Reflected Hamiltonian Monte Carlo

Khai Xiang Au
Integrative Sciences and Engineering Programme
National University of Singapore
khai@u.nus.edu

Alexandre Thiery
Department of Statistics and Applied Probability
National University of Singapore
a.h.thiery@nus.edu.sg

1 Background

We briefly describe the standard *Hamiltonian Monte Carlo* (HMC) method [4, 14] to prepare the machinery needed for *Reflected Hamiltonian Monte Carlo* (RHMC). Consider a d-dimensional *position* variable $\boldsymbol{q} \in \mathbb{R}^d$ distributed according to a target distribution with an unnormalized density $\pi : \mathbb{R}^d \to \mathbb{R}$. The HMC method augments the position space with an auxiliary *momentum* variable of the same dimension $\boldsymbol{p} \sim \mathrm{N}(\boldsymbol{p} \mid \boldsymbol{0}, \mathrm{M})$, where $\mathrm{M} \in \mathbb{R}^{d \times d}$ is the *mass matrix*. The *Hamiltonian* is then defined by $\mathrm{H}(\boldsymbol{q}, \boldsymbol{p}) \equiv -\log \pi(\boldsymbol{q}) + \frac{1}{2} \langle \boldsymbol{p}, \mathrm{M}^{-1} \boldsymbol{p} \rangle$. The sampler explores the joint density

$$\overline{\pi}(\boldsymbol{q}, \boldsymbol{p}) \equiv \exp\{-\mathrm{H}(\boldsymbol{q}, \boldsymbol{p})\} = \pi(\boldsymbol{q}) \times \mathrm{N}(\boldsymbol{p} \mid \boldsymbol{0}, \mathrm{M}) \tag{1}$$

by evolving the Hamiltonian dynamics $\frac{d\boldsymbol{p}}{dt} = -\partial_{\boldsymbol{q}} \mathrm{H}(\boldsymbol{q}, \boldsymbol{p}) = \nabla \log \pi(\boldsymbol{q})$; $\frac{d\boldsymbol{q}}{dt} = \partial_{\boldsymbol{p}} \mathrm{H}(\boldsymbol{q}, \boldsymbol{p}) = \mathrm{M}^{-1} \boldsymbol{p}$. This is achieved numerically usually with the *leapfrog integrator* [8, 13]

$$\boldsymbol{p}_{1/2} = \boldsymbol{p}_0 + \tfrac{\delta}{2} \nabla \log \pi(\boldsymbol{q}_0). \quad \boldsymbol{q}_1 = \boldsymbol{q}_0 + \delta\, \mathrm{M}^{-1} \boldsymbol{p}_{1/2}, \quad \boldsymbol{p}_1 = \boldsymbol{p}_{1/2} + \tfrac{\delta}{2} \nabla \log \pi(\boldsymbol{q}_1) \tag{2}$$

for a time-discretization parameter $\delta > 0$ called the *step size*. Iterating $L \geq 1$ leapfrog steps of step size $\delta > 0$ approximately evolves the dynamics for an integration time of $T = L \times \delta$. As the leapfrog integrator imperfectly conserves the Hamiltonian, a standard Metropolis-Hastings accept-reject step is necessary to maintain detailed balance. To ensure ergodicity, the momentum variable $\boldsymbol{p} \in \mathbb{R}^d$ is regenerated from $\mathrm{N}(0, \mathrm{M})$ regularly.

2 Reflected Hamiltonian Monte Carlo

The RHMC is composed of the (deterministic) alternation of two Markov kernels: **(1)** a reflected Hamiltonian dynamic [15, 20, 16, 1], **(2)** a momentum update. The resulting Markov kernel lets the joint density $\overline{\pi}$ invariant. We detail the RHMC Markov kernel below and present the pseudo-code in Algorithm 1.

Reflected Hamiltonian Dynamics. Starting from $(\boldsymbol{q}_k, \boldsymbol{p}_k)$, a proposal $(\boldsymbol{q}', \boldsymbol{p}')$ is obtained by the Hamiltonian flow for a duration $\Delta T > 0$, realised by taking 1 leapfrog step (2). The proposal $(\boldsymbol{q}', \boldsymbol{p}')$ is accepted with probability

$$\alpha_1(\boldsymbol{q}_k, \boldsymbol{p}_k) = 1 \wedge \exp\left[-\mathrm{H}(\boldsymbol{q}', \boldsymbol{p}') + \mathrm{H}(\boldsymbol{q}_k, \boldsymbol{p}_k)\right]. \tag{3}$$

Upon acceptance, the new position is set as $(\widehat{\boldsymbol{q}}_k, \widehat{\boldsymbol{q}}_k) = (\boldsymbol{q}', \boldsymbol{p}')$. Upon rejection, a new proposal $(\boldsymbol{q}'', \boldsymbol{p}'')$ is generated by reflecting the momentum with respect to the affine hyperplane orthogonal to

$\boldsymbol{g} \equiv \nabla \log \pi(\boldsymbol{q}')$ and passing through $\boldsymbol{q}'$, and then following the Hamiltonian dynamics for a duration $\Delta T > 0$. Precisely, the reflected momentum $\boldsymbol{p}'_R$ defined as

$$\boldsymbol{p}'_R = \boldsymbol{p}' - 2\frac{\langle \boldsymbol{p}', \mathrm{M}^{-1}\boldsymbol{g}\rangle}{\langle \boldsymbol{g}, \mathrm{M}^{-1}\boldsymbol{g}\rangle}\boldsymbol{g}. \tag{4}$$

The proposal $(\boldsymbol{q}'', \boldsymbol{p}'')$ is accepted with the usual delayed-rejection probability [19, 7]

$$\alpha_2(\boldsymbol{q}_k, \boldsymbol{p}_k) = 1 \wedge \left\{ \left[\frac{1 - \alpha_1(\boldsymbol{q}'', -\boldsymbol{p}'')}{1 - \alpha_1(\boldsymbol{q}_i, \boldsymbol{p}_i)} \right] \exp\left[-\mathrm{H}(\boldsymbol{q}'', \boldsymbol{p}'') + \mathrm{H}(\boldsymbol{q}_k, \boldsymbol{p}_k)\right] \right\}, \tag{5}$$

Upon acceptance, the new state is set as $(\widehat{\boldsymbol{q}}_k, \widehat{\boldsymbol{q}}_k) = (\boldsymbol{q}'', \boldsymbol{p}'')$. Upon rejection, the final state is obtained by negating the momentum so that $(\widehat{\boldsymbol{q}}_k, \widehat{\boldsymbol{p}}_k) = (\boldsymbol{q}_k, -\boldsymbol{p}_k)$.

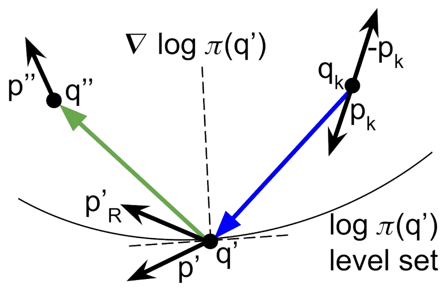

Figure 1: A 2D visualisation of the *Reflected Hamiltonian Dynamics*. Starting from $(\boldsymbol{q}_k, \boldsymbol{p}_k)$, the state $(\boldsymbol{q}', \boldsymbol{p}')$ is proposed (blue arrow). Upon rejection, the delayed-rejection state $(\boldsymbol{q}'', \boldsymbol{p}'')$ is proposed (green arrow). If the second proposal is also rejected, then the sampler remains at $\boldsymbol{q}_k$ and has its momentum negated.

Momentum Updates. We discuss two standard mechanisms [16, 11], both depending on a crucial parameter $\kappa > 0$. Consider an initial state $(\widehat{\boldsymbol{q}}_k, \widehat{\boldsymbol{p}}_k)$.

1. **Full refreshment:** with probability $p = 1 - \exp[-\kappa\,\delta]$, the new state is set as $(\boldsymbol{q}_{k+1}, \boldsymbol{p}_{k+1}) = (\widehat{\boldsymbol{q}}_k, \xi)$ where $\xi \sim \mathrm{N}(0, \mathrm{M})$ is a newly generated momentum. With probability $(1-p)$, the state is unchanged, i.e. $(\boldsymbol{q}_{k+1}, \boldsymbol{p}_{k+1}) = (\widehat{\boldsymbol{q}}_k, \widehat{\boldsymbol{p}}_k)$. This can be understood as a discretization on the time-interval $(t, t+\delta)$ of the Markov process that completely refreshes the momentum at rate $\kappa > 0$.
2. **Auto-regressive refreshment:** set $\alpha = \exp[-\kappa\,\delta/2]$ and $\beta = \sqrt{1-\alpha^2}$ so that $\alpha^2 + \beta^2 = 1$ and generate $\xi \sim \mathrm{N}(0, \mathrm{M})$. The new state $(\boldsymbol{q}_{k+1}, \boldsymbol{p}_{k+1})$ is defined as $\boldsymbol{q}_{k+1} = \widehat{\boldsymbol{q}}_k$ and
$$\boldsymbol{p}_{k+1} = \alpha\,\widehat{\boldsymbol{p}}_k + \beta\,\boldsymbol{\xi}.$$
This is similar to the momentum update date in the Horowitz's *second-order Lanvegin Monte Carlo* (L2MC) method [11] and can be understood as the discretization on the time-interval $(t, t+\delta)$ of an Ornstein-Uhlenbeck process $dP = -\kappa/2\,P\,dt + \kappa^{1/2}\,\mathrm{M}^{1/2}\,dB_t$, where B_t is a standard Brownian motion in $\mathbb{R}^d$.

Note the relation between the update rate κ in RHMC and the number of leapfrog steps $L \geq 1$ in HMC. In the RHMC method with *full refreshment*, on average, the momentum is refreshed after a duration $\mathcal{O}(1/\kappa)$. Similarly, when implementing a standard HMC method with parameter $\delta > 0$ and $L \geq 0$, the momentum variable is fully refreshed after a duration $T = L \times \delta$. In other words, parameter κ can be thought of as the equivalent of $1/L\delta$. An unsuitably large κ leads to overly frequent momentum updates leading to a diffusive and inefficient behaviour. On the other hand, when κ is set too small the RHMC sampler gets trapped on the same Hamiltonian level set for many iterations, diminishing its exploration of the target distribution.

2.1 Tuning the update rate parameter

We advocate to tune the parameter κ based on short preliminary runs. Consider a function of interest $\varphi : \mathbb{R}^d \to \mathbb{R}$. Our proposed tuning approach is based on the remark [17] that in high dimensions the

Algorithm 1 RHMC Kernel

Input: Current position $\boldsymbol{q}$, current momentum $\boldsymbol{p}$
Parameter: Step size δ, update rate κ, mass matrix M .
Output: Updated position $\tilde{\boldsymbol{q}}$, updated momentum $\tilde{\boldsymbol{p}}$.

```
1: Function RHMCKernel ($\boldsymbol{q}, \boldsymbol{p}$)
2: $(\boldsymbol{q}', \boldsymbol{p}') \leftarrow$ Leapfrog$(\boldsymbol{q}, \boldsymbol{p})$ from Equation 2
3: $A_1 \leftarrow \alpha_1(\boldsymbol{q}, \boldsymbol{p})$ from from Equation 3
4: if Uniform$(0, 1) < A_1$ then
5:     $(\tilde{\boldsymbol{q}}, \hat{\boldsymbol{p}}) \leftarrow (\boldsymbol{q}', \boldsymbol{p}')$
6: else
7:     $\boldsymbol{p}'_R \leftarrow$ Reflect$(\boldsymbol{p}', \boldsymbol{q}')$ from Equation 4
8:     $(\boldsymbol{q}'', \boldsymbol{p}'') \leftarrow$ Leapfrog$(\boldsymbol{q}', \boldsymbol{p}'_R)$ from Equation 2
9:     $A_2 \leftarrow \alpha_2(\boldsymbol{q}, \boldsymbol{p})$ from Equation 5
10:    if Uniform$(0, 1) < A_2$ then
11:        $(\tilde{\boldsymbol{q}}, \hat{\boldsymbol{p}}) \leftarrow (\boldsymbol{q}'', \boldsymbol{p}'')$
12:    else
13:        $(\tilde{\boldsymbol{q}}, \hat{\boldsymbol{p}}) \leftarrow (\boldsymbol{q}, -\boldsymbol{p})$
14:    end if
15: end if
16: $\tilde{\boldsymbol{p}} \leftarrow$ Refresh $\hat{\boldsymbol{p}}$ following Momentum update scheme 1 or 2
17: return $\tilde{\boldsymbol{q}}, \tilde{\boldsymbol{p}}$
18: EndFunction
```

auto-correlation function of $\{\varphi(\boldsymbol{q}_j^\kappa)\}_{j=1}^N$ decays approximately exponentially at some rate $\lambda > 0$. Therefore, the aim is to tune the parameter κ to approximately maximize λ. We advocate choosing a fixed correlation threshold $\gamma \in (0, 1)$, set to $\gamma = 10\%$ in our experiments, and record the first lag $\widehat{\Delta}(\kappa)$ when the auto-correlation falls below this threshold, i.e. $\mathrm{Corr}[\varphi(\boldsymbol{q}_k), \varphi(\boldsymbol{q}_{k+\widehat{\Delta}})] \leq \gamma$. This leads to an estimate of the rate of decay $\widehat{\lambda}(\kappa) \equiv -\log(\gamma)/\widehat{\Delta}(\kappa)$, which can be maximized over the parameter κ over a small grid-search. The preliminary chains used for our work is 2000 samples long.

3 Experiments

We test the samplers on a strongly-correlated Gaussian setting, a high-dimensional parameter estimation problem, and eight classification problems commonly used in the Bayesian neural network literature [9, 18, 3, 12].

Evaluation metric. We define sampling efficiency as the *effective sample size* (ESS) [6, 5] per number of gradient computations. For parameter estimation tasks, we are interested in the mixing of the log-target process [16], i.e. the process $\{\log \pi(\boldsymbol{q}_k)\}_{k \geq 0}$. For neural-network classification tasks, we study of the mixing of the negative averaged log-likelihood evaluated on a test set [12]

Baseline methods. We benchmark the performance of RHMC with *full refreshment* (RHMC-F) and RHMC with *auto-regressive refreshment* (RHMC-AR) against *Metropolis-adjusted Langevin algorithm* (MALA), *No-U-Turn Sampler* (NUTS), Horowitz'sL2MC (Generalised HMC with $L = 1$) [11], and HMC with jittering in the number of leapfrog steps L [21, 9], where L is tuned by an exhaustive grid-search. Specifically, at each iteration the number of leapfrog steps taken is drawn independently from DiscreteUniform$(1, L)$.

4 Discussion

Optimal RHMC vs optimal L2MC vs optimal HMC. The optimal samplers tuned by grid-search over the tuning parameters κ and L respectively using long chains. In more than half of the examples do either RHMC-AR or RHMC-F achieve overall best performance. Indeed, the results above are

Table 1: ESS per gradient call relative to NUTS, with best results (aside from exhaustively-tuned HMC) underlined and in brackets optimal results from an exhaustive grid-search of κ.

Name	HMC	RHMC-F	RHMC-AR	L2MC	MALA	NUTS
Gaussian	6.73	$\underline{1.37}$ (3.33)	0.70 (5.08)	1.02 (4.75)	0.11	1.00
Australian Credit (NN)	5.46	0.69 (1.87)	0.72 (1.70)	0.22 (0.98)	0.19	$\underline{1.00}$
German Credit (NN)	3.07	0.80 (1.45)	$\underline{1.53}$ (1.71)	0.52 (0.77)	0.14	1.00
Iris (NN)	2.65	$\underline{2.42}$ (3.77)	1.85 (5.58)	1.74 (2.06)	0.60	1.00
Banknote (NN)	5.98	3.18 (6.93)	$\underline{3.40}$ (9.10)	2.65 (5.02)	1.81	1.00
Red Wine (LR)	4.98	$\underline{6.33}$ (7.77)	4.38 (8.70)	4.26 (4.47)	0.71	1.00
Australian Credit (LR)	1.37	2.09 (3.57)	$\underline{2.65}$ (3.54)	2.39 (2.62)	1.80	1.00
German Credit (LR)	1.21	1.11 (2.09)	$\underline{2.03}$ (2.03)	1.30 (1.74)	0.48	1.00
Item Response (LR)	8.74	0.01 (7.78)	9.15 (11.75)	$\underline{12.85}$ (12.85)	6.57	1.00

naive in that running long chains over a large grid is unrealistic in practical applications. That said, this comparison serves to demonstrate the prospects of a well-tuned RHMC.

Performance of RHMC, tuned as suggested. Using our proposed tuning strategy for κ, we observe that either RHMC-AR or RHMC-F delivering overall better sampling performance compared to NUTS [10], MALA, and L2MC in all but one test example. In half of the examples, the RHMC even beats HMC, even with L tuned via an impractically large grid-search. Unsurprisingly, the tuning strategy fails to identify good κ settings in some scenarios, e.g. RHMC-F in the Item Response example. With respect to the optimal κ settings, on average RHMC-F achieves 55% of its optimal efficiency; RHMC-AR achieves 57% of its optimal efficiency.

Benefits of an auto-regressive momentum update. Note that if we replace the momentum refresh of MALA with the auto-regressive momentum refresh described in 2, we arrive at the L2MC. We observe that L2MC beat MALA in every experiment and attribute this improved sampling efficiency of L2MC over MALA to the auto-regressive momentum refresh.

Benefits of the reflected Hamiltonian dynamics. RHMC-AR can be thought of as L2MC with the addition of a delayed-rejection step described in the reflected Hamiltonian dynamics. In this view, we see that on average RHMC-AR performs 50% better than L2MC, which showcases the improved mixing due to the reflected Hamiltonian dynamics. Note that we have taken into account the additional cost due to the delayed-rejection proposal computations.

Practical considerations. A practical advantage of RHMC is that it can be implemented on the GPU with ease, making the tuning of update rate κ inexpensive as multiple chains can be run in parallel, e.g. using `vmap` function in `JAX` [2]. With a rough tuning of RHMC based on our suggestion to run many short chains, we observe competitive results when compared to baseline methods.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section **??**
   (b) Did you describe the limitations of your work? [Yes] See Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] It is available upon request / in the full paper
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [No] It is available upon request / in the full paper
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] It is available upon request / in the full paper
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] see Section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] see Section 3
   (b) Did you mention the license of the assets? [Yes] see Section 3 for citation of UCI datasets
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No personal data were used
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No personal data were used
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Alexandre Bouchard-Côté, Sebastian J Vollmer, and Arnaud Doucet. The bouncy particle sampler: A nonreversible rejection-free markov chain monte carlo method. *Journal of the American Statistical Association*, 113(522):855–867, 2018.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

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

[5] Charles J Geyer. Practical markov chain monte carlo. *Statistical science*, pages 473–483, 1992.

[6] Peter W Glynn and Ward Whitt. Estimating the asymptotic variance with batch means. *Operations Research Letters*, 10(8):431–435, 1991.

[7] Peter J Green and Antonietta Mira. Delayed rejection in reversible jump metropolis–hastings. *Biometrika*, 88(4):1035–1053, 2001.

[8] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the störmer–verlet method. *Acta numerica*, 12:399–450, 2003.

[9] Matthew Hoffman, Alexey Radul, and Pavel Sountsov. An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 3907–3915. PMLR, 2021.

[10] Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[11] Alan M Horowitz. A generalized guided monte carlo algorithm. *Physics Letters B*, 268(2):247–252, 1991.

[12] Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, and Andrew Gordon Wilson. What are bayesian neural network posteriors really like? In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4629–4640. PMLR, 2021.

[13] Benedict J Leimkuhler and Robert D Skeel. Symplectic numerical integrators in constrained Hamiltonian systems. *Journal of Computational Physics*, 112(1):117–125, 1994.

[14] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[15] Elias AJF Peters et al. Rejection-free monte carlo sampling for general potentials. *Physical Review E*, 85(2):026703, 2012.

[16] Chris Sherlock and Alexandre H. Thiery. A discrete bouncy particle sampler, 2021.

[17] Alan Sokal. Monte carlo methods in statistical mechanics: foundations and new algorithms. In *Functional integration*, pages 131–192. Springer, 1997.

[18] Stan Development Team. Stan modeling language users guide and reference manual, 2019. Version 2.27.

[19] Luke Tierney and Antonietta Mira. Some adaptive monte carlo methods for bayesian inference. *Statistics in medicine*, 18(17-18):2507–2515, 1999.

[20] Paul Vanetti, Alexandre Bouchard-Côté, George Deligiannidis, and Arnaud Doucet. Piecewise-deterministic markov chain monte carlo. *arXiv preprint arXiv:1707.05296*, 2017.

[21] Ziyu Wang, Shakir Mohamed, and Nando Freitas. Adaptive hamiltonian and riemann manifold monte carlo. In *International conference on machine learning*, pages 1462–1470. PMLR, 2013.