Fast Riemannian-Manifold Hamiltonian Monte Carlo for Hierarchical Gaussian-Process Models

Takashi Hayakawa * and Satoshi Asai

Posted Date: 11 November 2025

doi: 10.20944/preprints202511.0763.v1

Keywords: Gaussian process; Monte Carlo method; Bayesian statistics

 doi:10.20944/preprints202511.0763.v1

Article

Fast Riemannian-Manifold Hamiltonian Monte Carlo for Hierarchical Gaussian-Process Models

Takashi Hayakawa [1,2,*] **and Satoshi Asai** [1,2]

[1] Division of Pharmacology, Department of Biomedical Sciences, Nihon University School of Medicine, Oyaguchi-Kamicho 30-1, Itabashi, Tokyo 1738610, Japan
[2] Division of Genomic Epidemiology and Clinical Trials, Clinical Trials Research Centre, Nihon University School of Medicine, Oyaguchi-Kamicho 30-1, Itabashi, Tokyo 1738610, Japan
* Correspondence: hayakawa.takashi@nihon-u.ac.jp

Abstract

Hierarchical Bayesian models based on Gaussian processes are considered useful for describing complex nonlinear statistical dependencies among variables in real-world data. However, effective Monte Carlo algorithms for inference with these models have not yet been established, except for several simple cases. In this study, we show that, compared with the slow inference achieved with existing program libraries, the performance of Riemannian-manifold Hamiltonian Monte Carlo (RMHMC) can be drastically improved by optimising the computation order according to the model structure and dynamically programming the eigendecomposition. This improvement cannot be achieved when using an existing library based on a naive automatic differentiator. We numerically demonstrate that RMHMC effectively samples from the posterior, allowing the calculation of model evidence, in a Bayesian logistic regression on simulated data and in the estimation of propensity functions for the American national medical expenditure data using several Bayesian multiple-kernel models. These results lay a foundation for implementing effective Monte Carlo algorithms for analysing real-world data with Gaussian processes, and highlight the need to develop a customisable library set that allows users to incorporate dynamically programmed objects and finely optimises the mode of automatic differentiation depending on the model structure.

Keywords: Gaussian process; Monte Carlo method; Bayesian statistics

MSC: 65C05

1. Introduction

Bayesian analysis based on Gaussian processes (GPs) associated with positive-semidefinite kernels has been studied extensively from modelling, algorithmic, and theoretical points of view [1–8]. A GP serves as a prior process for generating nonlinear functions that describe relationships between variables. The goal of Bayesian analysis is to infer the posterior process for such nonlinear functions conditioned on given data or to estimate the average of quantities of interest over such a posterior process. Although the posterior process can be represented as a closed-form solution for the simplest problems (e.g., Chapter 6 of Ref. [9]), such a solution cannot be expected when GPs are used as building blocks for hierarchical models that capture complex statistical dependencies among variables in real-world data. Algorithms for inference with hierarchical GP models can be broadly divided into variational algorithms and Monte Carlo (MC) algorithms. Variational algorithms (e.g., Refs.[5,7]) are usually much faster than MC algorithms but introduce unavoidable bias due to discrepancies between the original and variational models. In contrast, MC algorithms (e.g., Refs.[6,10,11]) allow for precise computation of averages over the posterior, provided the algorithm is run until convergence. The preferred algorithm type depends on the purpose of the computation.

 doi:10.20944/preprints202511.0763.v1

With biometric and econometric applications in mind, we seek an efficient MC algorithm in this study. In these applications, biased results can be misleading and may adversely affect public health or the economy; therefore, the use of variational algorithms is not appropriate. However, to our knowledge, a standard MC algorithm for general hierarchical GP models has not yet been established. Svensson et al. [6] sampled a function from the posterior process using a closed-form solution for the GP component of their model conditioned on other parameters. Their formula can be applied only when the likelihood term is Gaussian and is thus not applicable to, for example, binary classification. Hensman et al. [12] introduced a non-Gaussian variational approximation of the posterior with the aid of inducing points sampled using Hamiltonian MC (HMC). Their method, however, is variational and not guaranteed to describe the posterior precisely, and its scalability with increasing numbers or dimensions of GPs remains unclear. Pandita et al. [11] introduced an adaptive sequential MC for GPs to solve problems in mechanical engineering. However, the authors used hundreds of CPU cores for inference, and the efficiency of their sampling scheme itself was not clearly demonstrated.

One fundamental issue in MC sampling from the posterior in a GP model is that the posterior may be highly stretched or compressed along unknown directions in the parameter space used for sampling. To address this problem, Paquet and Fraccaro [10] implemented Riemannian-manifold HMC (RMHMC) for GP models. With the Hessian metric they employed, RMHMC adjusts the direction of MC moves according to the curvature of the log-posterior density surface. They demonstrated that RMHMC is much more efficient than the ordinary HMC method based on the Euclidean metric. However, their method is applicable only to cases with log-concave posterior densities, whereas hierarchical models that describe complex dependencies among variables typically have non-log-concave posterior densities. To overcome this limitation, we develop an efficient implementation of RMHMC for hierarchical GP models using a soft-absolute Hessian metric introduced by Betancourt [13]. This metric transforms an indefinite Hessian into a positive-semidefinite matrix that shares eigenvectors with the Hessian but has associated eigenvalues set close to the absolute values of the original Hessian eigenvalues. Although a general-purpose implementation of RMHMC with this metric [14] based on PyTorch [15] is available, its performance is poor. We identify the reasons for this, and show that the computational complexity of the implementation can be considerably reduced.

The article is organised as follows. In section 2.1, we introduce our problem setting in which multiple GPs are used as building blocks for a hierarchical model, employing the representation of GPs introduced by Solin et al. [16]. In section 2.2, we provide a brief introduction to RMHMC with a soft-absolute Hessian metric. In sections 2.3 and 2.4, we identify two sources of redundancy in the calculation of the gradient flow and metric, respectively. We show how performance deteriorates when reverse-mode automatic differentiation and a divide-and-conquer algorithm for the eigendecomposition of symmetric matrices in PyTorch are naively applied. In section 3, we numerically demonstrate that our implementation considerably outperforms an implementation based on a general-purpose library. Specifically, in section 3.1, we compare the performance of different implementations using toy examples of varying sizes and show that both sources of redundancy contribute to the improvement. In section 3.2, we compare the performance of our implementation with that of no-U-turn (NUT)-HMC based on the Euclidean metric. We show that NUT-HMC becomes trapped within a very narrow region and fails to converge to the posterior, suggesting that the use of a data-adaptive metric is essential for efficient sampling. In section 3.3, we show, using a toy example and a real-world dataset, that our implementation successfully calculates the marginal likelihood, (that is, the Bayesian model evidence) within a reasonable computation time. Finally, in section 4, we discuss, based on these results, the advantages and limitations of the proposed method compared with existing approaches and suggest directions for future work. Mathematical notations used in this article are summarised in Table A1.

 doi:10.20944/preprints202511.0763.v1

2. Methods and Algorithms

2.1. Hierarchical GP Models and Their Approximate Representations

In this study, we consider hierarchical GP models for which the samplewise likelihood of *i.i.d.* data $X_i \in \mathbf{R}^d$ (with sample index $i \in \{1, 2, \ldots, N\}$) is parameterised by J nonlinear functions $\{f_j(X_i)\}_{1 \le j \le J}$ as

$$P(X_i | \{f_j\}_{1 \le j \le J}) = \exp(-U(\{f_j(X_i)\}_{1 \le j \le J})). \tag{1}$$

For instance, the simplest examples describing the relationship between a target variable $X_i^{(\mathrm{tgt})}$ and covariates $X_i^{(\mathrm{cov})}$ are as follows.

Example 1. Covariate-dependent target mean and variance ($J = 2$):

$$X_i^{(\mathrm{tgt})} = f_1(X_i^{(\mathrm{cov})}) + \epsilon_i, \;\; \epsilon_i \sim \mathcal{N}(0, \delta + \exp f_2(X_i^{(\mathrm{cov})})), \tag{2}$$

with fixed $\delta (> 0)$.

Example 2. Nonlinear Gaussian mixture model ($J = 4$):

$$\begin{aligned} X_i^{(\mathrm{tgt})} &= (1 - s_i) f_1(X_i^{(\mathrm{cov})}) + s_i f_2(X_i^{(\mathrm{cov})}) + \epsilon_i, \\ \epsilon_i \sim \mathcal{N}(0, \delta + \exp f_3(X_i^{(\mathrm{cov})})), &\quad s_i \sim \mathrm{Bernoulli}[1/(1 + \exp f_4(X_i^{(\mathrm{cov})}))]. \end{aligned} \tag{3}$$

In the above modelling, we assume that each of $\{f_j\}_{1 \le j \le J}$ is the sum of nonlinear functions generated by GP priors $\{\mathcal{G}_{jk}\}_{1 \le j \le J, k \in \mathcal{K}_j}$:

$$f_j = \sum_{k \in \mathcal{K}_j} f_{jk} + b_j, \;\; f_{jk} \sim \mathcal{G}_{jk}(\{\theta_{jk\ell}\}_{\ell \in \mathcal{L}_{jk}}), \;\; b_j \sim \mathcal{N}(0, \Sigma). \tag{4}$$

Each GP $\mathcal{G}_{jk}$ is associated with a positive-semidefinite kernel $k(\cdot, \cdot)$ that describes the prior covariance of the generated functions and is parameterised by scalar hyperparameters $\{\theta_{jk\ell}\}_{\ell \in \mathcal{L}_{jk}} \in \mathbf{R}^{|\mathcal{L}_{jk}|}$. Thus, each of $\{f_j\}_{j \in \mathcal{J}}$ is a Bayesian multiple-kernel model for which statistical properties were investigated in a previous study [4]. The prior density of the hyperparameters $\Pi_{jk}(\{\theta_{jk\ell}\}_{\ell \in \mathcal{L}_{jk}})$ is assumed to be given.

As we jointly sample functions $\{f_{jk}\}_{j,k}$ and hyperparameters $\{\theta_{jk\ell}\}_{j,k,\ell}$, the complicated dependence of the prior density of $\{f_{jk}(X_i)\}_i$ on $\{\theta_{jk\ell}\}_{j,k,\ell}$ hinders efficient sampling. To alleviate this problem, one previous study [16] introduced a useful approximation scheme that took the following form:

$$f_{jk}(X) \approx \sum_{m=1}^{M_{jk}} a_{jkm} \phi_{jkm}(X), \;\; a_{jkm} \sim \mathcal{N}(0, c_{jk} V_{jkm}(\sigma_{jk})), \tag{5}$$

where the generated function was represented as the sum of sinusoidal feature functions ϕ_{jkm} weighted by the normally distributed coefficients a_{jkm}. For the Gaussian kernels that we mainly use in this study, hyperparameters $\{\theta_{jk\ell}\}_{\ell \in \mathcal{L}_{jk}} = \{c_{jk}, \sigma_{jk}\}$ tune the amplitude and bandwidth, respectively, of the GP through the above equation. Concrete representations of ϕ_{jkm} and V_{jkm} are given in Appendix A. In this study, for simplicity, we restrict ourselves to the use of one-dimensional Gaussian kernels and linear kernels, as well as the use of a single set of hyperparameters $\{c_g, \sigma_g\}$ for all the GPs associated with Gaussian kernels and a single hyperparameter c_ℓ for all the GPs associated with linear kernels (equivalently Gaussian random variables). Note that V_{jkm} can be set to 1 for linear kernels. We assign inverse Gamma priors to the hyperparameters as follows:

$$\theta \sim \mathrm{InvGamma}(\alpha_\theta, \beta_\theta), \;\; (\theta = c_g, \; \sigma_g \text{ or } c_\ell). \tag{6}$$

 doi:10.20944/preprints202511.0763.v1

In the $L, M \to \infty$ limit (see Appendix A for the definition of L), the above approximation tends to be exact (Theorem 4 of Ref.[16]). The advantage of this approximation is the use of features (ϕ_{jkm}) that are independent of the bandwidth parameter σ_{jk}, and the use of a tractably computed prior density for $\{a_{jkm}\}_{j,k,m}$. However, the prefixed sinusoidal features are less flexible than are data-adaptive features (obtained, e.g., by the incomplete Cholesky decomposition [17]) and are not suitable for approximating high-dimensional functions. In biometric and econometric applications (e.g., epidaemiology), modelling a target variable as the sum of the effects of several factors is preferred for the sake of interpretability and robustness. Thus, modelling with the sum of several low-dimensional functions in Eq.(4) is justifiable.

The above modelling motivates our investigation of efficient MC algorithms. The variance $c_{jk}V_{jkm}$ varies with σ_{jk} over a few orders of magnitude. Combined with the structure of the data likelihood, the posterior distribution is often extremely stretched or compressed along unknown directions. Thus, a data-adaptive sampling scheme, such as RMHMC, is required.

2.2. RMHMC with a Soft-Absolute Hessian Metric

An HMC algorithm [18–20] obtains samples of parameters $q \in \mathbf{R}^d$ from a target density $P(q)$ by regarding q as the position of particles and simulating its time evolution together with that of the associated momentum vector $p \in \mathbf{R}^d$ according to the Hamilton equation of motion with a suitably designed Hamiltonian $\mathcal{H}(q,p)$. The time evolution over discretised timesteps is determined by

$$\begin{aligned} p\Big(t+\frac{\epsilon}{2}\Big) &= p(t) - \frac{\epsilon}{2}\frac{\partial}{\partial q}\mathcal{H}\Big(q(t), p\Big(t+\frac{\epsilon}{2}\Big)\Big), \\ q(t+\epsilon) &= q(t) + \frac{\epsilon}{2}\Big\{\frac{\partial}{\partial p}\mathcal{H}\Big(q(t), p\Big(t+\frac{\epsilon}{2}\Big)\Big) + \frac{\partial}{\partial p}\mathcal{H}\Big(q(t+\epsilon), p\Big(t+\frac{\epsilon}{2}\Big)\Big)\Big\}, \\ p(t+\epsilon) &= p\Big(t+\frac{\epsilon}{2}\Big) - \frac{\epsilon}{2}\frac{\partial}{\partial q}\mathcal{H}\Big(q(t+\epsilon), p\Big(t+\frac{\epsilon}{2}\Big)\Big). \end{aligned} \tag{7}$$

This time evolution for a time period of duration ϵ is called a leapfrog. The mapping from $(q(t), p(t))$ to $(q(t+\epsilon), p(t+\epsilon))$ preserves the volume and approximately preserves the value of $\mathcal{H}$. As we show in Algorithm 1, HMC repeats the combination of C leapfrogs and the sampling of a new value for p from a suitable q-dependent Gaussian distribution. At the end of every C leapfrogs, the change in $\mathcal{H}$ due to discretisation is adjusted via the Metropolis-Hastings procedure. Samples obtained from a long-time simulation after a burn-in period approximate the target distribution, and the statistics calculated with the samples converge to the average over the target at the infinite time limit. Girolami and Calderhead [21] introduced the use of a nontrivial Riemannian metric $q \mapsto G(q) \in \mathbf{R}^d \times \mathbf{R}^d$ for the parameter space and showed that, in this case, the Hamiltonian should be defined as

$$\mathcal{H}(q,p) \stackrel{\text{def}}{=} -\ln P(q) + \frac{1}{2}\ln\Big\{(2\pi)^d |G(q)|\Big\} + \frac{1}{2}p^T G(q)^{-1} p. \tag{8}$$

For this version of Hamiltonian, their derivatives with respect to q and p read

$$\begin{aligned} \frac{\partial}{\partial p_i}\mathcal{H} &= \Big(G(q)^{-1}p\Big)_i \\ \frac{\partial}{\partial q_i}\mathcal{H} &= -\frac{\partial}{\partial q_i}\ln P(q) + \frac{1}{2}p^T G(q)^{-1}\frac{\partial G(q)}{\partial q_i}G(q)^{-1}p - \frac{1}{2}tr\Big(G(q)^{-1}\frac{\partial G(q)}{\partial q_i}\Big). \end{aligned} \tag{9}$$

The choice of $G(q) = 1$ recovers the ordinary HMC based on the Euclidean metric.

The Hessian of $-\ln P(q)$ is a natural choice for $G(q)$, as we explained in the previous section. However, the Hessian of the negative log-posterior density for a hierarchical probabilistic model is

 doi:10.20944/preprints202511.0763.v1

Algorithm 1 (RM)HMC

Input: Target density $P(q)$, initial value $q(0) \in \mathbf{R}^d$, number of total and burn-in MC moves A and A_0, stepsize ϵ, number of leapfrogs C in a single move, metric $G : q \mapsto G(q) \in \mathbf{R}^d \times \mathbf{R}^d$ and Hamiltonian $\mathcal{H} : (q,p) \mapsto \mathcal{H}(q,p) \in \mathbf{R}$ defined in Eq.(8).
Output: A collection of parameter values $\{q(\epsilon\ell C)\}_{A_0 \le \ell \le A}$ useful for approximating $P(q)$.
Generate an initial value for the momentum vector: $p(0) \sim \mathcal{N}(0, G(q(0)))$.
for ℓ in $\{0 \dots A-1\}$ **do**
  for s in $\{0 \dots C-1\}$ **do**
    Carry out a leapfrog step according to Eq.(7)
  end for
  Generate $r_\ell \sim \text{Uniform}(0,1)$ for the following Metropolis-Hastings procedure:
  if $r_\ell < \exp(-\mathcal{H}(q(\epsilon(\ell+1)C), p(\epsilon(\ell+1)C)) + \mathcal{H}(q(\epsilon\ell C), p(\epsilon\ell C)))$ **then**
    $q(\epsilon(\ell+1)C) \leftarrow q(\epsilon\ell C)$
  end if
  Generate an updated value for the momentum vector: $p(\epsilon(\ell+1)C) \sim \mathcal{N}(0, G(q(\epsilon(\ell+1)C)))$.
end for

often not positive definite and therefore cannot be used as a metric. For this problem, Betancourt [13] proposed transforming the Hessian into a positive-definite matrix as

$$H(q) = \sum_{1 \le i \le d} \lambda_i \psi_i \psi_i^T \mapsto G(q) = \sum_{1 \le i \le d} g(\lambda_i) \psi_i \psi_i^T, \tag{10}$$

where the differentiable function g approximates the absolute value, i.e., $g(\lambda) \approx |\lambda|$. In the above, $\{\lambda_i\}_{1 \le i \le d}$ and $\{\psi_i\}_{1 \le i \le d}$ are the eigenvalues and eigenvectors of the Hessian H. Employing this transformed metric is equivalent to rescaling the motion of particles according to the absolute value of the curvature of the log-density's graph. Betancourt [13] further showed how to compute the right-hand sides of Eq.(9) for the soft-absolute Hessian as follows: for $\Psi = \text{mat}(\{(\psi_i)_j\}_{j,i})$,

$$\begin{aligned} \Psi^T \partial_{q_i} G(q) \Psi &= T \odot (\Psi^T \partial_{q_i} H(q) \Psi), \\ T_{j\ell} &= \begin{cases} \frac{g(\lambda_j) - g(\lambda_\ell)}{\lambda_j - \lambda_\ell} & (\lambda_j \neq \lambda_\ell) \\ g'(\lambda_j) & (\lambda_j = \lambda_\ell) \end{cases}, \quad (1 \le i,j,\ell, \le d), \end{aligned} \tag{11}$$

where $\odot$ denotes the Hadamard product. Given that eigenvectors and eigenvalues were already obtained, Betancourt [13] showed that the formula in Eq.(11) allows one to compute the right-hand side of the second equivalence in Eq.(9) for $1 \le i \le d$ with an $O(d^3)$ computational cost by first caching

$$W_1 = \Psi B T B \Psi^T, \quad W_2 = \Psi (R \odot T) \Psi^T \tag{12}$$

for $B = \text{diag}((\Psi^T p)_i / g(\lambda_i))_i$ and $R = \text{diag}(g(\lambda_i)^{-1})_i$, and then calculating

$$\begin{aligned} p^T G(q)^{-1} \frac{\partial G(q)}{\partial q_i} G(q)^{-1} p &= tr\left(W_1 \frac{\partial}{\partial q_i} H(q) \right), \\ tr\left(G(q)^{-1} \frac{\partial G(q)}{\partial q_i} \right) &= tr\left(W_2 \frac{\partial}{\partial q_i} H(q) \right). \end{aligned} \tag{13}$$

Here, W_1 and W_2 are obtained by $O(d^{2.38})$ matrix multiplications, the full gradient $\frac{\partial}{\partial q} H(q)$ is obtained by an $O(d^3)$ computation, and the $2d$ trace operations are carried out by taking the sum of the elementwise products $2d$ times; hence, the overall complexity of the computation is $O(d^3)$.

2.3. Structure-Dependent Removal of Redundancy

As we show below, a straightforward application of the above algorithm to the model described by Eqs.(1), (4), (5) and (6) is time-consuming. We show that the computational cost can be reduced

 doi:10.20944/preprints202511.0763.v1

to $O(d^{\omega(r)} + d^{1+r})$ for $d = \dim(\{\{a_{jkm}\}_{j,k,m}, \{b_j\}_j\}) + \dim(\{\theta_{jk}\}_{j,k})$, $N = \lceil d^r \rceil$ and fixed J. Here, the exponent $\omega(r)$ denotes the computational complexity of multiplying matrices of sizes $d^r \times d$ and $d \times d$ (equivalently matrices of sizes $d \times d^r$ and $d^r \times d$) (see Table 2 and Footnote 1 of Ref.[22]). This can be verified by substituting a concrete representation of the third derivatives of the negative log-posterior density for $\frac{\partial}{\partial q} H(q)$ in Eq.(13) as follows: for $s = 1, 2$, $a_j = \mathrm{vec}(\{a_{jkm}\}_{(k,m)})$, $f_{j,i} = f_j(X_i)$, $\phi_{j,i} = \mathrm{vec}(\{\phi_{jkm}(X_i)\}_{(k,m)})$, $\Phi_j = \mathrm{mat}(\{\phi_{jkm}(X_i)\}_{i,(k,m)})$ and $W_{sj_1j_2} = \mathrm{mat}(\{W_{s,j_1k_1m_1,j_2k_2m_2}\}_{(k_1,m_1),(k_2,m_2)})$,

$$
\begin{aligned}
& tr\left(W_s \frac{\partial}{\partial a_j} H(q)\right) \\
= & \sum_{i,j_1,j_2,k_1,k_2,m_1,m_2} \frac{\partial^3 U_i}{\partial f_{j_1,i} \partial f_{j_2,i} \partial f_{j,i}} \phi_{j_1k_1m_1,i} \phi_{j_2k_2m_2,i} \phi_{j,i} W_{s,(j_1k_1m_1),(j_2k_2m_2)} + \cdots \\
= & \sum_{j_1,j_2} \Phi_j \left[\mathrm{diag}\left(\frac{\partial^3 U_i}{\partial f_{j_1,i} \partial f_{j_2,i} \partial f_{j,i}} \right)_i \left\{ \left((\Phi_{j_1} W_{sj_1j_2}) \odot \Phi_{j_2} \right) \mathrm{vec}(\{1\}) \right\} \right] + \cdots,
\end{aligned} \tag{14}
$$

where we omit writing down the terms that involve the derivatives with respect to the bias term $\{b_j\}_j$ and the hyperparameters $\{\theta_{jk\ell}\}_{j,k,\ell}$ whose contribution can be computed in the same manner as those of the derivatives with respect to $\{a_{jkm}\}_{j,k,m}$. For each combination of (j_1, j_2), the suitably ordered computation in the second line requires, $O(d^{\omega(r)})$, $O(d^{1+r})$, $O(d^{1+r})$, $O(d^r)$ and $O(d^{1+r})$ computations for calculating $\Phi_{j_1} W_{s,j_1,j_2}$, calculating the Hadamard product, calculating the matrix-vector product with vector $\mathrm{vec}(\{1\})$ (a vector all of whose entries are 1), multiplying the diagonal matrix from the left and calculating the matrix-vector product with Φ_j, respectively.

The important point that should be noted here is that the standard library based on the automatic differentiation of the Hamiltonian does not follow the above order of computations. First, the direct derivatives of the Hamiltonian cannot be automatically computed when the soft-absolute metric is used. If the formula in Eq.(9) is naively implemented by computing the third-order derivatives of the log-likelihood with the aid of an automatic differentiator in the reverse mode (the default mode in PyTorch), at least an $O(Nd^3) = O(d^{3+r})$ computation is required. This differs greatly from the efficient computation described above. Even if the joint likelihood is log-concave and the formula in Eq.(9) can be avoided, a naive application of a reverse-mode automatic differentiator to the Hamiltonian results in the same computational cost. In order to obtain an efficient implementation using an automatic differentiator, one must first implement a differentiator that recognises the formula in Eq.(9) and then suitably assign the forward or reverse mode to the differentiation of each component of the Hamiltonian, according to the model structure.

2.4. Dynamically Programmed Eigendecomposition

The formula given by Betancourt [13] relies on the availability of the complete set of eigenvectors Ψ. The fact that $H(q(t))$ and $H(q(t+\epsilon))$ are close to each other motivates us to reduce computational costs by dynamically computing $\Psi(q(t+\epsilon))$ to take advantage of $\Psi(q(t))$. The boundedness of the gradient of $H(q)$ and p in the region where the joint probability for (q, p) is concentrated implies that

$$
\Psi(q(t))^T H(q(t+\epsilon)) \Psi(q(t)) = \mathrm{diag}(\lambda_i(q(t))) + O(\epsilon) \tag{15}
$$

holds with a high probability in terms of the Frobenius norm and can be efficiently eigendecomposed as

$$
\Psi(q(t))^T H(q(t+\epsilon)) \Psi(q(t)) = Q(t+\epsilon, t) \mathrm{diag}(\lambda_i(q(t+\epsilon))) Q(t+\epsilon, t)^T \tag{16}
$$

 doi:10.20944/preprints202511.0763.v1

using the Jacobi method. Note that cyclic versions of the Jacobi method quadratically converge [23] and are suitable for parallelisation [24], and thus are expected to carry out the above decomposition very quickly with a small error tolerance ζ. With this decomposition, we update the eigenvectors as

$$\Psi(q(t+\epsilon)) = \Psi(q(t))Q(t+\epsilon, t). \tag{17}$$

In practice, we perform the Gram-Schmidt orthogonalisation of $\Psi(q(t))$ every ten steps before applying it to $H(q(t+\epsilon))$ to remove accumulated numerical errors.

2.5. Numerical Experiments on Computational Complexity

We investigate the efficiency of the proposed algorithm by performing Bayesian logistic regression with artificially generated data and measuring its computation time. For comparison, we also perform posterior sampling using Hamiltorch [14], a publicly available library for different types of HMCs based on PyTorch that works on CUDA devices. First, we generate standardised explanatory variables $X_i^{(\mathrm{exp})} \in \mathbf{R}^D \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$ for each sample i ($1 \le i \le 500$). Then, we calculate the true log-odds ratio $f_1^*(X_i^{(\mathrm{exp})})$ according to

$$f_1^*(X_i^{(\mathrm{exp})}) = \sum_{1\le k\le D} \sum_{m=1}^{M_{1k}} a_{1km}^* \phi_{1km}(X_{ik}^{(\mathrm{exp})}) + b_1^*, \tag{18}$$

with $b_1^* = -0.5$, $a_{1km}^* \overset{i.i.d.}{\sim} \mathcal{N}(0, c^* m^{-1})$ for $4 \le m \le 16$ and $a_{1km}^* = 0$ otherwise. For any k, the feature function ϕ_{1mk} is given by Eq.(A2) in Appendix A with $d = 1$ and $L_1 = 8$. Here, the value of constant c^* is determined so that the empirical standard deviation $\widehat{\mathrm{SD}}[f_1(X^{(\mathrm{exp})})]$ is 1.5.

We then generate the sample label according to

$$X_i^{(\mathrm{lab})} = \begin{cases} +1 & \text{prob. } \frac{1}{1+e^{-f_1^*(X_i^{(\mathrm{exp})})}} \\ -1 & \text{otherwise} \end{cases}. \tag{19}$$

For this dataset, we perform posterior sampling, where the model is determined by Eq.(1) with $J = 1$ and

$$U(f_1(\{X_i^{(\mathrm{exp})}, X_i^{(\mathrm{lab})}\})) = \ln(1 + \exp(-X_i^{(\mathrm{lab})} f_1(X_i^{(\mathrm{exp})}))), \tag{20}$$

where function f_1 is further decomposed into functions associated with one-dimensional Gaussian kernels:

$$f_1(X_i^{(\mathrm{exp})}) = \sum_{1\le k\le D} \sum_{1\le m\le M_{1k}} a_{1km} \phi_{1km}(X_{ik}^{(\mathrm{exp})}) + b_1. \tag{21}$$

Here, we abuse a notation for brevity by identifying index k with a Gaussian kernel function $k(X_i, X_i') = \exp(-(X_{ik} - X_{ik}')^2/\sigma_{1k})$, for which, again, the feature function ϕ_{1mk} is given by Eq.(A2) in Appendix A with $d = 1$, $L_1 = 8$ and $M_{1k} = 30$. The associated GPs and their hyperparameters are described by Eqs.(4), (5) and (6). For the transformation into the soft-absolute Hessian in the proposed method, we used $g(\lambda) = \sqrt{\kappa^2 + \lambda^2}$, which takes values close to $g(\lambda) = \lambda \coth(\kappa^{-1}\lambda)$ employed in Hamiltorch. We did not implement the latter because of the singularity of coth at 0 and the lack of a straightforward implementation in the library we used (NVIDIA Inc., NVHPC23.1 [25]). In addition to the values specified above, we used the following values: (for the comparison of sampling speed) $\alpha_\theta = \beta_\theta = 2.0$ for any hyperparameter θ, $\Sigma = 1$, $A = 9600$, $A_0 = 2400$, $C = 100$, $\epsilon = 0.001$, $\kappa = 1$, $\zeta = 1.0 \times 10^{-13}$; (for the calculation of BME (see section B)) $\alpha_{c_g} = \alpha_{c_\ell} = 5$, $\beta_{c_g} = \beta_{c_\ell} = 0.5$, $\alpha_{\sigma_g} = \beta_{\sigma_g} = 1$, $\Sigma = 1$, $A = 50$, $C = 100$, $\epsilon = 0.001$, $\kappa = 1$, $\zeta = 1.0 \times 10^{-13}$, $Z = 50$, $\tau_s = 1 - 0.02(s-1)$ ($1 \le s \le 41$), $\tau_s = 0.2 - 0.005(s-41)$ ($41 < s \le 71$), $\tau_s = 0.05 - 0.002(s-71)$ ($71 < s \le 91$) and $\tau_s = 0.01 - 0.001(s-91)$

 doi:10.20944/preprints202511.0763.v1

($91 < s \leq 101$). The initial parameter values for MC we used were $a_{jkm} = 0$, $b_j = 0$ and $\theta = 1$ for any hyperparameter θ. We confirm the convergence of the proposed algorithm by performing Wilcoxon's rank-sum test for the values of the log-posterior density in the first and second halves of the trajectory after an initial burn-in period. The proposed method was implemented by writing codes in C++ and OpenACC that offload most of the calculation to a CUDA device retaining all necessary data in its own memory and invoke cuBLAS and cuSolver for linear-algebra routines as much as possible. For comparison, we wrote a code in PyTorch for the same model that invokes Hamiltorch for carrying out RMHMC or NUT-HMC with a manually coded likelihood. To conduct a comparison of different implementations, we measure the wall time spent on MC moves run on a single CPU core (a recent version of Intel Xeon processor) connected to a single NVIDIA Tesla A100 PCIe 80GB GPU card, for each of five datasets generated with different seeds for the random number generator.

2.6. Analysis of 1987 National Medical Expenditure Survey (NMES)

To demonstrate that our implementation is acceptably efficient in a real-world application, we perform Bayesian estimation with excerpted data from the 1987 national medical expenditure survey (NMES) of the United States. A few groups of authors performed causal inference with this dataset about the effects of smoking on medical expenditures [26–28]. In the present analysis, we attempt to increase the precision of the propensity function used for causal inference. As shown by Imai and van Dyk [27], the identification of a set of parameters f that characterise the conditional distribution $p_f(X^{(\mathrm{tgt})}|X^{(\mathrm{cov})})$ of the actual treatment variable $X^{(\mathrm{tgt})}$ for the given values of covariates $X^{(\mathrm{cov})}$ reduces the dimensionality of subsequent causal analysis. In this dataset, $X^{(\mathrm{tgt})}$ denotes the packyear of smoking (i.e., the product of the number of packs of cigarette consumed by the subject and the duration of smoking measured in years). The covariates $X^{(\mathrm{cov})}$ include the age at the time of the survey, the age at the initiation of smoking, gender, race, marital status, education level, census region, poverty status and seat belt usage. The previous study focused mainly on the estimation of the mean of the conditional distribution as f and not its variance. We compare the performance of linear and nonlinear models that describe only the conditional mean, and the performance of linear and nonlinear models that describe both the conditional mean and variance (Eq.(2)). In particular, as the model that best describes the given data is determined by the value of the Bayesian model evidence (BME) [9], we investigate whether the proposed method computes this value within a reasonable computation time (see section B for the technical details of the calculation of BME). Carrying out causal inference with the estimated propensity functions requires further theoretical development and is not within the scope of the present study. Thus, we restrict our analysis to the estimation of propensity functions and discuss the further development that is needed in section 4.

We obtained the dataset included in a library for causal inference [28]. The categorical covariates in this dataset were transformed to a set of binary covariates that retained the original information. The continuous covariates were standardised to have a zero mean and unit variance.

Next, we consider the data likelihood described by Eq.(2). The function f_j ($j = 1, 2$) is described as

$$f_j(X_i^{(\mathrm{cov})}) = \sum_{1 \leq k \leq D} \sum_{1 \leq m \leq M_{jk}} a_{jkm} \phi_{jkm}(X_{ik}^{(\mathrm{cov})}) + b_j. \tag{22}$$

For the linear models, we use a linear kernel indexed by k for each variable $X_{ik}^{(\mathrm{cov})}$, and thus we have $\phi_{jk1}(X_i^{(\mathrm{cov})}) = X_{ik}^{(\mathrm{cov})}$ with $M_{jk} = 1$. For the nonlinear models, we use a one-dimensional Gaussian kernel for each continuous variable with $L_1 = 8$ and $M_{jk} = 30$, and a linear kernel for each binary variable. Models that describe the conditional mean and variance are determined in this manner. Models that describe only the conditional mean determine f_1 in the manner described above, whereas their variance is described by $f_2 = b_2$. The priors for GPs and hyperparameters are described by Eqs.(4), (5) and (6) in the same manner as for the simulated data. In addition to the values specified above, we used the following values: $\alpha_{c_g} = \alpha_{c_\ell} = 5$, $\beta_{c_g} = \beta_{c_\ell} = 0.5$, $\alpha_{\sigma_g} = \beta_{\sigma_g} = 1$, $\Sigma = 1$, $A = 50$, $C = 400$, $\epsilon = 0.0001$ ($\epsilon = 0.00008$ for the nonlinear model describing both of the conditional mean and variance),

 doi:10.20944/preprints202511.0763.v1

$\delta = 0.2$, $\kappa = 1$, $\zeta = 1.0 \times 10^{-13}$, $Z = 10$, $\tau_s = 1 - 0.02(s-1)$ $(1 \leq s \leq 41)$, $\tau_s = 0.2 - 0.005(s-41)$ $(41 < s \leq 71)$, $\tau_s = 0.05 - 0.002(s-71)$ $(71 < s \leq 91)$ and $\tau_s = 0.01 - 0.001(s-91)$ $(91 < s \leq 101)$. The initial parameter values for MC we used were $a_{jkm} = 0$, $b_j = 0$ and $\theta = 1$ for any hyperparameter θ.

3. Results

3.1. Comparison Among Different Implementations of RMHMC with a Soft-Absolute Hessian Metric

The proposed implementation is fast enough to converge to equilibrium in both small and moderately large models, as seen from the trajectories in Fig.1(A) and confirmed by a Wilcoxon rank-sum test for $\sim$ 14,000- and $\sim$ 60,000-second simulations ($p = 0.31, 0.47$), respectively. As shown in Fig.1(A) and (B), our implementation is roughly 10 times faster than the RMHMC based on Hamiltorch. To investigate the relative impact of the order of computations compared with that of the dynamically programmed eigendecomposition, we also measured the wall time for implementations in which the eigendecomposition was replaced by static ones based on either the Jacobi method or the divide-and-conquer algorithm. The inset in Fig.1(B) suggests that the difference in computation time between our implementation and the implementation based on Hamiltorch is mainly attributed to the computation of the gradient flow rather than eigendecomposition. However, a close examination of Fig.1(B) also shows differences among the algorithms for eigendecomposition. Although little difference is observed in computation time among the examined algorithms for small D, the dynamic eigendecomposition saves substantial computation time for large D. Examining the number of sweeps in the cyclic Jacobi method in static and dynamic implementations (Fig.1(C)), we confirm the advantage of the dynamic implementation, which requires only one to two sweeps at each step regardless of the model dimensionality, whereas the static implementation requires an increasing number of sweeps for greater model dimensionality on average.

 doi:10.20944/preprints202511.0763.v1

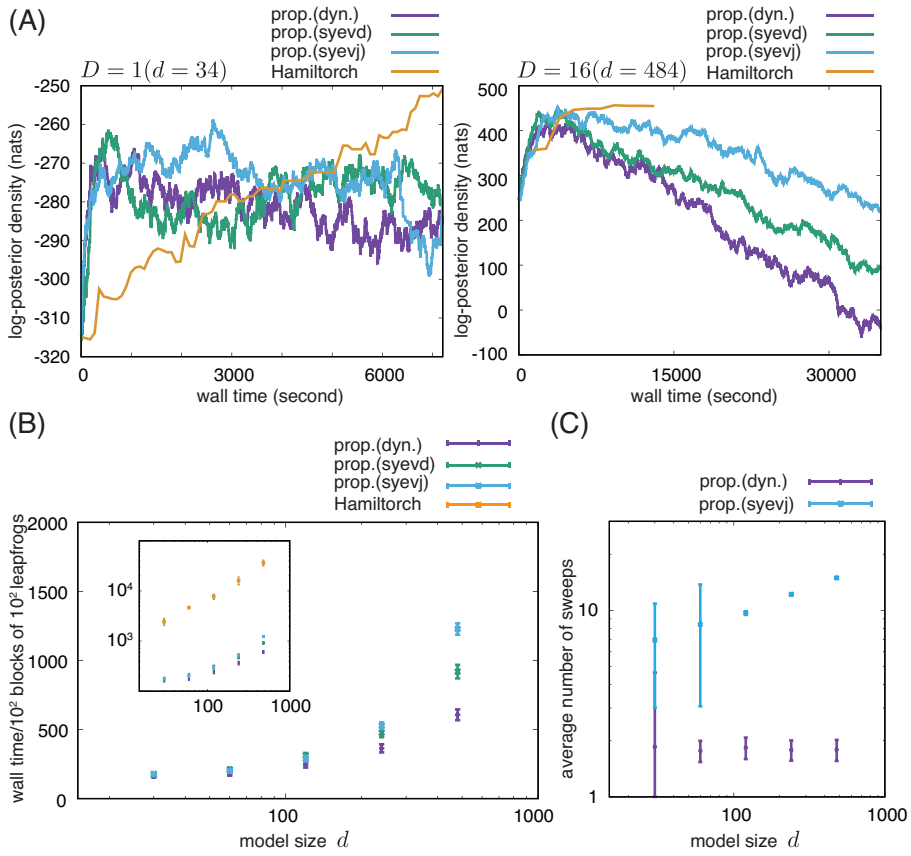

Figure 1. RMHMC for Bayesian logistic regression on simulated data. Comparison between our implementations based on the dynamic and static eigendecomposition of the Hessian and the implementation based on Hamiltorch. (A) Representative trajectories of log-posterior density for $D = 1$ ($d = 34$) and $D = 16$ ($d = 484$). (B) Mean and standard deviation of the wall time spent on computing 100 blocks of 100 leapfrogs are shown on the linear (main panel) and logarithmic (inset) scales for different model sizes. (C) Mean and standard deviation of the number of sweeps in the cyclic Jacobi method for a single dynamic and static eigendecomposition of the metric are shown on the logarithmic scale for different model sizes. In (B) and (C), model sizes are shown on the logarithmic scales. (Abbreviations) prop.(dyn.): the proposed method based on dynamic eigendecomposition, prop.(syevd): the proposed method based on static eigendecomposition with the divide-and-conquer algorithm, prop.(syevj): the proposed method based on static eigendecomposition with the Jacobi method.

3.2. Comparison with NUT-HMC Sampler

We also confirm the advantage of using RMHMC by comparing its performance with that of NUT-HMC. As shown in Fig.2A, NUT-HMC reaches samples of parameter values that have large values of log-posterior density very quickly and continues generating similar samples (Fig.2(A)). This contrasts with the slow convergence of RMHMC to generate samples with lower log-posterior density. We investigate which samplers generate representative samples from the posterior as follows. First, we examine the eigenvalues of the Hessian of the log-posterior density with representative samples from the two samplers (Fig.2(B)). We see that the eigenvalues for the sample from NUT-HMC are 10 times larger than those for the sample from RMHMC, except for several small positive or negative eigenvalues. This suggests that the samples from NUT-HMC have been taken from a much narrower region having a larger density. We therefore doubt that the total posterior probability for this narrow region is smaller than that for the region explored by RMHMC. Indeed, as we run RMHMC using for the initial parameter value a sample obtained using NUT-HMC, we observe that RMHMC swiftly

 doi:10.20944/preprints202511.0763.v1

moves to a region with the same lower range of log-density as that of the samples obtained by using the default initial condition $f_{jk} = 0$ and $\theta_{jk\ell} = 1$ (Fig.2(A)).

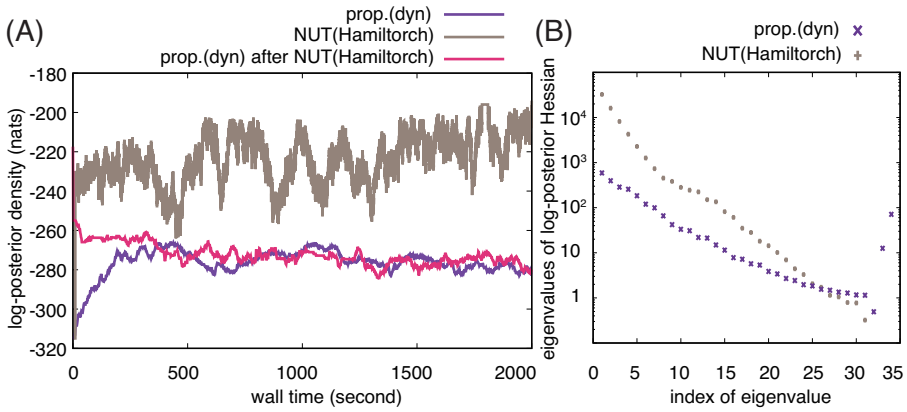

Figure 2. Comparison of proposed implementation of RMHMC and NUT-HMC in Hamiltorch. (A) Representative trajectories of the log-posterior density for $D = 1$ ($d = 34$) obtained using the proposed implementation of RMHMC based on dynamic eigendecomposition (prop.(dyn)) and NUT-HMC in Hamiltorch from the same initial condition $f = 0$ and $c_g, \sigma_g, c_\ell = 1$. A representative trajectory for the proposed implementation of RMHMC starting with the parameter values obtained after 2000 seconds of sampling with NUT-HMC is also shown. (B) Eigenvalues of the Hessians of the negative log-posterior density for the samples of parameter values obtained after running the proposed implementation of RMHMC and NUT-HMC in Hamiltorch for 2,000 seconds. The eigenvalues are plotted in descending order on the logarithmic scale. For negative eigenvalues, their absolute values are plotted. The absolute values of the 32nd to 34th eigenvalues for the sample from NUT-HMC were extremely small and are not shown in the plot.

3.3. Calculation of BME beyond the Laplace approximation with simulated data and NMES data

The existence of negative eigenvalues in Fig.2(B) suggests that the posterior cannot be regarded as being approximately normal. This is not surprising, since hierarchical models are known to be often singular [29]. Thus, the BME of the proposed model cannot be calculated by simply applying the Laplace approximation to the posterior density. Even in this case, we successfully carried out the calculation of BME through MC integration with reasonably high precision, as confirmed by the standard error of the estimate determined with multiple sequences of RMHMC [Table 1]. The obtained value was in good agreement with the BME value obtained by numerically integrating the Laplace approximation of the likelihood conditioned on each combination of hyperparameter values (see Appendix B for technical details). The latter calculation based on the Laplace approximation is justified by the fact that the model has log-concave posterior density and is regular when conditioned on hyperparameters.

Table 1. Calculation of BME using the proposed implementation of RMHMC. The mean and standard error of the estimated model evidence and the mean and standard deviation of the wall time for its computation in a single RMHMC (among Z RMHMCs) are shown. For simulated data, we also show the BME values estimated by iterating and integrating the Laplace approximation for different values of the hyperparameters. For NMES data, linear (L) and nonlinear (NL) models that describe only the conditional mean (mean) or both of the conditional mean and variance (mean/var) were used.

Data	Model	Estimated BME	Laplace Approx.	Wall time (sec)	# MC (Z)
Simulation	NL-logistic	-279.81 ± 0.40	-282.31	16243 ± 1102	50
NMES	L-mean	-11716.78 ± 2.11	—	43314 ± 5120	10
NMES	NL-mean	-11602.37 ± 2.65	—	49648 ± 3613	10
NMES	L-mean/var	-9540.63 ± 1.87	—	49424 ± 4509	10
NMES	NL-mean/var	-9163.53 ± 3.43	—	52284 ± 3501	10

 doi:10.20944/preprints202511.0763.v1

To demonstrate the usefulness of the proposed method for analysing real-world data, we also calculate the BME for the NMES dataset using linear or nonlinear models that estimate the mean (and variance) of the distribution of the treatment variable conditioned on the covariates, namely, a propensity function. We successfully calculated BME values for this estimation problem within an acceptable computation time (Table 1). The computed BME values showed improved estimation of the propensity function with the nonlinear model for both the mean and variance of the conditional treatment density.

4. Discussion

In the present study, we have investigated MC methods for the posterior sampling of functions and hyperparameters in hierarchical GP models, and we have shown that a straightforward implementation of RMHMC based on currently available general-purpose libraries is highly redundant and its performance can be greatly improved. The main source of redundancy was the order of computation in the calculation of the gradient flow on the Riemannian manifold, whereas the eigendecomposition of the metrics was also a major source of redundancy for the larger models. These findings have non-trivial implications for future modelling based on GP and its implementations, because our results indicate that the current standard practice of coding only the likelihood of the model and allowing libraries to carry out inference with the aid of automatic differentiation results in poor performance, unless an intelligent library that optimises overall computational complexity is developed. This problem was not recognised in the previous study of RMHMC for GP [10], because they sampled in the space for dual variables ($\{f_{j,i}\}_{1\leq j\leq J, 1\leq i\leq N}$ with $J = 1$ in our notation) for which the entries of the third-order derivative tensor were sparse. In a hierarchical model with multiple GPs, the use of dual variables incurs a large computational cost. In this case, sampling in the space for multiple sets of dual variables $\{f_{jki}\}_{1\leq j\leq J, k\in\mathcal{K}_j, 1\leq i\leq N}$ ($J > 1$) is very high-dimensional. In this case, the eigendecomposition of the metric required for the Betancourt's formula (Eq.(9)) is essentially impossible to compute. Furthermore, the derivatives of posterior density with respect to both of the dual variables and the hyperparameters (c_{jk} and σ_{jk} in our notations) are also difficult to compute. The previous study avoided these difficulties by using only a single GP and fixing the hyperparameter values. In contrast we use a representation [16] with a reduced number of primal variables, which makes the dependence on hyperparameters easier to compute.

In our numerical study, the dependence of computation time on model size does not precisely agree with theoretical expectations (section 2.3), presumably because the GPU accelerator carries out many arithmetic operations in parallel, and the effect of the size of multiplied matrices in the examined range is masked by this parallelism. Because this effect of parallelism is also observed in the multiplication of much larger matrices [30], we do not attempt to extend the range of dimension of our experiment. The computation time saved by dynamically programmed eigendecomposition was not striking in our simulation study; however, this effect could be much greater for larger model sizes, because eigendecomposition requires $O(d^3)$ computation, which eventually outweighs the computation of the gradient flow at sufficiently large d. We could not directly observe this phenomenon because tuning the step size and the threshold parameter for the soft-absolute Hessian metric becomes difficult for much larger d. We also note that the scaling of wall time for eigendecomposition depends heavily on the available GPU card. Several years ago, it was shown that the Jacobi method is outperformed by the divide-and-conquer algorithm for matrices larger than 512×512 on a NVIDIA Tesla K40 GPU card [31]; however, a recent study using dynamically programmed eigendecomposition for matrix optimisation showed that one sweep of the cyclic Jacobi method for matrices of size $4,096 \times 4,096$ takes much less time than the divide-and-conquer algorithm for matrices of the same size [32]. Because the dynamic programming successfully reduced the number of sweeps to fewer than two, regardless of model size, in our study, we expect that users can also benefit from the dynamically programmed eigendecomposition in larger models if a recent version of a GPU card is used.

 doi:10.20944/preprints202511.0763.v1

In the numerical study, we showed that RMHMC outperforms NUT-HMC. The latter apparently reaches a parameter region with higher log-posterior density more quickly, but this region has turned out to be a narrow spurious region from which RMHMC swiftly moved away. Such entrapment is a well-known phenomenon and was the main motivation for the development of RMHMC [21]. However, various elaborations have been introduced to Euclidean HMC and related Langevin algorithms but not yet to RMHMC. For example, avoiding random-walk behavior with a no-U-turn mechanism was proposed for Euclidean HMC [33] and RMHMC [34], but it has not yet been properly implemented for the latter. The friction mechanism introduced to Euclidean Langevin MC [35,36] suppresses inefficient oscillatory behaviour and could also be beneficial if applied to RMHMC. One factor that might have hindered elaboration of RMHMC in this regard is the previously poor performance of the plain RMHMC. Now that we have demonstrated improved RMHMC performance, further development in this direction should be encouraged.

Although we have successfully shown that RMHMC can be greatly accelerated, it is fair to note that we restricted our investigation to simple model settings. Bayesian multiple-kernel models have been shown to have favourable statistical properties when irrelevant GP is plugged out with sparsifying mechanisms such as a prior that imposes a penalty according to the number of included GPs [4]. For this purpose, reversible-jump mechanisms [37,38] must be introduced to RMHMC. Roughly speaking, this amounts to performing a shorter version of the MC integration in the calculation of BME multiple times in the simulation. For real-world applications, more elaborate structured models such as those for time-series data must sometimes be used. In this case, RMHMC for GP must be combined with other MC methods, such as sequential MC [39,40]. Whether the proposed method works within a reasonable computation time when it is tailored to the models described above needs to be investigated. Although the present work was intended to solve a biometrical and econometrical problem, unbiased effect estimation with Bayesian models requires further theoretical development, not just an accelerated implementation. The posterior we inferred with NMES data was asymptotically biased, as are essentially all machine-learning estimators. Therefore, in order to complete the causal inference with the biased propensity function, one needs to construct a corrected estimator for the treatment effect. This may be carried out by using the framework of doubly-robust debiased machine learning (DML) [41] (see our previous work [42] for the application of DML to a model based on multiple reproducing kernel Hilbert spaces). However, its application is not straightforward, because the hierarchical Bayesian model is apparently singular (Fig.2(D)), whereas DML relies on the asymptotic normality of the estimators. The model is expected to be regular for fixed hyperparameter values and the framework of DML may be applied to a regular submodel conditioned on suitable hyperparameter values. As the focus of the present study is on the efficiency of RMHMC, we leave this development to a future work.

Author Contributions: Conceptualization, T.H.; methodology, T.H.; software, T.H.; validation, T.H. and S.A.; investigation, T.H. and S.A. ; writing—original draft preparation, T.H.; writing—review and editing, T.H. and S.A. ; project administration, T.H.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Education, Culture, Sports, Sciences and Technology (MEXT) of the Japanese government and the Japan Agency for Medical Research and Development (AMED) under grant numbers JP18km0605001 and JP223fa627011.

Data Availability Statement: All of the source codes that support the findings of this study are available as supplementary materials

Acknowledgments: We would like to express our gratitude to two medical IT companies, 4DIN Ltd. (Tokyo, Japan) and Phenogen Medical Corporation (Tokyo, Japan) for financial support. Neither company had a role in the research design, analysis, data collection, interpretation of data, or review of the manuscript, and no honoraria or payments were made for authorship.

Conflicts of Interest: The authors declare no conflicts of interest.

 doi:10.20944/preprints202511.0763.v1

Appendix A. Reduced-Rank Representation of GPs (Svensson et al. and Solin and Särkkä

For a d-dimensional translation-invariant isotropic kernel function $k(x, x')$ describing the covariance of a GP, Solin and Särkkaä introduced the following approximation in a rectangular domain $[-L_1, L_1] \times \cdots \times [-L_d, L_d]$:

$$k(x, x') \approx \sum_{m_1, \cdots, m_d = 1}^{\hat{M}^d} S(\sqrt{\lambda_{m_1 \cdots m_d}}) \phi_{m_1 \cdots m_d}(x) \phi_{m_1 \cdots m_d}(x') \tag{A1}$$

with the eigenfunctions and eigenvalues

$$\phi_{m_1 \cdots m_d}(x) = \prod_{\ell=1}^{d} \frac{1}{L_\ell} \sin\left(\frac{\pi m_\ell (x_\ell + L_\ell)}{2L_\ell} \right), \tag{A2}$$

and

$$\lambda_{m_1, \cdots, m_d} = \sum_{\ell=1}^{d} \left(\frac{\pi m_\ell}{2L_\ell} \right)^2. \tag{A3}$$

In the above, $S(\cdot)$ is the spectral density of the GP related to the kernel function $k(r) = k(x, x + r)$ via the Wiener-Khinchin theorem:

$$\begin{aligned} k(r) &= \frac{1}{(2\pi)^d} \int_{\mathbf{R}^d} S(\omega) e^{\mathrm{i}\omega^T r} d\omega, \\ S(\omega) &= \int_{\mathbf{R}^d} k(r) e^{-i\omega^T r} dr. \end{aligned} \tag{A4}$$

In the present study, we focus on the one-dimensional ($d = 1$) case. Then, the GP is represented in the form of Eq.(5). We also refer readers to Svensson et al. [6] for more details about its implementation. Unlike Svensson et al. [6], we used scalar-valued GPs, and thus, the matrix normal distributions and inverse-Wishart distributions they used simply reduces to normal distributions and inverse-Gamma distributions in our case. By applying the Fourier transform to $k(r) = \exp(-r^2/\sigma)$, we obtain $S(\omega) = \sqrt{\pi\sigma} \exp(-\sigma\omega^2/4)$. Considering also Eq.(A1), we have

$$V_{jkm} = \sqrt{\pi\sigma} \exp\left(-\frac{\pi m^2 \sigma}{16 L^2} \right), \tag{A5}$$

and

$$\phi_{jkm}(X_i) = \frac{1}{L} \sin\left(\frac{\pi m (X_{ik} + L)}{2L} \right). \tag{A6}$$

In practice, we dropped $\frac{1}{L}$ from ϕ_{jkm}, which amounts to rescaling of c_g.

 doi:10.20944/preprints202511.0763.v1

Table A1. Mathmatical notations.

Symbols	Description
$\mathbf{R}, \mathbf{R}^d$	The sets of real numbers and d-dimensional Euclidean space
$\lceil a \rceil$	The smallest integer that is greater than or equals a
v^T, A^T	Transposition of vector v and matrix A
$(v)_i$	The i-th element of vector v
$(A)_{ij}$	The element of matrix A in the i-th row of the j-th column
$\mathrm{vec}(\{v_i\}_i)$ [1]	Vector whose i-th element is v_i
$\mathrm{mat}(\{A_{ji}\}_{i,j})$ [2]	Matrix whose i-th row of the j-th column is A_{ji}
$\mathrm{diag}(a_i)_i$	Diagonal matrix whose i-th element is a_i
$\mathrm{tr}A$	the trace of matrix A
$a \in \mathcal{A}$	Element a of a set $\mathcal{A}$
$A \odot B$	The Hadamard product of A and B
$\lvert\mathcal{A}\rvert$	The number of elements in a set $\mathcal{A}$
$E[\cdot]$ ($E_{X\sim p}[\cdot]$)	Expectation of the argument random variable (for the specified distribution)
$\mathrm{Var}[\cdot]$	Variance of the argument random variable
$\widehat{\mathrm{SD}}[\cdot]$	Empirical standard deviation of the argument variable
$\stackrel{\mathrm{def}}{=}$	Equation defining the object on the left-hand side
$i.i.d.$, ($\overset{i.i.d.}{\sim}$)	Independently and identically distributed (objects drawn from the right-hand side)
$\mathrm{Uniform}(a,b)$	Uniform probability distribution over the open interval (a,b)
$\mathrm{Normal}(\mu,\Sigma)$	Gaussian probability distribution with mean μ and (co)variance Σ
$\mathrm{InvGamma}(\alpha,\beta)$	Inverse Gamma probability distribution with shape and scale parameters α, β
$\mathrm{Bernoulli}(p)$	Bernoulli probability distribution (value 1 with probability p, or 0 otherwise)

Appendix B. MC Integration for the Calculation of BME (Calderhead and Girolami)

BME for the model with parameters $a = \{\{a_{jkm}\}_{j,k,m}, \{b_j\}_j\}$ for approximately describing $\{f_j\}_j$ and hyperparameter $\theta = \{c_g, \sigma_g, c_\ell\}$ (with prior densities $\mathcal{G}_\theta(a)$ and $\Pi(\theta)$) is defined as the following marginal likelihood [9]:

$$\mathrm{BME} \stackrel{\mathrm{def}}{=} \int P(X|a)\mathcal{G}_\theta(a)\Pi(\theta)dad\theta. \tag{A7}$$

In a hierarchical model, carrying out the above integration is usually intractable. If the joint density is approximately Gaussian, the following Laplace approximation can be used: for $v = \mathrm{vec}(a,\theta)$,

$$\begin{aligned} \mathrm{BME} &\approx \int \widehat{P}\exp(-(v-\widehat{v})^T\widehat{H}(v-\widehat{v}))dv \\ &= \widehat{P}(2\pi)^{d/2}|\widehat{H}|^{-1/2} \end{aligned} \tag{A8}$$

where $\widehat{v}$ denotes the maximum-a-posteriori value of v, and $\widehat{P}$ and $\widehat{H}$ denotes the peak value and the negative Hessian of the logarithm of the joint density at $\widehat{v}$.

The above formula cannot be used when the deviation of the joint density from the Gaussian approximation is large. This is often the case when we use a singular (or nearly singular) hierarchical model [29]. If the model is parametric, asymptotic formula for the BME of singular models can be used [43,44]. However, these formula do not apply to the semiparametric models that we consider in this study.

[1] Sometimes a combinatorial index such as $\mathrm{vec}(\{v_{km}\}_{(k,m)})$ is used. In this example, the index runs through all possible values for the combination (k,m).

[2] The first and second subscripts of the bracket $\{\cdot\}$ specify the indices for the row and column, respectively.

 doi:10.20944/preprints202511.0763.v1

Even in this case, the BME can be calculated by performing the following integration [45]:

$$\mathrm{BME} = \int_0^1 \mathrm{E}_{v\sim P_\tau}[\ln P(X|a)]d\tau, \tag{A9}$$

with a set of probability densities parameterised by τ ($0 \le \tau \le 1$):

$$P_\tau(v) \propto P(X|a)^\tau \mathcal{G}_\theta(a)\Pi(\theta). \tag{A10}$$

This integration can be carried out by sampling from P_τ for each value of τ in discretised steps interpolating 0 and 1 and approximating the integrand with the statistics over the samples.

We carry out the above integration with multiple RMHMC indexed by $z = 1, 2, \ldots, Z$ each of which determines the parameter value $(a_s^{(z)}, \theta_s^{(z)})$ used for the calculation of the integrand $\mathrm{E}_{v\sim P_\tau}[\ln P(X|a)] \approx \frac{1}{Z}\sum_z \ln P(X|a_s^{(z)})$ in Eq.(A9) for $\tau = \tau_s$ ($s = 1, 2, \ldots, S$; $\tau_1 = 1$ and $\tau_S = 0$) after performing A blocks of C leapfrogs from the initial parameter value (a_{s-1}, θ_{s-1}). To determine (a_1, θ_1) for $\tau_1 = 1$, we performed a single RMHMC until convergence and obtained a shared initial condition for Z RMHMCs that perform further 500 blocks of 400 leapfrogs to obtain $(a_1^{(z)}, \theta_1^{(z)})$ from this initial condition.

For the above calculation, it should noted that, if $\tau_s - \tau_{s-1}$ is small enough for all s and the number of MC moves A for each τ_s is large enough,

$$\mathrm{BME} \approx \sum_{s=2}^{S} \frac{1}{2}\left(\ln P(X|a_s^{(z)}) + \ln P(X|a_{s-1}^{(z)})\right)(\tau_s - \tau_{s-1}) \tag{A11}$$

holds for each value of z. In this case, since A is large enough, $\{a_s\}_{1\le s\le S}$, can be considered independent. Assuming this independence, we have

$$\begin{aligned}\mathrm{Var}[\mathrm{BME}] \approx & \sum_{s=2}^{S-1} \frac{1}{4}\mathrm{Var}[\ln P(X|a_s^{(z)})]\{(\tau_s - \tau_{s-1})^2 + (\tau_{s+1} - \tau_s)^2\} \\ & + \frac{1}{4}\left\{\mathrm{Var}[\ln P(X|a_1^{(z)})](\tau_2 - \tau_1)^2 + \mathrm{Var}[\ln P(X|a_S^{(z)})](\tau_S - \tau_{S-1})^2\right\},\end{aligned} \tag{A12}$$

and its right-hand side vanishes as the discretisation of the integration interval becomes infinitely finer.

The use of multiple RMHMCs (that is, $Z > 1$) is expected to accelerate the convergence. In practice, we used $A = 50$ and $Z = 10$ and confirmed that the change in the values of A and Z ($A = 12, 25, 50, 100$ and $Z = 5, 10, 20$) does not affect the decision about the best model.

Since $P(X|a)\mathcal{G}_\theta(a)$ is log-concave for fixed θ, in the numerical experiment with simulated data, we also perform the following integration:

$$\begin{aligned}\mathrm{BME} &\approx \int \widehat{P}_\theta \exp(-(a - \widehat{a}(\theta))^T \widehat{H}_{aa}(\theta)(a - \widehat{a}(\theta)))da\Pi(\theta)d\theta \\ &= \int \widehat{P}_\theta (2\pi)^{d/2} |\widehat{H}_{aa}(\theta)|^{-1/2}\Pi(\theta)d\theta,\end{aligned} \tag{A13}$$

where $\widehat{a}(\theta)$ denotes the value of a that maximises $P(X|a)\mathcal{G}_\theta(a)$ for the given value of θ, and $\widehat{P}_\theta$ and $\widehat{H}_{aa}(\theta)$ denote the peak value and the negative Hessian of $\ln P(X|a)\mathcal{G}_\theta(a)$ at $\widehat{a}(\theta)$. The two-dimensional integration in the second line was performed by discretising the rectangular region $[0, 4] \times [0, 4]$ with a mesh size 0.01×0.02. For the discrete values of hyperparameters, we obtained $\widehat{a}(\theta)$ by performing limited-memory BFGS [46] using PyTorch. Since we did not use linear kernels for simulated data, we can simply ignore the hyperparameter for linear kernels.

References

1. Williams, C.K.; Rasmussen, C.E. *Gaussian processes for machine learning*; Vol. 2, MIT press Cambridge, MA, 2006.

 doi:10.20944/preprints202511.0763.v1

2. van der Vaart, A.W.; van Zanten, J.H. Rates of contraction of posterior distributions based on Gaussian process priors. *The Annals of Statistics* **2008**, *36*, 1435 – 1463. https://doi.org/10.1214/009053607000000613.
3. van der Vaart, A.; van Zanten, H. Information Rates of Nonparametric Gaussian Process Methods. *Journal of Machine Learning Research* **2011**, *12*, 2095–2119.
4. Suzuki, T. PAC-Bayesian Bound for Gaussian Process Regression and Multiple Kernel Additive Model. In Proceedings of the Proceedings of the 25th Annual Conference on Learning Theory; Mannor, S.; Srebro, N.; Williamson, R.C., Eds., Edinburgh, Scotland, 25–27 Jun 2012; Vol. 23, *Proceedings of Machine Learning Research*, pp. 8.1–8.20.
5. Frigola, R.; Lindsten, F.; Schön, T.B.; Rasmussen, C.E. Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC. In Proceedings of the Advances in Neural Information Processing Systems; Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; Weinberger, K., Eds. Curran Associates, Inc., 2013, Vol. 26.
6. Svensson, A.; Solin, A.; Särkkä, S.; Schön, T. Computationally Efficient Bayesian Learning of Gaussian Process State Space Models. In Proceedings of the Proceedings of the 19th International Conference on Artificial Intelligence and Statistics; Gretton, A.; Robert, C.C., Eds., Cadiz, Spain, 09–11 May 2016; Vol. 51, *Proceedings of Machine Learning Research*, pp. 213–221.
7. Yerramilli, S.; Iyer, A.; Chen, W.; Apley, D.W. Fully Bayesian Inference for Latent Variable Gaussian Process Models. *SIAM/ASA Journal on Uncertainty Quantification* **2023**, *11*, 1357–1381, [https://doi.org/10.1137/22M1525600]. https://doi.org/10.1137/22M1525600.
8. Finocchio, G.; Schmidt-Hieber, J. Posterior Contraction for Deep Gaussian Process Priors. *Journal of Machine Learning Research* **2023**, *24*, 1–49.
9. Bishop, C.M. *Pattern recognition and machine learning*; Vol. 4, Springer, 2006.
10. Paquet, U.; Fraccaro, M. An Efficient Implementation of Riemannian Manifold Hamiltonian Monte Carlo for Gaussian Process Models. *ArXiv* **2018**, *abs/1810.11893*.
11. Pandita, P.; Tsilifis, P.; Ghosh, S.; Wang, L. Scalable Fully Bayesian Gaussian Process Modeling and Calibration With Adaptive Sequential Monte Carlo for Industrial Applications. *Journal of Mechanical Design* **2021**, *143*, 074502. https://doi.org/10.1115/1.4050246.
12. Hensman, J.; Matthews, A.G.d.G.; Filippone, M.; Ghahramani, Z. MCMC for variationally sparse Gaussian processes. In Proceedings of the Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1, Cambridge, MA, USA, 2015; NIPS'15, p. 1648–1656.
13. Betancourt, M.J. A general metric for Riemannian manifold Hamiltonian Monte Carlo. *International Conference on Geometric Science of Information* **2013**, pp. 327–334.
14. Cobb, A.D.; Baydin, A.G.; Markham, A.; Roberts, S.J. Introducing an Explicit Symplectic Integration Scheme for Riemannian Manifold Hamiltonian Monte Carlo. *arXiv preprint arXiv:1910.06243* **2019**.
15. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al., PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*; Curran Associates Inc.: Red Hook, NY, USA, 2019.
16. Solin, A.; Särkkä, S. Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing* **2020**, *30*, 419–446. https://doi.org/10.1007/s11222-019-09886-w.
17. Bach, F.; Jordan, M. Kernel independent component analysis. *Journal of Machine Learning Research* **2003**.
18. Duane, S.; Kennedy, A.; Pendleton, B.J.; Roweth, D. Hybrid Monte Carlo. *Physics Letters B* **1987**, *195*, 216–222. https://doi.org/https://doi.org/10.1016/0370-2693(87)91197-X.
19. Neal, R.M. Probabilistic inference using Markov chain Monte Carlo methods **1993**.
20. Neal, R.M. *Bayesian learning for neural networks*; Vol. 118, Springer Science & Business Media, 2012.
21. Girolami, M.; Calderhead, B. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **2011**, *73*, 123–214, [https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2010.00765.x]. https://doi.org/https://doi.org/10.1111/j.1467-9868.2010.00765.x.
22. Gall, F.L.; Urrutia, F. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* **2018**, p. 1029–1046.
23. van Kempen, H.P.M. On the quadratic convergence of the special cyclic Jacobi method. *Numerische Mathematik* **1966**, *9*, 19–22. https://doi.org/10.1007/BF02165225.
24. Golub, G.H.; Van Loan, C.F. *Matrix computations*; JHU press, 2013.

 doi:10.20944/preprints202511.0763.v1

25. Inc., N. NVIDA HPC SDK Version 23.1 Documentation. https://docs.nvidia.com/hpc-sdk/archive/23.1/index.html, 2023. accessed on 5th Nov. 2025.
26. Johnson, E.; Dominici, F.; Griswold, M.; Zeger, S.L. Disease cases and their medical costs attributable to smoking: an analysis of the national medical expenditure survey. *Journal of Econometrics* **2003**, *112*, 135–151.
27. Imai, K.; Van Dyk, D.A. Causal inference with general treatment regimes: Generalizing the propensity score. *Journal of the American Statistical Association* **2004**, *99*, 854–866.
28. Galagate, D.; Schafer, J.L. Causal inference with a continuous treatment and outcome: Alternative estimators for parametric dose-response functions with applications. *Digital repository at the University of Maryland* **2016**. https://doi.org/https://doi.org/10.13016/M2Q48K.
29. Watanabe, S. *Algebraic geometry and statistical learning theory*; Vol. 25, Cambridge university press, 2009.
30. Inc., N. NVIDIA Developer Forum: CUBLAS dgemm performance query. https://forums.developer.nvidia.com/t/reasonable-timing-with-cublas-dgemm-and-sgemm/14261, 2012. accessed on 5th Nov. 2025.
31. Lung-Sheng, C. Jacobi-Based Eigenvalue Solver on GPU. A shared slide presented in GPU technology conference: https://on-demand.gputechconf.com/gtc/2017/presentation/s7121-lung-sheng-chien-jacobi-based-eigenvalue-solver.pdf, 2017. accessed on 1st June 2022.
32. Fawzi, H.; Goulbourne, H. Faster proximal algorithms for matrix optimization using Jacobi-based eigenvalue methods. In Proceedings of the Advances in Neural Information Processing Systems; Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; Vaughan, J.W., Eds. Curran Associates, Inc., 2021, Vol. 34, pp. 11397–11408.
33. Hoffman, M.D.; Gelman, A. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* **2014**, *15*, 1593–1623.
34. Betancourt, M.J. Generalizing the No-U-Turn Sampler to Riemannian Manifolds, 2013, [arXiv:stat.ME/1304.1920].
35. Cheng, X.; Chatterji, N.S.; Bartlett, P.L.; Jordan, M.I. Underdamped Langevin MCMC: A non-asymptotic analysis. In Proceedings of the Proceedings of the 31st Conference On Learning Theory; Bubeck, S.; Perchet, V.; Rigollet, P., Eds. PMLR, 06–09 Jul 2018, Vol. 75, *Proceedings of Machine Learning Research*, pp. 300–323.
36. Dalalyan, A.S.; Riou-Durand, L. On sampling from a log-concave density using kinetic Langevin diffusions. *Bernoulli* **2020**, *26*, 1956 – 1988. https://doi.org/10.3150/19-BEJ1178.
37. GREEN, P.J. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **1995**, *82*, 711–732, [https://academic.oup.com/biomet/article-pdf/82/4/711/699533/82-4-711.pdf]. https://doi.org/10.1093/biomet/82.4.711.
38. Karagiannis, G.; Andrieu, C. Annealed Importance Sampling Reversible Jump MCMC Algorithms. *Journal of Computational and Graphical Statistics* **2013**, *22*, 623–648, [https://doi.org/10.1080/10618600.2013.805651]. https://doi.org/10.1080/10618600.2013.805651.
39. Moral, P. *Feynman-Kac formulae: genealogical and interacting particle systems with applications*; Springer, 2004.
40. Chopin, N.; Papaspiliopoulos, O.; et al. *An introduction to sequential Monte Carlo*; Vol. 4, Springer, 2020.
41. Chernozhukov, V.; Chetverikov, D.; Demirer, M.; Duflo, E.; Hansen, C.; Newey, W.; Robins, J. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal* **2018**, *21*, C1–C68, [https://onlinelibrary.wiley.com/doi/pdf/10.1111/ectj.12097]. https://doi.org/https://doi.org/10.1111/ectj.12097.
42. Hayakawa, T.; Asai, S. Debiased Maximum Likelihood Estimators of Hazard Ratios Under Kernel-Based Machine Learning Adjustment. *Mathematics* **2025**, *13*. https://doi.org/10.3390/math13193092.
43. Watanabe, S. A widely applicable Bayesian information criterion. *Journal of Machine Learning Research* **2013**, *14*, 867–897.
44. Drton, M.; Plummer, M. A Bayesian Information Criterion for Singular Models. *Journal of the Royal Statistical Society Series B: Statistical Methodology* **2017**, *79*, 323–380. https://doi.org/10.1111/rssb.12187.
45. Calderhead, B.; Girolami, M. Estimating Bayes factors via thermodynamic integration and population MCMC. *Computational Statistics and Data Analysis* **2009**, *53*, 4028–4045. https://doi.org/https://doi.org/10.1016/j.csda.2009.07.025.
46. Liu, D.C.; Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming* **1989**, *45*, 503–528.

 doi:10.20944/preprints202511.0763.v1

disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.