end{pmatrix}=\begin{pmatrix}\rho/m_\rho\\ \nabla_\theta\left\{logp(\theta)+log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}\\ \mathbf{p}\\ -\mathbf{u}+\nabla_{\mathbf{u}}\left\{logp(\theta)+log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}\end{pmatrix}\rightarrow\begin{pmatrix}\rho/m_\rho\\ \nabla_\theta\left\{logp(\theta)+logp\left(y|\theta\right)\right\}\\ \mathbf{p}\\ -\mathbf{u}+\nabla_{\mathbf{u}}\left\{logp(\theta)+logp\left(y|\theta\right)\right\}\end{pmatrix}$$

as $N\rightarrow\infty$, because $log\hat{p}\left(y|\theta,\mathbf{u}\right)\xrightarrow{p}logp\left(y|\theta\right)$. More specifically, for any $h>0$,

$$log\hat{p}\left(y|\theta,\mathbf{u}cos\left(h\right)+\mathbf{p}sin\left(h\right)\right)\xrightarrow{p}logp\left(y|\theta\right).$$

Let HNN output $\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\}$. Now we already have $\nabla_\theta\left\{logp(\theta)+log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}$, we can use it to train $\nabla_\theta\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}$.

The I can apply

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \rho\end{pmatrix}=\begin{pmatrix}\rho/m_\rho\\ \nabla_\theta\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}\end{pmatrix}$$

The Algorithm is that, the HNN takes input $\{\theta\}$,output $\left\{\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\},\nabla_\theta\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}\right\}$, then construct the loss function

$$w_1\left\|\left\{logp(\theta)+log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}-\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}\right\|^2+w_2\left\|\nabla_\theta\left\{logp(\theta)+log\hat{p}\left(y|\theta,\mathbf{u}\right)\right\}-\nabla_\theta\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}\right\|^2$$

After using HNN to obtain $\nabla_\theta\left\{log\tilde{p}(\theta|y)\right\}$, we can then apply the simple two-step leapfrog:

$$\theta\leftarrow\theta+\frac{h}{2m_\rho}\rho(t)$$

$$\rho\leftarrow\rho+h\nabla_\theta\left\{log\tilde{p}(\theta)+log\tilde{p}\left(y|\theta\right)\right\}$$

$$\theta\leftarrow\theta+\frac{h}{2m_\rho}\rho(t)$$

The merit of this method is that we don't have to train for a high dimension of input, and make the use of the property of convergence of PM-HMC to ideal HMC. The problem with this method is that it does not preserve the symplectic structure implied by the Hamiltonian. The result of this method is also not ideal.

13.1.4 Method 4: Simpler case using Hamiltonian

We know

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \rho\\ \mathbf{u}\\ \mathbf{p}\end{pmatrix}=\begin{pmatrix}+\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\rho\\ -\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\theta\\ +\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\mathbf{p}\\ -\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\mathbf{u}\end{pmatrix}\rightarrow\begin{pmatrix}+\partial H(\theta,\rho)/\partial\rho\\ -\partial H(\theta,\rho)/\partial\theta\\ -\\ -\end{pmatrix}$$

Let HNN output $H(\theta,\rho)$. Now we already have $\nabla_\theta H(\theta,\rho,\mathbf{u},\mathbf{p})$, we can use it to train $\nabla_\theta H(\theta,\rho)$ and $\nabla_\rho H(\theta,\rho)$. Because

$$H(\theta,\rho,u,p)=-logp(\theta)-log\hat{p}(y|\theta,u)+\frac{1}{2}\left\{\frac{\rho^T\rho}{m_\rho}+u^Tu+p^Tp\right\}$$

$$H(\theta,\rho)=-logp(\theta)-logp(y|\theta)+\frac{1}{2}\frac{\rho^T\rho}{m_\rho}$$

If $log\hat{p}(y|\theta,u)$ converges to $logp(y|\theta)$ in probability, then as N gets larger, $\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\rho$ converges to $\partial H(\theta,\rho)/\partial\rho$, and $\partial H(\theta,\rho,\mathbf{u},\mathbf{p})/\partial\theta$ converges to $\partial H(\theta,\rho)/\partial\theta$

The Algorithm is that, the HNN takes input $\{\theta,\rho\}$,output $\{\tilde{H}(\theta,\rho),\nabla_\theta\tilde{H}(\theta,\rho),\nabla_\rho\tilde{H}(\theta,\rho)\}$, then construct the loss function:

$$w_1\left\|\nabla_\theta H(\theta,\rho,\mathbf{u},\mathbf{p})-\nabla_\theta\tilde{H}(\theta,\rho)\right\|^2+w_2\left\|\rho/m_\rho-\nabla_\rho\tilde{H}(\theta,\rho)\right\|^2$$

13.1.5 Method 5: Learning dynamics

Using the merit above, suppose the shape of the training data is $[Batch, N, 2, size(\theta)]$. The first dimension is the dimension of each sample, the second dimension is total number of leapfrogs when collecting a single sample. The third dimension is $\{\theta, \rho\}$.

Now, we use HNN to predict $\{\tilde{\theta}, \tilde{\rho}\}$ with shape $[Batch, N, 2, size(\theta)]$. The algorithm is as below:

1. For each sample, denote the start of $\{\tilde{\theta}, \tilde{\rho}\}$ as $\{\tilde{\theta^0}, \tilde{\rho^0}\}$, and let $\{\tilde{\theta^0}, \tilde{\rho^0}\} = \{\theta^0, \rho^0\}$, where $\{\theta^0, \rho^0\}$ comes from the starting point of the training data, which is $Data[Batch, 0, 2, size(\theta)]$.
2. The HNN takes input $\{\tilde{\theta}, \tilde{\rho}\}$,output $\{\nabla_\theta H(\tilde{\theta}, \tilde{\rho}), \nabla_\rho H(\tilde{\theta}, \tilde{\rho})\}$.
3. Simulate the path of $\{\tilde{\theta}, \tilde{\rho}\}$through iteration, until it reaches $\{\tilde{\theta^N}, \tilde{\rho^N}\}$, using

$$\theta^{n\tilde{+}0.5} \leftarrow \tilde{\theta^n} + \frac{h}{2}\nabla_\rho H(\tilde{\theta^n}, \tilde{\rho^n})$$

$$\rho^{\tilde{n+1}} \leftarrow \tilde{\rho^n} - h\nabla_\theta H(\theta^{n\tilde{+}0.5}, \tilde{\rho^n})$$

$$\theta^{\tilde{n+1}} \leftarrow \theta^{n\tilde{+}0.5} + \frac{h}{2}\nabla_\rho H(\theta^{n\tilde{+}0.5}, \rho^{\tilde{n+1}})$$

only $\{\tilde{\theta^i}, \tilde{\rho^i} | i = integer\}$ is stored.

4. Construct the loss

$$w_1 \left\| \tilde{\theta} - \theta \right\|^2 + w_2 \left\| \tilde{\rho} - \rho \right\|^2$$

13.2 Comparison of 5 methods

Method 1 is the most general one but is the most difficult to train, because the input and output dimension is high. Method reduces the complexity of method 1 by around $1/2$, but breaks the symplectic structure of HNN. Method 3 applies the property that $log\hat{p}\,(y|\theta, \mathbf{u}) \xrightarrow{p} logp\,(y|\theta)$, which significantly reduces the dimension of inputs and outputs, but the symplectic condition still does not hold. Method 4 trains the gradient of both position and momentum so that symplectic structure holds. Method 5 is similar to method 4 but the loss is constructed by the whole trajectories of momentum and positions.