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 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

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

```
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.

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 $\mathsf{Supp}(w)$.

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

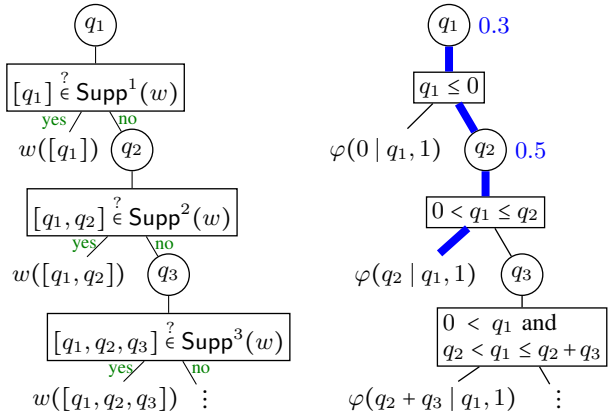


Figure 1. (left) Generic TR function w; (right) TR function w for the probabilistic program in Listing 1.

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

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

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 $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \le 0]$, 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.

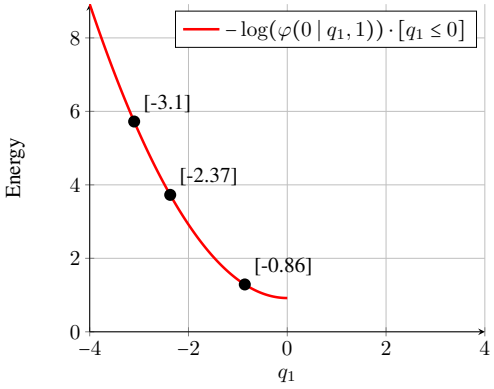
At the third time step, the particle is at the position $[1.15]$, which is no longer on the surface. To search for a suitable state, NP-HMC increments the dimension of the current surface (line) as follows.

First, the surface is extended to $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \le 0] - \log(\varphi(q_2 \mid q_1, 1)) \cdot [0 < q_1 \le q_2]$, 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 $\{[q_1, q_2] \in \mathbb{R}^2 \mid q_1 \le 0\}$ and $\{[q_1, q_2] \in \mathbb{R}^2 \mid 0 < q_1 \le q_2\}$, are disjoint; and hence the states of the particle on the previous surface $-\log(\varphi(0 \mid q_1, 1)) \cdot [q_1 \le 0]$ 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.

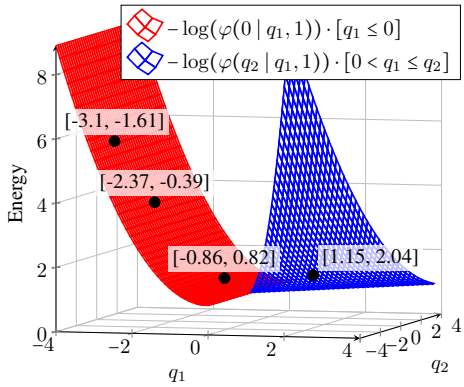
Assumption. Henceforth we assume that the input TR function w satisfies the following:

[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 nonparametric 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 ϵ, 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 $\boldsymbol{q}^{1\ldots k}$ where $w(\boldsymbol{q}^{1\ldots k}) > 0$
else
  return $\boldsymbol{q_0}^{1\ldots k}$ where $w(\boldsymbol{q_0}^{1\ldots k}) > 0$
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 ϵ, 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

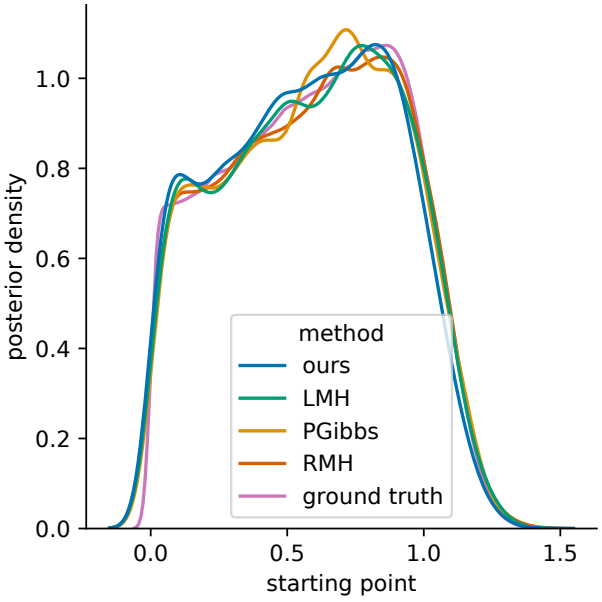
Geometric distribution A classic example to illustrate recursion in a universal PPL is sampling from a geometric distribution with parameter p by repeatedly flipping a biased coin with probability p (see e.g. Ch. 5 in (van de Meent et al., 2018)). The pseudocode for it is:

```
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.

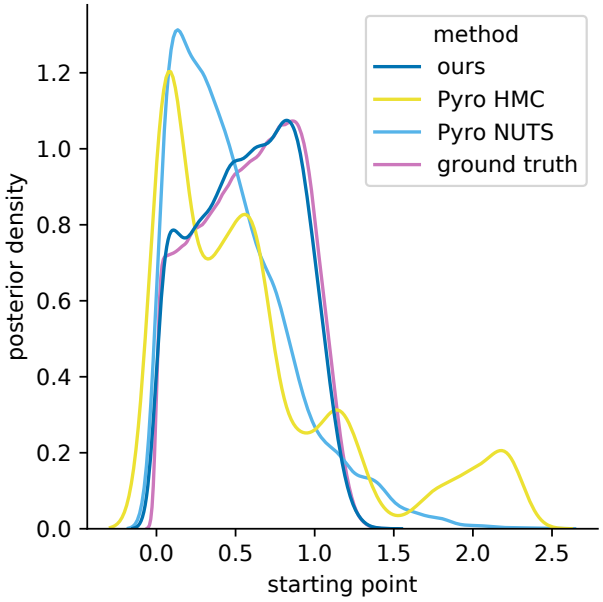


Figure 6. Kernel density estimate for the random walk example compared to Pyro, averaged over 10 runs. Each run: 10^3 samples with 10^2 burn-in, 50 leapfrog steps of size 0.1.

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.

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

This model samples parameters $\theta = (K \in \mathbb{N}, \mu_k \in [0, 100]^3)$ and N data points $X = \{x_1, \dots, x_N\} \subseteq \mathbb{R}^3$. 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 $\theta^* = (K^* = 9, \mu^*_{1 \dots K^*})$. 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

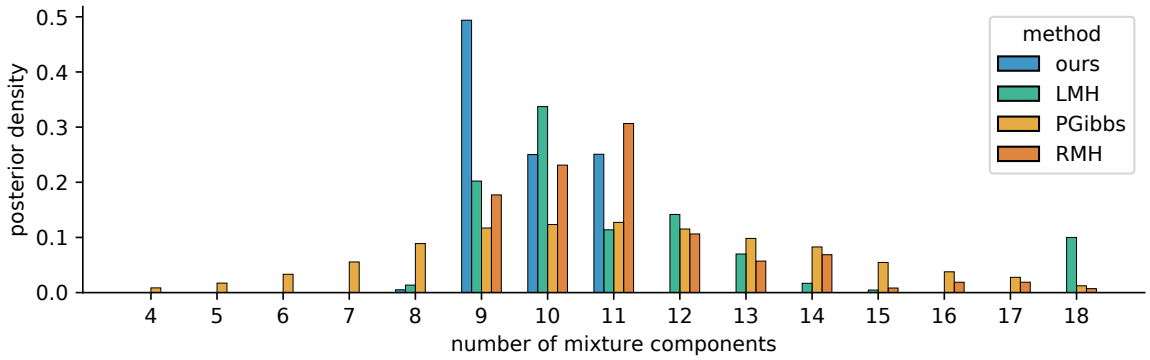


Figure 7. Histogram of the no. of mixtures for the GMM example; correct posterior = 9, averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 50 leapfrog steps of size 0.05; and 5×10^4 LMH, PGibbs and RMH samples.

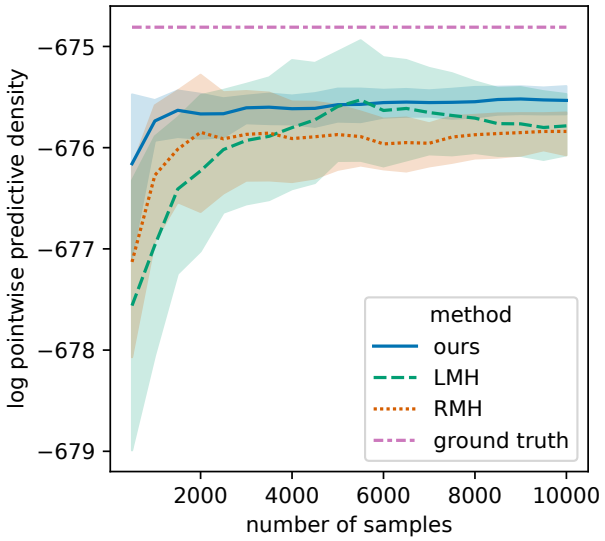


Figure 8. LPPD for the GMM example, averaged over 10 runs. Each run: 10^3 NP-DHMC samples with 10^2 burn-in, 50 leapfrog steps of size 0.05; and 5×10^4 LMH, PGibbs and RMH samples. The shaded area is one standard deviation. PGibbs (with final LPPD -716.85 ± 0.64) is omitted to show the top contenders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

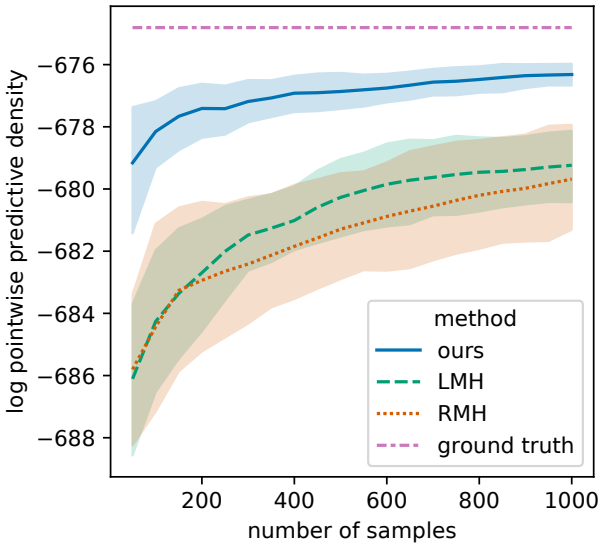


Figure 9. LPPD for the DPMM example, averaged over 10 runs. Each run: 10^2 NP-DHMC samples with 50 burn-in, 20 leapfrog steps of size 0.05; and 2×10^3 LMH, PGibbs and RMH samples. The shaded area is one standard deviation. PGibbs (with final LPPD -725.96 ± 9.83) is omitted to show the top contenders 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 $Y = \{y_1, \ldots, y_{N'}\}$, 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 $(\theta_j)_{j=1\ldots M}$ 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

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

Sampling from a DP is usually implemented by the stick-breaking method (Sethuraman, 1994). However, one can-

not 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. $H = \mathrm{Uniform}([0, 100]^3)$ 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.