Nonparametric Hamiltonian Monte Carlo

Carol Mak [1] Fabian Zaiser [1] Luke Ong [1]

Abstract

Probabilistic programming uses programs to express generative models whose posterior probability is then computed by built-in inference engines. A challenging goal is to develop general purpose inference algorithms that work out-of-the-box for arbitrary programs in a universal probabilistic programming language (PPL). The densities defined by such programs, which may use stochastic branching and recursion, are (in general) *nonparametric*, in the sense that they correspond to models on an infinite-dimensional parameter space. However standard inference algorithms, such as the Hamiltonian Monte Carlo (HMC) algorithm, target distributions with a fixed number of parameters. This paper introduces the *Nonparametric Hamiltonian Monte Carlo* (NP-HMC) algorithm which generalises HMC to nonparametric models. Inputs to NP-HMC are a new class of measurable functions called "*tree representable*", which serve as a language-independent representation of the density functions of probabilistic programs in a universal PPL. We provide a correctness proof of NP-HMC, and empirically demonstrate significant performance improvements over existing approaches on several nonparametric examples.

1. Introduction

Probabilistic programming is a general purpose means of expressing probabilistic models as programs, and automatically performing Bayesian inference. Probabilistic programming systems enable data scientists and domain experts to focus on designing good models; the task of developing efficient inference engines can be left to experts in Bayesian statistics, machine learning and programming languages. To realise the full potential of probabilistic programming, it is essential to automate the inference of latent variables in the model, conditioned on the observed data.

Church (Goodman et al., 2008) introduced *universal probabilistic programming*, the idea of writing probabilistic models in a Turing-complete functional programming language. Typically containing only a handful of basic programming constructs such as branching and recursion, universal probabilistic programming languages (PPLs) can nonetheless specify all computable probabilistic models (Vákár et al., 2019). In particular, ***nonparametric models***—models with an unbounded number of random variables—can be described naturally in universal PPLs using recursion. These include probabilistic models with an unknown number of components, like Bayesian nonparametric models (Richardson & Green, 1997), variable selection in regression (Ratner, 2010), signal processing (Murray et al., 2018); and models that are defined on infinite-dimensional spaces, such as probabilistic context free grammars (Manning & Schütze, 1999), birth-death models of evolution (Kudlicka et al., 2019) and statistical phylogenetics (Ronquist et al., 2021). Examples of practical universal PPL include Anglican (Wood et al., 2014), Venture (Mansinghka et al., 2014), Web PPL (Goodman & Stuhlmüller, 2014), Hakaru (Narayanan & Shan, 2020), Pyro (Bingham et al., 2019), Turing (Ge et al., 2018) and Gen (Cusumano-Towner et al., 2019).

However, because universal PPLs are expressively complete, it is a challenging problem to design and implement general purpose inference engines for them. The parameter space of a nonparametric model is a union of spaces of varying dimensions. To approximate the posterior via an Markov chain Monte Carlo (MCMC) algorithm, the transition kernel will have to efficiently switch between a potentially unbounded number of configurations of different dimensions. This difficulty explains why there are so few *general purpose* MCMC algorithms for universal PPLs (Wingate et al., 2011; Wood et al., 2014; Tolpin et al., 2015; Hur et al., 2015). We believe it is also the reason why these algorithms struggle with nonparametric models, as we show in Sec. 5. A case in point is the widely used universal PPL Pyro. Even though it allows the specification of nonparametric models, its HMC and No-U-Turn Sampler (Hoffman & Gelman, 2014) inference engines do not support them reliably: in one of our benchmark tests, they produced a wrong posterior (Fig. 6).

[1]Department of Computer Science, University of Oxford, United Kingdom. Correspondence to: Carol Mak <pui.mak@cs.ox.ac.uk>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

In this paper, we introduce the *Nonparametric Hamiltonian Monte Carlo* (NP-HMC) algorithm, which generalises the Hamiltonian Monte Carlo (HMC) algorithm (Duane et al., 1987) to nonparametric models. The input to NP-HMC is what we call a *tree representable* (TR) function, which is a large class of measurable functions of type $w : \bigcup_{n\in\mathbb{N}} \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, designed to be a language-independent representation for the density functions of programs written in any universal PPL. The parameter space of the standard HMC algorithm is $\mathbb{R}^n$, a Euclidean space of a fixed dimension. By contrast, the parameter space of NP-HMC is $\bigcup_{n\in\mathbb{N}} \mathbb{R}^n$. The key innovation of NP-HMC is a method by which the dimension of the configuration of the current sample is incremented lazily, while preserving the efficacy of HMC by keeping the Hamiltonian approximately invariant. We prove that NP-HMC is correct, i.e., the induced Markov chain converges to the posterior distribution. To evaluate the practical utility of NP-HMC, we compare an implementation of the algorithm against existing out-of-the-box MCMC inference algorithms on several challenging models with an unbounded number of random variables. Our results suggest that NP-HMC is applicable to a large class of probabilistic programs written in universal PPLs, offering significantly better performance than existing algorithms.

Notation We write $\boldsymbol{q}$ to mean a (possibly infinite) real-valued sequence; $\boldsymbol{q}^{1\dots i}$ the prefix of $\boldsymbol{q}$ consisting of the first i coordinates; $\boldsymbol{q}_i$ the i-th coordinate of $\boldsymbol{q}$; and $|\boldsymbol{q}|$ the length of $\boldsymbol{q}$. We write sequence as lists, such as $[3.6, 1.0, 3, 55, -4.2]$, and the concatenation of sequences $\boldsymbol{q}$ and $\boldsymbol{q}'$ as $\boldsymbol{q} + \boldsymbol{q}'$.

We write $\mathcal{B}_n$ for the Borel σ-algebra of $\mathbb{R}^n$; $\mathcal{N}_n$ for the standard n-dimensional normal distribution with mean $\mathbf{0}$ and covariance $\boldsymbol{I}$; $\varphi_n(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for the density of $\boldsymbol{x} \in \mathbb{R}^n$ in the n-dimensional normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. For brevity we write $\varphi_n(\boldsymbol{x})$ for $\varphi_n(\boldsymbol{x} \mid \mathbf{0}, \boldsymbol{I})$ and φ for φ_1.

For any $\mathbb{R}_{\geq 0}$-valued function $f : \mathsf{Dom}(f) \to \mathbb{R}_{\geq 0}$, we write $\mathsf{Supp}(f) := f^{-1}(\mathbb{R}_{>0})$ for the ***support*** of f; and $\mathsf{Supp}^n(f) := \mathsf{Supp}(f) \cap \mathbb{R}^n$ for the support of f in $\mathbb{R}^n$. We say $x \in X$ is f-***supported*** if $x \in \mathsf{Supp}(f)$.

2. Tree Representable Functions

Conventional HMC samples from a distribution with a density function $w : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ where the dimension of the target (parameter) space $\mathbb{R}^n$ is fixed. However this is too restrictive for probabilistic programs, because—with branching and recursion—the target space has a variable, even unbounded, number of dimensions.

Example 1 (Working)**.** Consider the probabilistic program in Listing 1 where `sample(normal(0, 1))` denotes sampling from the standard normal distribution. The dimension of the target space, i.e. the number of samples drawn, can vary from run to run because of the branching behaviour of the while loop. (Recall that by ***trace***, we mean the sequence of samples drawn in the course of a particular run, one for each random primitive encountered.) The density function then records the weight of this trace, computed by multiplying the probability densities of all sampled values and the likelihoods of all observations during the run. For the above program, we could have a trace $[0.3, 0.5] \in \mathbb{R}^2$ of length 2 or a trace $[1.0, 0.5, 0.5] \in \mathbb{R}^3$ of length 3, and so on. Hence we have to consider density functions of type $w : \bigcup_{n\in\mathbb{N}} \mathbb{R}^n \to \mathbb{R}_{\geq 0}$. However, not every such function makes sense as the density of a probabilistic program. For example, if $w([1.0, 0.5, 0.5]) > 0$ then the program can execute successfully with the trace $[1.0, 0.5, 0.5]$, but not with $[1.0, 0.5]$ or any other prefix. In other words, no proper prefix of $[1.0, 0.5, 0.5]$ is in $\mathsf{Supp}(w)$.

```
q = sample(normal(0, 1))
sum = 0
while sum < q:
  sum += sample(normal(0, 1))
observe(sum, normal(q, 1))
```

Listing 1. A simple probabilistic program.

Thus we set our target space to be the ***measure space of traces*** $\mathbb{T} := \bigcup_{n\in\mathbb{N}} \mathbb{R}^n$ equipped with the standard disjoint union σ-algebra $\Sigma_{\mathbb{T}} := \{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \mathcal{B}_n\}$, with measure given by summing the respective (higher-dimensional) normals $\mu_{\mathbb{T}}(\bigcup_{n\in\mathbb{N}} U_n) := \sum_{n\in\mathbb{N}} \mathcal{N}_n(U_n)$.

We consider density functions that are measurable functions $w : \mathbb{T} \to \mathbb{R}_{\geq 0}$ satisfying the ***prefix property***: whenever $\boldsymbol{q} \in \mathsf{Supp}^n(w)$ then for all $k < n$, we have $\boldsymbol{q}^{1\dots k} \notin \mathsf{Supp}^k(w)$. We call them ***tree representable (TR) functions*** because any such function w can be represented as a possibly infinite but finitely branching tree, which we call *program tree*. This is exemplified in Fig. 1 (left), where a circular node denotes an element of the input of type $\mathbb{R}$; a rectangular node gives the condition for $\boldsymbol{q} \in \mathsf{Supp}^n(w)$ (with the left, but not the right, child satisfying the condition); and a leaf node gives the result of the function on that branch. Any ***branch*** (i.e. path from root to leaf) in a program tree of w represents a set of finite sequences $[q_1, \dots, q_n]$ in $\mathsf{Supp}(w)$. In fact, every program tree of a TR function w specifies a countable partition of $\mathsf{Supp}(w)$ via its branches. The prefix property guarantees that for each TR function w, there are program trees of the form in Fig. 1 (left) representing w.

We target TR functions as densities for our new sampler NP-HMC because they are a naturally large class of functions. In particular, every program of a universal PPL has a density function that is tree representable.[1] (See Prop. 7 in App. A for a formal account.) For instance, the program in Listing 1

[1] With (additional) suitable assumptions about the computability of w, we can view any such tree as the abstract syntax tree of a

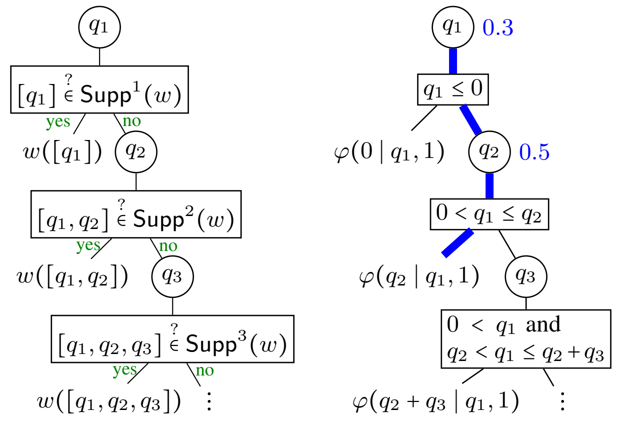

q_1
$[q_1] \overset{?}{\in} \mathsf{Supp}^1(w)$
yes
no
$w([q_1])$
q_2
$[q_1, q_2] \overset{?}{\in} \mathsf{Supp}^2(w)$
$w([q_1, q_2])$
q_3
$[q_1, q_2, q_3] \overset{?}{\in} \mathsf{Supp}^3(w)$
$w([q_1, q_2, q_3])$
q_1 0.3
$q_1 \leq 0$
$\varphi(0 \mid q_1, 1)$
q_2 0.5
$0 < q_1 \leq q_2$
$\varphi(q_2 \mid q_1, 1)$
q_3
$0 < q_1$ and $q_2 < q_1 \leq q_2 + q_3$
$\varphi(q_2 + q_3 \mid q_1, 1)$

Figure 1. (left) Generic TR function w; (right) TR function w for the probabilistic program in Listing 1.

has density[2] w (w.r.t. the stock measure $\mu_{\mathbb{T}}$) given by

$$w(\boldsymbol{q}) := \begin{cases} \varphi(\sum_{i=2}^{n} \boldsymbol{q}_i \mid \boldsymbol{q}_1, 1) & \text{if } \forall k < |\boldsymbol{q}| \\ & \sum_{i=2}^{k} \boldsymbol{q}_i < \boldsymbol{q}_1 \leq \sum_{i=2}^{n} \boldsymbol{q}_i, \\ 0 & \text{otherwise} \end{cases}$$

which is TR and it has a program tree as depicted in Fig. 1 (right). Notice that every element in the support of w belongs to a branch in this tree: for example, the trace $[0.3, 0.5]$ belongs to the blue branch in Fig. 1 (right).

As we will explain in Sec. 3, the prefix property (satisfied by TR functions) is essential for the correctness of NP-HMC.

3. Nonparametric HMC

Nonparametric Hamiltonian Monte Carlo (NP-HMC) (Fig. 4) is a MCMC algorithm that, given a TR function w, iteratively proposes a new sample $\boldsymbol{q} \in \bigcup_{n \in \mathbb{N}} \mathbb{R}^n$ and accepts it with a suitable Hastings acceptance probability, such that the invariant distribution of the induced Markov chain is

$$\nu : A \mapsto \frac{1}{Z} \int_A w \, \mathrm{d}\mu_{\mathbb{T}}$$

with normalising constant $Z := \int_{\mathbb{T}} w \, \mathrm{d}\mu_{\mathbb{T}}$. As the name suggests, NP-HMC is a generalisation of the standard HMC algorithm (Rem. 2.i) to *nonparametric models*, in the form of TR functions whose support is a subspace of $\mathbb{T}$ of unbounded dimension.

In this section, we first explain our generalised algorithm, using a version (Alg. 1) that is geared towards conceptual clarity, and defer discussions of more efficient variants.

program that computes w, but with any recursion unravelled (so that the tree is potentially infinite).

[2]Notice that, even though the program samples from a normal distribution, w does not factor in Gaussian densities from those sample statements, just the observe statement, since they are already accounted for by $\mu_{\mathbb{T}}$.

Idea We assume basic familiarity with the HMC algorithm; see (Betancourt, 2018) for the intuition behind it and (Neal, 2011) for details. Like the HMC algorithm, NP-HMC views a sample $\boldsymbol{q}$ as a particle at position $\boldsymbol{q}$, with a randomly initialised momentum $\boldsymbol{p}$, moving on a frictionless surface derived from the density function w. The key innovation lies in our treatment when the particle moves *beyond* the surface, i.e. outside the support of the density function. This procedure is called extend (Alg. 3), which we will now illustrate.

Let's trace the movement of a particle at position $\boldsymbol{q} = [-3.1]$ with a randomly chosen momentum $\boldsymbol{p} = [1.2]$, on the surface (a line in 1D) determined by the TR function w in Fig. 1 (right). The first two steps taken by the particle are simulated according to the Hamiltonian dynamics on the surface $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \leq 0]$, which is derived from the restriction of w to $\mathbb{R}$. The positions on the surface and states[3] of the particle at each step are given in Fig. 2.

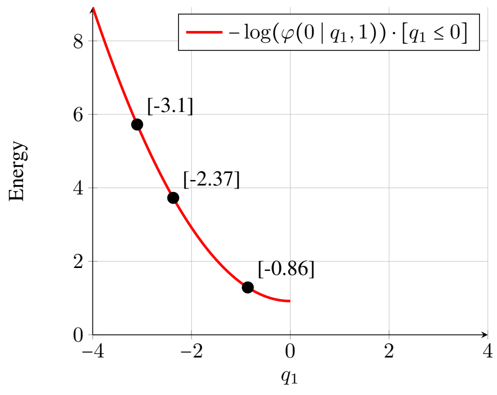
At the third time step, the particle is at the position $[1.15]$, which is no longer on the surface. To search for a suitable state, NP-HMC increments the dimension of the current surface (line) as follows.

First, the surface is extended to $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \leq 0] - \log(\varphi(q_2 \mid q_1, 1)) \cdot [0 < q_1 \leq q_2]$, which is derived from the sum of the respective restrictions of w to $\mathbb{R}$ and to $\mathbb{R}^2$, as depicted in Fig. 3. Since w satisfies the prefix property, the respective supports of these restrictions, namely $\{[q_1, q_2] \in \mathbb{R}^2 \mid q_1 \leq 0\}$ and $\{[q_1, q_2] \in \mathbb{R}^2 \mid 0 < q_1 \leq q_2\}$, are disjoint; and hence the states of the particle on the previous surface $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \leq 0]$ can be reused on the updated surface. Notice that the respective first coordinates of the particle's positions on the surface in Fig. 3 are identical to the particle's positions on the surface (line) in Fig. 2.

Next, the initial state $(\boldsymbol{q} = [-3.1], \boldsymbol{p} = [1.2])$ is extended by appending a randomly chosen value to both the position and momentum components, so that the particle is positioned on the updated surface with an initial momentum. In our example, -1.61 and 3.04 are sampled and the initial state of the particle becomes $(\boldsymbol{q} = [-3.1, -1.61], \boldsymbol{p} = [1.2, 3.04])$ which is located on the surface as shown in Fig. 3. The states at times 1, 2 and 3 are updated accordingly and are given in the table in Fig. 3. Notice that the particle at time 3 is now positioned on the updated surface, and hence Hamiltonian dynamics can resume. The rest of this section is devoted to formalising our algorithm.

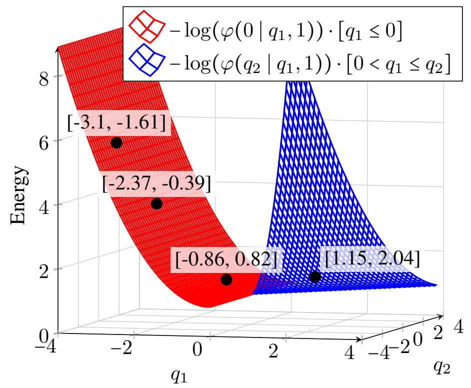
Assumption. Henceforth we assume that the input TR function w satisfies the following:

[3]As in HMC, a *state* of the NP-HMC algorithm is a position-momentum pair $(\boldsymbol{q}, \boldsymbol{p})$ with $|\boldsymbol{q}| = |\boldsymbol{p}|$; but unlike HMC, $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{T}$.



Time	0	1	2	3
$\boldsymbol{q}$	[-3.1]	[-2.37]	[-0.86]	[1.15]
$\boldsymbol{p}$	[1.2]	[3.29]	[3.94]	[5.26]

Figure 2. The Hamiltonian dynamics of a particle on the surface $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \le 0]$ on $\mathbb{R}$.



Time	0	1	2	3
$\boldsymbol{q}$	[-3.1, -1.61]	[-2.37, -0.39]	[-0.86, 0.82]	[1.15, 2.04]
$\boldsymbol{p}$	[1.2, 3.04]	[3.29, 3.04]	[3.94, 3.04]	[5.26, 3.04]

Figure 3. The Hamiltonian dynamics of a particle on the updated surface $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \le 0] - \log(\varphi(q_2 \mid q_1, 1)) \cdot [0 < q_1 \le q_2]$ on $\mathbb{R}^2$.

1 *Integrability*, i.e., the normalising constant $Z < \infty$. (Otherwise, the inference problem would be ill-defined.)

2 The function w is *almost everywhere continuously differentiable* on $\mathbb{T}$. (Since Hamiltonian dynamics exploits the derivative of w in simulating the position of a particle, this ensures that the derivative exists "often enough" for the Hamiltonian integrator to be used.)

3 For *almost* every infinite real-valued sequence $\boldsymbol{q}$, there is a k such that w is positive on $\boldsymbol{q}^{1\dots k}$. (This ensures the extend subroutine in Alg. 3 terminates almost surely.)

Remark 2. (i) NP-HMC is a generalisation of HMC to non-parametric models. Precisely, NP-HMC specialises to standard HMC (with the leapfrog integrator) in case the density function w satisfies $\mathsf{Dom}(w) = \mathbb{R}^n$ for some n, in addition to Assumptions 1, 2 and 3.

(ii) All closed integrable *almost surely terminating* programs of a universal PPL[4] induce densities that satisfy Assumptions 1, 2 and 3. (See App. A and Lem. 12 for an account.)

Truncations The surfaces on which the particle is positioned are derived from a sum of appropriate restrictions of the input TR function w, defined as follows. The n-th ***truncation*** $w_{\le n} : \mathbb{R}^n \to \mathbb{R}_{\ge 0}$ is $\boldsymbol{q} \mapsto \sum_{k=1}^{n} w(\boldsymbol{q}^{1\dots k})$, which returns the cumulative sum of the weight on the prefixes of an n-dimensional trace $\boldsymbol{q}$. Thanks to the prefix property, for each $\boldsymbol{q}$, at most one summand is non-zero. *So any real-valued $w_{\le n}$-supported sequence $\boldsymbol{q} \in \mathbb{R}^n$ has a prefix in the support of w; and any w-supported sequence of length n is also $w_{\le n}$-supported.*

We define a family $U = \{U_n\}_{n\in\mathbb{N}}$ of ***potential energies*** where each $U_n : \mathbb{R}^n \rightharpoonup \mathbb{R}_{\ge 0}$ is a partial function defined as $U_n := -\log w_{\le n}$ with domain $\mathsf{Dom}(U_n) := \mathsf{Supp}(w_{\le n})$. These are the surfaces on which the particle is positioned.

Proposal step The ***nonparametric (NP) integrator*** Ψ_{NP} (Alg. 2) proposes a state by simulating the Hamiltonian motion of a particle at position $\boldsymbol{q} \in \mathbb{R}^n$, with potential energy $U_n(\boldsymbol{q})$ and a randomly chosen momentum $\boldsymbol{p} \in \mathbb{R}^n$.[5] The simulation runs L discrete update steps (also called *leapfrog steps*) of size $\epsilon > 0$, or until the particle leaves the domain of U_n (i.e. the support of $w_{\le n}$). At that moment, the simulation stops and NP-HMC "extends" the state $(\boldsymbol{q}, \boldsymbol{p})$ via the extend subroutine (Alg. 3), until the position of the particle falls into the domain of some potential energy (i.e. support of some higher dimensional truncation).[6] Once the extended position of the particle is settled, simulation resumes. If no extension is necessary, the behaviour is the same as that of the standard HMC leapfrog integrator.

Extend The heart of NP-HMC is the extend subroutine (Alg. 3). Suppose extend is called after i position steps and $i - 1/2$ momentum steps are completed, i.e., at time $t = i\,\epsilon$. If $\boldsymbol{q}$ is in the domain of $U_{|\boldsymbol{q}|}$, extend leaves the state unchanged; otherwise $\boldsymbol{q}$ is not long enough and the while loop extends it as follows.

- Sample a pair (x_0, y_0) of real numbers from the standard normal distribution respectively.
- Trace the motion of a particle with constant potential energy 1 for i position and $i - 1/2$ momentum steps, starting from (x_0, y_0), to obtain (x, y). Notice that

[4]An almost surely terminating program (as defined in App. A) almost never observes a value with zero probability density.

[5]The Hamiltonian motion is almost always defined, by Ass. 2.
[6]This happens almost surely, thanks to Ass. 3.

Algorithm 1 NP-HMC Step

```
Input: current sample $\boldsymbol{q_0}$, density function $w$, step size $\epsilon$, number of steps $L$
Output: next sample $\boldsymbol{q}$
$\boldsymbol{p_0} \sim \mathcal{N}_{|\boldsymbol{q_0}|}$                                    {Initialise}
$U = \{U_n := \lambda \boldsymbol{q}. - \log(w_{\leq n}(\boldsymbol{q}))\}_{n\in\mathbb{N}}$
$((\boldsymbol{q},\boldsymbol{p}),(\boldsymbol{q_0},\boldsymbol{p_0})) = \Psi_{\mathsf{NP}}((\boldsymbol{q_0},\boldsymbol{p_0}),U,\epsilon,L)$
if $\mathcal{U}(0,1) < \min\left(1, \frac{w_{\leq|\boldsymbol{q}|}(\boldsymbol{q})\,\varphi_{2|\boldsymbol{q}|}(\boldsymbol{q}+\boldsymbol{p})}{w_{\leq|\boldsymbol{q}|}(\boldsymbol{q_0})\,\varphi_{2|\boldsymbol{q}|}(\boldsymbol{q_0}+\boldsymbol{p_0})}\right)$ then
    return $\boldsymbol{q}^{1\dots k}$ where $w(\boldsymbol{q}^{1\dots k}) > 0$
else
    return $\boldsymbol{q_0}^{1\dots k}$ where $w(\boldsymbol{q_0}^{1\dots k}) > 0$
end if
```

Algorithm 2 NP Integrator Ψ_{NP}

```
Input: current state $(\boldsymbol{q_0},\boldsymbol{p_0})$, family of potential energies $U = \{U_n\}_{n\in\mathbb{N}}$, step size $\epsilon$, number of steps $L$
Output: new state $(\boldsymbol{q},\boldsymbol{p})$, extended initial state $(\boldsymbol{q}_0,\boldsymbol{p}_0)$
$(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q_0},\boldsymbol{p_0})$                                    {Initialise}
for $i = 0$ to $L$ do
    $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U_{|\boldsymbol{q}|}(\boldsymbol{q})$                    {1/2 momentum step}
    $\boldsymbol{q} = \boldsymbol{q} + \epsilon\boldsymbol{p}$                              {1 position step}
    $((\boldsymbol{q},\boldsymbol{p}),(\boldsymbol{q_0},\boldsymbol{p_0})) = \mathsf{extend}((\boldsymbol{q},\boldsymbol{p}),(\boldsymbol{q_0},\boldsymbol{p_0}), i\,\epsilon, U)$
    $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U_{|\boldsymbol{q}|}(\boldsymbol{q})$                    {1/2 momentum step}
end for
return $((\boldsymbol{q},\boldsymbol{p}),(\boldsymbol{q_0},\boldsymbol{p_0}))$
```

Algorithm 3 extend

```
Input: current state $(\boldsymbol{q},\boldsymbol{p})$, initial state $(\boldsymbol{q_0},\boldsymbol{p_0})$, time $t$, family of potential energies $U = \{U_n\}_{n\in\mathbb{N}}$
Output: extended state $(\boldsymbol{q},\boldsymbol{p})$, extended initial state $(\boldsymbol{q}_0,\boldsymbol{p}_0)$
while $\boldsymbol{q} \notin \mathsf{Dom}(U_{|\boldsymbol{q}|})$ do
    $x_0 \sim \mathcal{N}_1; y_0 \sim \mathcal{N}_1$                       {sample from normal}
    $(x, y) = (x_0 + t\,y_0, y_0)$                          {run for time t}
    $(\boldsymbol{q_0},\boldsymbol{p_0}) = (\boldsymbol{q_0} + [x_0], \boldsymbol{p_0} + [y_0])$       {update initial}
    $(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q} + [x], \boldsymbol{p} + [y])$            {update current}
end while
return $((\boldsymbol{q},\boldsymbol{p}),(\boldsymbol{q_0},\boldsymbol{p_0}))$
```

Figure 4. Pseudocode for Nonparametric Hamiltonian Monte Carlo

the momentum update is simply the identity, hence we only need to consider i position updates which takes x to $x_0 + t\,y_0$.

- Append (x_0, y_0) to the initial state $(\boldsymbol{q_0},\boldsymbol{p_0})$ and (x, y) to the current state $(\boldsymbol{q},\boldsymbol{p})$.

Thus the length of the position $\boldsymbol{q}$ is incremented, and by Assumption 3, this loop terminates almost surely at a position $\boldsymbol{q}$ in the domain of the potential energy of dimension $|\boldsymbol{q}|$.

Putting them together A single NP-HMC iteration, as shown in Alg. 1, produces a proposed sample $\boldsymbol{q}$ from the current sample $\boldsymbol{q_0}$, by applying the NP integrator Ψ_{NP} to the state $(\boldsymbol{q_0},\boldsymbol{p_0})$ with a freshly sampled momentum $\boldsymbol{p_0}$. The proposed state $(\boldsymbol{q},\boldsymbol{p})$ is then accepted with probability given by the Hastings acceptance ratio.

Remark 3. The prefix property of the input TR function w plays an important role in the NP-HMC inference algorithm. If Alg. 1 returns $\boldsymbol{q}$ as the next sample, then for any extension $\boldsymbol{q}'$ of $\boldsymbol{q}$, we have $w(\boldsymbol{q}') = 0$, and so, all such q' are irrelevant for inference (because $U_{|\boldsymbol{q}'|}(\boldsymbol{q}') = U_{|\boldsymbol{q}|}(\boldsymbol{q})$). If the prefix property weren't satisfied, the algorithm would fail to account for the weight on such $\boldsymbol{q}'$.

Other extensions and efficiency considerations We have presented a version of NP-HMC that eschews runtime efficiency in favour of clarity. An advantage of such a presentation (in deliberately purified form) is that it becomes easy to see that the same method is just as applicable to such HMC variants as reflective/refractive HMC (Afshar & Domke, 2015) and discontinuous HMC (Nishimura et al., 2020); we call the respective extensions NP-RHMC and NP-DHMC. For details, see App. B.2.

Several efficiency improvements to NP-HMC are possible. If the density function is given by a probabilistic program, one can interleave its execution with extend, by gradually extending $\boldsymbol{q}$ at every encountered **`sample`** statement. Similarly, the truncations $w_{\leq n}$ don't require an expensive summation to compute. For details, see App. B.3.

Our implementation (Sec. 5) also improves the extend function (Alg. 3): it not only extends a trace $\boldsymbol{q}$ if necessary, it also trims it to the unique prefix $\boldsymbol{q}'$ of $\boldsymbol{q}$ with positive $w(\boldsymbol{q}')$. This version works better in our experiments. For details, see App. B.3.

4. Correctness

The NP-HMC algorithm is correct in the sense that the generated Markov chain converges to the target distribution $\nu : A \mapsto \frac{1}{Z}\int_A w \,\mathrm{d}\mu_{\mathbb{T}}$ with normalising constant Z. We present an outline of our proof here. The full proof can be found in App. C.

Invariant distribution By iterating Alg. 1, the NP-HMC algorithm generates a Markov chain $\{\boldsymbol{q}^{(i)}\}_{i\in\mathbb{N}}$ on the target space $\mathbb{T}$. The first step to correctness is to show that the invariant distribution of this chain is indeed the target distribution.

This proof is non-trivial, since the length of the generated sample depends on the values of the random samples drawn in the extend subroutine (Alg. 3). To work around this, we define an auxiliary algorithm, which induces the same

Markov chain as NP-HMC, but does not increase the dimension dynamically. Instead, it finds the smallest N such that all intermediate positions in the L leapfrog steps stay in the domain of U_N, and performs leapfrog steps as in standard HMC. In this algorithm, all stochastic primitives are executed outside of the Hamiltonian dynamics simulation, and the simulation has a fixed dimension. Hence we can proceed to identify the invariant distribution. We then show (in Lem. 28 and Thm. 4) that the Markov chain generated by this auxiliary algorithm has the target distribution ν as its invariant distribution, and hence the same holds for NP-HMC (Alg. 1).

Theorem 4. *Given Assumptions 1, 2 and 3, the target distribution ν is the invariant distribution of the Markov chain generated by iterating Alg. 1.*

Convergence In App. C.4, we extend the proof of Cances et al. (2007) to show that the chain converges for a small enough step size ϵ, as long as the following additional assumptions are met:

(C1) w is continuously differentiable on a non-null set A with measure-zero boundary.

(C2) $w|_{\mathsf{Supp}(w)}$ is bounded below by a positive constant.

(C3) For each n, the function $\frac{\nabla w_{\leq n}}{w_{\leq n}}$ is uniformly bounded from above and below on $\mathsf{Supp}(w_{\leq n}) \cap A$.

(C4) For each n, the function $\frac{\nabla w_{\leq n}}{w_{\leq n}}$ is Lipschitz continuous on $\mathsf{Supp}(w_{\leq n}) \cap A$.

Theorem 5. *If Assumptions (C1)–(C4) are satisfied in addition to Assumptions 1, 2 and 3, the Markov chain generated by iterating Alg. 1 converges to the target distribution ν.*

5. Experiments

We implemented the NP-HMC algorithm and its variants (NP-RHMC and NP-DHMC) in Python, using PyTorch (Paszke et al., 2019) for automatic differentiation. We implemented it from scratch rather than in an existing system because NP-DHMC needs additional information about each sample (does the density function depend discontinuously on it?), so it requires a deeper integration in the probabilistic programming system. In our empirical evaluation, we focus on the NP-DHMC algorithm because it inherits discontinuous HMC's efficient handling of discontinuities: contrary to NP-RHMC, it does not need to compute the intersections of the particle's trajectory with the regions of discontinuity. Our implementation also uses the efficiency improvements discussed in App. B.3. The code for our implementation and experiments is available at `https://github.com/fzaiser/nonparametric-hmc` and archived as (Zaiser & Mak, 2021).

We compare our implementation with Anglican's (Wood et al., 2014) inference algorithms that are applicable out-of-the-box to nonparametric models: lightweight Metropolis-Hastings (LMH), particle Gibbs (PGibbs) and random-walk lightweight Metropolis-Hastings (RMH).[7] NP-DHMC performs more computation per sample than its competitors because it evaluates the density function in each of the L leapfrog steps, not just once like the other inference algorithms. To equalise the computation budgets, we generate L times as many samples for each competitor algorithm, and apply thinning (taking every L-th sample) to get a comparable sample size.

Table 1. Total variation distance from the ground truth for the geometric distribution, averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 5 leapfrog steps of size 0.1; and 5×10^3 LMH, PGibbs and RMH samples.

method	**ours**	LMH	PGibbs	RMH
TVD	**0.0136**	0.0224	0.0158	0.0196

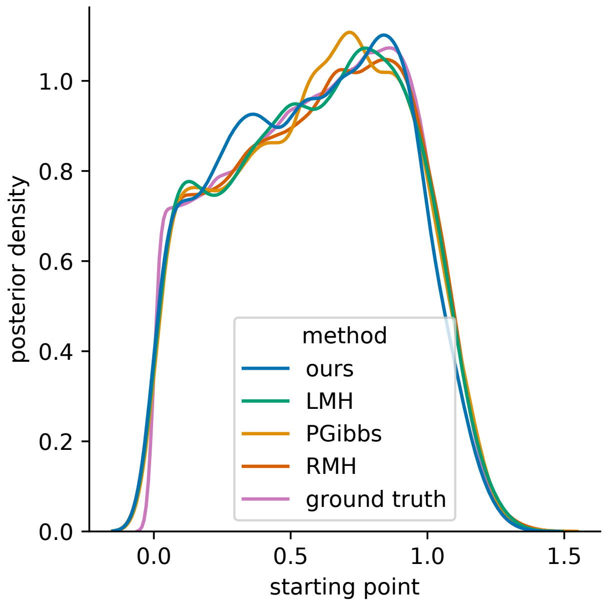
Geometric distribution A classic example to illustrate recursion in a universal PPL is sampling from a geometric distribution with parameter p by repeatedly flipping a biased coin with probability p (see e.g. Ch. 5 in (van de Meent et al., 2018)). The pseudocode for it is:

```
def geometric():
  if sample(uniform(0, 1)) < p: return 1
  else: return 1 + geometric()
```

We tested our algorithm on this problem with $p = 0.2$. Our implementation works well on this example and has no trouble jumping between traces of different length. To quantify this, we computed the total variation distance to the ground truth for each approach and report it in Table 1. The result is perhaps surprising given that the odds are "stacked against" NP-DHMC in this model: it is a discrete model, so there is no gradient information, and there are no observations (only sampling from the prior). These properties should favour the competitors, making the performance of NP-DHMC rather remarkable.

Random walk To better evaluate our algorithm on probabilistic programs with unbounded loops (such as the example from Sec. 2), we considered the one-sided random walk on $\mathbb{R}_{\geq 0}$ described in (Mak et al., 2021): A pedestrian starts from a random point in $[0, 3]$ and walks a uniformly random distance of at most 1 in either direction, until they pass 0. Given a (noisily) measured total distance of 1.1 travelled, what is the posterior distribution of the starting point? As this process has infinite expected running time, we need a stopping

[7]We also compared Interacting Particle MCMC (IPMCMC), but it performed consistently worse than PGibbs in our experiments and hence is omitted.



method	**ours**	LMH	PGibbs	RMH
ESS	**679**	526	310	508

Figure 5. Kernel density estimate (top) averaged over 10 runs and estimated effective sample size (bottom) averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 50 leapfrog steps of size 0.1; and 5×10^4 LMH, PGibbs and RMH samples.

condition if the pedestrian is too far away from zero, (`distance < 10`), as shown in the following pseudocode:

```
start = sample(uniform(0,3))
position = start; distance = 0
while position > 0 and distance < 10:
  step = sample(uniform(-1, 1))
  position += step; distance += abs(step)
observe(distance, normal(1.1, 0.1))
return start
```

This example is interesting and challenging because the true posterior is difficult to determine precisely. Therefore we took 10^7 importance samples (effective sample size $\approx 4.4 \times 10^5$) and considered those as the ground truth. As one can see from Fig. 5, NP-DHMC comes closest. Since it is not clear what measure to use for the distance from these "ground truth" samples, we instead computed the effective sample size[8] for each method (Fig. 5). Our method does best in that regard as well.

The popular PPL Pyro accepts nonparametric models as input. We therefore tried to ascertain the performance of its HMC implementation on this example. We ran both Pyro's HMC sampler (with the same hyperparameters as ours) and Pyro's No-U-Turn sampler (NUTS) (Hoffman & Gelman, 2014), which aims to automatically infer good hyperparameter settings. The inferred posterior distributions (Fig. 6) are far away from the ground truth, and clearly wrong. Pyro's inference was very slow, sometimes taking almost a minute to produce a single sample. For this reason, we didn't run more experiments with it.

[8]We used the standard ESS estimator (based on weighted samples) for the ground-truth importance samples, and an autocorrelation-based MCMC ESS estimator (Sec. 11.5 in (Gelman et al., 2014)) for the rest.

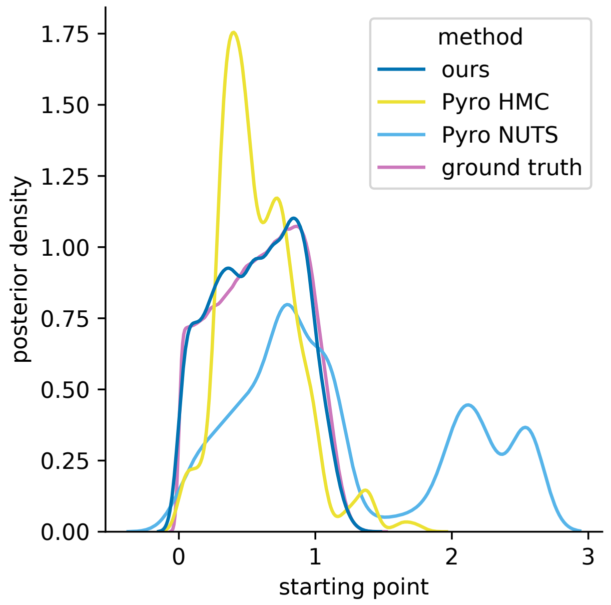


Figure 6. Kernel density estimate for the random walk example compared to Pyro averaged over 10 runs, some of which were discarded because of low acceptance rate (< 0.1). Each run: 10^3 samples with 10^2 burn-in, 50 leapfrog steps of size 0.1.

Gaussian mixture model We also considered the following mixture model adapted from (Zhou et al., 2020), where the number of mixture components K is unbounded.

$$\begin{aligned} K &\sim \text{Poisson}(10) + 1 \\ \mu_k &\sim \text{Uniform}([0,100]^3) && \text{for } k = 1, \ldots, K \\ x_n &\sim \frac{1}{K}\sum_{k=1}^{K} \mathcal{N}_3(\mu_k, 10^2 I_3) && \text{for } n = 1, \ldots, N \end{aligned}$$

This model samples parameters $\theta = (K \in \mathbb{N}, \mu_k \in [0,100]^3)$ and N data points $X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^3$. Note that this model uses much higher standard deviations (10 instead of 0.1) for the Gaussian mixture components compared to (Zhou et al., 2020). This is to avoid typical problems of MCMC algorithms with multimodal distributions, which is an issue inherent to MCMC algorithms and tangential to this work. We used this model to generate $N = 200$ training data points for a fixed $\theta^* = (K^* = 9, \mu^*_{1\ldots K^*})$. We let our inference algorithms sample from $p(\theta \mid X)$ and compared the posterior on the number of mixture components K with the other inference algorithms. The histogram (Fig. 7) shows

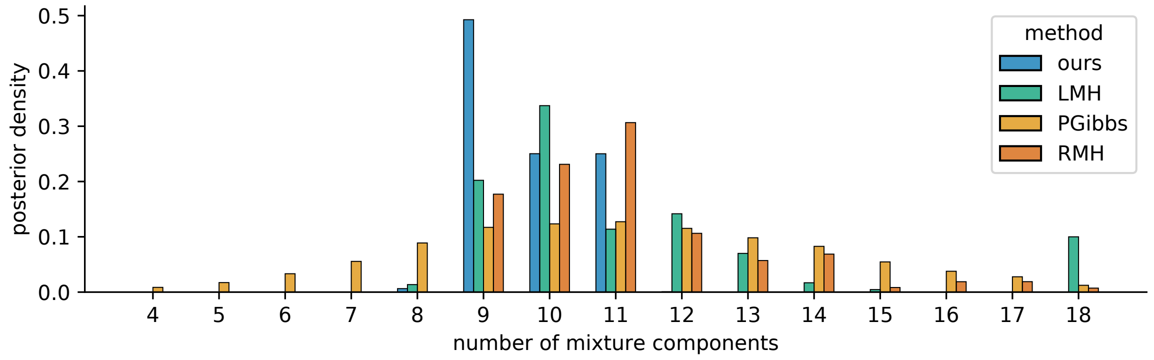


Figure 7. Histogram of the no. of mixtures for the GMM example; correct posterior = 9, averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 50 leapfrog steps of size 0.05; and 5×10^4 LMH, PGibbs and RMH samples.

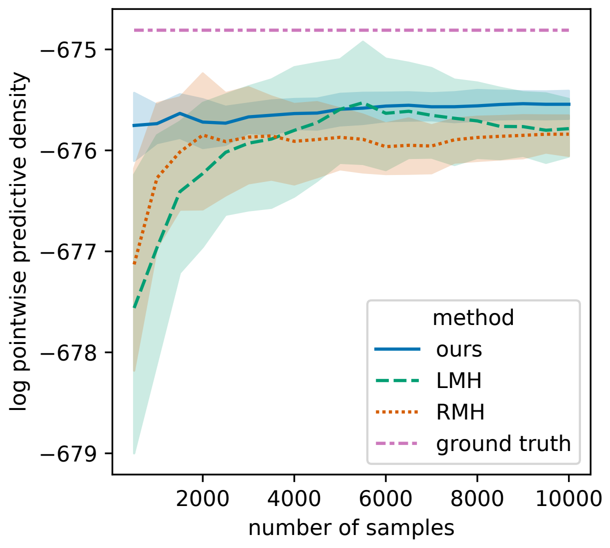


Figure 8. LPPD for the GMM example, averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 50 leapfrog steps of size 0.05; and 5×10^4 LMH, PGibbs and RMH samples. The shaded area is one standard deviation. PGibbs (with final LPPD -716.85 ± 0.64) is omitted to show the top contenders clearly.

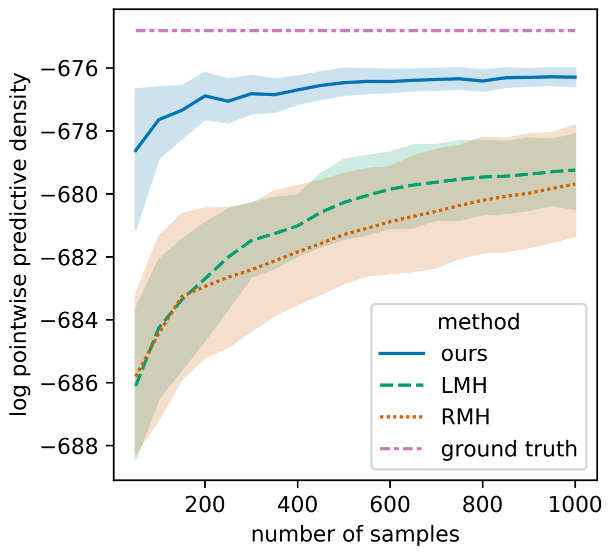


Figure 9. LPPD for the DPMM example, averaged over 10 runs. Each run: 10^2 NP-DHMC samples with 50 burn-in, 20 leapfrog steps of size 0.05; and 2×10^3 LMH, PGibbs and RMH samples. The shaded area is one standard deviation. PGibbs (with final LPPD -725.96 ± 9.83) is omitted to show the top contenders clearly.

that NP-DHMC usually finds the correct number of mixture components ($K^* = 9$).

In addition, we computed the log pointwise predictive density (LPPD) for a test set with $N' = 50$ data points $Y = \{y_1, \dots, y_{N'}\}$, generated from the same θ^* as the training data. The LPPD is defined as $\sum_{i=1}^{N'} \log \int p(y_i \mid \theta) p(\theta \mid X) \mathrm{d}\theta$ and can be approximated by $\sum_{i=1}^{N'} \log \frac{1}{M} \sum_{j=1}^{M} p(y_i \mid \theta_j)$ where $(\theta_j)_{j=1\dots M}$ are samples from $p(\theta \mid X)$ (Gelman et al., 2014). The results (Fig. 8) include the "true" LPPD of the test data under the point estimate θ^*. As we can see, NP-DHMC outperforms the other methods and has the lowest variance over multiple runs.

Dirichlet process mixture model Finally, we consider a classic example of nonparametric models: the Dirichlet process $\mathrm{DP}(\alpha, H)$ (Ferguson, 1973), which is a stochastic process parametrised by concentration parameter $\alpha > 0$ and a probability distribution H. For practical purposes, one can think of samples from $\mathrm{DP}(\alpha, H)$ as an infinite sequence $(w_k, h_k)_{n\in\mathbb{N}}$ where w_n are weights that sum to 1 and h_n are samples from H. Conceptually, a DP Gaussian mixture model takes the form

$$\begin{aligned}(w_k, h_k)_{k\in\mathbb{N}} &\sim \mathrm{DP}(\alpha, H) \\ x_n &\sim \sum_{k=1}^{\infty} w_k \cdot \mathcal{N}(h_k, \Sigma) \quad \text{for } n = 1, \dots, N\end{aligned}$$

Sampling from a DP is usually implemented by the stick-breaking method (Sethuraman, 1994). However, one cannot actually compute an infinite sequence. A practical workaround is to cap the number of mixture components by a fixed K and only consider $(w_k, h_k)_{k=1,\ldots,K}$.[9] This renders the model parametric, enabling the use of standard HMC. However such a treatment is clearly unsatisfactory: if the data actually requires more mixture components than K, the model would be found wanting.

We propose a different approach. Instead of choosing a fixed K, we allow it to depend on the weights w_k: we pick the minimal $K \in \mathbb{N}$ such that $\sum_{i=1}^{K} w_i > 1 - \epsilon$ for some $\epsilon > 0$. With this restriction, we only discard insignificant mixture components (with a weight $w_k < \epsilon$) and allow as many mixture components as necessary to model the data accurately. This model is not parametric anymore, but still tractable by our algorithm.

We implemented the above model with the parameters $\alpha = 5$, $\epsilon = 0.01$, and the remaining parameters as chosen in the previous GMM example, i.e. $H = \text{Uniform}([0, 100]^3)$ and $\Sigma = 10^2 I_3$. We used the same training and test data as in the previous GMM example and present the LPPD results in Fig. 9. As we can see, NP-DHMC outperforms the other methods and has the lowest variance over multiple runs.

6. Related Work and Conclusion

The standard MCMC algorithm for PPLs that is widely implemented (for example, in Anglican, Venture, and Web PPL) is the Lightweight Metropolis-Hastings (LMH) algorithm and its extensions (Yang et al., 2014; Tolpin et al., 2015; Ritchie et al., 2016), which performs single-site updates on the current sample and re-executes the program. Unlike NP-HMC, where Hamiltonian motion is simulated on the resulting extended trace, LMH suffers from a lack of predictive accuracy in its proposal (as shown in Sec. 5).

The Reversible Jump Markov chain Monte Carlo (RJMCMC) algorithm (Green, 1995) is similar to NP-HMC in that it is a trans-dimensional MCMC sampler. However, NP-HMC is a *general purpose* inference algorithm that works out-of-the-box when given an input density function, whereas RJMCMC additionally requires the user to specify a transition kernel. Various RJMCMC transition kernels have been suggested for specific models, e.g. split-merge proposal for infinite Gaussian mixture models.

Some PPLs such as Hakaru, Pyro and Gen give users the flexibility to hand-code the proposal in a MCMC setting. For instance, Cusumano-Towner et al. (2020) implement the split-merge proposal (Richardson & Green, 1997) of RJMCMC in Gen. Though this line of research is orthogonal to ours, PPLs such as Gen could play a useful role in the implementation of NP-HMC and similar extensions of inference algorithms to nonparametric models.

The HMC algorithm and its variants, notably the No-U-Turn Sampler, are the workhorse inference methods in the influential PPL Stan (Gelman et al., 2015). The challenges posed by stochastic branching in PPLs are the focus of reflective/refractive HMC (Afshar & Domke, 2015); discontinuous HMC (Nishimura et al., 2020); mixed HMC (Zhou, 2020); and the first-order PPL in (Zhou et al., 2019) which is equipped with an implementation of discontinuous HMC. By contrast, our work is an attempt to tackle the language constructs of branching *and* recursion.

Unlike Monte Carlo methods, variational inference (VI) (Blei et al., 2017) solves the Bayesian inference problem by treating it as an optimisation problem. When adapted to models expressed as probabilistic programs, the score function VI (Ranganath et al., 2014) can in principle be applied to a large class of branching and recursive programs because only the variational density functions need to be differentiable. Existing implementations of VI algorithms in probabilistic programming systems are however far from automatic: in the main, the guide programs (that express variational distributions) still need to be hand-coded.

Recently, Zhou et al. (2020) introduced the Divide, Conquer, and Combine (DCC) algorithm, which is applicable to programs definable using branching and recursion. As a hybrid algorithm, DCC solves the problem of designing a proposal that can efficiently transition between configurations by performing local inferences on submodels, and returning an appropriately weighted combination of the respective samples. Thanks to a judicious resource allocation scheme, it exhibits strong performance on multimodal distributions.

Conclusion

We have presented the NP-HMC algorithm, the first extension of the HMC algorithm to nonparametric models. We have proved that NP-HMC is correct. We have also empirically demonstrated that it enjoys significant performance improvements over state-of-the-art MCMC algorithms for universal PPLs on four nonparametric models, thereby illustrating that the key advantage of HMC—the proposal of moves to distant states with a high acceptance probability—has been preserved by NP-HMC.

Acknowledgements We thank the reviewers for their insightful feedback and pointing out important related work. We are grateful to Hugo Paquet, Dominik Wagner and Yuan Zhou, who gave detailed comments on an early draft, and to Tom Rainforth and Arnaud Doucet for their helpful comments and advice. We gratefully acknowledge support from the EPSRC and the Croucher Foundation.

[9]For instance, see `https://pyro.ai/examples/dirichlet_process_mixture.html` (accessed: 2021-06-06), the Pyro tutorial on DP mixture models.

References

Afshar, H. M. and Domke, J. Reflection, refraction, and Hamiltonian Monte Carlo. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS 2015)*, volume 28, pp. 3007–3015. Curran Associates, Inc., 2015.

Betancourt, M. A conceptual introduction to hamiltonian monte carlo. arXiv, 2018. URL https://arxiv.org/abs/1701.02434.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Borgström, J., Dal Lago, U., Gordon, A. D., and Szymczak, M. A lambda-calculus foundation for universal probabilistic programming. In Garrigue, J., Keller, G., and Sumii, E. (eds.), *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP 2016)*, pp. 33–46. Association for Computing Machinery, 2016.

Bou-Rabee, N. and Sanz-Serna, J. M. Geometric integrators and the Hamiltonian Monte Carlo method. *Acta Numerica*, 27:113–206, 2018.

Cances, E., Legoll, F., and Stoltz, G. Theoretical and numerical comparison of some sampling methods for molecular dynamics. *ESAIM: Mathematical Modelling and Numerical Analysis*, 41(2):351–389, 2007.

Culpepper, R. and Cobb, A. Contextual Equivalence for probabilistic programs with continuous Random Variables and scoring. In Yang, H. (ed.), *Proceedings of the 26th European Symposium on Programming (ESOP 2017)*, volume 10201 of *Lecture Notes in Computer Science*, pp. 368–392. Springer, 2017.

Cusumano-Towner, M., Lew, A. K., and Mansinghka, V. K. Automating involutive MCMC using probabilistic and differentiable programming. arXiv, 2020. URL https://arxiv.org/abs/2007.09871.

Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. Gen: A general-purpose probabilistic programming system with programmable inference. In McKinley, K. S. and Fisher, K. (eds.), *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*, pp. 221–236. Association for Computing Machinery, 2019.

Duane, S., Kennedy, A., Pendleton, B. J., and Roweth, D. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

Ferguson, T. S. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209 – 230, 1973.

Ge, H., Xu, K., and Ghahramani, Z. Turing: Composable inference for probabilistic programming. In Storkey, A. J. and Pérez-Cruz, F. (eds.), *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*, volume 84, pp. 1682–1690. PMLR, 2018.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. *Bayesian data analysis*. Texts in Statistical Science Series. CRC Press, 3rd edition, 2014.

Gelman, A., Lee, D., and Guo, J. Stan: A probabilistic programming language for Bayesian inference and optimization. *Journal of Educational and Behavioral Statistics*, 40(5):530–543, 2015.

Goodman, N. D. and Stuhlmüller, A. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2021-5-24.

Goodman, N. D., Mansinghka, V. K., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. Church: A language for generative models. In McAllester, D. A. and Myllymäki, P. (eds.), *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, pp. 220–229. AUAI Press, 2008.

Green, P. J. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Hoffman, M. D. and Gelman, A. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1): 1593–1623, 2014.

Hur, C., Nori, A. V., Rajamani, S. K., and Samuel, S. A provably correct sampler for probabilistic programs. In Harsha, P. and Ramalingam, G. (eds.), *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2015)*, volume 45 of *LIPIcs*, pp. 475–488. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

Kudlicka, J., Murray, L. M., Ronquist, F., and Schön, T. B. Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling. In Globerson, A. and Silva, R. (eds.), *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, (UAI 2019*, volume 115, pp. 679–689. AUAI Press, 2019.

Mak, C., Ong, C. L., Paquet, H., and Wagner, D. Densities of almost surely Terminating probabilistic programs are differentiable almost Everywhere. In Yoshida, N. (ed.), *Proceedings of the 30th European Symposium on Programming (ESOP 2021)*, volume 12648 of *Lecture Notes in Computer Science*, pp. 432–461. Springer, 2021.

Manning, C. and Schütze, H. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

Mansinghka, V. K., Selsam, D., and Perov, Y. N. Venture: a higher-order probabilistic programming platform with programmable inference. arXiv, 2014. URL `http://arxiv.org/abs/1404.0099`.

Murray, L. M., Lundén, D., Kudlicka, J., Broman, D., and Schön, T. B. Delayed sampling and automatic Rao-blackwellization of probabilistic programs. In Storkey, A. J. and Pérez-Cruz, F. (eds.), *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*, volume 84, pp. 1037–1046. PMLR, 2018.

Narayanan, P. and Shan, C.-c. Symbolic disintegration with a variety of base measures. *ACM Transactions on Programming Languages and Systems*, 42(2):1–60, 2020.

Neal, R. M. MCMC using Hamiltonian dynamics. In Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (eds.), *Handbook of Markov Chain Monte Carlo*, chapter 5, pp. 113–162. Chapman & Hall CRC Press, 2011.

Nishimura, A., Dunson, D. B., and Lu, J. Discontinuous Hamiltonian Monte Carlo for discrete parameters and discontinuous likelihoods. *Biometrika*, 107(2):365–380, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32, pp. 8024–8035. Curran Associates, Inc., 2019.

Ranganath, R., Gerrish, S., and Blei, D. M. Black box variational inference. In Kaski, S. and Corander, J. (eds.), *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33, pp. 814–822. PMLR, 2014.

Ratner, B. Variable selection methods in regression: Ignorable problem, outing notable solution. *Journal of Targeting, Measurement and Analysis for Marketing*, 18: 65–75, 2010.

Richardson, S. and Green, P. J. On Bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(4):731–792, 1997.

Ritchie, D., Stuhlmüller, A., and Goodman, N. D. C3: lightweight incrementalized MCMC for probabilistic programs using continuations and callsite caching. In Gretton, A. and Robert, C. C. (eds.), *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS 2016)*, volume 51, pp. 28–37. PMLR, 2016.

Ronquist, F., Kudlicka, J., Senderov, V., Borgström, J., Lartillot, N., Lundén, D., Murray, L., Schön, T. B., and Broman, D. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *Communications Biology*, 4(244), 2021.

Schütte, C. Conformational dynamics: Modelling, theory, algorithm, and application to biomolecules. Technical report, Freie Universität Berlin, 1999.

Scott, D. S. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1&2):411–440, 1993.

Sethuraman, J. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4(2):639–650, 1994.

Sieber, K. Relating full abstraction Results for different programming languages. In Nori, K. V. and Madhavan, C. E. V. (eds.), *Proceedings of the 10th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1990)*, volume 472 of *Lecture Notes in Computer Science*, pp. 373–387. Springer, 1990.

Tierney, L. Markov Chains for exploring posterior distributions. *The Annals of Statistics*, 22(4):1701–1728, 1994.

Tolpin, D., van de Meent, J., Paige, B., and Wood, F. D. Output-sensitive adaptive Metropolis-Hastings for probabilistic programs. In Appice, A., Rodrigues, P. P., Costa, V. S., Gama, J., Jorge, A., and Carlos Soares (eds.), *Proceedings of Machine Learning and Knowledge Discovery in Databases - European Conference Part II (ECML PKDD 2015)*, volume 9285 of *Lecture Notes in Computer Science*, pp. 311–326. Springer, 2015.

Vákár, M., Kammar, O., and Staton, S. A domain theory for statistical probabilistic programming. *Proceedings of the ACM on Programming Languages*, 3(36):1–29, 2019.

van de Meent, J., Paige, B., Yang, H., and Wood, F. An introduction to probabilistic programming. arXiv, 2018. URL `http://arxiv.org/abs/1809.10756`.

Verlet, L. Computer “experiments” on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, 159(1):98–103, 1967.

Wingate, D., Stuhlmüller, A., and Goodman, N. D. Lightweight implementations of probabilistic programming languages via transformational compilation. In Gordon, G. J., Dunson, D. B., and Dudík, M. (eds.), *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15, pp. 770–778. PMLR, 2011.

Wood, F. D., van de Meent, J., and Mansinghka, V. A new approach to probabilistic programming inference. In Kaski, S. and Corander, J. (eds.), *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33, pp. 1024–1032. PMLR, 2014.

Yang, L., Hanrahan, P., and Goodman, N. D. Generating efficient MCMC kernels from probabilistic programs. In Kaski, S. and Corander, J. (eds.), *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33, pp. 1068–1076. PMLR, 2014.

Zaiser, F. and Mak, C. Artifact for ”Nonparametric Hamiltonian Monte Carlo” (ICML 2021), June 2021. URL https://doi.org/10.5281/zenodo.4897900.

Zhou, G. Mixed Hamiltonian Monte Carlo for mixed discrete and continuous variables. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pp. 17094–17104. Curran Associates, Inc., 2020.

Zhou, Y., Gram-Hansen, B. J., Kohn, T., Rainforth, T., Yang, H., and Wood, F. LF-PPL: a low-level first order probabilistic programming language for non-differentiable models. In Chaudhuri, K. and Sugiyama, M. (eds.), *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS 2019)*, volume 89, pp. 148–157. PMLR, 2019.

Zhou, Y., Yang, H., Teh, Y. W., and Rainforth, T. Divide, conquer, and combine: a new inference strategy for probabilistic programs with stochastic support. In *Proceedings of the 37th International Conference on Machine Learning, (ICML 2020)*, volume 119, pp. 11534–11545. PMLR, 2020.

Nonparametric Hamiltonian Monte Carlo (Appendix)

A. Statistical PCF

In this section, we present a simply-typed statistical probabilistic programming language with (stochastic) branching and recursion, and its operational semantics.

This language serves two purposes for the NP-HMC algorithm. First, it is a purified universal probabilistic programming language (PPL) widely considered (Borgström et al., 2016; Vákár et al., 2019; Mak et al., 2021) which specifies tree-representable functions that satisfies Ass. 1, 2 and 3 (Prop. 7 and Lem. 12) and hence NP-HMC can be applied. Second, its (operational) semantics is used to prove correctness of NP-HMC in App. C.

A.1. Syntax

SPCF is a simply-typed higher-order universal PPL with branching and recursion. More formally, it is a statistical probabilistic version of call-by-value PCF (Scott, 1993; Sieber, 1990) with reals as the ground type. The terms and part of the typing system of SPCF are presented in Fig. 10. Free variables and closed terms are defined in the usual way. In the interest of readability, we sometimes use pseudocode (e.g. Listing 1) in the style of Python to express SPCF terms.

There are two probabilistic constructs of SPCF: the sampling construct normal draws from $\mathcal{N}$, the standard Gaussian distribution with mean 0 and variance 1; the scoring construct $\mathsf{score}(M)$ enables conditioning on observed data by multiplying the weight of the current execution with the real number denoted by M. Note this is not limiting as the standard uniform distribution with endpoints 0 and 1 can be described as $\underline{\mathsf{cdfNormal}}(\mathsf{normal})$ where $\mathsf{cdfNormal}$ is the cumulative distribution function (cdf) of the standard normal distribution. And any real-valued distribution with inverse cdf f can be described as $\underline{f}(\underline{\mathsf{cdfNormal}}(\mathsf{normal}))$.

Remark 6. The main difference between our variant of SPCF and the others (Vákár et al., 2019; Mak et al., 2021) is that our sampling construct draws from the *standard normal distribution* instead of the standard uniform distribution. This does not restrict nor extend our language and is only considered since the target (parameter) space of the standard HMC algorithm matches that of the support of a standard n-dimensional normal distribution.

Types (typically denoted σ, τ) and ***terms*** (typically M, N, L):

$$\sigma, \tau ::= \mathsf{R} \mid \sigma \Rightarrow \tau$$
$$M, N, L ::= y \mid \underline{r} \mid \lambda y.M \mid M\,N \mid \mathsf{if}\big(L \leq 0, M, N\big) \mid \underline{f}(M_1, \ldots, M_\ell) \mid \mathsf{Y}M \mid \mathsf{normal} \mid \mathsf{score}(M)$$

Typing system:

$$\frac{}{\Gamma \vdash \mathsf{normal} : \mathsf{R}} \qquad \frac{\Gamma \vdash M : \mathsf{R}}{\Gamma \vdash \mathsf{score}(M) : \mathsf{R}} \qquad \frac{\Gamma \vdash M : (\sigma \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau)}{\Gamma \vdash \mathsf{Y}M : \sigma \Rightarrow \tau}$$

Figure 10. Syntax of SPCF, where $r \in \mathbb{R}$, x, y are variables, and $f : \mathbb{R}^n \to \mathbb{R}$ ranges over a set $\mathcal{F}$ of partial, measurable primitive functions.

A.2. Operational Semantics

The small-step reduction of SPCF is standard (see Borgström et al. (2016)). We present it as a rewrite system of ***configurations***, which are triples of the form $\langle M, w, \boldsymbol{t}\rangle$ where M is a closed SPCF term, $w \in \mathbb{R}_{\geq 0}$ is a ***weight***, and $\boldsymbol{t} \in \mathbb{T}$ a trace, as defined in Fig. 11.

Values (typically denoted V), ***redexes*** (typically R) and ***evaluation contexts*** (typically E):

$$\begin{aligned}
V &::= \underline{r} \mid \lambda y.M\\
R &::= (\lambda y.\,M)\,V \mid \mathsf{if}\big(\underline{r} \le 0, M, N\big) \mid \underline{f}(\underline{r_1},\ldots,\underline{r_\ell}) \mid \mathsf{Y}(\lambda y.\,M) \mid \mathsf{normal} \mid \mathsf{score}(\underline{r})\\
E &::= [\,] \mid E\,M \mid (\lambda y.M)\,E \mid \mathsf{if}\big(E \le 0, M, N\big) \mid \underline{f}(\underline{r_1},\ldots,\underline{r_{i-1}}, E, M_{i+1},\ldots,M_\ell) \mid \mathsf{Y}E \mid \mathsf{score}(E)
\end{aligned}$$

Redex contractions:

$$\begin{aligned}
\langle(\lambda y.M)\,V, w, \boldsymbol{t}\rangle &\longrightarrow \langle M[V/y], w, \boldsymbol{t}\rangle\\
\big\langle \underline{f}(\underline{r_1},\ldots,\underline{r_\ell}), w, \boldsymbol{t}\big\rangle &\longrightarrow \begin{cases}\big\langle \underline{f(r_1,\ldots,r_\ell)}, w, \boldsymbol{t}\big\rangle & \text{if } (r_1,\ldots,r_\ell) \in \mathsf{Dom}(f),\\ \mathsf{fail} & \text{otherwise.}\end{cases}\\
\langle \mathsf{Y}(\lambda y.M), w, \boldsymbol{t}\rangle &\longrightarrow \langle \lambda z.M[\mathsf{Y}(\lambda y.M)/y]\,z, w, \boldsymbol{t}\rangle && \text{(for fresh variable } z)\\
\big\langle \mathsf{if}\big(\underline{r} \le 0, M, N\big), w, \boldsymbol{t}\big\rangle &\longrightarrow \begin{cases}\langle M, w, \boldsymbol{t}\rangle & \text{if } r \le 0,\\ \langle N, w, \boldsymbol{t}\rangle & \text{otherwise.}\end{cases}\\
\langle \mathsf{normal}, w, \boldsymbol{t}\rangle &\longrightarrow \langle \underline{r}, w, \boldsymbol{t} +\!\!+ [r]\rangle && \text{(for some } r \in \mathbb{R})\\
\langle \mathsf{score}(\underline{r}), w, \boldsymbol{t}\rangle &\longrightarrow \begin{cases}\langle \underline{r}, r\cdot w, \boldsymbol{t}\rangle & \text{if } r > 0,\\ \mathsf{fail} & \text{otherwise.}\end{cases}
\end{aligned}$$

Evaluation contexts:

$$\frac{\langle R, w, \boldsymbol{t}\rangle \longrightarrow \langle R', w', \boldsymbol{t}'\rangle}{\langle E[R], w, \boldsymbol{t}\rangle \longrightarrow \langle E[R'], w', \boldsymbol{t}'\rangle} \qquad \frac{\langle R, w, \boldsymbol{t}\rangle \longrightarrow \mathsf{fail}}{\langle E[R], w, \boldsymbol{t}\rangle \longrightarrow \mathsf{fail}}$$

Figure 11. Operational small-step semantics of SPCF

In the rule for normal, a random value $r \in \mathbb{R}$ is generated and recorded in the trace, while the weight remains unchanged: even though the program samples from a normal distribution, the weight does not factor in Gaussian densities as they are already accounted for by $\mu_{\mathbb{T}}$. In the rule for $\mathsf{score}(\underline{r})$, the current weight is multiplied by $r \in \mathbb{R}$: typically this reflects the likelihood of the current execution given some observed data. Similarly to (Borgström et al., 2016) we reduce terms which cannot be reduced in a reasonable way (i.e. scoring with nonpositive constants or evaluating functions outside their domain) to fail.

We write $\longrightarrow^+$ for the transitive closure of $\longrightarrow$, and $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

A.2.1. Value and Weight Functions.

Recall the measure space of traces $\mathbb{T} := \bigcup_{n\in\mathbb{N}} \mathbb{R}^n$ is equipped with the standard disjoint union σ-algebra $\Sigma_{\mathbb{T}} := \{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \mathcal{B}_n\}$, with measure given by summing the respective (higher-dimensional) normals $\mu_{\mathbb{T}}(\bigcup_{n\in\mathbb{N}} U_n) := \sum_{n\in\mathbb{N}} \mathcal{N}_n(U_n)$. Following Borgström et al. (2016), we write Λ to denote the set of all SPCF terms and view it as $\bigcup_{n\in\mathbb{N}}(\mathsf{SK}_n \times \mathbb{R}^n)$ where SK_n is the set of SPCF terms with exactly n numerals place-holders. The measurable space of terms is equipped with the σ-algebra Σ_Λ that is the Borel algebra of the countable disjoint union topology of the product topology of the discrete topology on SK_n and the standard topology on $\mathbb{R}^n$. Similarly the subspace Λ^0_v of closed values inherits the Borel algebra on Λ.

Let M be a closed SPCF term. Its ***value function*** $\mathsf{value}_M : \mathbb{T} \to \Lambda^0_v \cup \{\bot\}$ returns, given a trace, the output value of the program, if the program terminates in a value. The ***weight function*** $\mathsf{weight}_M : \mathbb{T} \to \mathbb{R}_{\ge 0}$ returns the final weight of the corresponding execution. Formally:

$$\mathsf{value}_M(\boldsymbol{t}) := \begin{cases} V & \text{if } \langle M, 1, [\,]\rangle \longrightarrow^* \langle V, w, \boldsymbol{t}\rangle\\ \bot & \text{otherwise}\end{cases} \qquad \mathsf{weight}_M(\boldsymbol{t}) := \begin{cases} w & \text{if } \langle M, 1, [\,]\rangle \longrightarrow^* \langle V, w, \boldsymbol{t}\rangle\\ 0 & \text{otherwise}\end{cases}$$

It follows already from (Borgström et al., 2016) that the functions value_M and weight_M are measurable.

Finally, every closed SPCF term M has an associated ***value measure***

$$\begin{aligned}\llbracket M \rrbracket : \Sigma_{\Lambda_v^0} &\longrightarrow \mathbb{R}_{\geq 0} \\ U &\longmapsto \int_{\mathsf{value}_M{}^{-1}(U)} \mathsf{weight}_M \, \mathsf{d}\mu_{\mathbb{T}}\end{aligned}$$

This corresponds to the denotational semantics of SPCF in the ω-quasi-Borel space model via computational adequacy (Vákár et al., 2019).

Proposition 7. *Every closed SPCF term has a tree representable weight function.*

Proof. Assume M is a closed SPCF term and $\boldsymbol{q} \in \mathsf{Supp}^n(\mathsf{weight}_M)$. The reduction of M must be $\langle M, 1, [] \rangle \longrightarrow^* \langle V, w, \boldsymbol{q} \rangle$ for some value V and weight $w > 0$. Assume for contradiction that there is some $k < n$ where $\langle M, 1, [] \rangle \longrightarrow^* \left\langle V', w', \boldsymbol{q}^{1\dots k} \right\rangle$ for some value V' and weight $w' > 0$. Since $\boldsymbol{q}^{1\dots k}$ is a prefix of $\boldsymbol{q}$ and $\longrightarrow$ is deterministic if the trace is given, we must have $\langle M, 1, [] \rangle \longrightarrow^+ \left\langle V', w', \boldsymbol{q}^{1\dots k} \right\rangle \longrightarrow^+ \langle V, w, \boldsymbol{q} \rangle$, which contradicts the fact that V' is a value. $\square$

A.3. Almost-sure Termination

Definition 8. We say that a SPCF term M ***terminates almost surely*** if M is closed and $\mu_{\mathbb{T}}(\{\boldsymbol{t} \in \mathbb{T} \mid \exists V, w \,.\, \langle M, 1, [] \rangle \longrightarrow^* \langle V, w, \boldsymbol{t} \rangle\}) = 1$;

The following proposition is used in Prop. 24 to support the correctness proof.

Proposition 9. *The value measure $\llbracket M \rrbracket$ of a closed almost surely terminating SPCF term M which does not contain* $\mathsf{score}(-)$ *as a subterm is probabilistic.*

One of the main contribution of (Mak et al., 2021) is to find a suitable class of primitive functions such that their main theorem (Lem. 10) holds.

For our purposes, we take the set of ***analytic functions*** with co-domain $\mathbb{R}$ as our class $\mathcal{F}$ of primitive functions which, as shown in Example 3 of (Mak et al., 2021), satisfies the conditions for which the following lemma holds.

Lemma 10 (Mak et al. (2021), Theorem 3)**.** *Let M be an SPCF term which terminates almost surely. Then its weight function* weight_M *and value function* value_M *are differentiable almost everywhere.*

Definition 11. We say that a SPCF term M is ***integrable*** if M is closed and its value measure is finite, i.e. $\llbracket M \rrbracket(\Lambda_v^0) < \infty$;

We conclude with the following lemma which shows that NP-HMC is an adequate inference algorithm for closed SPCF terms.

Lemma 12. *The weight function of a closed integrable almost surely terminating SPCF term satisfies Assumptions 1, 2 and 3 of the NP-HMC algorithm.*

Proof. Let M be a closed integrable almost surely terminating SPCF term, and w be its weight function. w is tree representable by Prop. 7. Integrability of w (Assumption 1) is given as an assumption, and w is almost everywhere continuously differentiable (Assumption 2) by Lem. 10.

Assume for contradiction that Assumption 3 does not hold. i.e. There is a non-null set U of infinite real-valued sequence where w is zero on all prefixes of sequences in U. Let $U_p := \{\boldsymbol{q}^{1\dots k} \mid \boldsymbol{q} \in U, k \in \mathbb{N}\}$ be the set of prefixes of sequences in U. Since U is non-null, U_p must also be non-null. Moreover, w is zero on all traces in U_p. By the definition of weight function, $\boldsymbol{q} \in U_p$ implies $\langle M, 1, [] \rangle \not\longrightarrow^* \langle V, w', \boldsymbol{q} \rangle$ for some V and w'. Hence, the probability of a non-terminating run of M is non-zero and M is not almost surely terminating. $\square$

Remark 13. The weight function as defined in App. A.2.1 is the input density function of the target distribution to which an inference algorithm typically samples from. In this paper, we call this function the "weight function" when considering semantics following (Culpepper & Cobb, 2017; Vákár et al., 2019; Mak et al., 2021), and use the notion "density" when referring it in an inference algorithm similar to (Zhou et al., 2019; 2020; Cusumano-Towner et al., 2020).

B. Hamiltonian Monte Carlo Algorithm and its Variants

Hamiltonian Monte Carlo (HMC) algorithm (Duane et al., 1987; Cances et al., 2007; Neal, 2011) is a Markov chain Monte Carlo inference algorithm that generates samples from a continuous (finite) distribution ν on the measure space $(\mathbb{R}^n, \mathcal{B}_n, \mathsf{Leb}_n)$, where $\mathcal{B}_n$ denotes the Borel σ-algebra.

B.1. HMC Algorithm

To generate a Markov chain $\{\boldsymbol{q_i}\}_{i\in\mathbb{N}}$ of samples from ν, HMC simulates the *Hamiltonian* motion of a particle on the negative logarithm of the density function of ν with some auxiliary momentum. Hence regions with high probability in ν have low potential energy and are more likely to be visited by the simulated particle. In each iteration, the particle is given some random momentum. We formalise the algorithm here.

B.1.1. Hamiltonian Dynamics

Say $\rho : \mathbb{R}^n \to \mathbb{R}$ is the (not necessarily normalized) probability density function of ν. The simulated particle has two types of energies: ***potential energy*** $U : \mathbb{R}^n \to \mathbb{R}$ given by $U(\boldsymbol{q}) := -\log\rho(\boldsymbol{q})$ and ***kinetic energy*** $K : \mathbb{R}^n \to \mathbb{R}$ given by $K(\boldsymbol{p}) := -\log \mathsf{pdf}_D(\boldsymbol{p})$ where D is some momentum distribution, typically a n-dimensional normal distribution. Henceforth, we take $K(\boldsymbol{p}) := \sum_{i=1}^n \frac{\boldsymbol{p}_i^2}{2}$.

The ***Hamiltonian*** $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ of a system is defined quite simply to be the sum of the potential and kinetic energies, i.e.

$$H(\boldsymbol{q},\boldsymbol{p}) := U(\boldsymbol{q}) + K(\boldsymbol{p}).$$

The trajectories $\{(\boldsymbol{q}^t, \boldsymbol{p}^t)\}_{t\geq 0}$, where $\boldsymbol{q}^t$ and $\boldsymbol{p}^t$ are the position and momentum of the particle at time t respectively, defined by the Hamiltonian H, can be determined by the ***Hamiltonian equations***:

$$\frac{\mathrm{d}\boldsymbol{q}(t)}{\mathrm{d}t} := \frac{\partial H}{\partial \boldsymbol{p}}(\boldsymbol{q}(t),\boldsymbol{p}(t)) = \nabla K(\boldsymbol{p}(t)) = \boldsymbol{p}(t) \qquad \text{and} \qquad \frac{\mathrm{d}\boldsymbol{p}(t)}{\mathrm{d}t} := -\frac{\partial H}{\partial \boldsymbol{q}}(\boldsymbol{q}(t),\boldsymbol{p}(t)) = -\nabla U(\boldsymbol{q}(t)).$$

with initial conditions $(\boldsymbol{q}(0), \boldsymbol{p}(0)) = (\boldsymbol{q}^0, \boldsymbol{p}^0)$.

The ***canonical distribution*** (also called Boltzmann-Gibbs distribution) π on the measure space $(\mathbb{R}^n \times \mathbb{R}^n, \Sigma_{\mathbb{R}^n\times\mathbb{R}^n}, \mathsf{Leb}_{2n})$ corresponding to H is given by the probability density function

$$\zeta(\boldsymbol{q},\boldsymbol{p}) := \frac{1}{Z}\exp(-H(\boldsymbol{q},\boldsymbol{p})) = \frac{1}{Z}\exp(-U(\boldsymbol{q}) - K(\boldsymbol{p})) \qquad \text{where } Z := \int_{\mathbb{R}^n} \rho \, \mathsf{d}\mathsf{Leb}_n$$

B.1.2. The Algorithm

Since computers cannot simulate continuous motions like Hamiltonian, the equations of motion are generally numerically integrated by the ***leapfrog*** method (also called the velocity-Verlet algorithm (Verlet, 1967)):

$$\begin{aligned}\boldsymbol{p}^{n+1/2} &= \boldsymbol{p}^n - \epsilon/2 \cdot \nabla U(\boldsymbol{q}^n)\\ \boldsymbol{q}^{n+1} &= \boldsymbol{q}^n + \epsilon \cdot \boldsymbol{p}^{n+1/2}\\ \boldsymbol{p}^{n+1} &= \boldsymbol{p}^{(n+1)/2} - \epsilon/2 \cdot \nabla U(\boldsymbol{q}^{n+1})\end{aligned}$$

where ϵ is the time step.

The ***integrator*** $\Psi_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$ as given in Alg. 4, takes a state $(\boldsymbol{q},\boldsymbol{p})$ and performs L leapfrog steps with initial condition $(\boldsymbol{q}^0, \boldsymbol{p}^0) := (\boldsymbol{q},\boldsymbol{p})$ and time step ϵ, and return the state $(\boldsymbol{q}^L, -\boldsymbol{p}^L)$.

Proposition 14 (Bou-Rabee & Sanz-Serna (2018), Theorem 4.1 and 4.2)**.** *The integrator Ψ_n is volume preserving (i.e. ${\Psi_n}_* \mathsf{Leb}_{2n} = \mathsf{Leb}_{2n}$) and reversible (i.e. $\Psi_n = {\Psi_n}^{-1}$) on $\mathbb{R}^n \times \mathbb{R}^n$.*

Proof. Let $\phi_k^P, \phi_k^Q : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ be the transition of momentum and position variables with step size k respectively, i.e. $\phi_k^P(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q}, \boldsymbol{p} - k\nabla U(\boldsymbol{q}))$, and $\phi_k^Q(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q} + k\nabla K(\boldsymbol{p}), \boldsymbol{p})$. Hence, we can write the integrator Ψ_n as the composition $S \circ \phi_{\epsilon/2}^P \circ \phi_\epsilon^Q \circ \phi_{\epsilon/2}^P$, where $S(\boldsymbol{q},\boldsymbol{p}) := (\boldsymbol{q}, -\boldsymbol{p})$.

Algorithm 4 HMC Integrator Ψ_n

```
**Input:** current state $(\boldsymbol{q_0}, \boldsymbol{p_0})$, potential energy $U$, step size $\epsilon$, number of steps $L$
**Output:** new state $(\boldsymbol{q}, \boldsymbol{p})$
$(\boldsymbol{q}, \boldsymbol{p}) = (\boldsymbol{q_0}, \boldsymbol{p_0})$    {initialise}
**for** $i = 0$ **to** $L$ **do**
    $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U(\boldsymbol{q})$    {1/2 momentum step}
    $\boldsymbol{q} = \boldsymbol{q} + \epsilon \boldsymbol{p}$    {1 position step}
    $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U(\boldsymbol{q})$    {1/2 momentum step}
**end for**
$\boldsymbol{p} = -\boldsymbol{p}$
**return** $(\boldsymbol{q}, \boldsymbol{p})$
```

Algorithm 5 HMC Step

```
**Input:** current sample $\boldsymbol{q_0}$, potential energy $U$, step size $\epsilon$, number of steps $L$
**Output:** next sample $\boldsymbol{q}$
$\boldsymbol{p_0} \sim \mathcal{N}_n$    {Kick}
$(\boldsymbol{q}, \boldsymbol{p}) = \Psi_n((\boldsymbol{q_0}, \boldsymbol{p_0}), U, \epsilon, L)$    {Integrate}
**if** $\mathcal{U}(0,1)$[a] $< \min\{1, \frac{\zeta(\boldsymbol{q},\boldsymbol{p})}{\zeta(\boldsymbol{q_0},\boldsymbol{p_0})}\}$ **then**
    **return** $\boldsymbol{q}$    {MH acceptance ratio}
**else**
    **return** $\boldsymbol{q_0}$
**end if**
```

[a] $\mathcal{U}(0,1)$ is the standard uniform distribution.

It is easy to see that $(\phi_k^P)^{-1} = S \circ \phi_k^P \circ S$ and $(\phi_k^Q)^{-1} = S \circ \phi_k^Q \circ S$. Hence, ${\Psi_n}^{-1} = (\phi_{\epsilon/2}^P)^{-1} \circ (\phi_\epsilon^Q)^{-1} \circ (\phi_{\epsilon/2}^P)^{-1} \circ S = S \circ \phi_{\epsilon/2}^P \circ \phi_\epsilon^Q \circ \phi_{\epsilon/2}^P = \Psi_n$ and Ψ_n is reversible.

Similarly it is easy to see that the shear transformations ϕ_k^P, ϕ_k^Q and momentum flip S preserves measure on $\mathbb{R}^{2n}$, i.e. $\phi_k^P(D)$, $\phi_k^Q(D)$, $S(D)$ and D have the same measure for all measurable set D in $\mathbb{R}^{2n}$. Hence, $({\Psi_n}_* \mathsf{Leb}_{2n})(D) = \mathsf{Leb}_{2n}({\Psi_n}^{-1}(D)) = \mathsf{Leb}_{2n}(\Psi_n(D)) = \mathsf{Leb}_{2n}(D)$ and Ψ_n is volume preserving. □

Alg. 5 shows how HMC generates a sample from the current one $\boldsymbol{q_0}$. It first performs leapfrog steps on $(\boldsymbol{q_0}, \boldsymbol{p_0})$ via the integrator Ψ_n with a randomly chosen initial momentum $\boldsymbol{p_0}$. The result $(\boldsymbol{q}, \boldsymbol{p})$ of Ψ_n is then accepted with probability $\min\{1, \frac{\zeta(\boldsymbol{q},\boldsymbol{p})}{\zeta(\boldsymbol{q_0},\boldsymbol{p_0})}\}$. Note that if Hamiltonian is preserved (i.e. $H(\boldsymbol{q}, \boldsymbol{p}) = H(\boldsymbol{q_0}, \boldsymbol{p_0})$), the acceptance probability is one and the proposal will always be accepted.

A Markov chain $\{\boldsymbol{q_i}\}_{i\in\mathbb{N}}$ is generated by iterating Alg. 5.

B.1.3. Correctness

The HMC algorithm is only effective if its generated Markov chain $\{\boldsymbol{q_i}\}_{i\in\mathbb{N}}$ does converge to the target distribution ν. Here we consider the typical convergence result of the total variation norm for the probability measure generated.

Formally, we say a Markov chain $\{\boldsymbol{q_i}\}_{i\in\mathbb{N}}$ converges to the target distribution ν on $\mathbb{R}^n$ if

$$\forall \boldsymbol{q} \in \mathbb{R}^n, \quad \lim_{m\to\infty} \|Q^m(\boldsymbol{q}, -) - \nu\| = 0,$$

where $Q^m(\boldsymbol{q}, A)$ is the probability for which the Markov chain is in $A \in \mathcal{B}_n$ after m steps starting at $\boldsymbol{q} \in \mathbb{R}^n$ and $\|-\|$ denotes the total variation norm on $\mathbb{R}^n$ (i.e. $\|\mu\| := \sup_{A\in\mathcal{B}_n} \mu(A) - \inf_{A\in\mathcal{B}_n} \mu(A)$).

Here we present the necessary conditions to prove such a result for the HMC algorithm. Let $Q : \mathbb{R}^n \times \mathcal{B}_n \to \mathbb{R}_{\geq 0}$ be the transition kernel specified by Alg. 5, so that $Q(\boldsymbol{q}, A)$ is the probability for which the next sample returned by Alg. 5 is in $A \in \mathcal{B}_n$ given the current sample is $\boldsymbol{q} \in \mathbb{R}^n$. We write Q^m to be m compositions of Q. (i.e. $Q^0(\boldsymbol{q}, A) := [\boldsymbol{q} \in A]$; for $k > 0$, $Q^{k+1}(\boldsymbol{q}, A) := \int_{\mathbb{R}^n} Q^k(\boldsymbol{q}', A)\ Q(\boldsymbol{q}, \mathrm{d}\boldsymbol{q}')$).

First, we make sure that ν is the invariant distribution of the Markov chain.

Proposition 15 (Bou-Rabee & Sanz-Serna (2018), Theorem 5.2)**.** *ν is invariant against Q.*

While showing ν is the invariant distribution for the Markov chain is relatively simple, we would be wrong to think that convergence follows trivially. In fact, as shown in the following example, the Markov chain can easily be periodic.

Example 16 (Bou-Rabee & Sanz-Serna (2018), Example 5.1)**.** Consider the case where the target distribution is a (unnormalised) one-dim. normal distribution. In particular say the potential energy is $U(q) := q^2/2$. Then, the Hamiltonian flow ($H(q,p) = U(q) + K(p) = q^2/2 + p^2/2$) is a rotation in the (q,p)-plane with period 2π. If the duration of the simulation is π, the exact flow returns $q_1 = -q_0$.

There are known conditions for which HMC converges to the right distribution (Schütte, 1999). Here we follow the treatment given by Cances et al. (2007).

Results from (Tierney, 1994; Borgström et al., 2016) tell us that it is enough to show that the transition kernel Q is ***strongly*** ***ν-irreducible***: for all a and B, $\nu(B) > 0$ implies $Q(a, B) > 0$.

Lemma 17 (Cances et al. (2007), Lemma 2 and 3 (Strong irreducibility))**.** *Assume U is continuously differentiable, bounded above on $\mathbb{R}^n$ and ∇U is globally Lipschitz. Then the transition kernel Q is strongly ν-irreducible.*

Lemma 18 (Borgström et al. (2016), Lemma 33 (Aperiodicity))**.** *A strongly ν-irreducible transition kernel is also ν-aperiodic.*

Lemma 19 (Tierney (1994), Theorem 1 and Corollary 2)**.** *If the transition kernel Q with invariant distribution ν is ν-irreducible and ν-aperiodic, then for all $\boldsymbol{q}$, $\lim_{n\to\infty} \|Q^n(\boldsymbol{q}, -) - \nu\| = 0$.*

Theorem 20. *If U is continuously differentiable, bounded above on $\mathbb{R}^n$ and ∇U is globally Lipschitz, the Markov chain generated by iterating Alg. 5 converges to the target distribution ν.*

B.2. HMC Variants

B.2.1. Reflective/Refractive HMC

Reflective/refractive HMC (RHMC) (Afshar & Domke, 2015) is an extension of HMC that improves its behaviour for discontinuous density functions. Standard HMC is correct for such distributions as well, but the acceptance probability may be very low and convergence extremely slow.

We need to quickly discuss what discontinuities mean in our setting: In addition to discontinuities of each $U_n : \mathbb{R}^n \to \mathbb{R}$ itself, we also regard it as a discontinuity when $\boldsymbol{q}$ leaves the support of U_n, since this means that a different branch in the tree representing function is chosen. The set of these discontinuities is $\partial\mathsf{Supp}(w)$, i.e. the boundary of the support of the density function.

Fortunately, the extension of RHMC to our nonparametric setting is straightforward. The algorithm is described in Alg. 6. The only relevant difference is the need for an extend call in the algorithm.

The rest of the algorithm is the same as (Afshar & Domke, 2015): It uses two additional functions that deal with the discontinuities of U: decompose and nextBoundary. Just like in (Afshar & Domke, 2015), we assume that these are given to the algorithm because their implementation depends on the kind of discontinuities in the density function. In the original paper, they only consider discontinuities that are given by affine subspaces.

The function nextBoundary$(\boldsymbol{q}, \boldsymbol{p}, T, U)$ takes a position $\boldsymbol{q} \in \mathbb{R}^n$, a momentum $\boldsymbol{p} \in \mathbb{R}^n$, a time limit $T > 0$, and family of potential energies $\{U_n\}_{n\in\mathbb{N}}$. It then checks whether a particle starting at $\boldsymbol{q}$ moving with momentum $\boldsymbol{p}$ will hit a discontinuity of U in time $\leq T$. If so, it returns the time t of "impact", the position $\boldsymbol{q}_<$ just before the discontinuity and $\boldsymbol{q}_>$ just after the discontinuity.

The function decompose$(\boldsymbol{q}, \boldsymbol{p}, U)$ takes a position $\boldsymbol{q}$ on the discontinuity, a momentum $\boldsymbol{p}$, and U as before. It then decomposes the momentum $\boldsymbol{p}$ into a component $\boldsymbol{p}_\parallel$ that is parallel to the discontinuity and $\boldsymbol{p}_\perp$ that is perpendicular to it.

The basic idea of the algorithm is inspired by reflection and refraction in physics. We simulate the trajectory of a particle according to Hamiltonian dynamics. When hitting a discontinuity, we compute the potential difference. If the kinetic energy is big enough to overcome it, refraction occurs: the perpendicular component of $\boldsymbol{p}$ is scaled down. Otherwise, the particle is reflected.

The only difference to the original algorithm in (Afshar & Domke, 2015) is the call to extend. Why is it necessary? When hitting a discontinuity (and only then!), we may have to switch to a different branch on the tree representing the density function. Hence we may have to extend the position $q_>$ just after the discontinuity, which is why we call extend on it.

B.2.2. Laplace Momentum and Discontinuous HMC

The Hamiltonian Monte Carlo method usually uses Gaussian momentum because it corresponds to the physical interpretation of kinetic energy being $\frac{1}{2}\sum_i \boldsymbol{p}_i^2$ for a momentum vector $\boldsymbol{p}$. Nishimura et al. (2020) propose to use Laplace momentum where the kinetic energy for a momentum vector $\boldsymbol{p}$ is given by $\sum_i |\boldsymbol{p}_i|$. This means that the momentum vector must follow a Laplace distribution, denoted as $\mathcal{L}(0, 1)$, with density proportional to $\prod_i \exp(-|\boldsymbol{p}_i|)$. Hamilton's equations have to be changed to

$$\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = \operatorname{sign}(p), \quad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\nabla_q U.$$

Note that the time derivative of $\boldsymbol{q}$ only depends on the sign of the p_i's. Hence, if the sign does not change, the change of $\boldsymbol{q}$ can be computed, irrespective of the intermediate values of $U_{|\boldsymbol{q}|}(\boldsymbol{q})$. The integrator of discontinuous HMC (Nishimura et al., 2020) takes advantage of this for "discontinuous parameters", i.e. parameters that U is not continuous in. Thus it can jump through multiple discontinuities of U without evaluating it at every boundary.

We adapt the integrator from (Nishimura et al., 2020) to NP-HMC. Following them, we assume for simplicity that each coordinate of the position space either corresponds to a continuous or discontinuous parameter, irrespective of which path is chosen. The set C records all the continuous parameters and $D = \mathbb{N} \setminus C$ the discontinuous ones. We use a Gaussian distribution for the continuous parameters of the momentum vector and a Laplace distribution for the discontinuous parameters. Our integrator updates the continuous coordinates by half a step size just as before, but then the discontinuous ones are updated coordinate by coordinate, a technique called *operator splitting*. Afterwards, the continuous coordinates are updated by half a step size again. Algorithm 8 contains all the details.

Again, the main difference to the original algorithm is a call to extend. Note we also have to modify the extend function itself (given in Alg. 7) because some momentum coordinates have to be sampled from a Laplace distribution, and not a Gaussian as before.

We also make the following modification: we update the q_0 position to current time t instead of q because this avoids having to re-run the probabilistic program. If we update q, a re-run might be necessary if q changed again after an extension, but for q_0 this is not the case because the extended part does not affect the weight.

B.3. Efficiency Improvements

As touched upon in the main text, our implementation includes various performance improvements compared to the pseudocode presentation of NP-HMC.

(i) The extend function (Alg. 3) as presented may seem inefficient. While it terminates almost surely (thanks to Assumption 3), the expected number of iterations may be infinite. In practice, however, the density function w will arise from a probabilistic program, such as Listing 1. Therefore, to evaluate w, it would be natural to run the program. The length of $\boldsymbol{q}$ returned by extend is exactly the number of `sample` statements encountered during the program's execution. In particular, if the program has finite expected running time, then the same is true of extend.

(ii) On top of that, efficient implementations of NP-HMC will interleave the execution of the program with extend, by gradually extending $\boldsymbol{q}$ (if necessary) at every encountered `sample` statement. This way, extend increases the running time only by a small constant factor.

(iii) For this to work, we also make the following modification: we update the q_0 position to current time t instead of q because this avoids having to re-run the probabilistic program. If we update q, a re-run might be necessary if q changed again after an extension, but for q_0 this is not the case because the extended part does not affect the weight.

(iv) In a similar vein, we do not have to compute the sum $w_{\leq n}(\boldsymbol{q}) = \sum_{k=1}^{n} w(\boldsymbol{q}^{1\dots k})$ each time $U_n = -\log w_{\leq n}$ is accessed. By the prefix property, only one of the summands of $w_{\leq n}(\boldsymbol{q})$ is actually nonzero. Moreover, if w is given by a probabilistic program, then the weight computed during the execution of the program on $\boldsymbol{q}$ is exactly this nonzero summand, assuming that the trace $\boldsymbol{q}$ is long enough for a successful run (which the extend function ensures).

(v) Another notable way our implementation differs from the algorithm presented above is that it not only extends a trace $\boldsymbol{q}$ in extend (if necessary), but also trims it (if necessary) to the unique prefix $\boldsymbol{q}'$ of $\boldsymbol{q}$ with positive $w(\boldsymbol{q}')$. The dimension of $\boldsymbol{p}$ is adjusted accordingly. This seems to work much better for certain examples, such as the geometric distribution described in Sec. 5. The reason is most likely that the unused suffix (which may have been adapted to the state before the current call of extend) is a hindrance when trying to extend to a different state later on.

C. Proof of Correctness

In this section, we show that the NP-HMC algorithm is correct, in the sense that the Markov chain generated by iterating Alg. 1 converges to the target distribution $\nu : A \mapsto \frac{1}{Z} \int_A w \, d\mu_{\mathbb{T}}$ where $Z := \int_{\mathbb{T}} w \, d\mu_{\mathbb{T}}$.

Henceforth, we assume that the density function w of the target distribution ν is tree-representable and satisfies Assumptions 1, 2 and 3.

C.1. An Equivalent Algorithm

We write Alg. 1 as the program `NPHMCstep` (Alg. 2 as `NPint` and Alg. 3 as `extend`) in Listing 2. We present input sample as `q0`; the density function as `w` and define potential energy U, which is a family of partial functions, as a function `U`, such that `U(n)` is a partial function denoting U_n; step size as `ep`; and number of steps as `L`. We also assume the following primitive functions are implemented: `normal` is the sampling construct in the language which samples a real number from the standard normal distribution $\mathcal{N}_1$. `domain(f)` gives the domain of the partial function `f`. `pdfN(x,n)` gives the probability density of `x` on the standard `n`-dimensional normal distribution. `cdfN(x)` gives the cumulative distribution of `x` on the standard normal distribution. `grad(f,x)` gives the gradient of the partial function `f` at `x` if defined and `None` if not.

The program `NPHMC` generates a Markov chain on $\mathbb{T}$ by iterating `NPHMCstep`.

Instead of a direct proof, we consider an auxiliary program `eNPHMC` *equivalent* to `NPHMC` (in the sense of Prop. 22), which does not increase the dimension dynamically; instead it finds the smallest N such that all intermediate positions during the L leapfrog steps stay in the domain of U_N, and performs leapfrog steps as in standard HMC.

The program `eNPHMC` is given in Listing 4, which iterates `eNPHMCstep` to generate a Markov chain on *states* and then marginalise it using the helper function `supported` to obtain a Markov chain on $\mathbb{T}$. The program `validstate` determines whether the input state `(q0,p0)` goes beyond the domain of the potential energy `U` in `L` leapfrog steps, and the program `HMCint` is the leapfrog integrator of the standard HMC algorithm.

Remark 21. Programs in Listings 2 to 4 are given in Python syntax, but they can be translated into SPCF. First, note we can represent pairs and lists using Church encoding as follows:

$$\begin{aligned} \mathsf{Pair}(\sigma,\tau) &:= \sigma \to \tau \to (\sigma \to \tau \to \mathsf{R}) \to \mathsf{R} & \qquad \mathsf{List}(\sigma) &:= (\sigma \to \mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \\ \langle M, N \rangle &\equiv \lambda z.z\, M\, N & [M_1, \dots, M_\ell] &\equiv \lambda f x. f\, M_1 (f\, M_2 \dots (f\, M_\ell\, \underline{0})) \end{aligned}$$

Hence a state $(\boldsymbol{q},\boldsymbol{p}) \in \mathbb{R}^\ell \times \mathbb{R}^\ell$ can be encoded as a value $[\langle \underline{\boldsymbol{q}_1}, \underline{\boldsymbol{p}_1} \rangle, \dots, \langle \underline{\boldsymbol{q}_\ell}, \underline{\boldsymbol{p}_\ell} \rangle]$ with type $\mathsf{List}(\mathsf{Pair}(\mathsf{R},\mathsf{R}))$.

Now we look at all the primitive functions used in the programs. It is easy to see that `cdfN`, `pdfN` and `log` are analytic functions. `len`, `append` and `sum` can be defined on Church lists. `grad` can be defined using the simple numerical differentiation method using analytic functions like subtraction and division. We can change `domain` in such a way that it takes `q` and `w` as inputs and tests whether `sum([w(q[:i]) for i in range(len(q))])` is zero (instead of testing whether `q` is in the domain of `U(len(q))`).

Now we give a formal definition of equivalence. We say two SPCF programs are ***equivalent*** if they induce the same value and weight functions, as specified in App. A.2.1.

Proposition 22. *`NPHMC` and `eNPHMC` are equivalent.*

Proof. We give an informal explanation here.

First note that `NPHMCstep` is a Markov process on samples, and `eNPHMCstep` on states. However, it is easy to see that some minor changes to `NPHMCstep` and `NPHMC` make `NPHMCstep` a Markov process on states. Precisely, the following does not alter the meaning of program `NPHMC`:

(1) Given a state `(q0,p0)` in `NPHMCstep`, apply `supported` to `q0` at the start of initialisation and return the state `(q0,p0)` or `(q,p)` at the MH acceptance step.
(2) In `NPHMC`, add the marginalisation step just like in `eNPHMC`.

Hence, it is enough to show that all steps in programs `NPHMCstep` and `eNPHMCstep` are equivalent, i.e. they give the same weight and value functions.

After the modification, `NPHMCstep` and `eNPHMCstep` have the same initialisation and MH acceptance step. So it remains to show that the NP-HMC integration as described in `NPint` behaves the same as searching for a valid initial state (step 2) and HMC integration (step 3) in `eNPHMCstep`.

In `NPHMCstep`, `((q,p),(q0,p0)) = NPint((q0,p0),U,ep,L)` "integrates" from the initial state `(q0,p0)` until it goes beyond the domain of `U(len(q0))`, at which moment it `extend`s.

While in `eNPHMCstep`, it increments the dimension of the state `(q0,p0)` until it has *just* enough dimension to "integrate" for time `ep*L` through `U(len(q0))` without going beyond the domain of `U(len(q0))`. This ensures the state `(q0,p0)` is safe to be an input to the standard HMC integrator `HMCint`.

Notice that given the same values for the samples, the resulting initial state `(q0,p0)` in `NPHMCstep` would be the same as that in `eNPHMCstep`. Hence, the proposal state `(q,p)` in both programs would be the same. □

Remark 23. The discussion in the proof of Prop. 22 argues informally that `NPHMC` and `eNPHMC` are equivalent. We outline a formal proof here. To show that `NPHMC` and `eNPHMC` are equivalent, we first demonstrate that one program can be obtained form another by a series of meaning-preserving transformations (i.e. transformations that preserves the value and weight functions). After that we show that the convergence result (Thm. 5) is invariant over equivalent programs.

Since `NPHMC` and `eNPHMC` are equivalent, it is enough to show that `eNPHMC` is correct, i.e. generates a Markov chain that converges to the target distribution. We present a three-step proof.

1. We first identify the invariant distribution π of the Markov chain $\{(\boldsymbol{q}^{(i)},\boldsymbol{p}^{(i)})\}_{i\in\mathbb{N}}$ generated by iterating `eNPHMCstep`. (Eq. (1))
2. We then show that the *marginalised* chain $\{f(\boldsymbol{q}^{(i)},\boldsymbol{p}^{(i)})\}_{i\in\mathbb{N}}$ is invariant under the target distribution ν, where $f(\boldsymbol{q},\boldsymbol{p})$ is the unique prefix of $\boldsymbol{q}$ that has positive weight according to w. (Thm. 4)
3. Finally, we show this chain converges for a small enough step size ϵ. (Thm. 5)

C.2. Invariant Distribution

By iterating `eNPHMCstep`, a Markov chain $\{(\boldsymbol{q}^{(i)},\boldsymbol{p}^{(i)})\}_{i\in\mathbb{N}}$ is generated. We now analyse this Markov chain by studying its invariant distribution π and transition kernel.

Let $(\mathbb{S},\Sigma_{\mathbb{S}},\mu_{\mathbb{S}})$ be the ***state space*** where $\mathbb{S} := \biguplus_{n\in\mathbb{N}}(\mathbb{R}^n\times\mathbb{R}^n)$, $\Sigma_{\mathbb{S}} := \{\biguplus_{n\in\mathbb{N}}U_n \mid U_n\in\mathcal{B}_{2n}\}$ and $\mu_{\mathbb{S}}(\biguplus_{n\in\mathbb{N}}U_n) := \sum_{n\in\mathbb{N}}(\mathcal{N}_n\times\mathcal{N}_n)(U_n)$. It is easy to see that all output states in `eNPHMCstep`, and hence all elements of the Markov chain, is in $\mathbb{S}$.

However not all states have a positive weight. In fact not even the union of the support of invariant distributions of the fixed dimension HMC on each of the truncations works. This is because if `eNPHMCstep` returns $(\boldsymbol{q},\boldsymbol{p})\in\mathbb{R}^{2k}$, then it cannot return states of the form $(\boldsymbol{q}+\boldsymbol{q}',\boldsymbol{p}+\boldsymbol{p}')\in\mathbb{R}^{2n}$, which is a valid returning state for the fixed dimension HMC. Hence we define a subset of states which precisely capture all possible returning states of `eNPHMCstep`, and define a distribution on it.

We say a state $(\boldsymbol{q},\boldsymbol{p})$ is (ϵ,L)-***valid*** (or simply ***valid*** whenever the parameters ϵ and L are clear from the context) if a particle starting from the state $(\boldsymbol{q},\boldsymbol{p})$ does not "fall beyond" the domain of $U_{|\boldsymbol{q}|} := -\log w_{\le|\boldsymbol{q}|}$ in the course of L discrete leapfrog steps of size ϵ, and the states $(\boldsymbol{q}^{1\dots k},\boldsymbol{p}^{1\dots k})$ are not (ϵ,L)-valid for all $k<n$.

Let $\mathbb{S}^{\text{valid}}$ denote the set of all valid states and $\mathbb{S}^{\text{valid}}_n := \mathbb{S}^{\text{valid}}\cap(\mathbb{R}^n\times\mathbb{R}^n)$ denote the the set of all n-dimension valid states. The program `validstate` verifies valid states, i.e `validstate` always returns True when the input state is valid.

Let π be a distribution on $\mathbb{S}$ with density ζ (with respect to $\mu_{\mathbb{S}}$) given by

$$\zeta(\boldsymbol{q},\boldsymbol{p}) := \begin{cases}\frac{1}{Z}w_{\le|\boldsymbol{q}|}(\boldsymbol{q}) & \text{if } (\boldsymbol{q},\boldsymbol{p})\in\mathbb{S}^{\text{valid}},\\ 0 & \text{otherwise.}\end{cases} \tag{1}$$

Since the the position component of all valid states must have a w-supported prefix, the set of valid states can be written as

$$\mathbb{S}^{\text{valid}} = \bigcup_{n=1}^{\infty}\bigcup_{m=n}^{\infty}\{(\boldsymbol{q}+\boldsymbol{x},\boldsymbol{y})\in\mathbb{S}^{\text{valid}}_m \mid \boldsymbol{q}\in\mathsf{Supp}^n(w),\boldsymbol{x}\in\mathbb{R}^{m-n},\boldsymbol{y}\in\mathbb{R}^m\},$$

and hence the distribution π can be written as

$$\begin{aligned}\pi : X \mapsto \int_X [(\boldsymbol{q},\boldsymbol{p})\in\mathbb{S}^{\text{valid}}]\cdot\frac{1}{Z}w_{\le|\boldsymbol{q}|}(\boldsymbol{q})\,\mu_{\mathbb{S}}(\mathrm{d}(\boldsymbol{q},\boldsymbol{p})) &= \int_X [(\boldsymbol{q},\boldsymbol{p})\in\mathbb{S}^{\text{valid}}]\cdot\frac{1}{Z}\sum_{n=1}^{|\boldsymbol{q}|}w(\boldsymbol{q}^{1\dots n})\,\mu_{\mathbb{S}}(\mathrm{d}(\boldsymbol{q},\boldsymbol{p}))\\ &= \sum_{n=1}^{\infty}\sum_{m=n}^{\infty}\int_{\mathbb{R}^n}\int_{\mathbb{R}^{m-n}}\int_{\mathbb{R}^m}[(\boldsymbol{q}+\boldsymbol{x},\boldsymbol{y})\in X\cap\mathbb{S}^{\text{valid}}_m]\cdot\frac{1}{Z}w(\boldsymbol{q})\,\mathcal{N}_m(\mathrm{d}\boldsymbol{y})\,\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{x})\,\mathcal{N}_n(\mathrm{d}\boldsymbol{q})\end{aligned} \tag{2}$$

We claim that π is the ***invariant distribution*** of the Markov chain determined by `eNPHMCstep`. The rest of this subsection is devoted to a proof of the claim.

For any state $(\boldsymbol{q},\boldsymbol{p}) \in \mathbb{S}$, we write $[\![(\boldsymbol{q},\boldsymbol{p})]\!]$ to be the term $[\langle \underline{\boldsymbol{q}_1}, \underline{\boldsymbol{p}_1}\rangle, \dots, \langle \underline{\boldsymbol{q}_{|\boldsymbol{q}|}}, \underline{\boldsymbol{p}_{|\boldsymbol{q}|}}\rangle]$ of type $\mathsf{List}(\mathsf{Pair}(\mathsf{R},\mathsf{R}))$. Take a SPCF term M of type $\{x : \mathsf{List}(\mathsf{Pair}(\mathsf{R},\mathsf{R}))\} \vdash M : \mathsf{List}(\mathsf{Pair}(\mathsf{R},\mathsf{R}))$. We define a function $v_M : \mathbb{S} \times \mathbb{T} \to \mathbb{S}$ such that $[\![v_M(\boldsymbol{s},\boldsymbol{t})]\!] = \mathsf{value}_{M[[\![\boldsymbol{s}]\!]/x]}(\boldsymbol{t})$. Then, the ***transition kernel*** $k_M : \mathbb{S} \times \Sigma_{\mathbb{S}} \to \mathbb{S}$ of M given by

$$k_M(\boldsymbol{s},U) := \int_{v_M(\boldsymbol{s},-)^{-1}(U)} \mathsf{weight}_{M[[\![\boldsymbol{s}]\!]/x]} \, \mathrm{d}\mu_{\mathbb{T}}.$$

returns the probability of M returning a state in U given the input $\boldsymbol{s}$.

We say M leaves the distribution μ on $\mathbb{S}$ invariant if for all $U \in \Sigma_{\mathbb{S}}$, $\int_{\mathbb{S}} k_M(\boldsymbol{s},U)\,\mu(\mathrm{d}\boldsymbol{s}) = \mu(U)$.

C.2.1. Initialisation and Search (Steps 1 and 2)

Given $(\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}^{\text{valid}}$ and $X \in \Sigma_{\mathbb{S}}$, where $w({\boldsymbol{q_0}}^{1\dots n}) > 0$, the initialisation (step 1) of `eNPHMCstep` returns a pair of the w-supported prefix of $\boldsymbol{q_0}$ and a randomly drawn momentum. Hence, its transition kernel k_1 is given by $k_1((\boldsymbol{q_0},\boldsymbol{p_0}),X) := \int_{\mathbb{T}} [({\boldsymbol{q_0}}^{1\dots n},\boldsymbol{t}) \in X]\,\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t})$. Note that $\boldsymbol{p}_0$ (of the input state $(\boldsymbol{q}_0,\boldsymbol{p}_0)$) is ignored by `eNPHMCstep`.

If the input state $(\boldsymbol{q_0},\boldsymbol{p_0})$ is not a valid state, we have $k_1((\boldsymbol{q_0},\boldsymbol{p_0}),X) = 0$. This is required for technical reasons but is excluded in the program `eNPHMCstep` for ease of readability. At it stands in Listing 4, `eNPHMCstep` does not care whether the input state is valid as long as it has a prefix which is w-supported. To define such a transition kernel for `eNPHMCstep`, we can simply call `validstate` on the input state at the start of initialisation and fail this execution if the input state is not valid.

After that, given $(\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}$ and $X \in \Sigma_{\mathbb{S}}$ where $w({\boldsymbol{q_0}}^{1\dots n}) > 0$, step 2 of `eNPHMCstep` searches for a valid state by repeating drawing from the standard normal distribution. We can write its transition kernel k_2 as $k_2((\boldsymbol{q_0},\boldsymbol{p_0}),X) := \int_{\mathbb{T}} [(\boldsymbol{q_0} + \boldsymbol{t}^{\text{odd}}, \boldsymbol{p_0} + \boldsymbol{t}^{\text{even}}) \in X \cap \mathbb{S}^{\text{valid}}]\,\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t})$ where $\boldsymbol{t}^{\text{odd}}$ and $\boldsymbol{t}^{\text{even}}$ are subsequences of $\boldsymbol{t}$ containing the values of odd and even indexes respectively.

For any $X \in \Sigma_{\mathbb{T}}$, the (combined) transition kernel $k_{1,2}$ of steps 1 and 2 of `eNPHMCstep` is given by

$$\begin{aligned}
k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),X) &= \int_{\mathbb{T}}\int_{\mathbb{T}} [({\boldsymbol{q_0}}^{1\dots n} + \boldsymbol{t}'^{\text{odd}}, \boldsymbol{t} + \boldsymbol{t}'^{\text{even}}) \in X \cap \mathbb{S}^{\text{valid}}]\,\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}')\,\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}) \\
&= \int_{\mathbb{R}^n} \sum_{m=n}^{\infty} \int_{\mathbb{R}^{m-n}} \int_{\mathbb{R}^{m-n}} [({\boldsymbol{q_0}}^{1\dots n} + \boldsymbol{t}'', \boldsymbol{t} + \boldsymbol{t}') \in X \cap \mathbb{S}^{\text{valid}}]\,\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{t}'')\,\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{t}')\,\mathcal{N}_n(\mathrm{d}\boldsymbol{t}) \\
&= \sum_{m=n}^{\infty} \int_{\mathbb{R}^m} \int_{\mathbb{R}^{m-n}} [({\boldsymbol{q_0}}^{1\dots n} + \boldsymbol{x}, \boldsymbol{y}) \in X \cap \mathbb{S}^{\text{valid}}]\,\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{x})\,\mathcal{N}_m(\mathrm{d}\boldsymbol{y})
\end{aligned}$$

if $(\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}^{\text{valid}}$; and $k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),X) = 0$ otherwise.

Proposition 24. *The transition kernel is probabilistic, i.e.* $k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),\mathbb{S}) = k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),\mathbb{S}^{\mathit{valid}}) = 1$ *for any valid state* $(\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}^{\mathit{valid}}$.

Proof. Let $(\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}^{\text{valid}}$. We can see $k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),-)$ as the value measure of steps 1 and 2 of `eNPHMCstep` (with the initial states substituted by $[\![(\boldsymbol{q_0},\boldsymbol{p_0})]\!]$) which does not contain $\mathsf{score}(-)$ as a subterm. Moreover, Assumption 3 ensures step 2 almost always terminates and returns a valid state. Hence, Prop. 9 tells us that $k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),-)$ is probabilistic and $k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),\mathbb{S}) = k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),\mathbb{S}^{\text{valid}}) = 1$. □

Proposition 25. π *is invariant with respect to step 1 and 2 of* `eNPHMCstep`.

Proof. We aim to show: $\int_{\mathbb{S}} k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),X)\,\pi(\mathrm{d}(\boldsymbol{q_0},\boldsymbol{p_0})) = \pi(X)$ for any measurable set $X \in \Sigma_{\mathbb{S}}$.

$$\begin{aligned}
&\int_{\mathbb{S}} k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),X)\,\pi(\mathrm{d}(\boldsymbol{q_0},\boldsymbol{p_0})) = \int_{\mathbb{S}^{\text{valid}}} k_{1,2}((\boldsymbol{q_0},\boldsymbol{p_0}),X)\,\pi(\mathrm{d}(\boldsymbol{q_0},\boldsymbol{p_0})) \\
= &\quad \{\ \text{Eq. (2), definition of } k_{1,2} \text{ and writing } (\boldsymbol{q_0},\boldsymbol{p_0}) \in \mathbb{S}^{\text{valid}} \text{ as } (\boldsymbol{q}+\boldsymbol{x},\boldsymbol{y}) \text{ where } \boldsymbol{q} \in \mathsf{Supp}(w)\ \}
\end{aligned}$$

$$\sum_{n=1}^{\infty}\sum_{m=n}^{\infty}\int_{\mathbb{R}^n}\int_{\mathbb{R}^{m-n}}\int_{\mathbb{R}^m}\left(\sum_{k=n}^{\infty}\int_{\mathbb{R}^k}\int_{\mathbb{R}^{k-n}}[(\boldsymbol{q}+\boldsymbol{x}',\boldsymbol{y}')\in X\cap\mathbb{S}^{\text{valid}}]\,\mathcal{N}_{k-n}(\mathrm{d}\boldsymbol{x}')\,\mathcal{N}_k(\mathrm{d}\boldsymbol{y}')\right)\cdot$$
$$\left([(\boldsymbol{q}+\boldsymbol{x},\boldsymbol{y})\in\mathbb{S}^{\text{valid}}]\cdot\frac{1}{Z}w(\boldsymbol{q})\right)\mathcal{N}_m(\mathrm{d}\boldsymbol{y})\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{x})\mathcal{N}_n(\mathrm{d}\boldsymbol{q})$$
$$= \quad \{\text{ Rearranging (allowed because everything is nonnegative) }\}$$
$$\sum_{n=1}^{\infty}\sum_{k=n}^{\infty}\int_{\mathbb{R}^n}\int_{\mathbb{R}^{k-n}}\int_{\mathbb{R}^k}[(\boldsymbol{q}+\boldsymbol{x}',\boldsymbol{y}')\in X\cap\mathbb{S}^{\text{valid}}]\cdot\frac{1}{Z}w(\boldsymbol{q})$$
$$\left(\sum_{m=n}^{\infty}\int_{\mathbb{R}^{m-n}}\int_{\mathbb{R}^m}[(\boldsymbol{q}+\boldsymbol{x},\boldsymbol{y})\in\mathbb{S}^{\text{valid}}]\,\mathcal{N}_m(\mathrm{d}\boldsymbol{y})\,\mathcal{N}_{m-n}(\mathrm{d}\boldsymbol{x})\right)\mathcal{N}_k(\mathrm{d}\boldsymbol{y}')\mathcal{N}_{k-n}(\mathrm{d}\boldsymbol{x}')\mathcal{N}_n(\mathrm{d}\boldsymbol{q})$$
$$= \quad \{\text{ Definition of } k_{1,2} \text{ where } (\hat{\boldsymbol{q}},\hat{\boldsymbol{p}}) \text{ is an arbitrary valid state such that } \hat{\boldsymbol{q}}^{1\dots n}=\boldsymbol{q}\ \}$$
$$\sum_{n=1}^{\infty}\sum_{k=n}^{\infty}\int_{\mathbb{R}^n}\int_{\mathbb{R}^{k-n}}\int_{\mathbb{R}^k}[(\boldsymbol{q}+\boldsymbol{x}',\boldsymbol{y}')\in X\cap\mathbb{S}^{\text{valid}}]\cdot\frac{1}{Z}w(\boldsymbol{q})\cdot k_{1,2}((\hat{\boldsymbol{q}},\hat{\boldsymbol{p}}),\mathbb{S}^{\text{valid}})\ \mathcal{N}_k(\mathrm{d}\boldsymbol{y}')\,\mathcal{N}_{k-n}(\mathrm{d}\boldsymbol{x}')\,\mathcal{N}_n(\mathrm{d}\boldsymbol{q})$$
$$= \quad \{\text{ Definition of } \zeta \text{ and Prop. 24 for some valid state } (\hat{\boldsymbol{q}},\hat{\boldsymbol{p}})\ \}$$
$$\int_X \zeta\ \mathrm{d}\mu_{\mathbb{S}}$$

□

C.2.2. Integration and Acceptance (Steps 3 and 4)

Let $(\boldsymbol{q_0},\boldsymbol{p_0})\in\mathbb{S}$ and $X\in\Sigma_{\mathbb{S}}$. Now we check that the HMC integration (step 3) and acceptance (step 4) preserve the invariant distribution π.

Similar to HMC, the transition kernel for steps 3 and 4 is given by

$$k_{3,4}((\boldsymbol{q_0},\boldsymbol{p_0}),X)=\begin{cases}\alpha(\boldsymbol{q_0},\boldsymbol{p_0})\cdot[\Psi_{|\boldsymbol{q_0}|}(\boldsymbol{q_0},\boldsymbol{p_0})\in X]+(1-\alpha(\boldsymbol{q_0},\boldsymbol{p_0}))\cdot[(\boldsymbol{q_0},\boldsymbol{p_0})\in X] & \text{if } (\boldsymbol{q_0},\boldsymbol{p_0})\in\mathbb{S}^{\text{valid}},\\ 0 & \text{otherwise.}\end{cases}$$

where $\alpha(\boldsymbol{q_0},\boldsymbol{p_0})=\min\{1,\frac{w_{\le N}(\boldsymbol{q})\cdot\varphi_{2N}(\boldsymbol{q},\boldsymbol{p})}{w_{\le N}(\boldsymbol{q_0})\cdot\varphi_{2N}(\boldsymbol{q_0},\boldsymbol{p_0})}\}$ for $N=|\boldsymbol{q_0}|$ and $(\boldsymbol{q},\boldsymbol{p})=\Psi_N(\boldsymbol{q_0},\boldsymbol{p_0})$.

Proposition 26. *The HMC integrator Ψ_n with respect to the potential energy U_n is volume preserving with respect to Leb_{2n} (i.e. $\Psi_{n*}\mathsf{Leb}_{2n}=\mathsf{Leb}_{2n}$) and reversible (i.e. $\Psi_n={\Psi_n}^{-1}$) on $\mathbb{S}_n^{valid}$.*

Proof. Since measurable subsets of and states in $\mathbb{S}_n^{\text{valid}}$ are also in the n-dimension Euclidean Space, and Ψ_n always map valid states to valid states, Prop. 14 is sufficient. □

Proposition 27. *π is invariant against integration and acceptance (steps 3 and 4) of* `eNPHMCstep`.

Proof. We aim to show: $\int_{\mathbb{S}} k_{3,4}(x,X)\ \pi(\mathrm{d}x)=\pi(X)$ for all $X\in\Sigma_{\mathbb{S}}$. By Prop. 26, for all n, HMC integrator Ψ_n is volume preserving against Leb_{2n} and reversible on $\mathbb{S}_n^{\text{valid}}$. Hence, we have

$$\int_{\mathbb{S}} k_{3,4}(x,X)\ \pi(\mathrm{d}x)=\int_{\mathbb{S}^{\text{valid}}} k_{3,4}(x,X)\ \pi(\mathrm{d}x)=\sum_{n=1}^{\infty}\int_{\mathbb{S}_n^{\text{valid}}} k_{3,4}(x,X)\cdot\zeta(x)\ (\mathcal{N}_n\times\mathcal{N}_n)(\mathrm{d}x)$$
$$=\int_X\zeta\ \mathrm{d}\mu_{\mathbb{S}}+\sum_{n=1}^{\infty}\left(\int_{\mathbb{S}_n^{\text{valid}}}[\Psi_n(x)\in X\cap\mathbb{S}_n^{\text{valid}}]\cdot\alpha(x)\cdot\zeta(x)\cdot\varphi_{2n}(x)\ \mathsf{Leb}_{2n}(\mathrm{d}x)\right.$$
$$\left.-\int_{\mathbb{S}_n^{\text{valid}}}[x\in X\cap\mathbb{S}_n^{\text{valid}}]\cdot\alpha(x)\cdot\zeta(x)\cdot\varphi_{2n}(x)\ \mathsf{Leb}_{2n}(\mathrm{d}x)\right)$$

The second and third integrals are the same since the pushforward measure of Leb_{2n} along the integrator Ψ_n is the same as Leb_{2n} (Ψ_n is volume preserving on $\mathbb{S}_n^{\text{valid}}$) for all n and $\alpha(x)\cdot\zeta(x)\cdot\varphi_{2n}(x)=\alpha(\Psi_n(x))\cdot\zeta(\Psi_n(x))\cdot\varphi_{2n}(\Psi_n(x))$ for all $x\in\mathbb{S}_n^{\text{valid}}$ (all Ψ_n are reversible on $\mathbb{S}_n^{\text{valid}}$). □

Since the transition kernel P of `eNPHMCstep` is the composition of $k_{1,2}$ and $k_{3,4}$, i.e. $P(x, X) := \int_{\mathbb{S}} k_{3,4}(x', X)\ k_{1,2}(x, \mathrm{d}x')$ for $x \in \mathbb{S}$ and $X \in \Sigma_{\mathbb{S}}$, and both $k_{1,2}$ and $k_{3,4}$ are invariant against π (Propositions 25 and 27), we conclude with the following lemma.

Lemma 28. *π is the invariant distribution of the Markov chain generated by iterating* `eNPHMCstep`.

C.3. Marginalised Markov Chains

It is important to notice that the Markov chain $\{(\boldsymbol{q_i}, \boldsymbol{p_i})\}_{i\in\mathbb{N}}$ generated by iterating `eNPHMCstep` with invariant distribution π is *not* the samples we are seeking. The chain we are in fact interested in is the *marginalised* chain $\{f(\boldsymbol{q_i}, \boldsymbol{p_i})\}_{i\in\mathbb{N}}$ where the measurable[10] function f finds the prefix of $\boldsymbol{q}$ which is w-supported, formally defined as

$$\begin{aligned} f: \quad \mathbb{S}^{\text{valid}} &\longrightarrow \mathbb{T} \\ (\boldsymbol{q}, \boldsymbol{p}) &\longmapsto \boldsymbol{q}^{1\ldots n} \quad \text{for } \boldsymbol{q}^{1\ldots n} \in \mathsf{Supp}(w). \end{aligned}$$

This function is realised by the `supported` program in Listing 3.

In this section we show that this marginalised chain has the target distribution ν as its invariant distribution. Let Q : $\mathsf{Supp}(w) \times \Sigma_{\mathbb{T}} \to \mathbb{R}_{\geq 0}$ be the transition kernel of this marginalised chain. We can write it as $Q(f(x), A) = P(x, f^{-1}(A))$ for $x \in \mathbb{S}^{\text{valid}}$ and $A \in \Sigma_{\mathbb{T}}$.

Remark 29. In the standard HMC algorithm, the function f would simply be the first projection, and it is trivial to check that the pushforward of the invariant distribution along the first projection is exactly the target distribution. Hence this step tends to be skipped in the correctness proof of HMC (Neal, 2011; Bou-Rabee & Sanz-Serna, 2018).

Lemma 30. *Writing $\mathbb{S}^{valid}_{\leq n} := \bigcup_{k=1}^{n} \mathbb{S}^{valid}_{k}$, we let π_n be a probability distribution on measurable space $(\mathbb{R}^{2n}, \mathcal{B}^{2n}, \mathcal{N}_{2n})$ given by*

$$\pi_n(X) := \int_X \frac{1}{Z_n} w_{\leq n}(\boldsymbol{q})\ \mathcal{N}_{2n}(d(\boldsymbol{q},\boldsymbol{p})) \qquad \textit{where } Z_n := \int_{\mathbb{R}^n} w_{\leq n}\ d\mathcal{N}_n \textit{ and } X \in \mathcal{B}_{2n}.$$

(1) $\pi(\mathbb{S} \setminus \mathbb{S}^{valid}_{\leq n}) \to 0$ as $n \to \infty$.
(2) For $m \geq n$, $Z_n \cdot \pi_n = Z_m \cdot e^{(m,n)}_{*}\pi_m$ on $\mathbb{S}^{valid}_{n}$ where $e^{(m,n)} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n \times \mathbb{R}^n$ with $e^{(m,n)}(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q}^{1\ldots n}, \boldsymbol{p}^{1\ldots n})$.
(3) $Z \cdot \pi = Z_n \cdot g^{(n)}_{*}\pi_n$ on $\mathbb{S}^{valid}_{\leq n}$ where $g^{(n)} : \mathbb{R}^n \times \mathbb{R}^n \rightharpoonup \mathbb{S}^{valid}_{\leq n}$ such that $g^{(n)}(\boldsymbol{q},\boldsymbol{p}) = (\boldsymbol{q}^{1\ldots k}, \boldsymbol{p}^{1\ldots k}) \in \mathbb{S}^{valid}_{\leq n}$.

Proof. (1) π is an invariant distribution, and hence it is probabilistic. The sum $\sum_{n=1}^{\infty} \pi(\mathbb{S}^{\text{valid}}_n)$ which equals $\pi(\bigcup_{n=1}^{\infty} \mathbb{S}^{\text{valid}}_n) = \pi(\mathbb{S}^{\text{valid}})$ must converge. Hence $\pi(\mathbb{S} \setminus \mathbb{S}^{\text{valid}}_{\leq n}) = \sum_{i=n+1}^{\infty} \pi(\mathbb{S}^{\text{valid}}_i) \to 0$ as $n \to \infty$.
(2) Simple to show.
(3) Let X be a measurable subset of $\mathbb{S}^{\text{valid}}_{\leq n}$. Then,

$$\begin{aligned} Z \cdot \pi(X) &= \sum_{k=1}^{n} Z_k \cdot \pi_k(X \cap \mathbb{S}^{\text{valid}}_k) = Z_n \sum_{k=1}^{n} e^{(n,k)}_{*}\pi_n(X \cap \mathbb{S}^{\text{valid}}_k) \\ &= Z_n \cdot \pi_n\Big(\bigcup_{k=1}^{n} \{(\boldsymbol{q},\boldsymbol{p}) \in \mathbb{R}^{2n} \mid (\boldsymbol{q}^{1\ldots k}, \boldsymbol{p}^{1\ldots k}) \in X \cap \mathbb{S}^{\text{valid}}_k\}\Big) \\ &= Z_n \cdot g^{(n)}_{*}\pi_n(X). \end{aligned}$$

□

Theorem 4. *Given Assumptions 1, 2 and 3, the target distribution ν is the invariant distribution of the Markov chain generated by iterating Alg. 1.*

Proof. For any $A \in \Sigma_{\mathbb{T}}$, if (1) $\nu = f_{*}\pi$ on $\mathbb{T}$ and (2) $\mu_{\mathbb{T}} = f_{*}\mu_{\mathbb{S}}$ on $\mathsf{Supp}(w)$, then

$$\begin{aligned} \nu(A) = f_{*}\pi(A) &= \int_{\mathbb{S}} P(x, f^{-1}(A))\ \mu_{\mathbb{S}}(\mathrm{d}x) && \text{(Lem. 28)} \\ &= \int_{\mathbb{S}^{\text{valid}}} P(x, f^{-1}(A))\ \mu_{\mathbb{S}}(\mathrm{d}x) = \int_{\mathbb{S}^{\text{valid}}} Q(f(x), A)\ \mu_{\mathbb{S}}(\mathrm{d}x) \\ &= \int_{\mathsf{Supp}(w)} Q(q, A)\ f_{*}\mu_{\mathbb{S}}(\mathrm{d}q) = \int_{\mathsf{Supp}(w)} Q(q, A)\ \mu_{\mathbb{T}}(\mathrm{d}q) = \int_{\mathbb{T}} Q(q, A)\ \mu_{\mathbb{T}}(\mathrm{d}q). \end{aligned}$$

[10]For any measurable set $A \in \Sigma_{\mathbb{T}}$, $f^{-1}(A) = \big(\bigcup_{n=1}^{\infty} \bigcup_{m=n}^{\infty} ((A \cap \mathbb{R}^n) \times \mathbb{R}^{m-n}) \times \mathbb{R}^m\big) \cap \mathbb{S}^{\text{valid}}$ is measurable in $\mathbb{S}$.

Hence it is enough to show (1) and (2).

(1) Let $A \subseteq \mathbb{R}^n$ be a measurable set on $\mathbb{T}$ and $\delta > 0$. Then partitioning $f^{-1}(A) = \{(\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{S}^{\text{valid}} \mid \boldsymbol{q}^{1\dots n} \in A\}$ using $\mathbb{S}_k^{\text{valid}}$, we have for sufficiently large m,

$$\begin{aligned} f_*\pi(A) &= \pi\left(\bigcup_{k=1}^{m} f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\right) + \pi\left(\bigcup_{k=m+1}^{\infty} f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\right) \\ &< \frac{Z_m}{Z} \cdot g_*^{(m)}\pi_m\left(\bigcup_{k=1}^{m} f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\right) + \delta && \text{(by Lem. 30 (1) and (3))} \\ &\leq \frac{Z_m}{Z} \cdot \pi_m(A \times \mathbb{R}^{m-n} \times \mathbb{R}^m) + \delta \\ &= \nu(A) + \delta. \end{aligned}$$

For any measurable set $A \in \Sigma_{\mathbb{T}}$, we have $f_*\pi(A) = \sum_{n=1}^{\infty} f_*\pi(A \cap \mathbb{R}^n) \leq \sum_{n=1}^{\infty} \nu(A \cap \mathbb{R}^n) = \nu(A)$. Since both ν and π are probability distributions, we also have $\nu(A) = 1 - \nu(\mathbb{T} \setminus A) \leq 1 - f_*\pi(\mathbb{T} \setminus A) = 1 - (1 - f_*\pi(A)) = f_*\pi(A)$. Hence $f_*\pi = \nu$ on $\mathbb{T}$.

(2) Similarly, let $A \subseteq \mathsf{Supp}^n(w)$ be a measurable set on $\mathbb{T}$ and $\delta > 0$. Then for sufficiently large m, we must have $\mu_{\mathbb{S}}(\bigcup_{k=m+1}^{\infty} \mathbb{S}_k^{\text{valid}}) = \mu_{\mathbb{S}}(\mathbb{S}^{\text{valid}} \setminus \mathbb{S}_{\leq m}^{\text{valid}}) < \delta$. Hence,

$$\begin{aligned} f_*\mu_{\mathbb{S}}(A) &= \mu_{\mathbb{S}}\left(\bigcup_{k=1}^{m} f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\right) + \mu_{\mathbb{S}}\left(\bigcup_{k=m+1}^{\infty} f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\right) \\ &< \sum_{k=1}^{m} \mathcal{N}_{2k}(f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}) + \delta \\ &= \sum_{k=1}^{m} \mathcal{N}_{2m}(\{(\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{R}^{2m} \mid (\boldsymbol{q}^{1\dots k}, \boldsymbol{p}^{1\dots k}) \in f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\}) + \delta \\ &= \mathcal{N}_{2m}(\bigcup_{k=1}^{m} \{(\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{R}^{2m} \mid (\boldsymbol{q}^{1\dots k}, \boldsymbol{p}^{1\dots k}) \in f^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\}) + \delta \\ &\leq \mathcal{N}_{2m}(A \times \mathbb{R}^{m-n} \times \mathbb{R}^m) + \delta \\ &= \mu_{\mathbb{T}}(A) + \delta. \end{aligned}$$

Then the proof proceeds as in (1).

□

C.4. Convergence

Last but not least, we check for the convergence of the marginalised chain to the target distribution ν.

As shown in Ex. 16, it is not trivial that the standard HMC algorithm converges. The same can be said of the NP-HMC algorithm. Recall the conditions on the transition kernel to ensure convergence.

Lemma 19 (Tierney (1994), Theorem 1 and Corollary 2)**.** *If the transition kernel Q with invariant distribution ν is ν-irreducible and ν-aperiodic, then for all $\boldsymbol{q}$,* $\lim_{n\to\infty} \|Q^n(\boldsymbol{q}, -) - \nu\| = 0$.

Recall Q is the transition kernel of the Markov chain generated by iterating Alg. 1 on $\mathsf{Supp}(w)$. In Thm. 4, we have shown that Q has invariant distribution ν. Hence, most of this section is devoted to searching for sufficient conditions (Def. 35) in order to show that the transition kernel Q is ν-irreducible (Lem. 36) and aperiodic (Lem. 37). We conclude in Thm. 5 that this Markov chain converges to the target distribution ν.

We start by extending the result in (Cances et al., 2007) in two ways:

1. The density function is only continuously differentiable *almost everywhere*.
2. The position space is the target space $\mathbb{T}$.

Let $\mathcal{U}$ be the collection of measurable subsets of $\mathbb{T}$ with the property that their boundary has measure zero. Formally, $\mathcal{U} := \{A \in \Sigma_{\mathbb{T}} \mid \mu_{\mathbb{T}}(\partial A) = 0\}$. Not every set in $\Sigma_{\mathbb{T}}$ satisfies this property. A typical example would be the fat Cantor set. It is easy to see that $\mathcal{U}$ is closed under complementation. Moreover, for any non-null set A in $\mathcal{U}$, its interior $\mathring{A}$ is non-empty.

We assume the density function $w : \mathbb{T} \to \mathbb{R}_{\geq 0}$ is continuously differentiable on a non-null set $A \in \mathcal{U}$. We start by showing that the Markov chain can almost surely move between w-supported elements in A.

Lemma 31. *Assume w is continuously differentiable on a non-null set $A \in \mathcal{U}$ and $\{U_n\}$ is uniformly bounded above (i.e. there is an upper bound M, where $U_n(\boldsymbol{q}) < M$ for all $\boldsymbol{q} \in \mathsf{Dom}(U_n)$ for all $n \in \mathbb{N}$). For almost all $\boldsymbol{a}, \boldsymbol{b} \in A \cap \mathsf{Supp}(w)$, there exists some $k \geq \max\{|\boldsymbol{a}|, |\boldsymbol{b}|\}$ and $\boldsymbol{p} \in \mathbb{R}^k$ such that $\mathsf{proj}_1(\Psi_k(\boldsymbol{a} + \boldsymbol{0}^{1\dots k-|\boldsymbol{a}|}, \boldsymbol{p}))^{1\dots|\boldsymbol{b}|} = \boldsymbol{b}$, where $\mathsf{proj}_1(\boldsymbol{q}, \boldsymbol{p}) = \boldsymbol{q}$.*

Proof. Define a function V on the sequence space $\mathbb{R}^\omega$, which is a Fréchet space with a family of semi-norms $\{\|-\|_k\}_{k\in\mathbb{N}}$ where $\|\boldsymbol{x}\|_k = |\boldsymbol{x}_k|$, as

$$\begin{aligned} V : \quad \mathbb{R}^\omega &\longrightarrow \mathbb{R}_{\geq 0} \\ \boldsymbol{x} &\longmapsto -\log \sum_{k=1}^{\infty} w(\boldsymbol{x}^{1\dots k}). \end{aligned}$$

V is well-defined thanks to Assumption 3. Since w is continuously differentiable on A, V is continuously differentiable on the non-empty open set $\hat{A} := \bigcup_{n=1}^{\infty}(\mathring{A} \cap \mathbb{R}^n) \times \mathbb{R}^\omega$. Moreover, V must be bounded above, say by some M.

Now we consider the minimization of the function $S_\epsilon : (\mathbb{R}^\omega)^{L+1} \to \mathbb{R}^\omega$ where ϵ is the leapfrog step size,

$$(S_\epsilon(\boldsymbol{q}^0, \dots, \boldsymbol{q}^L))_k := \epsilon \sum_{i=0}^{L-1} \left(\frac{1}{2}\Big(\frac{\boldsymbol{q}_k^{i+1} - \boldsymbol{q}_k^i}{\epsilon}\Big)^2 - \frac{V(\boldsymbol{q}^{i+1}) + V(\boldsymbol{q}^i)}{2} \right) \qquad \text{for all } k \in \mathbb{N}$$

where $\boldsymbol{q}^0 = \boldsymbol{a} + \boldsymbol{0}$ and $\boldsymbol{q}^L = \boldsymbol{b} + \boldsymbol{0}$. Since V is bounded above by M, for all $\phi \in (\mathbb{R}^\omega)^{L+1}$, each component of $S_\epsilon(\phi) \in \mathbb{R}^\omega$ is bounded below by $-\epsilon(L-1)M$ (i.e. $\forall k \in \mathbb{N}$, $S_\epsilon(\phi)_k > -\epsilon(L-1)M$). Hence, S_ϵ is bounded below. By the completeness of $\mathbb{R}^\omega$, $\inf S_\epsilon \in \mathbb{R}^\omega$ exists.

Consider a minimising sequence $\{\phi_n\}_{n\in\mathbb{N}}$ on $(\mathbb{R}^\omega)^{L+1}$ where $S_\epsilon(\phi_{n+1})_k < S_\epsilon(\phi_n)_k$ for all $n, k \in \mathbb{N}$ and $S_\epsilon(\phi_n) \to \inf S_\epsilon$ as $n \to \infty$. Writing the sequence as $\{(\boldsymbol{q}^{0,n}, \dots, \boldsymbol{q}^{L,n})\}_{n\in\mathbb{N}}$, we say it is bounded on $(\mathbb{R}^\omega)^{L+1}$ if and only if for each $i = 0, \dots, L$, $\{\boldsymbol{q}^{i,n}\}_{n\in\mathbb{N}}$ is a bounded set on $\mathbb{R}^\omega$ which is equivalent to saying that for each $i = 0, \dots, L$ and for all $k \in \mathbb{N}$, $\{\|\boldsymbol{q}^{i,n}\|_k\}_{n\in\mathbb{N}}$ is bounded on $\mathbb{R}$. It is easy to see that for all $n \in \mathbb{N}$ and $i = 1, \dots, L$, $\|\boldsymbol{q}^{i+1,n} - \boldsymbol{q}^{i,n}\|_k \leq 2\epsilon S_\epsilon(\phi_0) + 2\epsilon^2 LM$ and $\|\boldsymbol{q}^{1,n}\|_k \leq 2\epsilon S_\epsilon(\phi_0) + 2\epsilon^2 LM + \|\boldsymbol{q}^0\|_k$, so for any $i = 0, \dots, L$ and $k \in \mathbb{N}$, $\{\|\boldsymbol{q}^{i,n}\|_k\}_{n\in\mathbb{N}}$ is bounded and hence the sequence $\{\phi_n\}_{n\in\mathbb{N}}$ is bounded. Moreover, its closure $\Phi := \overline{\{\phi_n\}_{n\in\mathbb{N}}}$ is bounded and closed.

Note that the Fréchet space $\mathbb{R}^\omega$ is a quasi-complete nuclear space and has the Heine–Borel property, i.e. all closed and bounded set is compact. So, the set Φ is compact. Moreover, since $\mathbb{R}^\omega$ is completely metrisable, the compact set Φ is also sequentially compact, i.e. every sequence in Φ has a subsequence converging to a point in Φ. Hence $\{\phi_n\}_{n\in\mathbb{N}} \subseteq \Phi$ must have a subsequence $\{\phi_{n_k}\}_{k\in\mathbb{N}}$ which converges to some point $\bar{\phi}$ in Φ.

We claim that $\bar{\phi}$ is almost surely in $\hat{A}^{L+1}$. We show that the set $(\mathbb{R}^\omega)^{L+1} \setminus \hat{A}^{L+1}$ has measure zero. First note that by Assumption 2, w is continuously differentiable almost everywhere and hence $\mathbb{T} \setminus A$ is a null set. Moreover, by the definition of $A \in \mathcal{U}$, $\mathbb{T} \setminus \mathring{A}$ is also a null set. Then this implies the set of infinite sequences with no prefixes in $\mathring{A}$ has measure zero, i.e. $\mathbb{R}^\omega \setminus \hat{A}$ is a null set. Hence $(\mathbb{R}^\omega)^{L+1} \setminus \hat{A}^{L+1} = \{(\boldsymbol{q}^0, \dots, \boldsymbol{q}^L) \in (\mathbb{R}^\omega)^{L+1} \mid \exists i \,.\, \boldsymbol{q}^i \notin \hat{A}\} = \bigcup_{i=0}^{L} (\mathbb{R}^\omega)^i \times (\mathbb{R}^\omega \setminus \hat{A}) \times (\mathbb{R}^\omega)^{L-i}$ has zero measure.

Since $\bar{\phi}$ is constrained by $\boldsymbol{q}^0 = \boldsymbol{a} + \boldsymbol{0}$ and $\boldsymbol{q}^L = \boldsymbol{b} + \boldsymbol{0}$, there can only be a null set of $\boldsymbol{a}, \boldsymbol{b} \in A \cap \mathsf{Supp}(w)$ which induces $\bar{\phi}$ in the null set $(\mathbb{R}^\omega)^{L+1} \setminus \hat{A}^{L+1}$. Hence $\bar{\phi}$ is almost surely in $\hat{A}^{L+1}$.

Assume $\bar{\phi}$ is in $\hat{A}^{L+1}$. Since V is continuously differentiable on $\hat{A}$, so is S_ϵ on $\hat{A}^{L+1}$. By the continuity of S_ϵ, we have $\inf S_\epsilon = \lim_{k\to\infty} S_\epsilon(\phi_{n_k}) = S_\epsilon(\lim_{k\to\infty} \phi_{n_k}) = S_\epsilon(\bar{\phi})$, so S_ϵ attains its infimum on $\hat{A}^{L+1}$.

By Prop. 32, the gradient of S_ϵ at its infimum $\bar{\phi} = (\bar{\boldsymbol{q}}^0, \dots, \bar{\boldsymbol{q}}^L)$ is $\boldsymbol{0}$. Hence $\bar{\boldsymbol{q}}^0 = \boldsymbol{a} + \boldsymbol{0}$, $\bar{\boldsymbol{q}}^L = \boldsymbol{b} + \boldsymbol{0}$ and

$$\bar{\boldsymbol{q}}^{i+1} = 2\bar{\boldsymbol{q}}^i - \bar{\boldsymbol{q}}^{i-1} - \epsilon^2 \nabla V(\bar{\boldsymbol{q}}^i) \qquad \text{for } i = 1, \dots, L-1$$

which is the solution to the leapfrog steps. In other words, the infimum $\bar{\phi}$ gives a path from $\boldsymbol{a} + \boldsymbol{0}$ to $\boldsymbol{b} + \boldsymbol{0}$ via the leapfrog trajectory with initial momentum $\boldsymbol{p} = \frac{1}{\epsilon}(\bar{\boldsymbol{q}}^1 - \boldsymbol{a} + \boldsymbol{0}) + \frac{\epsilon}{2}\nabla V(\boldsymbol{a} + \boldsymbol{0})$.

Last but not least, let k be the maximum of k_i's where $w(\bar{\boldsymbol{q}}^{i^{1\dots k_i}}) > 0$ for all $i = 0, \dots, L$. Then it is easy to see that $\mathsf{proj}_1(\Psi_k(\boldsymbol{a} + \boldsymbol{0}^{1\dots k-|\boldsymbol{a}|}), \boldsymbol{p}^{1\dots k})^{1\dots|\boldsymbol{b}|} = \boldsymbol{b}$. $\square$

Proposition 32. *Let $f : \mathbb{R}^\omega \to \mathbb{R}^\omega$ be a function with infimumat $x_0 \in \mathbb{R}^\omega$ and is continuously differentiable on $A \subseteq \mathbb{R}^\omega$ where $x_0 \in A$, then $\nabla f(x_0)$ is the zero map, i.e. $\nabla f(x_0)(h) = \mathbf{0}$ for all $h \in \mathbb{R}^\omega$.*

Proof. First note that f is continuously differentiable at $x_0 \in A$ means that for any $\epsilon > 0$ there exists an $\delta > 0$ such that for any $k \in \mathbb{N}$ and $x \in \mathbb{R}^\omega$ such that $\|x - x_0\|_k < \delta$ implies $\frac{\|f(x)-f(x_0)-L(x-x_0)\|_\ell}{\|x-x_0\|_k} < \epsilon$ for all $\ell \in \mathbb{N}$, where $L : \mathbb{R}^\omega \to \mathbb{R}^\omega$ is the bounded linear map defined as $L := (Df)(x_0)$. [11]

Assume for contradiction that L is not a zero map. i.e. There exists some $h \in \mathbb{R}^\omega$ such that $Lh \neq 0$. Let k be the coordinate such that $(Lh)_k \neq 0$ and $\epsilon > 0$.

Since x_0 is an infimum of f, $f(x)_\ell \geq f(x_0)_\ell$ for all $\ell \in \mathbb{N}$ and $x \in \mathbb{R}^\omega$. Moreover, f is continuously differentiable at x_0 so there exists an $\delta > 0$ such that for any $x \in \mathbb{R}^\omega$, $\|x - x_0\|_k < \delta$ implies $\frac{\|f(x)-f(x_0)-L(x-x_0)\|_\ell}{\|x-x_0\|_k} < \epsilon$ for all $\ell \in \mathbb{N}$.

Consider the sequence $\{y_n\}_{n\in\mathbb{N}}$ defined as $y_n := x_0 - \frac{1}{n}\frac{Lh}{\|Lh\|_k} \cdot h$. The distance between y_n and x_0 is $\|y_n - x_0\|_k = \left\|\frac{-1}{n}\frac{Lh}{\|Lh\|_k} \cdot h\right\|_k = \frac{1}{n}\|h\|_k$. So for large enough n, $\|y_n - x_0\|_k < \delta$.

Hence,

$$0 \leq \frac{(f(y_n) - f(x_0))_k}{\|y_n - x_0\|_k} < \frac{L(y_n - x_0)_k}{\|y_n - x_0\|_k} + \epsilon = \frac{n}{\|h\|_k} \cdot \left(\frac{-1}{n}\frac{(Lh)^2}{\|Lh\|_k}\right)_k + \epsilon = -\frac{\|Lh\|_k}{\|h\|_k} + \epsilon$$

which implies $\|Lh\|_k < \|h\|_k\epsilon$. Since ϵ is arbitrary, we have $\|Lh\|_k \leq 0$ which implies $(Lh)_k = 0$ and contradicts our assumption. □

Now we show that the Markov chain can move to any measurable set with positive measure on A from almost all w-supported element in A.

Lemma 33. *Assuming w is continuously differentiable on a non-null set $A \in \mathcal{U}$ and $\{U_n\}$ is uniformly bounded above (i.e. there is an upper bound M, where $U_n(\boldsymbol{q}) < M$ for all $\boldsymbol{q} \in \mathsf{Dom}(U_n)$ for all $n \in \mathbb{N}$) and ∇U_n is Lipschitz on $A \cap \mathsf{Dom}(U_n)$. For almost all $\boldsymbol{a} \in A \cap \mathsf{Supp}(w)$ and measurable subset $B \subseteq A$, $\nu(B) > 0$ implies $Q(\boldsymbol{a}, B) > 0$.*

Proof. It is enough to prove the statement for a non-null measurable set $B \subseteq A \cap \mathbb{R}^n$ where all elements of B have positive weight since all measurable subset B of A with $\nu(B) > 0$ must contains such a subset. Moreover we restrict B to the elements where the statement in Lem. 31 always hold w.r.t. $\boldsymbol{a}$.

Say $m = |\boldsymbol{a}|$ and $M = \max\{m, n\}$. Let $I_{\boldsymbol{a}}(B) = \{\boldsymbol{p} \in \mathbb{R}^k \mid k \geq M$ and all intermediate leapfrog steps starting from $(\boldsymbol{a} + \mathbf{0}^{1\dots k-m}, \boldsymbol{p}) \in \mathbb{S}^{\text{valid}}$ are in $A \cap \mathsf{Dom}(U_k)$ and $\mathsf{proj}_1(\Psi_k(\boldsymbol{a} + \mathbf{0}^{1\dots k-m}, \boldsymbol{p}))^{1\dots n} \in B\}$. It is enough to show that $\sum_{k=M}^\infty \mathsf{Leb}_k(I_{\boldsymbol{a}}(B) \cap \mathbb{R}^k) > 0$.

Let $\theta : I_{\boldsymbol{a}}(B) \to B$ be the function where $\theta(\boldsymbol{p})$ gives the next sample in B after L HMC leapfrog steps starting with initial state $(\boldsymbol{a} + \mathbf{0}^{1\dots|\boldsymbol{p}|-m}, \boldsymbol{p})$. By Lem. 31, θ is subjective.

We write $I^k_{\boldsymbol{a}}(B) = I_{\boldsymbol{a}}(B) \cap \mathbb{R}^k$ and show that $\theta_k : I^k_{\boldsymbol{a}}(B) \to B$ is Lipschitz. By assumption for any $\boldsymbol{p}^0 \in I^k_{\boldsymbol{a}}(B)$, all the intermediate positions are in $\mathsf{Dom}(U_k) \cap A$. Hence, we can write $\theta_k(\boldsymbol{p}) := \mathsf{proj}_1(\Psi_k(\boldsymbol{a} + \mathbf{0}^{1\dots k-m}, \boldsymbol{p})) = \boldsymbol{q}^L$ as

$$\boldsymbol{q}^0 + \epsilon L\boldsymbol{p}^0 - \epsilon^2\Big(\frac{L}{2}\nabla U_k(\boldsymbol{q}^0) + \sum_{k=1}^{L-1} k\nabla U_k(\boldsymbol{q}^{L-k})\Big).$$

Let $\boldsymbol{p}, \boldsymbol{p'} \in I^k_{\boldsymbol{a}}(B)$, and $\boldsymbol{q}^i, \boldsymbol{q'}^i$ be the position of the state after i leapfrog steps with momentum kick $\boldsymbol{p}, \boldsymbol{p'}$ respectively. Then,

$$\begin{aligned}|\theta_k(\boldsymbol{p}) - \theta_k(\boldsymbol{p'})| = |\boldsymbol{q}^L - \boldsymbol{q'}^L| &\leq \epsilon L|\boldsymbol{p} - \boldsymbol{p'}| + \epsilon^2 \sum_{i=1}^{L-1} i|\nabla U_k(\boldsymbol{q}^{L-i}) - \nabla U_k(\boldsymbol{q'}^{L-i})| \\ &\leq \epsilon L|\boldsymbol{p} - \boldsymbol{p'}| + \epsilon^2 \sum_{i=1}^{L-1} i|\boldsymbol{q}^{L-i} - \boldsymbol{q'}^{L-i}| \qquad (U_k \text{ is Lipschitz on } A \cap \mathsf{Dom}(U_k))\end{aligned}$$

[11] This can be easily seen by substituting h by $\frac{x-x_0}{\|x-x_0\|_k}$ in the standard definition of continuously differentiable functions f on $A \subseteq \mathbb{R}^\omega$.

hence $|\theta_k(\boldsymbol{p}) - \theta_k(\boldsymbol{p}')| \le c|\boldsymbol{p} - \boldsymbol{p}'|$ for some constant c and θ_k is Lipschitz.

Assume for contradiction that $\sum_{k=M}^{\infty} \mathsf{Leb}_k(I_{\boldsymbol{a}}(B) \cap \mathbb{R}^k) = 0$ which means that for all $k \ge M$, $\mathsf{Leb}_k(I^k_{\boldsymbol{a}}(B)) = 0$. However,

$$\begin{aligned}\mathsf{Leb}_n(B) = \mathsf{Leb}_n(\theta(I_{\boldsymbol{a}}(B))) = \mathsf{Leb}_n(\theta(\bigcup_{k=M}^{\infty} I^k_{\boldsymbol{a}}(B))) = \mathsf{Leb}_n(\bigcup_{k=M}^{\infty} \theta_k(I^k_{\boldsymbol{a}}(B)))\\ \le \sum_{k=M}^{\infty} \mathsf{Leb}_n(\theta_k(I^k_{\boldsymbol{a}}(B))) \le \sum_{k=M}^{\infty} \mathsf{Lip}(\theta_k)^{3N} \cdot \mathsf{Leb}_n(I^k_{\boldsymbol{a}}(B)) = 0\end{aligned}$$

implies that $\mathsf{Leb}_n(B) = 0$ which gives a contradiction. □

Lemma 34. *Assuming w is continuously differentiable on a non-null set A where $A \in \mathcal{U}$ and $\{\nabla U_n\}$ is uniformly bounded above and below (i.e. there are bounds M_1, M_2, where $M_1 \le \nabla U_n(\boldsymbol{q}) \le M_2$ for all $\boldsymbol{q} \in \mathsf{Dom}(\nabla U_n)$ for all $n \in \mathbb{N}$). Then there exists a step size ϵ such that for any sequence $\boldsymbol{q} \in \mathsf{Supp}(w)$, $\nu(A) > 0$ implies $Q(\boldsymbol{q}, A) > 0$.*

Proof. Let $\boldsymbol{q} \in \mathbb{R}^m$ be w-supported. Since $A \in \mathcal{U}$, its interior $\mathring{A}$ is an non-empty open set. Hence for some n, there is an non-empty open subset $\prod_{i=1}^n (a_i, b_i)$ of $A \cap \mathbb{R}^n$.

Now we consider the conditions on the starting momentum $\boldsymbol{p}^0$ in order for the position $\boldsymbol{q}^L$ at the end of the trajectory of the leapfrog steps to be in A assuming that the position of the intermediate states never leave the domain of U_k for some $k \ge M := \max\{m, n\}$.

$$\begin{aligned}\boldsymbol{q}^L \in \prod_{i=1}^n (a_i, b_i) \times \mathbb{R}^{k-n} \quad &\Leftrightarrow \quad \forall i = 1, \dots, n \quad \boldsymbol{q}^0_i + \epsilon L \boldsymbol{p}^0_i - \epsilon^2\Big(\frac{L}{2}\nabla U_k(\boldsymbol{q}^0) + \sum_{k=1}^{L-1} k \nabla U_k(\boldsymbol{q}^{L-k})\Big) \in (a_i, b_i)\\ &\Leftarrow \quad \forall i = 1, \dots, n \quad \boldsymbol{p}^0_i \in \Big(\frac{1}{\epsilon L}(a_i - \boldsymbol{q}^0_i + \frac{(\epsilon L)^2}{2} M_2), \frac{1}{\epsilon L}(b_i - \boldsymbol{q}^0_i + \frac{(\epsilon L)^2}{2} M_1)\Big) =: I_i\end{aligned}$$

For any $\boldsymbol{p} \in \prod_{i=1}^n I_i$, the union $\bigcup_{k=M}^{\infty} \{\boldsymbol{p}' \in \mathbb{R}^{k-n} \mid (\boldsymbol{q} + \boldsymbol{0}^{k-m}, \boldsymbol{p} + \boldsymbol{p}') \in \mathbb{S}^{\text{valid}}\}$ is non-null. This is because the measure of the union can be seen as the value measure of the almost surely terminating probabilistic program which given $\boldsymbol{q} \in \mathsf{Supp}^m(w)$ and $\boldsymbol{p} \in \prod_{i=1}^n I_i$ returns $\boldsymbol{p}' \in \mathbb{R}^{k-n}$ such that $(\boldsymbol{q} + \boldsymbol{0}^{k-m}, \boldsymbol{p} + \boldsymbol{p}')$ is a valid state,

For $\epsilon < \frac{1}{L}\sqrt{\frac{2(b_i - a_i)}{M_2 - M_1}}$ for all i, the intervals $\{I_i\}$ are non-empty and hence $Q(\boldsymbol{q}, A) \ge \sum_{k=M}^{\infty} \mathcal{N}_k(\{\boldsymbol{p}' \in \mathbb{R}^{k-n} \mid (\boldsymbol{q} + \boldsymbol{0}^{k-m}, \boldsymbol{p} + \boldsymbol{p}') \in \mathbb{S}^{\text{valid}}, \boldsymbol{p} \in \prod_{i=1}^n I_i\}) > 0$. □

Definition 35. We gather all the conditions so far.

(C1) w is continuously differentiable on a non-null set A with measure-zero boundary.
(C2) $w|_{\mathsf{Supp}(w)}$ is bounded below by a positive constant.
(C3) For each n, the function $\frac{\nabla w_{\le n}}{w_{\le n}}$ is uniformly bounded from above and below on $\mathsf{Supp}(w_{\le n}) \cap A$.
(C4) For each n, the function $\frac{\nabla w_{\le n}}{w_{\le n}}$ is Lipschitz continuous on $\mathsf{Supp}(w_{\le n}) \cap A$.

Note that

(C1) implies w is continuously differentiable on a non-null set $A \in \mathcal{U}$.
(C2) implies $\{U_n\}$ is uniformly bounded above (i.e. there is an upper bound M, where $U_n(\boldsymbol{q}) < M$ for all $\boldsymbol{q} \in \mathsf{Dom}(U_n)$ for all $n \in \mathbb{N}$).
(C3) implies $\{\nabla U_n\}$ is uniformly bounded above and below (i.e. there are bounds M_1, M_2, where $M_1 \le \nabla U_n(\boldsymbol{q}) \le M_2$ for all $\boldsymbol{q} \in \mathsf{Dom}(\nabla U_n)$ for all $n \in \mathbb{N}$).
(C4) implies ∇U_n is Lipschitz on $A \cap \mathsf{Dom}(U_n)$.

Now we are ready to prove irreducibility.

Lemma 36 (Irreducible)**.** *If Assumptions (C1)–(C4) are satisfied, there exists a step size ϵ such that for any sequence $\boldsymbol{q} \in \mathsf{Supp}(w)$ and measurable set $B \in \Sigma_{\mathbb{T}}$, $\nu(B) > 0$ implies $Q^i(\boldsymbol{q}, B) > 0$ for $i \in \{1, 2\}$.*

Proof. Let A be the non-null set in $\mathcal{U}$ where w is continuously differentiable on A *and* $\mu_{\mathbb{T}}(\mathbb{T} \setminus A) = 0$ *and* Lem. 33 holds for all elements in A. Such A must exist by Assumption 2 and (C1).

First note that $\nu(A \cap B) > 0$. Otherwise, we must have $\nu((\mathbb{T} \setminus A) \cap B) > 0$. But this implies $\mu_{\mathbb{T}}(\mathbb{T} \setminus A) \geq \mu_{\mathbb{T}}((\mathbb{T} \setminus A) \cap B) > 0$ which contradicts the assumption.

We do case analysis on $\boldsymbol{q} \in \mathbb{T}$.

- If $\boldsymbol{q} \in A$, then by Lem. 33, $Q(\boldsymbol{q}, A \cap B) > 0$.
- If $\boldsymbol{q} \notin A$, then by Lem. 34, $Q(\boldsymbol{q}, A) > 0$ and so

$$\begin{aligned} Q^2(\boldsymbol{q}, B) &\geq Q^2(\boldsymbol{q}, A \cap B) \\ &= \int_{\mathbb{T}} Q(\boldsymbol{q}', A \cap B)\, Q(\boldsymbol{q}, \mathrm{d}\boldsymbol{q}') \\ &\geq \int_{A} Q(\boldsymbol{q}', A \cap B)\, Q(\boldsymbol{q}, \mathrm{d}\boldsymbol{q}') > 0. \end{aligned}$$

□

Lemma 37 (Aperiodic)**.** *If Assumptions (C1)–(C4) are satisfied, Q is aperiodic.*

Proof. Assume for contradiction that Q is not aperiodic. Then, there exists disjoint $B_0, \dots, B_d$ for $d \geq 1$ such that $\nu(B_0) > 0$ and $x \in B_i$ implies $Q(x, B_{(i+1) \bmod (d+1)}) = 1$ for all $i = 0, \dots, d$.

Let A be the non-null set in $\mathcal{U}$ where w is continuously differentiable on A *and* $\mu_{\mathbb{T}}(\mathbb{T} \setminus A) = 0$ *and* Lem. 34 holds for all elements in A. Such A must exist by Assumption 2 and (C1). Let $C_i := B_i \cap A$ for all $i = 0, \dots, d$. Hence, $\nu(C_0) > 0$ and $x \in C_i$ implies $Q(x, C_{(i+1) \bmod (d+1)}) = 1$ for all $i = 0, \dots, d$.

Let $x \in C_0$ be a w-supported sequence. Such an x must exist as $\nu(C_0) > 0$. Then, $Q(x, C_1) = 1$ implies $Q(x, C_0) \leq Q(x, \mathbb{T} \setminus C_1) = 0$ which contradicts with Lem. 36 as $x \in A$.

□

Finally by Tierney's Theorem (Lem. 19), the ν-irreducible (Lem. 36) and ν-aperiodic (Lem. 37) transition kernel Q with invariant distribution ν (Thm. 4) converges to ν.

Theorem 5. *If Assumptions (C1)–(C4) are satisfied in addition to Assumptions 1, 2 and 3, the Markov chain generated by iterating Alg. 1 converges to the target distribution ν.*

D. Experiments

D.1. Details on the Experimental Setup

For our experimental evaluation, we implemented the algorithms in Python, using PyTorch for tensor and gradient computations. The source code for our implementation and experiments is available at https://github.com/fzaiser/nonparametric-hmc and archived as (Zaiser & Mak, 2021).

Inference algorithms The four inference algorithms we compared were:

1. NP-DHMC (ours): the nonparametric adaptation of (Nishimura et al., 2020), explained in App. B.2, using the efficiency improvements from App. B.3.
2. Lightweight Metropolis-Hastings (LMH),
3. Particle Gibbs (PGibbs) and
4. Random walk lightweight Metropolis-Hastings (RMH).

We used the Anglican implementations of the latter three algorithms.

Models For NP-DHMC, the models were given to the algorithm as probabilistic programs in the form of a Python function with a context argument for NP-DHMC. The context allows probabilistic primitives and records the trace and weight for the inference algorithms. This way, evaluating the density function w amounts to running the probabilistic programs. For LMH, PGibbs, and RMH, the Python models were translated to Clojure programs using Anglican's probabilistic programming constructs. The pseudocode for the geometric example and the random walk example can be found in the main text. The Gaussian and Dirichlet process mixture model is explained there as well, using statistical notation. Sampling from $\mathrm{DP}(\alpha, \mathrm{Uniform}([0,1]^3))$ is implemented using the stick-breaking procedure (Sethuraman, 1994). We use a cutoff of $\epsilon = 0.01$ for the stick size as explained in the text. In pseudocode, it looks as follows:

```
def dp(alpha, H):
    stick = 1.0
    beta = 0.0
    cumulative_product = 1.0
    weights = []
    means = []
    while stick > 0.01:
        cumulative_product *= 1 - beta
        beta = sample(Beta(1, alpha))
        theta = sample(H)
        weights.append(beta * cumulative_product)
        means.append(theta)
        stick -= beta * cumulative_product
    return weights, means
```

ESS computation For the random walk example, we computed the effective sample size. For this we used NumPyro's (Bingham et al., 2019) `diagnostics. effective_sample_size` function. It is designed to estimate the effective sample size for MCMC samplers using autocorrelation (Gelman et al., 2014). For importance samples used as the ground truth, we used the importance weights directly to compute the ESS: given importance weights $w_1, \ldots, w_n$, the ESS is $\frac{(\sum_{i=1}^n w_i)^2}{\sum_{i=1}^n w_i^2}$. We also computed the (autocorrelation-based) MCMC ESS for the importance samples and we obtained very similar results.

Hyperparameter choices We produced 10 runs with 1000 samples each for every example except the last, Dirichlet process mixture model (DPMM). For the DPMM example, we only produced 100 samples in each run because of the forbidding computational cost. We set the number of burn-in samples that are discarded to 10% of the total number of samples, i.e. 100 samples for each run. Since each run of the DPMM only had 100 samples, we set the burn-in higher there, namely to 50. We did not vary this hyperparameter much because higher values did not seem to make a difference. For the number of leapfrog steps we tried values $L \in \{5, 20, 50, 100\}$, and for the step size we tried values $\epsilon \in \{0.01, 0.05, 0.1, 0.5\}$. Generally, the simple geometric distribution example already works for very rough hyperparameters ($L = 5, \epsilon = 0.1$). Finer steps work as well, but are not necessary. However, more complex models generally require finer steps (GMM: $L = 50, \epsilon = 0.05$). The other inference algorithms we tested don't have any hyperparameters that need to be set.

Thinning Since NP-DHMC performs more computation than its competitors for each sample because it evaluates the density function in each of the L leapfrog steps, not just once like the other inference algorithms. To equalise the computation budgets, we generate L times as many samples for each competitor algorithm, and apply thinning (taking every L-th sample) to get a comparable sample size.

D.2. Additional Plots and Data

In addition to the ESS and LPPD computations, we also plotted both as a variable of the number of samples computed. The results can be seen in Fig. 12 and 13. As we can see, NP-DHMC performs the best consistently over the course of the inference, not just in terms of the final result.

Running time We report the wall-clock times for the different algorithms. Experiments were carried out on a computer with an Intel Core i7-8700 CPU @ 3.20 GHz x 12 and 16 GB RAM, running Ubuntu 20.04. The results are presented in Table 2.

NP-DHMC is significantly slower than the competition in the geometric and random walk examples, faster for GMM and comparable for DPMM. Due to the nature of the coordinate integrator of discontinuous HMC (Nishimura et al., 2020), NP-DHMC has to run the model $L \times d$ times per sample where d is the number of discontinuous variables in the model. We

Table 2. Running times for the different inference algorithms in seconds per sample.

method	ours	LMH	PGibbs	RMH	Pyro HMC	Pyro NUTS
geometric example	0.0418	0.0003	0.0001	0.0005	n/a	n/a
random walk example	0.2266	0.0077	0.0051	0.0095	≈ 0.41	≈ 5.7
GMM example	0.1879	1.6572	1.6835	1.6376	n/a	n/a
DPMM example	1.8516	2.1491	1.7855	2.0584	n/a	n/a

could improve the algorithm by only updating a subset of the discontinuous variables per iteration. In addition, NP-DHMC computes gradients and simulates Hamiltonian dynamics, which is computationally expensive. On the random walk example we also ran Pyro HMC and NUTS, as mentioned before. Both of them were a lot slower than our implementation, which speaks to the fact that HMC methods simply have an unavoidable performance overhead. Finally, the implementation of NP-DHMC is a research prototype, so it is not optimal and there is a lot of room for improvement.

Algorithm 6 NP-RHMC Integrator $\Psi_{\mathsf{NP-R}}$

```
Input: current state $(\boldsymbol{q_0}, \boldsymbol{p_0})$, family of potential energies $\{U_n\}_{n\in\mathbb{N}}$, step size $\epsilon$, number of steps $L$
Output: new state $(\boldsymbol{q}, \boldsymbol{p})$ computed according to Hamiltonian dynamics, extended initial state $(\boldsymbol{q}_0, \boldsymbol{p}_0)$
$(\boldsymbol{q}, \boldsymbol{p}) = (\boldsymbol{q_0}, \boldsymbol{p_0})$                    {initialise}
for $i = 0$ to $L$ do
  $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U_{|\boldsymbol{q_0}|}(\boldsymbol{q})$                {1/2 momentum step}
  $t = 0$                                  {start of position step}
  while nextBoundary$(\boldsymbol{q}, \boldsymbol{p}, \epsilon - t, U)$ exists do
    $(t', \boldsymbol{q}_<, \boldsymbol{q}_>)$ = nextBoundary$(\boldsymbol{q}, \boldsymbol{p}, \epsilon - t, U)$
    $t = t + t'$
    $((\boldsymbol{q}', \boldsymbol{p}'), (\boldsymbol{q}_0', \boldsymbol{p}_0'))$ = extend$((\boldsymbol{q}_>, \boldsymbol{p}), (\boldsymbol{q}_0, \boldsymbol{p}_0), i\epsilon + t, U)$
    $\Delta U = (U_{|\boldsymbol{q}'|}(\boldsymbol{q}') - U_{|\boldsymbol{q}_<|}(\boldsymbol{q}_<))$
    if $\|\boldsymbol{p}_\perp\|^2 > 2\Delta U$ then
      $(\boldsymbol{p}_\|, \boldsymbol{p}_\perp)$ = decompose$(\boldsymbol{q}', \boldsymbol{p}', U)$
      $\boldsymbol{p}_\perp = \sqrt{\|\boldsymbol{p}_\perp\|^2 - 2\Delta U}\frac{\boldsymbol{p}_\perp}{\|\boldsymbol{p}_\perp\|}$            {refraction}
      $\boldsymbol{q} = \boldsymbol{q}'$
    else
      $(\boldsymbol{p}_\|, \boldsymbol{p}_\perp)$ = decompose$(\boldsymbol{q}_<, \boldsymbol{p}, U)$
      $\boldsymbol{p}_\perp = -\boldsymbol{p}_\perp$                       {reflection}
      $\boldsymbol{q} = \boldsymbol{q}_<$
    end if
    $\boldsymbol{p} = \boldsymbol{p}_\perp + \boldsymbol{p}_\|$
  end while
  $\boldsymbol{q} = \boldsymbol{q} + (\epsilon - t)\boldsymbol{p}$                {rest of position step}
  $\boldsymbol{p} = \boldsymbol{p} - \frac{\epsilon}{2}\nabla U_{|\boldsymbol{q}|}(\boldsymbol{q})$                {1/2 momentum step}
end for
$\boldsymbol{p} = -\boldsymbol{p}$
return $((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q_0}, \boldsymbol{p_0}))$
```

Algorithm 7 extend for NP-DHMC

```
Input: current state $(\boldsymbol{q}, \boldsymbol{p})$, initial state $(\boldsymbol{q_0}, \boldsymbol{p_0})$, time $t$, family of potential energies $U = \{U_n\}_{n\in\mathbb{N}}$ family of potential energies $\{U_n\}_{n\in\mathbb{N}}$, step size $\epsilon$, number of steps $L$
Output: extended current state $(\boldsymbol{q}, \boldsymbol{p})$, extended initial state $(\boldsymbol{q}_0, \boldsymbol{p}_0)$
while $\boldsymbol{q} \notin$ Dom$(U_{|\boldsymbol{q}|})$ do
  $x \sim \mathcal{N}(0,1)$
  if $|q| + 1 \in C$ then
    $y \sim \mathcal{N}(0,1)$          {Gaussian for continuous params}
    $(x_0, y_0) = (x - t\,y, y)$        {update to current time $t$}
  else
    $y_0 \sim \mathcal{L}(0,1)$          {Laplace for discontinuous ones}
    $(x_0, y_0) = (x - t\,\mathrm{sign}(y), y)$ {update to current time $t$}
  end if
  $(\boldsymbol{q_0}, \boldsymbol{p_0}) = (\boldsymbol{q_0} +\!\!+ [x_0], \boldsymbol{p_0} +\!\!+ [y_0])$
  $(\boldsymbol{q}, \boldsymbol{p}) = (\boldsymbol{q} +\!\!+ [x], \boldsymbol{p} +\!\!+ [y])$       {increment dimension}
end while
return $((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q_0}, \boldsymbol{p_0}))$
```

Algorithm 8 NP-DHMC Integrator $\Psi_{\mathsf{NP-Dis}}$

```
Input: current state $(\boldsymbol{q_0}, \boldsymbol{p_0})$, family of potential energies $\{U_n\}_{n\in\mathbb{N}}$, step size $\epsilon$, number of steps $L$, discontinuous coordinates $D$
Output: new state $(\boldsymbol{q}, \boldsymbol{p})$ computed according to Hamiltonian dynamics, extended initial state $(\boldsymbol{q}_0, \boldsymbol{p}_0)$
$(\boldsymbol{q}, \boldsymbol{p}) = (\boldsymbol{q_0}, \boldsymbol{p_0})$                    {initialise}
$\boldsymbol{q}' = \boldsymbol{q}_0$
$\boldsymbol{p}' = \boldsymbol{p}_0$
$N = |\boldsymbol{q}_0|$
for $i = 0$ to $L$ do
  $\boldsymbol{p}_C = \boldsymbol{p}_C - \frac{\epsilon}{2}\nabla_{\boldsymbol{q}_C} U_N(\boldsymbol{q})$
  $\boldsymbol{q}_C = \boldsymbol{q}_C + \frac{\epsilon}{2}\boldsymbol{p}_C$
  for $j \in$ randomlyPermute$(D)$ do
    if $j < |\boldsymbol{q}|$ then
      {$|\boldsymbol{q}|$ may have changed, so must check $j < |\boldsymbol{q}|$}
      $((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q}', \boldsymbol{p}'))$ =
          coordIntegrator$((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q}', \boldsymbol{p}'), j, i\epsilon, \epsilon)$
    end if
  end for
  $N = |\boldsymbol{q}|$
  $\boldsymbol{q}_C = \boldsymbol{q}_C + \frac{\epsilon}{2}\boldsymbol{p}_C$
  $\boldsymbol{p}_C = \boldsymbol{p}_C - \frac{\epsilon}{2}\nabla_{\boldsymbol{q}_C} U_N(\boldsymbol{q})$
end for
$\boldsymbol{p} = -\boldsymbol{p}$
return $((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q}', \boldsymbol{p}'))$

function coordIntegrator$((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q}', \boldsymbol{p}'), j, t, \epsilon)$
$\boldsymbol{q}^* = \boldsymbol{q}$
$q_j^* = q_j^* + \epsilon\mathrm{sign}(p_j)$
$((\boldsymbol{q}^*, \boldsymbol{p}^*), (\boldsymbol{q}'^*, \boldsymbol{p}'^*))$ = extend$((\boldsymbol{q}^*, \boldsymbol{p}^*), (\boldsymbol{q}', \boldsymbol{p}'), t, U)$
$\Delta U = U(\boldsymbol{q}^*) - U(\boldsymbol{q})$
if $|p_j| > \Delta U$ then
  $(\boldsymbol{q}, \boldsymbol{p}) = (\boldsymbol{q}^*, \boldsymbol{p}^*)$       {enough kinetic energy to jump}
  $(\boldsymbol{q}', \boldsymbol{p}') = (\boldsymbol{q}'^*, \boldsymbol{p}'^*)$
  $p_j = p_j - \mathrm{sign}(p_j)\Delta U$
else
  $p_j = -p_j$                {not enough kinetic energy, reflect}
end if
return $((\boldsymbol{q}, \boldsymbol{p}), (\boldsymbol{q}', \boldsymbol{p}'))$
```

Listing 2. Python code for NPHMC

```
def extend((q,p),(q0,p0),t,U):
  while q not in domain(U(len(q))):
    x0 = normal
    y0 = normal
    x = x0 + t*y0
    y = y0
    q0.append(x0)
    p0.append(y0)
    q.append(x)
    p.append(y)
  return ((q,p),(q0,p0))

def NPint((q0,p0),U,ep,L):
  q = q0
  p = p0
  for i in range(L):
    p = p - ep/2*grad(U(len(q0)),q)
    q = q + ep*p
    ((q,p),(q0,p0)) =
      extend((q,p),(q0,p0),i*ep,U)
    p = p - ep/2*grad(U(len(q0)),q)
  return ((q,p),(q0,p0))

def NPHMCstep(q0,w,ep,L):
  # initialisation
  p0 = [normal for i in range(len(q0))]
  U = lambda n: lambda q:
    -log(sum([w(q[:i]) for i in range(n)]))
  # NP-HMC integration
  ((q,p),(q0,p0)) = NPint((q0,p0),U,ep,L)
  # MH acceptance
  if cdfN(normal) < accept((q,p),(q0,p0),w):
    return supported(q,w)
  else:
    return supported(q0,w)

def NPHMC(q0,w,ep,L,M):
  S = [q0]
  for i in range(M):
    S.append(NPHMCstep(S[i],w,ep,L))
  return S
```

Listing 3. Python code for helper functions

```
# the MH acceptance ratio
def accept((q,p),(q0,p0),w):
  N = len(q)
  N_trunc = lambda q':
    sum([w(q'[:i]) for i in range(N)])
  weight = (N_trunc(q)*pdfN((q,p),2N))/
           (N_trunc(q0)*pdfN((q0,p0),2N))
  return min(1,weight)

# the w-supported prefix of q
def supported(q,w):
  k = 1
  while w(q[:k]) == 0 and k < len(q):
    k += 1
  return q[:k]
```

Listing 4. Python code for eNPHMC

```
def validstate((q0,p0),U,ep,L):
  q = q0
  p = p0
  for i in range(L):
    p = p - ep/2*grad(U,q)
    q = q + ep*p
    if q not in domain(U):
      return False
    p = p - ep/2*grad(U,q)
  return True

def HMCint((q0,p0),U,ep,L):
  q = q0
  p = p0
  for i in range(L):
    p = p - ep/2*grad(U,q)
    q = q + ep*p
    p = p - ep/2*grad(U,q)
  # momentum flip
  p = -p
  return (q,p)

def eNPHMCstep((q0,p0),w,ep,L):
  # initialisation (step 1)
  q0 = supported(q0,w)
  p0 = [normal for i in range(len(q0))]
  U = lambda n: lambda q:
    -log(sum([w(q[:i]) for i in range(n)]))
  # search (step 2)
  while not validstate((q0,p0),U(len(q0)),ep
      ,L):
    x0 = normal
    y0 = normal
    q0.append(x0)
    p0.append(y0)
  # HMC integration (step 3)
  (q,p) = HMCint((q0,p0),U(len(q0)),ep,L)
  # MH acceptance (step 4)
  if cdfN(normal) < accept((q,p),(q0,p0),w):
    return (q,p)
  else:
    return (q0,p0)

def eNPHMC(q0,w,ep,L,M):
  mc = [(q0,0)]
  for i in range(M):
    mc.append(eNPHMCstep(mc[i],w,ep,L))
  # marginalisation
  S = [supported(q,w) for (q,p) in mc]
  return S
```

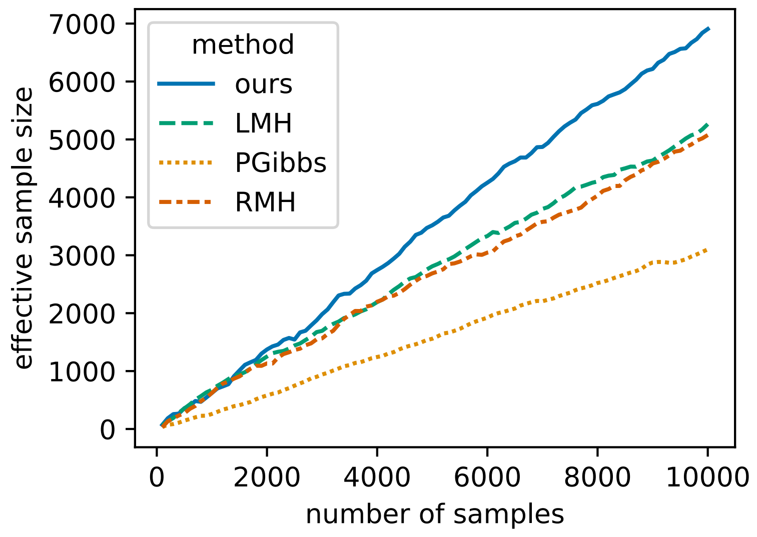


Figure 12. ESS for the random walk example in terms of number of samples

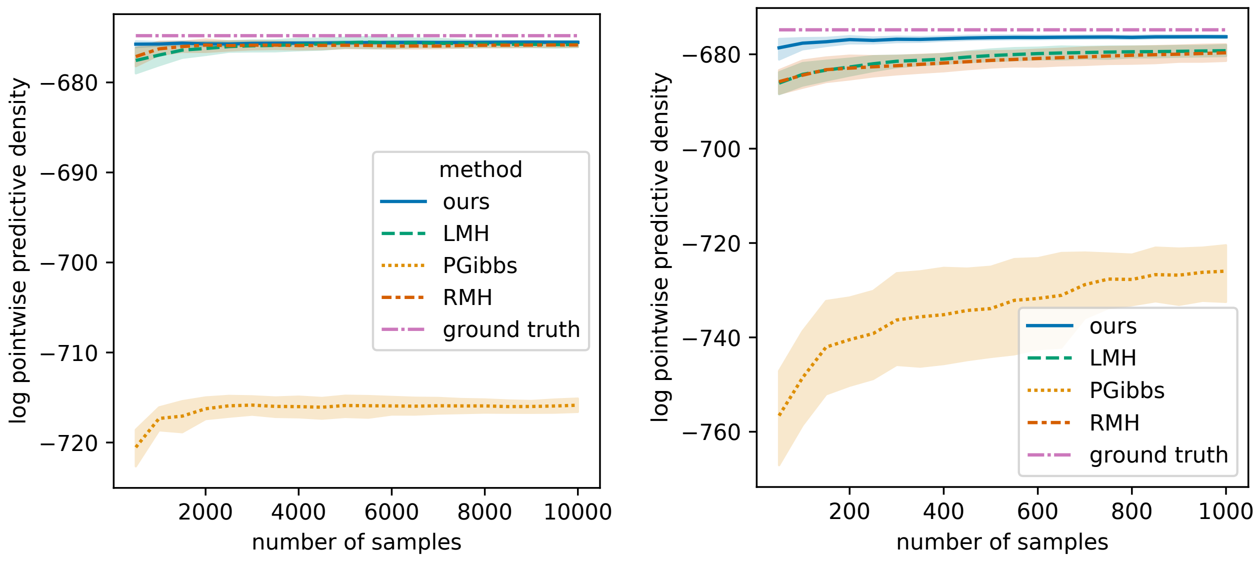


Figure 13. LPPD for the GMM and DP mixture model in terms of the number of samples from 10 runs. The shaded area is one standard deviation. These are the full plots of Fig. 8 and 9, respectively.