Nonparametric Involutive Markov Chain Monte Carlo

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

Abstract

A challenging problem in probabilistic programming is to develop inference algorithms that work for arbitrary programs in a universal probabilistic programming language (PPL). We present the *nonparametric involutive Markov chain Monte Carlo* (NP-iMCMC) algorithm as a method for constructing MCMC inference algorithms for nonparametric models expressible in universal PPLs. Building on the unifying *involutive MCMC* framework, and by providing a *general* procedure for driving state movement between dimensions, we show that NP-iMCMC can generalise numerous existing iMCMC algorithms to work on nonparametric models. We prove the correctness of the NP-iMCMC sampler. Our empirical study shows that the existing strengths of several iMCMC algorithms carry over to their nonparametric extensions. Applying our method to the recently proposed Nonparametric HMC, an instance of (Multiple Step) NP-iMCMC, we have constructed several nonparametric extensions (all of which new) that exhibit significant performance improvements.

1. Introduction

Universal probabilistic programming (Goodman et al., 2008) is the idea of writing probabilistic models in a Turing-complete programming language. A universal probabilistic programming language (PPL) can express all computable probabilistic models (Vákár et al., 2019), using only a handful of basic programming constructs such as branching and recursion. In particular, **nonparametric models**, where the number of random variables is not determined *a priori* and possibly unbounded, can be described naturally in a universal PPL. In programming language terms, this means the number of `sample` statements is unknown prior to

execution. On the one hand, such programs can describe probabilistic models with an unknown number of components, such as Bayesian nonparametric models (Richardson & Green, 1997), variable selection in regression (Ratner, 2010), and models for signal processing (Murray et al., 2018). On the other hand, there are models 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).

However, since universal PPLs are expressively complete, it is challenging to design and implement inference engines that work for arbitrary programs written in them. The parameter space of a nonparametric model is a disjoint union of spaces of varying dimensions. To approximate the posterior distribution via a Markov chain Monte Carlo (MCMC) algorithm (say), the transition kernel will have to switch between (possibly an unbounded number of) states of different dimensions, and to do so reasonably efficiently. This explains why providing theoretical guarantees for MCMC algorithms that work for universal PPLs (Wingate et al., 2011; Wood et al., 2014; Tolpin et al., 2015; Hur et al., 2015; Mak et al., 2021b) is very challenging. For instance, the original version of Lightweight MH (Wingate et al., 2011) was incorrect (Kiselyov, 2016). In fact, most applications requiring Bayesian inference rely on custom MCMC kernels, which are error-prone and time-consuming to design and build.

Contributions   We introduce *Nonparametric Involutive MCMC* (NP-iMCMC) for designing MCMC samplers for universal PPLs. It is an extension of the involutive MCMC (iMCMC) framework (Neklyudov et al., 2020; Cusumano-Towner et al., 2020) to densities arising from nonparametric models (for background on both, see Sec. 2). We explain how NP-iMCMC moves between dimensions and how a large class of existing iMCMC samplers can be extended for universal PPLs (Sec. 3). We also discuss necessary assumptions and prove its correctness. Furthermore, there are general transformations and combinations of NP-iMCMC, to derive more powerful samplers systematically (Sec. 4), for example by making them nonreversible to reduce mixing time. Finally, our experimental results show that our method yields significant performance improvements over existing general MCMC approaches (Sec. 5).

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

All missing proofs are presented in the appendix.

Notation We write $\mathcal{N}_n(\boldsymbol{x}, \Sigma)$ for the $\boldsymbol{x}$-mean Σ-covariance n-dimensional Gaussian with pdf $\varphi_n(\boldsymbol{x}, \Sigma)$. For the standard Gaussian $\mathcal{N}_n(\boldsymbol{0}, \boldsymbol{I})$, we abbreviate them to $\mathcal{N}_n$ and φ_n. In case $n = 1$, we simply write $\mathcal{N}$ and φ.

Given measurable spaces (X, Σ_X) and (Y, Σ_Y), we write $K : X \rightsquigarrow Y$ to mean a **kernel** of type $K : X \times \Sigma_Y \to [0, \infty)$. We say that K is a **probability kernel** if for all $x \in X$, $K(x, \cdot) : \Sigma_Y \to [0, \infty)$ is a probability measure. We write $\mathsf{pdf}\, K(x, y)$ as the density of $y \in Y$ in the measure $K(x, \cdot)$ assuming a derivative w.r.t. some reference measure exists.

Unless otherwise specified, the real space $\mathbb{R}$ is endowed with the Borel measurable sets $\mathcal{B}$ and the standard Gaussian $\mathcal{N}$ measure; the boolean space $2 := \{\mathsf{T}, \mathsf{F}\}$ is endowed with the discrete measurable sets $\Sigma_2 := \mathcal{P}(2)$ and the measure μ_2 which assigns either boolean the probability 0.5. We write $\boldsymbol{x}^{1..n}$ to mean the n-long prefix of the sequence $\boldsymbol{x}$. For any real-valued function $f : X \to \mathbb{R}$, we define its **support** as $\mathrm{Supp}(f) := \{x \in X \mid f(x) > 0\}$.

2. Background

2.1. Involutive MCMC

Given a target density ρ on a measure space (X, Σ_X, μ_X), the iMCMC algorithm generates a Markov chain of samples $\{\boldsymbol{x}^{(i)}\}_{i \in \mathbb{N}}$ by proposing the next sample $\boldsymbol{x}$ using the current sample $\boldsymbol{x}_0$, in three steps:

1. $\boldsymbol{v}_0 \sim K(\boldsymbol{x}_0, \cdot)$: sample a value $\boldsymbol{v}_0$ on an auxiliary measure space (Y, Σ_Y, μ_Y) from an *auxiliary kernel* $K : X \rightsquigarrow Y$ applied to the current sample $\boldsymbol{x}_0$.
2. $(\boldsymbol{x}, \boldsymbol{v}) \leftarrow \Phi(\boldsymbol{x}_0, \boldsymbol{v}_0)$: compute the new state $(\boldsymbol{x}, \boldsymbol{v})$ by applying an *involution*[a] $\Phi : X \times Y \to X \times Y$ to $(\boldsymbol{x}_0, \boldsymbol{v}_0)$.
3. Accept the proposed sample $\boldsymbol{x}$ as the next step with probability given by the *acceptance ratio*

$$\min \left\{ 1; \ \frac{\rho(\boldsymbol{x}) \cdot \mathsf{pdf}\, K(\boldsymbol{x}, \boldsymbol{v})}{\rho(\boldsymbol{x}_0) \cdot \mathsf{pdf}\, K(\boldsymbol{x}_0, \boldsymbol{v}_0)} \cdot |\det(\nabla \Phi(\boldsymbol{x}_0, \boldsymbol{v}_0))| \right\};$$

otherwise reject the proposal $\boldsymbol{x}$ and repeat $\boldsymbol{x}_0$.

---
[a]i.e. $\Phi = \Phi^{-1}$.

Figure 1. iMCMC Algorithm

Our sampler is built on the recently introduced **involutive Markov chain Monte Carlo** method (Neklyudov et al., 2020; Cusumano-Towner et al., 2020), a unifying framework for MCMC algorithms. Completely specified by a target density ρ, an (auxiliary) kernel K and an involution Φ, the iMCMC algorithm (Fig. 1) is conceptually simple. Yet it is remarkably expressive, describing many existing MCMC

Listing 1. Infinite Gaussian mixture model

```
K = floor(abs(sample(normal(0, 1))))
for i in range(K):
    xs[i] = sample(normal(0, 1))
for d in data:
    observe d from mixture([normal(x, 1)
        for x in xs])
return K
```

samplers, including Metropolis-Hastings (MH) (Metropolis et al., 1953; Hastings, 1970) with the "swap" involution $\Phi(\boldsymbol{x}, \boldsymbol{v}) := (\boldsymbol{v}, \boldsymbol{x})$ and the proposal distribution as its auxiliary kernel K; as well as Gibbs (Geman & Geman, 1984), Hamiltonian Monte Carlo (HMC) (Neal, 2011) and Reversible Jump MCMC (RJMCMC) (Green, 1995). Thanks to its schematic nature and generality, we find iMCMC an ideal basis for constructing our nonparametric sampler, NP-iMCMC, for (arbitrary) probabilistic programs. *We stress that NP-iMCMC is applicable to any target density function that is tree representable.*

2.2. Tree representable functions

As is standard in probabilistic programming, our sampler finds the posterior of a program M by taking as the target density a map w, which, given an execution trace, runs M on the sampled values specified by the trace, and returns the weight of such a run. Hence the support of w is the set of traces on which M terminates.

This density w must satisfy the **prefix property** (Mak et al., 2021b): for every trace, there is at most one prefix with strictly positive density. Such functions are called **tree representable** as they can be presented as a computation tree. We shall see how our sampler exploits this property to jump across dimensions in Sec. 3.

Formally the **trace space** $\mathbb{T}$ is the disjoint union $\bigcup_{n \in \mathbb{N}} \mathbb{R}^n$, endowed with σ-algebra $\Sigma_{\mathbb{T}} := \{\bigcup_{n \in \mathbb{N}} X_n \mid X_n \in \mathcal{B}_n\}$ and the standard Gaussian (of varying dimensions) as measure $\mu_{\mathbb{T}}(\bigcup_{n \in \mathbb{N}} X_n) := \sum_{n \in \mathbb{N}} \mathcal{N}_n(X_n)$. We present traces as lists, e.g. $[-0.2, 3.1, 2.8]$ and $[]$. Thus the prefix property is expressible as: for all traces $\boldsymbol{t} \in \mathbb{T}$, there is at most one $k \leq |\boldsymbol{t}|$ s.t. the prefix $\boldsymbol{t}^{1..k}$ is in $\mathrm{Supp}(w)$. Note that the prefix property is satisfied by any densities $w : \mathbb{T} \to [0, \infty)$ induced by a probabilistic program (Prop. A.6), so this is a mild restriction

Example 1. Consider the classic nonparametric infinite Gaussian mixture model (GMM), which infers the number of Gaussian components from a data set. It is describable as a program (Listing 1), where there is a mixture of `K` Gaussian distributions such that the `i`-th Gaussian has mean

`xs[i]` and unit variance. As `K` is not pre-determined, the possible number of components is unbounded, rendering the model nonparametric. Given a trace $[3.4, -1.2, 1.0, 0.5]$, the program describes a mixture of three Gaussians centred at -1.2, 1.0 and 0.5; and it computes the likelihood of generating the set $\mathcal{D}$ of data from such a mixture. The program has density $w : \mathbb{T} \to [0, \infty)$ (w.r.t. the trace measure $\mu_{\mathbb{T}}$) with $w(\boldsymbol{t})$ defined as:

$$\begin{cases} \prod_{d \in \mathcal{D}} \sum_{i=1}^{\lfloor |\boldsymbol{t}^1| \rfloor} \frac{1}{\lfloor |\boldsymbol{t}^1| \rfloor} \varphi_{\lfloor |\boldsymbol{t}^1| \rfloor}(d \mid \boldsymbol{t}^{1+i}, 1) & \text{if } |\boldsymbol{t}| - 1 = \lfloor |\boldsymbol{t}^1| \rfloor \\ 0 & \text{otherwise.} \end{cases}$$

We can check that the density w is tree representable.

3. Nonparametric involutive MCMC

3.1. Example: infinite GMM mixture

Consider how a sample for the infinite GMM (Ex. 1) can be generated using a nonparametric variant of Metropolis-Hastings (MH), an instance of iMCMC. Suppose the current sample is $\boldsymbol{x}_0 := [3.4, -1.2, 1.0, 0.5]$; and $[4.3, -3.4, -0.1, 1.4]$—a sample from the stock Gaussian $\mathcal{N}_4$—is the value of the initial auxiliary variable $\boldsymbol{v}_0$. Then, by application of the "swap" involution to $(\boldsymbol{x}_0, \boldsymbol{v}_0)$, the proposed state $(\boldsymbol{x}, \boldsymbol{v})$ becomes $([4.3, -3.4, -0.1, 1.4], [3.4, -1.2, 1.0, 0.5])$. A problem arises if we simply propose $\boldsymbol{x}$ as the next sample, as it describes a mixture of *four* Gaussians (notice `K` has value 4) but only *three* means are provided, viz., $-3.4, -0.1, 1.4$. Hence, the program does not terminate on the trace specified by $\boldsymbol{x}$, i.e., $\boldsymbol{x}$ is not in the support of w, the model's density.

The key idea of NP-iMCMC is to *extend* the initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ to $(\boldsymbol{x}_0 + [e], \boldsymbol{v}_0 + [p])$ where $+$ denotes trace concatenation, and e, p are random draws from the stock Gaussian $\mathcal{N}$. Say -0.7 and -0.3 are the values drawn; the initial state then becomes $(\boldsymbol{x}_0, \boldsymbol{v}_0) = ([3.4, -1.2, 1.0, 0.5, -0.7], [4.3, -3.4, -0.1, 1.4, -0.3])$, and the proposed state $(\boldsymbol{x}, \boldsymbol{v})$ becomes $([4.3, -3.4, -0.1, 1.4, -0.3], [3.4, -1.2, 1.0, 0.5, -0.7])$. Now the program does terminate on a trace specified by the proposed sample $\boldsymbol{x} = [4.3, -3.4, -0.1, 1.4, -0.3]$; equivalently $\boldsymbol{x} \in \text{Supp}(w)$.

Notice that if this is not the case, such a process—which extends the initial state by incrementing the dimension—can be repeated until termination happens. For an almost surely (a.s.) terminating program, this process a.s. yields a proposed sample.

Finally, we calculate the acceptance ratio for $\boldsymbol{x} \in \text{Supp}(w)$

from the initial sample $\boldsymbol{x}_0^{1..4} \in \text{Supp}(w)$ as

$$\min \left\{ 1; \frac{w(\boldsymbol{x}) \cdot \varphi_5(\boldsymbol{x}) \cdot \varphi_5(\boldsymbol{v})}{w(\boldsymbol{x}_0^{1..4}) \cdot \varphi_5(\boldsymbol{x}_0) \cdot \varphi_5(\boldsymbol{v}_0)} \right\}.$$

3.2. State space, target density and assumptions

Fix an *parameter (measure) space* $(\mathbb{X}, \Sigma_{\mathbb{X}}, \mu_{\mathbb{X}})$, which is (intuitively) the product of the respective measure space of the distribution of X, with X ranging over the random variables of the model in question. Assume an *auxiliary (probability) space* $(\mathbb{Y}, \Sigma_{\mathbb{Y}}, \mu_{\mathbb{Y}})$. For simplicity, we assume in this paper[1] that both $\mathbb{X}$ and $\mathbb{Y}$ are $\mathbb{R}$; further $\mu_{\mathbb{X}}$ has a derivative $\varphi_{\mathbb{X}}$ w.r.t. the Lebesgue measure, and $\mu_{\mathbb{Y}}$ also has a derivative $\varphi_{\mathbb{Y}}$ w.r.t. the Lebesgue measure. Note that it follows from our assumption that $\mathbb{X}^n \times \mathbb{Y}^n$ is a smooth manifold for each n.[2] Now a **state** is a pair of *parameter* and *auxiliary* variables of equal dimension. Formally the **state space** $\mathbb{S} := \bigcup_{n \in \mathbb{N}} (\mathbb{X}^n \times \mathbb{Y}^n)$ is endowed with the σ-algebra $\Sigma_{\mathbb{S}} := \sigma \{ X_n \times V_n \mid X_n \in \Sigma_{\mathbb{X}^n}, V_n \in \Sigma_{\mathbb{Y}^n}, n \in \mathbb{N} \}$ and measure $\mu_{\mathbb{S}}(S) := \sum_{n \in \mathbb{N}} \int_{\mathbb{Y}^n} \mu_{\mathbb{X}^n}(\{ \boldsymbol{x} \in \mathbb{X}^n \mid (\boldsymbol{x}, \boldsymbol{v}) \in S \}) \, \mu_{\mathbb{Y}^n}(\mathrm{d}\boldsymbol{v})$.

Besides the target density function w, our algorithm NP-iMCMC requires two additional inputs: **auxiliary kernels** (as an additional source of randomness) and **involutions** (to traverse the *state space*). Next we present what we assume about the three inputs and discuss some relevant properties.

Target density function We only target densities $w : \mathbb{T} \to [0, \infty)$ that are tree representable, where $\mathbb{T} := \bigcup_{n \in \mathbb{N}} \mathbb{R}^n$. Moreover, we assume two common features of real-world probabilistic programs:

(V1) w is **integrable**, i.e. $Z := \int_{\mathbb{T}} w \, \mathrm{d}\mu_{\mathbb{T}} < \infty$ (otherwise, the inference problem is undefined)

(V2) w is **almost surely terminating (AST)**, i.e. $\mu_{\mathbb{T}}(\{ \boldsymbol{t} \in \mathbb{T} \mid w(\boldsymbol{t}) > 0 \}) = 1$ (otherwise, the loop (Step 3) of the NP-iMCMC algorithm may not terminate a.s.).[3]

Auxiliary kernel We assume, for each dimension $n \in \mathbb{N}$, an **auxiliary (probability) kernel** $K^{(n)} : \mathbb{X}^n \rightsquigarrow \mathbb{Y}^n$ with density function $\text{pdf} K^{(n)} : \mathbb{X}^n \times \mathbb{Y}^n \to [0, \infty)$ (assuming a derivative w.r.t. $\mu_{\mathbb{X}^n \times \mathbb{Y}^n}$ exists).

Involution We assume, for each dimension $n \in \mathbb{N}$, a differentiable endofunction $\Phi^{(n)}$ on $\mathbb{X}^n \times \mathbb{Y}^n$ which is *in-*

---

[1] In App. B.1, we consider a more general case where $\mathbb{X}$ is set to be $\mathbb{R} \times 2$.

[2] Notation: For any probability space $(\mathtt{X}, \Sigma_{\mathtt{X}}, \mu_{\mathtt{X}})$ such that $\mu_{\mathtt{X}}$ has derivative $\varphi_{\mathtt{X}}$ w.r.t. the Lebesgue measure. $\mathtt{X}^n$ is the Cartesian product of n copies of $\mathtt{X}$; $\Sigma_{\mathtt{X}^n}$ is the σ-algebra generated by subsets of the form $\prod_{i=1}^n V_i$ where $V_i \in \Sigma_{\mathtt{X}}$; and $\mu_{\mathtt{X}^n}$ is the product of n copies of $\mu_{\mathtt{X}}$ which has derivative $\varphi_{\mathtt{X}^n}$ w.r.t. the Lebesgue measure. Note that $(\mathtt{X}^n, \Sigma_{\mathtt{X}^n}, \mu_{\mathtt{X}^n})$ is a probability space.

[3] If a program does not terminate on a trace $\boldsymbol{t}$, the density $w(\boldsymbol{t})$ is defined to be zero.

volutive, i.e. $\Phi^{(n)} = \Phi^{(n)}{}^{-1}$, and satisfies the ***projection commutation property***:

(V3) For all $(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}$ where $|\boldsymbol{x}| = m$, if $\boldsymbol{x}^{1..n} \in$ Supp(w) for some n, then for all $k = n, \ldots, m$, take$_k(\Phi^{(m)}(\boldsymbol{x}, \boldsymbol{v})) = \Phi^{(k)}(\text{take}_k(\boldsymbol{x}, \boldsymbol{v}))$

where take$_\ell$ is the projection that takes a state $(\boldsymbol{x}, \boldsymbol{v})$ and returns the state $(\boldsymbol{x}^{1..\ell}, \boldsymbol{v}^{1..\ell})$ with the first ℓ coordinates of each component. (Otherwise, the sample-component $\boldsymbol{x}$ of the proposal state tested in Step 3 may not be an extension of the sample-component of the preceding proposal state.)

3.3. Algorithm

Given a probabilistic program M with density function w on the trace space $\mathbb{T}$, a set $\{K^{(n)} : \mathbb{X}^n \leadsto \mathbb{Y}^n\}$ of auxiliary kernels and a set $\{\Phi^{(n)} : \mathbb{X}^n \times \mathbb{Y}^n \to \mathbb{X}^n \times \mathbb{Y}^n\}$ of involutions satisfying V1 to 3, we present the ***NP-iMCMC*** algorithm in Fig. 2.

The ***NP-iMCMC*** generates a Markov chain by proposing the next sample $\boldsymbol{x}$ using the current sample $\boldsymbol{x}_0$ as follows:

1. $\boldsymbol{v}_0 \sim K^{(k_0)}(\boldsymbol{x}_0, \cdot)$: sample a value $\boldsymbol{v}_0$ on the auxiliary space $\mathbb{Y}^{k_0}$ from the auxiliary kernel $K^{(k_0)} : \mathbb{X}^{k_0} \leadsto \mathbb{Y}^{k_0}$ applied to the current sample $\boldsymbol{x}_0$ where $k_0 = |\boldsymbol{x}_0|$.

2. $(\boldsymbol{x}, \boldsymbol{v}) \leftarrow \Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0)$: compute the *proposal state* $(\boldsymbol{x}, \boldsymbol{v})$ by applying the involution $\Phi^{(n)}$ on $\mathbb{X}^n \times \mathbb{Y}^n$ to the *initial state* $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ where $n = |\boldsymbol{x}_0|$.

3. Test if for some k, $\boldsymbol{x}^{1..k} \in$ Supp(w). (*Equivalently*: Test if program M terminates on the trace specified by the sample-component $\boldsymbol{x}$ of the proposal state, or one of its prefixes.) If so, proceed to the next step; otherwise
   - $(\boldsymbol{x}_0, \boldsymbol{v}_0) \leftarrow (\boldsymbol{x}_0 + [e], \boldsymbol{v}_0 + [p])$: extend the initial state to $(\boldsymbol{x}_0 + [e], \boldsymbol{v}_0 + [p])$ where e and p are samples drawn from $\mu_{\mathbb{X}}$ and $\mu_{\mathbb{Y}}$,
   - Go to Step 2.

4. Accept $\boldsymbol{x}^{1..k}$ as the next sample with probability

$$\min \left\{ 1; \frac{w(\boldsymbol{x}^{1..k}) \cdot \text{pdf} K^{(k)}(\boldsymbol{x}^{1..k}, \boldsymbol{v}^{1..k})}{w(\boldsymbol{x}_0{}^{1..k_0}) \cdot \text{pdf} K^{(k_0)}(\boldsymbol{x}_0{}^{1..k_0}, \boldsymbol{v}_0{}^{1..k_0})} \right.$$
$$\left. \cdot \frac{\varphi_{\mathbb{X}^n \times \mathbb{Y}^n}(\boldsymbol{x}, \boldsymbol{v})}{\varphi_{\mathbb{X}^n \times \mathbb{Y}^n}(\boldsymbol{x}_0, \boldsymbol{v}_0)} \cdot |\det(\nabla \Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0))| \right\}$$

where $n = |\boldsymbol{x}_0|$; otherwise reject the proposal and repeat $\boldsymbol{x}_0{}^{1..k_0}$.

Figure 2. NP-iMCMC Algorithm

The heart of NP-iMCMC is Step 3, which can drive a state across dimensions. Step 3 first checks if $\boldsymbol{x}^{1..k} \in$ Supp(w) for some $k = 1, \ldots, k_0$, (i.e. if the program M terminates on the trace specified by some prefix of $\boldsymbol{x}$). If so, the proposal state is set to $(\boldsymbol{x}^{1..k}, \boldsymbol{v}^{1..k})$, and the state moves from dimension k_0 to k. Otherwise, Step 3 repeatedly extends the

initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ to, say, $(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$, and computes the new proposal state $(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{v} + \boldsymbol{u})$ by Step 2, until the program M terminates on the trace specified by $\boldsymbol{x} + \boldsymbol{y}$. Then, the proposal state becomes $(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{v} + \boldsymbol{u})$, and the state moves from dimension k_0 to dimension $k_0 + |\boldsymbol{y}|$.

Remark 3.1.
 (i) The projection commutation property, V3, ensures that the new proposal state computed using $(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$ from Step 3 is of the form $(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{v} + \boldsymbol{u})$ where $(\boldsymbol{x}, \boldsymbol{v}) = \Phi^{(|\boldsymbol{x}_0|)}(\boldsymbol{x}_0, \boldsymbol{v}_0)$.

 (ii) V2, a.s. termination of the program M, ensures that the method of computing a proposal state in Step 3 almost surely finds a proposal sample $\boldsymbol{x}$ such that M terminates on a trace specified by $\boldsymbol{x}$.

 (iii) The prefix property of the target density w ensures that any proper extension of current sample $\boldsymbol{x}_0$ (of length k_0) has zero density, i.e. $w(\boldsymbol{x}_0 + \boldsymbol{y}) = 0$ for all $\boldsymbol{y} \neq []$. Hence only the weight of the current sample $\boldsymbol{x}_0{}^{1..k_0} \in$ Supp(w) is accounted for in Step 4 even when $\boldsymbol{x}_0$ is extended.

 (iv) If the program M is parametric, thus inducing a target density w on a fixed dimensional space, then the NP-iMCMC sampler coincides with the iMCMC sampler.

Using NP-iMCMC (Fig. 2), we can formally present the Nonparametric Metropolis-Hastings (NP-MH) sampler which was introduced in Sec. 3.1. See App. E.1 for details.

3.4. Generalisations

In the interest of clarity, we have presented a version of NP-iMCMC in deliberately purified form. Here we discuss three generalisations of the NP-iMCMC sampler.

Hybrid state space Many PPLs provide continuous and discrete samplers. The positions of discrete and continuous random variables in an execution trace may vary, because of branching. We get around this problem by defining the parameter space $\mathbb{X}$ to be the *product space* of $\mathbb{R}$ and $2 := \{\mathsf{F}, \mathsf{T}\}$. Each value $\boldsymbol{t}^i$ in a trace $\boldsymbol{t}$ is paired with a randomly drawn "partner" t of the other type to make a pair $(\boldsymbol{t}^i, t)$ (or $(t, \boldsymbol{t}^i)$). Hence, the same idea of "jumping" across dimensions can be applied to the state space $\bigcup_{n \in \mathbb{N}} \mathbb{X}^n \times \mathbb{Y}^n$. The resulting algorithm is called the *Hybrid NP-iMCMC* sampler. (See App. B for more details.)

Computationally heavy involutions Step 3 in the NP-iMCMC sampler may seem inefficient. While it terminates almost surely (thanks to V2), the expected number of iterations may be infinite. This is especially bad if the involution is computationally expensive such as the leapfrog integrator in HMC which requires gradient information of the target density function. This can be worked around if for each $n \in \mathbb{N}$, there is an inexpensive *slice* function $s^{(n)} : \mathbb{X}^n \times \mathbb{Y}^n \to \mathbb{X} \times \mathbb{Y}$ where $s^{(n)}(\boldsymbol{x}, \boldsymbol{v}) = (\text{drop}_{n-1} \circ \Phi^{(n)})(\boldsymbol{x}, \boldsymbol{v})$ if $(\boldsymbol{x}, \boldsymbol{v})$ is a n-

dimensional state such that $\boldsymbol{x}^{1..k} \in \mathrm{Supp}(w)$ for some $k < n$, and drop_ℓ is the projection that takes a state $(\boldsymbol{x}, \boldsymbol{v})$ and returns the state $(\boldsymbol{x}^{\ell+1..|\boldsymbol{x}|}, \boldsymbol{v}^{\ell+1..|\boldsymbol{x}|})$ with the first ℓ coordinates of each component dropped. Then the new proposal state in Step 3 can be computed by applying the function $s^{(n)}$ to the recently extended initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$, i.e. $(\boldsymbol{x}, \boldsymbol{v}) \leftarrow (\boldsymbol{x} + [e'], \boldsymbol{v} + [p'])$ where $(e', p') = s^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0)$ instead. (See App. D.2 for more details.)

Multiple step NP-iMCMC Suppose the involution is a composition of bijective endofunctions, i.e. $\Phi^{(n)} := f_L^{(n)} \circ \cdots \circ f_2^{(n)} \circ f_1^{(n)}$ and each endofunction $\{f_\ell^{(n)}\}_n$ satisfies the projection commutation property and has a slice function $s_\ell^{(n)}$. A new state can then be computed by applying the endofunctions to the initial state *one-by-one* (instead of in one go as in Step 2 and 3): For each $\ell = 1, \ldots, L$,

1. Compute the intermediate state $(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell)$ by applying $f_\ell^{(n)}$ to $(\boldsymbol{x}_{\ell-1}, \boldsymbol{v}_{\ell-1})$ where $n = |\boldsymbol{x}_{\ell-1}|$.
2. Test whether $\boldsymbol{x}_\ell^{1..k}$ is in $\mathrm{Supp}(w)$ for some k. If so, proceed to the next ℓ; otherwise
   - extend the initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ with samples drawn from $\mu_{\mathbb{X}}$ and $\mu_{\mathbb{Y}}$,
   - for $i = 1, \ldots, \ell$, extend the intermediate states $(\boldsymbol{x}_i, \boldsymbol{v}_i)$ with the result of $s_i^{(n)}(\boldsymbol{x}_{i-1}, \boldsymbol{v}_{i-1})$ where $n = |\boldsymbol{x}_{i-1}|$,
   - go to 2.

The resulting algorithm is called the *Multiple Step NP-iMCMC* sampler. (See App. D.3 for more details.) This approach was adopted in the recently proposed Nonparametric HMC (Mak et al., 2021b). (See App. E.3 for more details.)

3.5. Correctness

The NP-iMCMC algorithm is correct in the sense that the invariant distribution of the Markov chain generated by iterating the algorithm in Fig. 2 coincides with the target distribution $\nu : A \mapsto \frac{1}{Z} \int_A w \, \mathrm{d}\mu_{\mathbb{T}}$ with the normalising constant Z. We present an outline proof here. See App. B.4 for a full proof of the Hybrid NP-iMCMC algorithm, a generalisation of NP-iMCMC.

Note that we *cannot* reduce NP-iMCMC to iMCMC, i.e. the NP-iMCMC sampler cannot be formulated as an instance of the iMCMC sampler with an involution on the whole state space $\mathbb{S}$. This is because the dimension of involution depends on the values of the random samples drawn in Step 3. Instead, we define a helper algorithm (App. B.4.2), which induces a Markov chain on states and does not change the dimension of the involution.

This algorithm first extends the initial state to find the smallest N such that the program M terminates with a trace

specified by some prefix of the sample-component of the resulting state $(\boldsymbol{x}, \boldsymbol{v})$ after applying the involution $\Phi^{(N)}$. Then, it performs the involution $\Phi^{(N)}$ as per the standard iMCMC sampler. Hence all stochastic primitives are executed outside of the involution, and the involution has a fixed dimension. We identify the state distribution (App. B.4.2), and show that the Markov chain generated by the auxiliary algorithm has the state distribution as its invariant distribution (Lem. B.14). We then deduce that its marginalised chain is identical to that generated by Hybrid NP-iMCMC; and Hybrid NP-iMCMC has the target distribution ν as its invariant distribution (Lem. B.17). Since Hybrid NP-iMCMC is a generalisation of NP-iMCMC (Fig. 2), we have the following corollary.

Corollary 3.2 (Invariant). *If all inputs satisfy V1 to 3 then ν is the invariant distribution of the Markov chain generated by iterating the algorithm described in Fig. 2.*

4. Transforming NP-iMCMC samplers

The strength of the iMCMC framework lies in its flexibility, which makes it a useful tool capable of expressing important ideas in existing MCMC algorithms as "tricks", namely

- state-dependent mixture (Trick 1 and 2 in (Neklyudov et al., 2020)),
- smart involutions (Trick 3 and 4), and
- smart compositions (Trick 5 and 6).

In each of these tricks, the auxiliary kernel and involution take special forms to equip the resulting sampler with desirable properties such as higher acceptance ratio and better mixing times. This enables a "make to order" approach in the design of novel MCMC samplers.

A natural question is whether there are similar tricks for the NP-iMCMC framework. In this section, we examine the tricks discussed in (Neklyudov et al., 2020), giving requirements for and showing via examples how one can design *novel* NP-iMCMC samplers with bespoke properties by suitable applications of these "tricks" to simple NP-iMCMC samplers. Similar applications can be made to the generalisations of NP-iMCMC such as Hybrid NP-iMCMC (App. C) and Multiple NP-iMCMC (App. D.3.4). Throughout this section, we consider samplers for a program M expressed in a universal PPL which has target density function w that is integrable (V1) and almost surely terminating (V2).

4.1. State-dependent mixture

Suppose we want a sampler that chooses a suitable NP-iMCMC sampler depending on the current sample. This might be beneficial for models that are modular, and where there is already a good sampler for each module. We can form a ***state-dependent mixture*** of a family $\{\iota_m\}_{m \in M}$ of

NP-iMCMC samplers[4] which runs ι_m with a weight depending on the current sample. See App. C.1 for details of the algorithm.

Remark 4.1. This corresponds to Tricks 1 and 2 discussed in (Neklyudov et al., 2020) which generalises the Mixture Proposal MCMC and Sample-Adaptive MCMC samplers.

4.2. Auxiliary direction

Suppose we want to use sophisticated bijective but non-involutive endofunctions $f^{(n)}$ on $\mathbb{X}^n \times \mathbb{Y}^n$ to better explore the parameter space and return proposals with a high acceptance ratio. Assuming both families $\{f^{(n)}\}_n$ and $\{f^{(n)^{-1}}\}_n$ satisfy the projection commutation property (V 3), we can construct an NP-iMCMC sampler with *auxiliary direction*, which

- samples a direction $d \in \mathbb{D} := \{+, -\}$ with equal probability; and
- generates the next sample by running Step 1 to 4 of the NP-iMCMC sampler using $\{f^{(n)}\}_n$ to suggest the proposal sample if d is sampled to be $+$; otherwise $\{f^{(n)^{-1}}\}_n$ is used.

See App. C.2 for details of the algorithm.

Notice that since the distribution of the direction variable d is the discrete uniform distribution, we do not need to alter the acceptance ratio in Step 4.

Example 2 (NP-HMC). We can formulate the recently proposed Nonparametric Hamiltonian Monte Carlo sampler in (Mak et al., 2021b) using the (Multiple Step) NP-iMCMC framework with auxiliary direction, in which case the sophisticated non-involutive endofunction is the leapfrog method $\boldsymbol{L}$. (See App. E.3 for details.)

Remark 4.2. This corresponds to Trick 3 described in (Neklyudov et al., 2020). Trick 4 from (Neklyudov et al., 2020) cannot be applied in our framework because the projection commutation property (V3) is not closed under composition.

4.3. Persistence

Suppose we want a nonreversible sampler, so as to obtain better mixing times. A typical way of achieving nonreversibility from an originally reversible MCMC sampler is to reuse the value for a variable (that is previously resampled in the original reversible sampler) in the next iteration if the proposed sample is accepted. In this way, the value of such a variable is allowed to *persist*, making the sampler nonreversible.

A key observation made by Neklyudov et al. (2020) is that the composition of reversible iMCMC samplers can

---

[4]We treat ι_m as a piece of computer code that changes the sample via the NP-iMCMC method described in Fig. 2.

yield a nonreversible sampler. Two systematic techniques to achieve nonreversibility are persistent direction (Trick 5) and an auxiliary kernel (Trick 6). We present similar approaches for NP-iMCMC samplers.

Suppose there is an NP-iMCMC sampler that uses the auxiliary direction as described in Sec. 4.2, i.e. there is a non-involutive bijective endofunction $f^{(n)}$ on $\mathbb{X}^n \times \mathbb{Y}^n$ for each $n \in \mathbb{N}$ such that $\{f^{(n)}\}_n$ and $\{f^{(n)^{-1}}\}_n$ satisfy the projection commutation property (V 3). In addition, assume there are two distinct families of auxiliary kernels, namely $\{K^{(n)}{}_+\}_n$ and $\{K^{(n)}{}_-\}_n$. The corresponding NP-iMCMC sampler with *persistence*

- proposes the next sample by running Step 1 to 3 of the NP-iMCMC sampler with $\{K^{(n)}{}_+\}_n$ and $\{f^{(n)}\}_n$ if d is sampled to be $+$; otherwise $\{K^{(n)}{}_-\}_n$ and $\{f^{(n)^{-1}}\}_n$ are used;
- accepts the proposed sample with probability indicated in Step 4 of the NP-iMCMC sampler; otherwise repeats the current sample and *flips the direction d*.

See App. C.3 for details of the algorithm. The family of kernels and maps indeed *persist* across multiple iterations if the proposals of these iterations are accepted. The intuitive idea behind this is that if a family of kernels and maps perform well (proposals are accepted) in the current part of the sample space, we should keep it, and otherwise switch to its inverse.

Remark 4.3. This corresponds to Tricks 5 and 6 described in (Neklyudov et al., 2020), which can be found in non-reversible MCMC sampler like the Generalised HMC algorithm (Horowitz, 1991), the Look Ahead HMC sampler (Sohl-Dickstein et al., 2014; Campos & Sanz-Serna, 2015) and Lifted MH (Turitsyn et al., 2011).

Example 3 (NP-HMC with Persistence). The nonreversible HMC sampler in (Horowitz, 1991) uses persistence, and, in addition, (partially) reuses the momentum vector from the previous iteration. As shown in (Neklyudov et al., 2020), it can be viewed as a composition of iMCMC kernels. Using the method indicated above, we can also add persistence to NP-HMC. (See App. E.4 for details.)

Example 4 (NP-Lookahead-HMC). Look Ahead HMC (Sohl-Dickstein et al., 2014; Campos & Sanz-Serna, 2015) can be seen as an HMC sampler with persistence that generates a new state with a varying number of leapfrog steps, depending on the value of the auxiliary variable. Similarly, we can construct an NP-HMC sampler with Persistence that varies the numbers of leapfrog steps. (See App. E.5 for more details.)
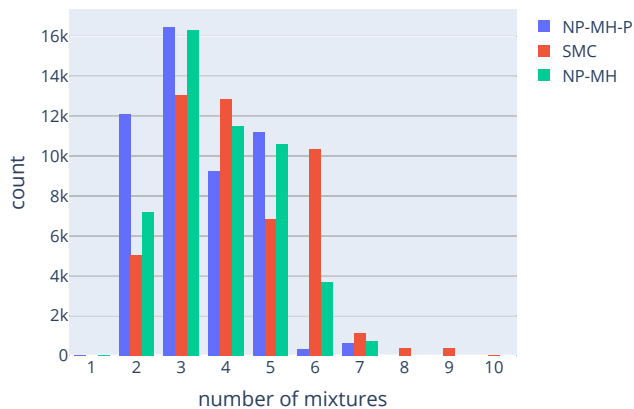
Figure 3. Histogram of the number of components for the infinite GMM; correct posterior is 3.

5. Experiments

5.1. Nonparametric Metropolis-Hastings

We first implemented two simple instances of the NP-iMCMC sampler, namely NP-MH (App. E.1) and NP-MH with Persistence (App. E.2) in the Turing language (Ge et al., 2018).[5] We compared them with Turing's built-in Sequential Monte Carlo (SMC) algorithm on an infinite Gaussian mixture model (GMM) where the number of mixture components is drawn from a normal distribution. Posterior inference is performed on 30 data points generated from a ground truth with three components. The results of ten runs with 5000 iterations each (Fig. 3) suggest that the NP-iMCMC samplers work pretty well.

5.2. Nonparametric Hamiltonian Monte Carlo

Secondly, we consider Nonparametric HMC (Mak et al., 2021b), mentioned in Ex. 2 before. We have seen how the techniques from Sec. 4 can yield nonreversible versions of NP-iMCMC inference algorithms. Here, we look at nonparametric versions of two extensions described in (Neklyudov et al., 2020): persistence (Ex. 3) and lookahead (Ex. 4). Persistence means that the previous momentum vector is reused in the next iteration. It is parametrised by $\alpha \in [0, 1]$ where $\alpha = 1$ means no persistence (standard HMC) and $\alpha = 0$ means full persistence (no randomness added to the momentum vector). Lookahead HMC is parametrised by $K \geq 0$, which is the number of extra iterations ("look ahead") to try

---

⁵ The code to reproduce the Turing experiments is available in https://github.com/cmaarkol/nonparametric-mh. Turing's SMC implementation is nondeterministic (even with a fixed random seed), so its results may vary somewhat, but everything else is exactly reproducible.

Table 1. Geometric distribution example: total variation difference from the ground truth, averaged over 10 runs, and standard deviation. Each run: 10^3 samples, L leapfrog steps, step size $\epsilon = 0.1$, persistence parameter $\alpha \in \{0.5\}$.

	persistence	TVD from ground truth
$L = 5$	—	0.0524 ± 0.0069
$L = 5$	$\alpha = 0.5$	0.0464 ± 0.0074
$L = 5$	$\alpha = 0.1$	0.0461 ± 0.0083
$L = 2$	—	0.0768 ± 0.0181
$L = 2$	$\alpha = 0.5$	0.0570 ± 0.0115
$L = 2$	$\alpha = 0.1$	0.0534 ± 0.0058

before rejecting a proposed sample (so $K = 0$ corresponds to standard HMC). Detailed descriptions of these algorithms and how they fit into the (Multiple Step) NP-iMCMC framework can be found in App. E.4 and E.5.

We evaluate these extensions of NP-HMC on the benchmarks from (Mak et al., 2021b): a model for the geometric distribution, a model involving a random walk, and an unbounded Gaussian mixture model. Note that similarly to (Mak et al., 2021b), we actually work with a discontinuous version of NP-HMC, called *NP-DHMC*, which is a nonparametric extension of discontinuous HMC (Nishimura et al., 2020).[6] The discontinuous version can handle the discontinuities arising from the jumps between dimensions more efficiently. We don't discuss it in this paper due to lack of space. However, the modifications necessary to this discontinuous version are the same as for the standard NP-HMC. Mak et al. (2021b) demonstrated the usefulness of NP-DHMC and how it can obtain better results than other general-purpose inference algorithms like Lightweight Metropolis-Hastings and Random-walk Lightweight Metropolis-Hastings. Here, we focus on the benefits of nonreversible versions of NP-DHMC, which were derived using the (Multiple Step) NP-iMCMC framework.

Geometric distribution The geometric distribution benchmark from (Mak et al., 2021b) illustrates the usefulness of persistence: we ran NP-DHMC for a step count $L \in \{2, 5\}$ with and without persistence. As can be seen in Table 1, persistence usually decreases the distance from the ground truth. In fact, the configuration $L = 2, \alpha = 0.1$ is almost as good as $L = 5$ without persistence, despite taking 2.5 times less computing time.

Random walk The next benchmark from (Mak et al., 2021b) models a random walk and observes the distance travelled. Fig. 4 shows the effective sample size (ESS) in terms of the number of samples drawn, comparing versions of NP-DHMC with persistence ($\alpha = 0.5$) and look-ahead

---

⁶The source code is available at https://github.com/fzaiser/nonparametric-hmc.
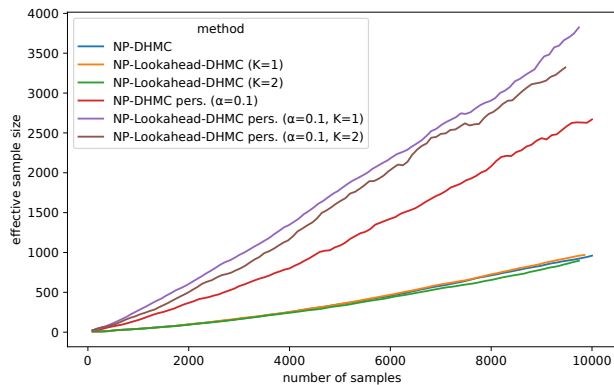
Figure 4. ESS for the random walk example in terms of number of samples, computed from 10 runs. Each run: 10^3 samples with $L = 5$ leapfrog steps of size $\epsilon = 0.1$, persistence parameter $\alpha \in \{0.5, 0.1\}$, and look-ahead $K \in \{1, 2\}$.

Figure 5. Gaussian mixture with Poisson prior: LPPD in terms of number of samples, averaged over 10 runs. The shaded area is one standard deviation. Each run: 10^3 samples with $L = 25$ leapfrog steps of size $\epsilon = 0.05$, persistence parameter $\alpha = 0.5$, and look-ahead $K \in \{1, 2\}$.

($K \in \{1, 2\}$). We can see again that persistence is clearly advantageous. Look-ahead ($K \in \{1, 2\}$) seems to give an additional boost on top. We ran all these versions with the same computation time budget, which is why the the lines for $K = 1, 2$ are cut off before the others.

Unbounded Gaussian mixture model   Next, we consider a Gaussian mixture model where the number of mixture components is drawn from a Poisson prior. Inference is performed on a training data set generated from a mixture of 9 components (the ground truth). We then compute the log pointwise predictive density (LPPD) on a test data set drawn from the same distribution as the training data. The LPPD is shown in Fig. 5 in terms of the number of samples. Note that again, all versions were run with the same computation budget, which is why some of the lines are cut off early. Despite this, we can see that the versions with lookahead ($K \in \{1, 2\}$) converge more quickly than the versions without lookahead. Persistent direction ($\alpha = 0.5$) also seems to have a (smaller) benefit.

Dirichlet process mixture model   Finally, we consider a Gaussian mixture whose weights are drawn from a Dirichlet process. The rest of the setup is the same as for the Poisson prior, and the results are shown in Fig. 6. The version with persistence is worse at the start but obtains a better LPPD at the end. Look-ahead ($K \in \{1, 2\}$) yields a small additional boost in the LPPD. It should be noted that the variance over the 10 runs is larger in this example than in the previous benchmarks, so the conclusion of this benchmark is less clear-cut.
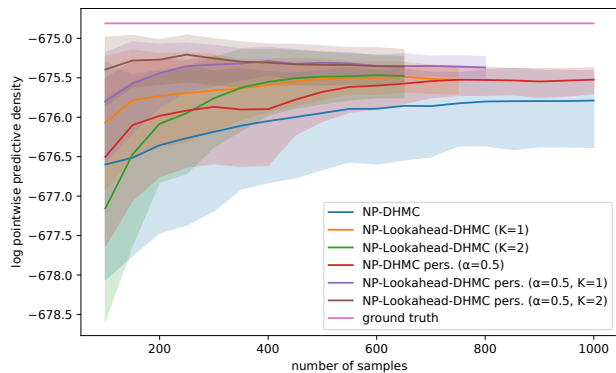
6. Related work and Conclusion

Involutive MCMC and its instances   The involutive MCMC framework (Neklyudov et al., 2020; Cusumano-Towner et al., 2019; Matheos et al., 2020) can in principle be used for nonparametric models by setting $X := \bigcup_{n \in \mathbb{N}} \mathbb{X}^n$ and $Y := \bigcup_{n \in \mathbb{N}} \mathbb{Y}^n$ in Fig. 1 and defining an auxiliary kernel on $X \rightsquigarrow Y := \bigcup_{n \in \mathbb{N}} \mathbb{X}^n \rightsquigarrow \bigcup_{n \in \mathbb{N}} \mathbb{Y}^n$ an involution on $X \times Y := \bigcup_{n \in \mathbb{N}} \mathbb{X}^n \times \mathbb{Y}^n$. For instance, Reversible Jump MCMC (Green, 1995) is an instance of iMCMC that works for the infinite GMM model, with the split-merge proposal (Richardson & Green, 1997) specifying when and how states can "jump" across dimensions. However, designing appropriate auxiliary kernels and involutions that enable the extension of an iMCMC sampler to nonparametric models remains challenging and model specific. By contrast, NP-iMCMC only requires the specification of involutions on the finite-dimensional space $\mathbb{X}^n \times \mathbb{Y}^n$; moreover, it provides a *general* procedure (via Step 3) that drives state movement between dimensions. For designers of nonparametric samplers who do not care to custom build trans-dimensional methods, we contend that NP-iMCMC is their method of choice.

The performance of NP-iMCMC and iMCMC depends on the complexity of the respective auxiliary kernels, involutions and the model in question. Take iGMM for example. RJMCMC with the split-merge proposal which computes the weight, mean, and variance of the new component(s) would be slower than NP-MH, an instance of NP-iMCMC with a computationally light involution (a swap), but more efficient than NP-HMC, an instance of (Multiple Step) NP-iMCMC with the computationally heavy leapfrog integrator as involution.
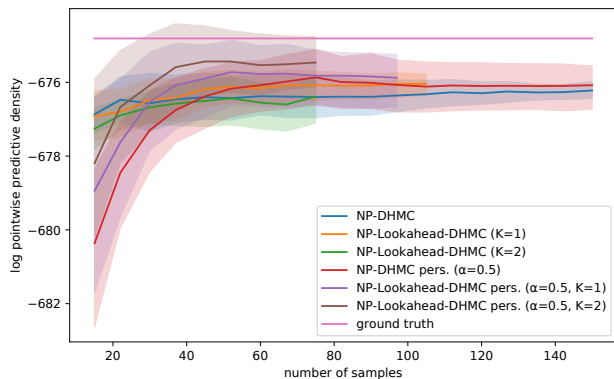
Figure 6. Dirichlet process mixture: LPPD in terms of number of samples, averaged over 10 runs. The shaded area is one standard deviation. Each run: 150 samples with $L = 20$ leapfrog steps of size $\epsilon = 0.05$, persistence parameter $\alpha = 0.5$, and look-ahead $K \in \{1, 2\}$.

Trans-dimensional samplers   A standard MCMC algorithm for universal PPLs is the Lightweight Metropolis-Hastings algorithm (LMH) (Yang et al., 2014; Tolpin et al., 2015; Ritchie et al., 2016). Widely implemented in several universal PPLs (Anglican, Venture, Gen, and Web PPL), LMH performs single-site updates on the current sample and re-executes the program from the resampling point.

Divide, Conquer, and Combine (DCC) (Zhou et al., 2020) is an inference algorithm that is applicable to probabilistic programs that use branching and recursion. 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.

Mak et al. (2021b) have recently introduced Nonparametric Hamiltonian Monte Carlo (NP-HMC), which generalises HMC to nonparametric models. As we've seen, NP-HMC is an instantiation of (Multiple Step) NP-iMCMC.

Conclusion   We have introduced the *nonparametric involutive MCMC algorithm* as a general framework for designing MCMC algorithms for models expressible in a universal PPL, and provided a correctness proof. To demonstrate the relative ease of make-to-order design of nonparametric extensions of existing MCMC algorithms, we have constructed several *new* algorithms, and demonstrated empirically that the expected features and statistical properties are preserved.

References

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

Campos, C. M. and Sanz-Serna, J. M. Extra chance generalized hybrid Monte Carlo. *Journal of Computational Physics*, 281:365–374, 2015.

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), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 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, 2020.

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. ACM, 2019.

Danos, V. and Ehrhard, T. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.

Devroye, L. Discrete univariate distributions. In *Non-Uniform Random Variate Generation*, chapter 10, pp. 485–553. Springer-Verlag, New York, NJ, USA, 1986.

Ehrhard, T., Tasson, C., and Pagani, M. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In Jagannathan, S. and Sewell, P. (eds.), *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014)*, pp. 309–320. ACM, 2014.

Ehrhard, T., Pagani, M., and Tasson, C. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *Proceedings of the ACM on Programming Languages*, 2(POPL):59:1–59:28, 2018.

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 of *Proceedings of Machine Learning Research*, pp. 1682–1690. PMLR, 2018.

Geman, S. and Geman, D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-6(6):721–741, 1984.

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, 12 1995.

Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1): 97–109, 04 1970.

Horowitz, A. M. A generalized guided Monte Carlo algorithm. *Physics Letters B*, 268(2):247–252, 1991. ISSN 0370-2693.

Hur, C., Nori, A. V., Rajamani, S. K., and Samuel, S. A provably correct sampler for probabilistic programs. In *Proceedings of the 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, pp. 475–488, 2015.

Kiselyov, O. Problems of the lightweight implementation of probabilistic programming. In *Proceedings of Workshop on Probabilistic Programming Semantics*, 2016.

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 of *Proceedings of Machine Learning Research*, 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), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2021)*, volume 12648 of *Lecture Notes in Computer Science*, pp. 432–461. Springer, 2021a.

Mak, C., Zaiser, F., and Ong, L. Nonparametric Hamiltonian Monte Carlo. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7336–7347. PMLR, 2021b.

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

Matheos, G., Lew, A. K., Ghavamizadeh, M., Russell, S., Cusumano-Towner, M., and Mansinghka, V. K. Transforming worlds: Automated involutive MCMC for open-universe probabilistic models. In *the 3rd Symposium on Advances in Approximate Bayesian Inference*, pp. 1–37, 2020.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

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 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*, volume 84 of *Proceedings of Machine Learning Research*, pp. 1037–1046. PMLR, 2018.

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. Chapman & Hall CRC Press, 2011.

Neklyudov, K., Welling, M., Egorov, E., and Vetrov, D. P. Involutive MCMC: a unifying framework. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7273–7282. PMLR, 2020.

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

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 of *JMLR Workshop and Conference Proceedings*, pp. 28–37. JMLR.org, 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):2399–3642, 2021.

Ścibior, A., Kammar, O., Vákár, M., Staton, S., Yang, H., Cai, Y., Ostermann, K., Moss, S. K., Heunen, C., and Ghahramani, Z. Denotational validation of higher-order Bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(POPL):60:1–60:29, 2018.

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

Sieber, K. Relating full abstraction results for different programming languages. pp. 373–387, 1990.

Sohl-Dickstein, J., Mudigonda, M., and DeWeese, M. R. Hamiltonian Monte Carlo without detailed balance. In *Proceedings of the 31th International Conference on Machine Learning (ICML 2014)*, volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 719–726. JMLR.org, 2014.

Staton, S. Commutative semantics for probabilistic programming. In Yang, H. (ed.), *Proceedings of the 26th European Symposium on Programming, (ESOP 2017), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2017)*, volume 10201 of *Lecture Notes in Computer Science*, pp. 855–879. Springer, 2017.

Staton, S., Yang, H., Wood, F. D., Heunen, C., and Kammar, O. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, pp. 525–534, 2016.

Tolpin, D., van de Meent, J.-W., Paige, B., and Wood, F. Output-sensitive adaptive metropolis-hastings for probabilistic programs. In Appice, A., Rodrigues, P. P., Santos Costa, V., Gama, J., Jorge, A., and Soares, C. (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 311–326, Cham, 2015. Springer International Publishing. ISBN 978-3-319-23525-7.

Turitsyn, K. S., Chertkov, M., and Vucelja, M. Irreversible Monte Carlo algorithms for efficient sampling. *Physica D: Nonlinear Phenomena*, 240(4):410–414, 2011. ISSN 0167-2789.

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

Wand, M., Culpepper, R., Giannakopoulos, T., and Cobb, A. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *Proceedings of the ACM on Programming Languages*, 2(ICFP):87:1–87:30, 2018.

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 of *JMLR Proceedings*, pp. 770–778. JMLR.org, 2011.

Wood, F. D., van de Meent, J., and Mansinghka, V. A new approach to probabilistic programming inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33 of *JMLR Workshop and Conference Proceedings*, pp. 1024–1032. JMLR.org, 2014.

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

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 of *Proceedings of Machine Learning Research*, 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 of *Proceedings of Machine Learning Research*, pp. 11534–11545. PMLR, 2020.

Nonparametric Hamiltonian Monte Carlo (Appendix)

A. Statistical PCF

In this section, we present a functional probabilistic programming language (PPL) with (stochastic) branching and recursion, and its operational semantics. We also define what it means for a program to be almost surely terminating and integrable. We conclude the section by showing that a broad class of programs satisfies the assumptions for the NP-iMCMC inference algorithm described in Sec. 3.

A.1. Syntax

Statistical PCF (SPCF) is a statistical probabilistic extension of the call-by-value PCF (Scott, 1993; Sieber, 1990) with the reals and Booleans as the ground types. The terms and part of the typing system of SPCF are presented in Fig. 7.

SPCF has three probabilistic constructs:

(1) The continuous sampler normal draws from the standard Gaussian distribution $\mathcal{N}$ with mean 0 and variance 1.
(2) The discrete sampler coin is a fair coin (formally coin draws from the Bernoulli distribution $\mathsf{Bern}(0.5)$ with probability 0.5).
(3) 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.

Remark A.1 (Continuous Sampler). The continuous sampler in most PPLs (Culpepper & Cobb, 2017; Wand et al., 2018; Ehrhard et al., 2018; Vákár et al., 2019; Mak et al., 2021a) draw from the standard uniform distribution $\mathcal{U}$ with endpoints 0 and 1. However, we decided against $\mathcal{U}$ since its support is not the whole of $\mathbb{R}$, which is a common target space for inference algorithms (e.g. Hamiltonian Monte Carlo (HMC) inference algorithm). Instead our continuous sampler draws from the *standard normal distribution* $\mathcal{N}$ which has the whole of $\mathbb{R}$ as its support. This design choice does not restrict nor extend our language as we will see in Ex. 6.

Remark A.2 (Discrete Sampler). Like (Danos & Ehrhard, 2011; Ścibior et al., 2018), we choose the fair coin as our discrete sampler for its simplicity. However, as shown in Ex. 7, this is not limiting. (Ehrhard et al., 2014), for example, samples from the discrete uniform distribution.

Following the convention, the set of all terms is denoted as Λ with meta-variables M, N, L, the set of free variables of a term M is denoted as $\mathsf{FV}(M)$ and the set of all closed terms is denoted as Λ^0. In the interest of readability, we sometimes use pseudocode in the style of ML (e.g. Ex. 5) to express SPCF terms.

Example 5. `let rec f x = if coin then f(x+normal) else x in f 0` is a simple program which keeps tossing a coin and sampling from the normal distribution until the first coin failure, upon which it returns the sum of samples from the normal distribution.

A.2. Primitive Functions

Primitive functions play an important role in the expressiveness of SPCF. To be concise, we only consider partial, measurable functions of types $\mathbb{R}^n \times 2^m \rightharpoonup \mathbb{R}$ or $\mathbb{R}^n \times 2^m \rightharpoonup 2$ for some $n, m \in \mathbb{N}$. Examples of these primitives include addition $+$, division $/$, comparison $<$ and equality $=$. As we will see in Ex. 6 and 7, it is important that the cumulative distribution functions (cdf) and probability density functions (pdf) of distributions are amongst the primitives in $\mathcal{F}$. However, we do not require all measurable functions to be primitives, unlike (Staton et al., 2016; Staton, 2017).

Example 6. (1) Let `cdfnormal` be the cdf of the standard normal distribution. Then, the standard uniform distribution with endpoints 0 and 1 can be described as `uniform = cdfnormal(normal)`.
(2) Any distribution with an inverse cdf `f` in the set of primitives can be described as `f(uniform)`. For instance, the

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

$$\sigma, \tau ::= \mathsf{R} \mid \mathsf{B} \mid \sigma \Rightarrow \tau$$

$$M, N, L ::= \underline{r} \mid \underline{a} \mid \underline{f}(M_1, \ldots, M_\ell) \qquad \text{(Constants and functions)}$$

$$\mid y \mid \lambda y.M \mid M\,N \qquad \text{(Higher-order)}$$

$$\mid \mathsf{if}(L, M, N) \mid \mathsf{Y}M \qquad \text{(Branching and recursion)}$$

$$\mid \mathsf{normal} \mid \mathsf{coin} \mid \mathsf{score}(M) \qquad \text{(Probabilistic)}$$

Typing system:

$$\frac{a \in 2}{\Gamma \vdash \underline{a} : \mathsf{B}} \qquad \frac{\{\Gamma \vdash M_i : \mathsf{R}\}_{i=1}^n \qquad \{\Gamma \vdash N_j : \mathsf{B}\}_{j=1}^m \qquad f : \mathbb{R}^n \times 2^m \rightharpoonup G}{\Gamma \vdash \underline{f}(M_1, \ldots, M_n, N_1, \ldots, N_m) : \begin{cases} \mathsf{R} & \text{if } G = \mathbb{R} \\ \mathsf{B} & \text{if } G = 2 \end{cases}}$$

$$\frac{\Gamma \vdash L : \mathsf{B} \qquad \Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{if}(L, M, N) : \sigma} \qquad \frac{\Gamma \vdash M : (\sigma \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau)}{\Gamma \vdash \mathsf{Y}M : \sigma \Rightarrow \tau}$$

$$\frac{}{\Gamma \vdash \mathsf{normal} : \mathsf{R}} \qquad \frac{}{\Gamma \vdash \mathsf{coin} : \mathsf{B}} \qquad \frac{\Gamma \vdash M : \mathsf{R}}{\Gamma \vdash \mathsf{score}(M) : \mathsf{R}}$$

Figure 7. Syntax of SPCF, where $r, q, p \in \mathbb{R}$, $a, b \in 2$, x, y, z are variables, and f, g, h ranges over a set $\mathcal{F}$ of primitive functions.

inverse cdf of the exponential distribution (with rate 1) is $f(p) := -\ln(1-p)$ and hence `-ln(1-uniform)` describes the distribution.

(3) The Poisson distribution can be specified using the uniform distribution ((Devroye, 1986)) as follows.

```
1   Poi(rate) = let p = exp(-rate) in
2               let rec f x p s = if s < uniform then f (x+1) (p*rate/x) (s+p) else x
3               in f 0 p p
```

Example 7. It might be beneficial for some inference algorithm if discrete distributions are specified using discrete random variables. Hence, we show how different discrete distributions can be specified by our discrete sampler `coin`.

(1) The Bernoulli distribution with probability $p \in [0, 1] \cap \mathbb{D}$, where $\mathbb{D} := \{\frac{n}{2^m} \mid n, m \in \mathbb{N}\}$ is the set of all Dyadic numbers, can be specified by

```
1   let rec bern(p) = if p = 0 then False else
2                     if p = 1 then True else
3                     if p < 0.5 then
4                        if coin then bern(2*p) else False
5                     else
6                        if coin then True else bern(2*(p-0.5))
```

(2) The geometric distribution with rate $p \in [0, 1] \cap \mathbb{D}$ can be specified by

```
1   geo(p) = count = 0; while bern(1-p): count += 1; return count
```

(3) The binomial distribution with $n \in \mathbb{N}$ trails and probability $p \in [0, 1] \cap \mathbb{D}$ can be specified by `bin(n,p) = sum([1 for i in range(n) if bern(p)])`.

(4) Let `pdfPio` and `pdfgeo` be the pdfs of the Poisson and geometric distributions respectively. Then, the Poisson distribution can be described by

```
1   Poi(rate) = n = geo(0.5);
2               score(pdfPoi(rate,n)/pdfgeo(0.5,n));
3               return n
```

A.3. Church Encodings

We can represent pairs and lists in SPCF using Church encoding as follows:

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

Moreover standard primitives on pairs and lists, such as `projection`, `len`, `append` and `sum`, can be defined easily.

A.4. Operational Semantics

A.4.1. TRACE SPACE

Since normal samples from the standard normal distribution $\mathcal{N}$ and coin from the Bernoulli distribution $\mathsf{Bern}(0.5)$, the **sample space** of SPCF is the union of the measurable spaces of $\mathbb{R}$ and 2. Formally it is the measurable space with set $\Omega := \mathbb{R} \cup 2$, σ-algebra $\Sigma_\Omega := \{V \cup W \mid V \in \mathcal{B}, W \in \Sigma_2\}$ and measure $\mu_\Omega(V \cup W) := \mathcal{N}(V) + \mathsf{Bern}(0.5)(W)$. We denote the product of n copies of the sample space as $(\Omega^n, \Sigma_{\Omega^n}, \mu_{\Omega^n})$ and call it the n-dimensional sample space.

A **trace** is a record of the values sampled in the course of an execution of a SPCF term. Hence, the **trace space** is the union of sample spaces of varying dimension. Formally it is the measurable space with set $\mathbb{T} := \bigcup_{n\in\mathbb{N}} \Omega^n$, σ-algebra $\Sigma_\mathbb{T} := \{\bigcup_{n\in\mathbb{N}} U_n \mid U_n \in \Sigma_{\Omega^n}\}$ and measure $\mu_\mathbb{T}(\bigcup_{n\in\mathbb{N}} U_n) = \sum_{n\in\mathbb{N}} \mu_{\Omega^n}(U_n)$. We present traces as lists, e.g. $[-0.2, \mathsf{T}, \mathsf{T}, 3.1, \mathsf{F}]$ and $[]$.

Remark A.3. Another way of recording the sampled value in a run of a SPCF term is to have separate records for the values of the continuous and discrete samples. In this case, the trace space will be the set $\bigcup_{n\in\mathbb{N}} \mathbb{R}^n \times \bigcup_{m\in\mathbb{N}} 2^m$. We find separating the continuous and discrete samples unnecessarily complex for our purposes and hence follow the more conventional definition of trace space.

A.4.2. SMALL-STEP REDUCTION

The small-step reduction of SPCF terms can be seen 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, $w > 0$ is a **weight**, and $\boldsymbol{t} \in \mathbb{T}$ a trace, as defined in Fig. 8.

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 $\mu_\mathbb{T}$. Similarly, in the rule for coin, a random boolean $a \in 2$ is sampled and recorded in the trace with an unchanged weight. In the rule for 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. Similar 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 and $\longrightarrow^*$ for the reflexive and transitive closure of the small-step reduction.

A.4.3. VALUE AND WEIGHT FUNCTIONS

Following (Borgström et al., 2016), we view the set Λ of all SPCF terms as $\bigcup_{n,m\in\mathbb{N}}(\mathsf{SK}_{n,m} \times \mathbb{R}^n \times 2^m)$ where $\mathsf{SK}_{n,m}$ is the set of SPCF terms with exactly n real-valued and m boolean-valued 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 $\mathsf{SK}_{n,m}$, the standard topology on $\mathbb{R}^n$ and the discrete topology on 2^m. Similarly the subspace Λ_v^0 of closed values inherits the Borel algebra on Λ.

Let M be a closed SPCF term. Its **value function** $\mathsf{value}_M : \mathbb{T} \to \Lambda_v^0 \cup \{\bot\}$ returns, given a trace, the output value of the program, if the program terminates in a value. Its **weight function** $\mathsf{weight}_M : \mathbb{T} \to [0, \infty)$ returns the final weight of the

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

$$V ::= \underline{r} \mid \underline{a} \mid \lambda y.M$$
$$R ::= \underline{f}(\underline{c_1}, \ldots, \underline{c_\ell}) \mid (\lambda y.M)\,V \mid \mathsf{if}(\underline{a}, M, N) \mid \mathsf{Y}(\lambda y.M)$$
$$\mid \mathsf{normal} \mid \mathsf{coin} \mid \mathsf{score}(\underline{r})$$
$$E ::= [] \mid E\,M \mid (\lambda y.M)\,E \mid \mathsf{if}(E, M, N) \mid \underline{f}(\underline{c_1}, \ldots, \underline{c_{i-1}}, E, M_{i+1}, \ldots, M_\ell) \mid \mathsf{Y}E$$
$$\mid \mathsf{score}(E)$$

Redex contractions:

$$\langle \underline{f}(\underline{c_1}, \ldots, \underline{c_\ell}), w, \boldsymbol{t} \rangle \longrightarrow \begin{cases} \langle \underline{f(c_1, \ldots, c_\ell)}, w, \boldsymbol{t} \rangle & \text{if } (c_1, \ldots, c_\ell) \in \mathsf{Dom}(f), \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

$$\langle (\lambda y.M)\,V, w, \boldsymbol{t} \rangle \longrightarrow \langle M[V/y], w, \boldsymbol{t} \rangle$$

$$\langle \mathsf{if}(\underline{a}, M, N), w, \boldsymbol{t} \rangle \longrightarrow \begin{cases} \langle M, w, \boldsymbol{t} \rangle & \text{if } a, \\ \langle N, w, \boldsymbol{t} \rangle & \text{otherwise} \end{cases}$$

$$\langle \mathsf{Y}(\lambda y.M), w, \boldsymbol{t} \rangle \longrightarrow \langle \lambda z.M[\mathsf{Y}(\lambda y.M)/y]\,z, w, \boldsymbol{t} \rangle \qquad \text{(for fresh variable } z)$$

$$\langle \mathsf{normal}, w, \boldsymbol{t} \rangle \longrightarrow \langle \underline{r}, w, t \mathbin{+\!\!+} [r] \rangle \qquad \text{(for some } r \in \mathbb{R})$$

$$\langle \mathsf{coin}, w, \boldsymbol{t} \rangle \longrightarrow \langle \underline{a}, w, t \mathbin{+\!\!+} [a] \rangle \qquad \text{(for some } a \in 2)$$

$$\langle \mathsf{score}(\underline{r}), w, \boldsymbol{t} \rangle \longrightarrow \begin{cases} \langle \underline{r}, r \cdot w, \boldsymbol{t} \rangle & \text{if } r > 0, \\ \mathsf{fail} & \text{otherwise.} \end{cases}$$

Evaluation contexts:

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

Figure 8. Small-step reduction of SPCF, where $r, q, p \in \mathbb{R}$, $a, b \in 2$, $c \in \mathbb{R} \cup 2$, x, y, z are variables, and f, g, h ranges over the set $\mathcal{F}$ of primitive functions.

corresponding execution. Formally:

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

It follows readily 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*** $\langle\!\langle M \rangle\!\rangle$ on Λ_v^0 given by

$$\langle\!\langle M \rangle\!\rangle : \Sigma_{\Lambda_v^0} \longrightarrow [0, \infty)$$
$$U \longmapsto \int_{\mathsf{value}_M^{-1}(U)} \mathsf{weight}_M \, \mathsf{d}\mu_{\mathbb{T}}$$

Remark A.4. A trace is in the support of the weight function if and only if the value function returns a (closed) value when given this trace. i.e. $\mathsf{Supp}(\mathsf{weight}_M) = \mathsf{value}_M^{-1}(\Lambda^0)$ for all closed SPCF term M.

Remark A.5. The weight function defined here is the density of the target distribution from which an inference algorithm typically samples. In this work, we call it the *weight function* when considering semantics following (Culpepper & Cobb, 2017; Vákár et al., 2019; Mak et al., 2021a), and call it ***density function*** when discussing inference algorithms following (Zhou et al., 2019; 2020; Cusumano-Towner et al., 2020).

A.5. Tree Representable Functions

We consider a necessary condition for the weight function of closed SPCF terms which would help us in designing inference algorithms for them. Note that not every function of type $\mathbb{T} \to [0, \infty)$ makes sense as a weight function. Consider the program
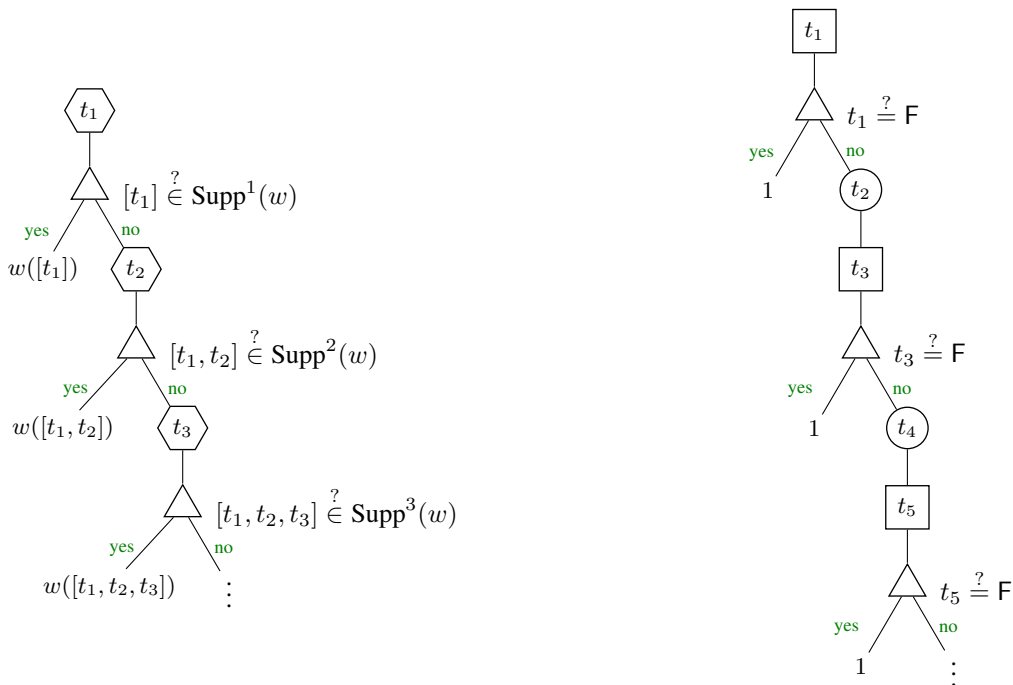
Figure 9. Program tree of a tree representable function w

`let rec f x = if coin then f(x+normal) else x in f 0` in Ex. 5. This program executes successfully with the trace $[\mathsf{T}, 0.5, \mathsf{F}]$. This immediately tells us that upon sampling T and 0.5, there must be a sample following them, and this third sample must be a boolean. In other words, the program does *not* terminate with any proper prefix of $[\mathsf{T}, 0.5, \mathsf{F}]$ such as $[\mathsf{T}, 0.5]$, nor any traces of the form $[\mathsf{T}, 0.5, r]$ for $r \in \mathbb{R}$.

Hence, we consider measurable functions $w : \mathbb{T} \to [0, \infty)$ satisfying

- ***prefix property***: whenever $\boldsymbol{t} \in \mathrm{Supp}^n(w)$[7] then for all $k < n$, we have $t^{1\ldots k} \notin \mathrm{Supp}^k(w)$; and

- ***type property***: whenever $\boldsymbol{t} \in \mathrm{Supp}^n(w)$ then for all $k < n$ and for all $t \in \Omega \setminus \mathsf{Type}(\boldsymbol{t}^{k+1})$[8] we have $\boldsymbol{t}^{1\ldots k} \mathbin{+\!\!+} [t] \notin \mathrm{Supp}^{k+1}(w)$.

They are called ***tree representable (TR) functions*** (Mak et al., 2021b) 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. 9 (left), where a hexagon node denotes an element of the input of type Ω; a triangular node gives the condition for $\boldsymbol{t} \in \mathrm{Supp}^n(w)$ (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 $[t_1, \ldots, t_n]$ in $\mathrm{Supp}(w)$. In fact, every program tree of a TR function w specifies a countable partition of $\mathrm{Supp}(w)$ via its branches. The prefix property guarantees that for each TR function w, there are program trees of the form in Fig. 9 representing w.

The program tree of M is depicted in Fig. 9 (right), where a circular node denotes a real-valued input and a squared node denotes a boolean-valued input.

The following proposition ties SPCF terms and TR functions together.

Proposition A.6. *Every closed SPCF term has a tree representable weight function.*

We will see in Sec. 3 how the TR functions, in particular the prefix property, is instrumental in the design of the inference algorithm.

---

[7]$\mathrm{Supp}^n(w) := \mathrm{Supp}(w) \cap \Omega^n$ for all $n \in \mathbb{N}$.

[8]The ***type*** $\mathsf{Type}(t)$ of a sample $t \in \Omega$ is $\mathbb{R}$ if $t \in \mathbb{R}$ and is 2 if $t \in 2$.

A.6. Almost Sure Termination and Integrability

Definition A.7. We say a SPCF term M ***terminates almost surely*** if M is closed and $\mu_{\mathbb{T}}(\{t \in \mathbb{T} \mid \exists V, w . \langle M, 1, [] \rangle \longrightarrow^* \langle V, w, t \rangle\}) = 1$.

We denote the set of terminating traces as $\mathbb{T}_{\mathsf{ter}} := \{t \in \mathbb{T} \mid \exists V, w . \langle M, 1, [] \rangle \longrightarrow^* \langle V, w, t \rangle\}$.

Remark A.8. The set of traces on which a closed SPCF term M terminates, i.e. $\{t \in \mathbb{T} \mid \exists V, w . \langle M, 1, [] \rangle \longrightarrow^* \langle V, w, t \rangle\}$, can be understood as the support of its weight function $\mathsf{Supp}(\mathsf{weight}_M)$, or as discussed in Rem. A.4, the traces on which the value function returns a value, i.e. $\mathsf{value}_M^{-1}(\Lambda_v^0)$. Hence, M almost surely terminates if and only if $\mu_{\mathbb{T}}(\mathsf{Supp}(\mathsf{weight}_M)) = \mu_{\mathbb{T}}(\mathsf{value}_M^{-1}(\Lambda_v^0)) = 1$.

Definition A.9. Following (Mak et al., 2021a), we say a trace $t \in \mathbb{T}$ is ***maximal*** w.r.t. a closed term M if there exists a term N, weight w where $\langle M, 1, [] \rangle \longrightarrow^* \langle N, w, t \rangle$ and for all $t' \in \mathbb{T} \setminus \{[]\}$ and all terms N', $\langle N, w, t \rangle \not\longrightarrow^* \langle N', w', t \mathbin{+\!\!+} t' \rangle$.

We denote the set of maximal traces as $\mathbb{T}_{\mathsf{max}}$.

Proposition A.10 ((Mak et al., 2021a), Lemma 9)**.** *A closed term M is almost surely terminating if $\mu_{\mathbb{T}}(\mathbb{T}_{\mathsf{max}} \setminus \mathbb{T}_{\mathsf{ter}}) = 0$.*

Proposition A.11. *The value measure $\langle\!\langle M \rangle\!\rangle$ of a closed almost surely terminating SPCF term M which does not contain* $\mathsf{score}(\cdot)$ *as a subterm is probabilistic.*

Definition A.12. We say a SPCF term M is ***integrable*** if M is closed and its value measure is finite, i.e. $\langle\!\langle M \rangle\!\rangle(\Lambda_v^0) < \infty$;

Proposition A.13. *An integrable term has an integrable weight function.*

Example 8. Now we look at a few examples in which we show that almost surely termination and integrability identify two distinct sets of SPCF terms.

(1) The term M_1 defined as `let rec f x = if coin then f (x+1) else x in score(2**(f 0))` almost surely terminates since it only diverges on the infinite trace $[\mathsf{F}, \mathsf{F}, \ldots]$ which has zero probability. However, it is *not* integrable as the value measure applied to all closed values $\langle\!\langle M_1 \rangle\!\rangle(\Lambda_v^0) = \int_{\{[\mathsf{T},\ldots,\mathsf{T},\mathsf{F}]\}} \mathsf{weight}_{M_1} \, d\mu_{\mathbb{T}} = \sum_{n=0}^{\infty} \int_{\{[\mathsf{T}]^n \mathbin{+\!\!+} [\mathsf{F}]\}} \mathsf{weight}_{M_1} \, d\mu_{2^n}{}^9 = \sum_{n=0}^{\infty} (\frac{1}{2})^{n+1} \cdot 2^n = \sum_{n=0}^{\infty} \frac{1}{2}$ is infinite.

(2) Consider the term M_2 defined as `if coin then Y (lambda x:x)0 else 1`. Since it reduces to a diverging term, namely `Y (lambda x:x)0`, with non-zero probability, it does not terminate almost surely. However, it is integrable, since $\langle\!\langle M_2 \rangle\!\rangle(\Lambda_v^0) = \int_{\{[\mathsf{F}]\}} \mathsf{weight}_{M_2} \, d\mu_{\mathbb{T}} = \frac{1}{2} < \infty$.

(3) The term M_3 defined as $\mathsf{if}(\mathsf{coin}, M_1, M_2)$ is neither almost surely terminating nor integrable, since M_1 is not integrable and M_2 is not almost surely terminating.

(4) All terms considered previously in Ex. 5 to 7 are both almost surely terminating and integrable.

---

[9] We write $[x]^n$ to be the list that contains n copies of x.

B. Hybrid Nonparametric Involutive MCMC and its Correctness

In this section, we present the ***Hybrid Nonparametric Involutive Markov chain Monte Carlo*** (Hybrid NP-iMCMC), an inference algorithm that simulates the probabilistic model specified by a given SPCF program that may contains both discrete and continuous samplers.

To start, we detail the Hybrid NP-iMCMC inference algorithm: its state space, conditions on the inputs and steps to generate the next sample; and study how the sampler moves between states of varying dimensions and returns new samples of a nonparametric probabilistic program. We then give an implementation of Hybrid NP-iMCMC in SPCF and demonstrate how the Hybrid NP-iMCMC method extends the MH sampler. Last but not least, we conclude with a discussion on the correctness of Hybrid NP-iMCMC.

B.1. State Spaces

A *state* in the Hybrid NP-iMCMC algorithm is a pair $(\boldsymbol{x}, \boldsymbol{v})$ of equal *dimension* (but not necessarily equal length) *parameter* and *auxiliary* variables. The parameter variable $\boldsymbol{x}$ is used to store traces and the auxiliary variable $\boldsymbol{v}$ is used to record randomness. Both variables are vectors of *entropies*, i.e. Real-Boolean pairs. This section gives the formal definitions of the entropy, parameter and auxiliary variables and the state, in preparation for the discussion of the Hybrid NP-iMCMC sampler.

B.1.1. ENTROPY SPACE

As shown in App. A, the reduction of a SPCF program is determined by the input trace $\boldsymbol{t} \in \mathbb{T} := \bigcup_{n \in \mathbb{N}} (\mathbb{R} \cup 2)^n$, a record of drawn values in a particular run of the program. Hence in order to simulate a probabilistic model described by a SPCF program, the Hybrid NP-iMCMC sampler should generate Markov chains on the trace space. However traversing through the trace space is a delicate business because the positions and numbers of discrete and continuous values in a trace given by a SPCF program may vary. (Consider `if coin: normal else: coin`.)

Instead, we pair each value $\boldsymbol{t}^i$ in a trace $\boldsymbol{t}$ with a random value t of the other type to make a Real-Boolean pair $(\boldsymbol{t}^i, t)$ (or $(t, \boldsymbol{t}^i)$). For instance, the trace $[\mathsf{T}, -3.1]$ can be made into a Real-Boolean vector $[(1.5, \mathsf{T}), (-3.1, \mathsf{T})]$ with randomly drawn values 1.5 and T. Now, the position of discrete and continuous random variables does not matter and the number of discrete and continuous random variables are fixed in each vector.

We call a Real-Boolean pair an ***entropy*** and define the ***entropy space*** $\mathbb{E}$ to be the product space $\mathbb{R} \times 2$ of the Borel measurable space and the Boolean measurable space, equipped with the σ-algebra $\Sigma_{\mathbb{E}} := \sigma(\{R \times B \mid R \in \mathcal{B}, B \in \Sigma_2\})$, and the product measure $\mu_{\mathbb{E}} := \mathcal{N} \times \mu_2$ where $\mu_2 := \mathsf{Bern}(0.5)$. Note the Radon-Nikodym derivative $\varphi_{\mathbb{E}}$ of $\mu_{\mathbb{E}}$ can be defined as $\varphi_{\mathbb{E}}(r, a) := \frac{1}{2}\varphi(r)$. A n-length ***entropy vector*** is then a vector of n entropies, formally an element in the product measurable space $(\mathbb{E}^n, \Sigma_{\mathbb{E}^n})$. We write $|\boldsymbol{x}|$ to mean the length of the entropy vector $\boldsymbol{x}$.

As mentioned earlier, the parameter variable of a state is an entropy vector that stores traces. Hence, it would be useless if a unique trace cannot be restored from an entropy vector. We found that such a recovery is possible if the trace is in the support of a tree representable function.

Say we would like to recover the trace $\hat{\boldsymbol{t}}$ that is used to form the entropy vector $\boldsymbol{x}$ by pairing each value in the trace with a random value of the other type. First we realise that traces can be made by selecting either the Real or Boolean component of each pair in a prefix of $\boldsymbol{x}$. For example, traces like $[]$, $[\mathsf{T}]$, $[-0.2]$, $[\mathsf{T}, 2.9]$ and $[-0.2, \mathsf{T}, \mathsf{F}]$ can be made from the entropy vector $[(-0.2, \mathsf{T}), (2.9, \mathsf{T}), (1.3, \mathsf{F})]$. We call these traces ***instances*** of the entropy vector. Formally, a trace $\boldsymbol{t} \in \mathbb{T}$ is an instance of an entropy vector $\boldsymbol{x} \in \mathbb{E}^n$ if $|\boldsymbol{t}| \leq n$ and $\boldsymbol{t}^i \in \{r, a \mid (r, a) = \boldsymbol{x}^i\}$ for all $i = 1, \ldots, |\boldsymbol{t}|$. We denote the set of all instances of $\boldsymbol{x}$ as $\mathsf{instance}(\boldsymbol{x}) \subseteq \mathbb{T}$. Then, the trace $\hat{\boldsymbol{t}}$ must be an instance of $\boldsymbol{x}$. Moreover, if we can further assume that $\hat{\boldsymbol{t}}$ is in the support of a tree representable function, then Prop. B.1 says we can uniquely identify $\hat{\boldsymbol{t}}$ amongst all instances of $\boldsymbol{x}$.

Proposition B.1. *There is at most one (unique) trace that is both an instance of an entropy vector and in the support of a tree representable function.*

Finally, we consider differentiability on the multi-dimensional entropy space. We say a function $f : \mathbb{E}^{k_1} \to \mathbb{E}^{k_2}$ is ***differentiable almost everywhere*** if for all $\boldsymbol{i} \in 2^{k_1}, \boldsymbol{j} \in 2^{k_2}$, the partial function $f_{\boldsymbol{i} \to \boldsymbol{j}} : \mathbb{R}^{k_1} \to \mathbb{R}^{k_2}$ where

$$f_{\boldsymbol{i} \to \boldsymbol{j}}(\boldsymbol{r}) = \boldsymbol{q} \qquad \Longleftrightarrow \qquad f(\mathsf{zip}(\boldsymbol{r}, \boldsymbol{i})) = (\mathsf{zip}(\boldsymbol{q}, \boldsymbol{j}))^{10}$$

---

[10]We write $\mathsf{zip}(\ell_1, \ell_2)$ to be the n-length vector $[(\ell_1{}^1, \ell_2{}^1), (\ell_1{}^2, \ell_2{}^2), \ldots, (\ell_1{}^n, \ell_2{}^n)] \in (L_1 \times L_2)^n$ for any vectors $\ell_1 \in L_1^{n_1}$ and $\ell_2 \in L_2^{n_2}$ with $n := \min\{n_1, n_2\}$.

is differentiable *almost everywhere* on its domain $\mathsf{Dom}(f_{i \rightarrow j}) := \{r \in \mathbb{R}^{k_1} \mid \exists q \in \mathbb{R}^{k_2} . f(\mathsf{zip}(r, i)) = (\mathsf{zip}(q, j))\}$. The Jacobian of f on $(\mathsf{zip}(r, i))$ is given by $\nabla f_{i \rightarrow j}(r)$, if it exists.

B.1.2. PARAMETER SPACE

A parameter variable x of dimension n is an entropy vector of length $\iota_{\mathbb{X}}(n)$ where $\iota_{\mathbb{X}} : \mathbb{N} \rightarrow \mathbb{N}$ is a strictly monotone map. For instance, the parameter variable $x := [(-0.2, \mathsf{T}), (2.9, \mathsf{T}), (1.3, \mathsf{F})]$ is of dimension two if $\iota_{\mathbb{X}}(n) := n + 1$, and dimension three if $\iota_{\mathbb{X}}(n) := n$. We write $\dim(x)$ to mean the dimension of x and $|x|$ to mean the length of x. Hence, $\dim(x) \leq |x|$ and $\iota_{\mathbb{X}}(\dim(x)) = |x|$. We extend the notion of dimension to traces and say a trace $t \in \mathbb{T}$ has dimension n if $|t| = \iota_{\mathbb{X}}(n)$. Importantly, we assume that every trace in the support of w has a dimension (w.r.t. $\iota_{\mathbb{X}}$), i.e. $\mathsf{Supp}(w) = \bigcup_{n \in \mathbb{N}} \mathsf{Supp}^{\iota_{\mathbb{X}}(n)}(w)$.

Formally, the n-dimensional parameter space $(\mathbb{X}^{(n)}, \Sigma_{\mathbb{X}^{(n)}})$ is the product of $\iota_{\mathbb{X}}(n)$ copies of the entropy space $(\mathbb{E}, \Sigma_{\mathbb{E}})$ and the base measure $\mu_{\mathbb{X}^{(n)}}$ on $\mathbb{X}^{(n)}$ is the product of $\iota_{\mathbb{X}}(n)$ copies of the entropy measure $\mu_{\mathbb{E}}$ with the Radon-Nikodym derivative $\varphi_{\mathbb{X}^{(n)}}$. For ease of reference, we write $(\mathbb{X}, \Sigma_{\mathbb{X}}, \mu_{\mathbb{X}})$ for the one-dimensional parameter space.

B.1.3. AUXILIARY SPACE

Similarly, an auxiliary variable v of dimension n is an entropy vector of length $\iota_{\mathbb{Y}}(n)$ where $\iota_{\mathbb{Y}} : \mathbb{N} \rightarrow \mathbb{N}$ is a strictly monotone map. The n-dimensional auxiliary space $(\mathbb{Y}^{(n)}, \Sigma_{\mathbb{Y}^{(n)}})$ is the product of $\iota_{\mathbb{Y}}(n)$ copies of the entropy space $(\mathbb{E}, \Sigma_{\mathbb{E}})$ and the base measure $\mu_{\mathbb{Y}^{(n)}}$ on $\mathbb{Y}^{(n)}$ is the product of $\iota_{\mathbb{Y}}(n)$ copies of the entropy measure $\mu_{\mathbb{E}}$ with the Radon-Nikodym derivative $\varphi_{\mathbb{Y}^{(n)}}$. For ease of reference, we write $(\mathbb{Y}, \Sigma_{\mathbb{Y}}, \mu_{\mathbb{Y}})$ for the one-dimensional auxiliary space.

B.1.4. STATE SPACE

A ***state*** is a pair of *equal dimension but not necessarily equal length* parameter and auxiliary variable. For instance with $\iota_{\mathbb{X}}(n) := n + 1$ and $\iota_{\mathbb{Y}}(n) := n$, the parameter variable $x := [(-0.2, \mathsf{T}), (2.9, \mathsf{T}), (1.3, \mathsf{F})]$ and the auxiliary variable $v := [(1.5, \mathsf{T}), (-2.1, \mathsf{F})]$ are both of dimension two and (x, v) is a two-dimensional state.

Formally, the ***state space*** $\mathbb{S}$ is the list measurable space of the product of parameter and auxiliary spaces of equal dimension, i.e. $\mathbb{S} := \bigcup_{n \in \mathbb{N}} (\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)})$, equipped with the σ-algebra $\Sigma_{\mathbb{S}} := \sigma\{X_n \times V_n \mid X_n \in \Sigma_{\mathbb{X}^{(n)}}, V_n \in \Sigma_{\mathbb{Y}^{(n)}}, n \in \mathbb{N}\}$ and measure $\mu_{\mathbb{S}}(S) := \sum_{n \in \mathbb{N}} \int_{\mathbb{Y}^{(n)}} \mu_{\mathbb{X}^{(n)}}(\{x \in \mathbb{X}^{(n)} \mid (x, v) \in S\}) \mu_{\mathbb{Y}^{(n)}}(dv)$. We write $\mathbb{S}^{(n)}$ for the set consisting of all n-dimensional states.

We extend the notion of instances to states and say a trace t is an instance of a state (x, v) if it is an instance of the parameter component x.

The distinction between dimension and length in parameter and auxiliary variables gives us the necessary pliancy to discuss techniques for further extension of the Hybrid NP-iMCMC sampler in App. C. Before that, we present the inputs to the Hybrid NP-iMCMC sampler.

B.2. Inputs of Hybrid NP-iMCMC Algorithm

Besides the target density function, the Hybrid NP-iMCMC sampler, like iMCMC, introduces randomness via ***auxiliary kernels*** and moves around the state space via ***involutions*** in order to propose the next sample. We now examine each of these inputs closely.

B.2.1. TARGET DENSITY FUNCTION

Similar to other inference algorithms for probabilistic programming, the Hybrid NP-iMCMC sampler takes the *weight function* $w : \mathbb{T} \rightarrow [0, \infty)$ as the target density function. Recall $w(t)$ gives the weight of a particular run of the given probabilistic program indicated by the trace t. By Prop. A.6, the weight function w is tree representable. For the sampler to work properly, we also require weight function w to satisfy the following assumptions.

(H1) w is ***integrable***, i.e. $\int_{\mathbb{T}} w \, d\mu_{\mathbb{T}} =: Z < \infty$ (otherwise, the inference problem is undefined).

(H2) w is ***almost surely terminating (AST)***, i.e. $\mu_{\mathbb{T}}(\{t \in \mathbb{T} \mid w(t) > 0\}) = 1$ (otherwise, the loop in the Hybrid NP-iMCMC algorithm may not terminate almost surely).

Virtually all useful probabilistic models can be specified by SPCF programs with densities satisfying H1 and 2. Exceptions are models that are not normalizable or diverge with non-zero probability. (See App. A.6 for more details.)

B.2.2. AUXILIARY KERNELS

To introduce randomness, the Hybrid NP-iMCMC sampler takes, for each $n \in \mathbb{N}$, a probability ***auxiliary kernel*** $K^{(n)}$: $\mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$ which gives a probability distribution $K^{(n)}(\boldsymbol{x}, \cdot)$ on $\mathbb{Y}^{(n)}$ for each n-dimensional parameter variable $\boldsymbol{x}$. We assume each auxiliary kernel $K^{(n)}$ has a probability density function (pdf) $\mathsf{pdf}\,K^{(n)} : \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)} \rightarrow [0, \infty)$ w.r.t. $\mu_{\mathbb{Y}^{(n)}}$.

B.2.3. INVOLUTIONS

To move around the state space $\mathbb{S}$, the Hybrid NP-iMCMC sampler takes, for each $n \in \mathbb{N}$, an endofunction $\Phi^{(n)}$ on $\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$ that is both involutive and differentiable almost everywhere. We require the set $\{\Phi^{(n)}\}_n$ of involutions to satisfy the *projection commutation property*:

(H3)   For all $(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}$ where $\dim(\boldsymbol{x}) = m$, if $\mathsf{Supp}^{\iota_{\mathbb{X}}(n)}(w) \cap \mathsf{instance}(\boldsymbol{x}) \neq \varnothing$ for some n, then for all $k = n, \dots, m$,
$\mathsf{take}_k(\Phi^{(m)}(\boldsymbol{x}, \boldsymbol{v})) = \Phi^{(k)}(\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v}))$

where take_k is the projection that given a state $(\boldsymbol{x}, \boldsymbol{v})$, takes the first $\iota_{\mathbb{X}}(k)$ coordinates of $\boldsymbol{x}$ and the first $\iota_{\mathbb{Y}}(k)$ coordinates of $\boldsymbol{v}$ and forms a k-dimensional state.

The projection commutation property ensures that the order of applying a projection and an involution to a state (which has an instance in the support of the target density function) does not matter.

B.3. The Hybrid NP-iMCMC Algorithm

After identifying the state space and the necessary conditions on the inputs of the Hybrid NP-iMCMC sampler, we have enough foundation to describe the algorithm.

Given a SPCF program M with weight function w on the trace space $\mathbb{T}$, the ***Hybrid Nonparametric Involutive Markov chain Monte Carlo (Hybrid NP-iMCMC)*** algorithm generates a Markov chain on $\mathbb{T}$ as follows. Given a current sample $\boldsymbol{t}_0$ of dimension k_0 (i.e. $|\boldsymbol{t}_0| = \iota_{\mathbb{X}}(k_0)$),

1. (Initialisation Step) Form a k_0-dimensional parameter variable $\boldsymbol{x}_0 \in \mathbb{X}^{(k_0)}$ by pairing each value $\boldsymbol{t}_0{}^i$ in $\boldsymbol{t}_0$ with a randomly drawn value t of the other type to make a pair $(\boldsymbol{t}_0{}^i, t)$ or $(t, \boldsymbol{t}_0{}^i)$ in the entropy space $\mathbb{E}$. Note that $\boldsymbol{t}_0$ is the unique instance of $\boldsymbol{x}_0$ that is in the support of w.

2. (Stochastic Step) Introduce randomness to the sampler by drawing a k_0-dimensional value $\boldsymbol{v}_0 \in \mathbb{Y}^{(k_0)}$ from the probability measure $K^{(k_0)}(\boldsymbol{x}_0, \cdot)$.

3. (Deterministic Step) Move around the n-dimensional state space $\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$ and compute the new state $(\boldsymbol{x}, \boldsymbol{v})$ by applying the involution $\Phi^{(n)}$ to the *initial state* $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ where $n = \dim(\boldsymbol{x}_0) = \dim(\boldsymbol{v}_0)$.

4. (Extend Step) Test whether any instance $\boldsymbol{t}$ of $\boldsymbol{x}$ is in the support of w. If so, proceed to the next step with $\boldsymbol{t}$ as the proposed sample; otherwise

    (i) Extend the n-dimensional initial state to a state $(\boldsymbol{x}_0 +\!\!+ \boldsymbol{y}_0, \boldsymbol{v}_0 +\!\!+ \boldsymbol{u}_0)$ of dimension $n+1$ where $\boldsymbol{y}_0$ and $\boldsymbol{u}_0$ are values drawn randomly from $\mu_{\mathbb{E}^{\iota_{\mathbb{X}}(n+1) - \iota_{\mathbb{X}}(n)}}$ and $\mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(n+1) - \iota_{\mathbb{Y}}(n)}}$ respectively,
    (ii) Go to Step 3 with the initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ replaced by $(\boldsymbol{x}_0 +\!\!+ \boldsymbol{y}_0, \boldsymbol{v}_0 +\!\!+ \boldsymbol{u}_0)$.

5. (Accept/reject Step) Accept the proposed sample $\boldsymbol{t}$ as the next sample with probability

$$\min\left\{1; \frac{w(\boldsymbol{t}) \cdot \mathsf{pdf}\,K^{(k)}(\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v})) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v})}{w(\boldsymbol{t}_0) \cdot \mathsf{pdf}\,K^{(k_0)}(\mathsf{take}_{k_0}(\boldsymbol{x}_0, \boldsymbol{v}_0)) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot |\det(\nabla\Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0))|\right\} \quad (1)$$

where $n = \dim(\boldsymbol{x}_0) = \dim(\boldsymbol{v}_0)$, k is the dimension of $\boldsymbol{t}$ and k_0 is the dimension of $\boldsymbol{t}_0$; otherwise reject the proposal and repeat $\boldsymbol{t}_0$.

Remark B.2. The integrable assumption on the target density (H1) ensures the inference problem is well-defined. The almost surely terminating assumption on the target density (H2) guarantees that the Hybrid NP-iMCMC sampler almost surely terminates. (See App. B.4.1 for a concrete proof.) The projection commutation property on the involutions (H3) allows us to define the invariant distribution

B.3.1. MOVEMENT BETWEEN SAMPLES OF VARYING DIMENSIONS

All MCMC samplers that simulate a nonparametric model must decide how to move between samples of varying dimensions. We now discuss how the Hybrid NP-iMCMC sampler as given in App. B.3 achieves this.

Form initial and new states in the same dimension Say the current sample t_0 has a dimension of k_0. Step 1 to 3 form a k_0-dimensional initial state (x_0, v_0) and a new k_0-dimensional state (x, v).

Move between dimensions The novelty of Hybrid NP-iMCMC is its ability to generate a proposed sample t in the support of the target density w which may not be of same dimension as t_0. This is achieved by Step 4.

Propose a sample of a lower dimension Step 4 first checks whether any instance of the parameter-component $x \in \mathbb{X}^{(k_0)}$ of the new state (computed in Step 3) is in the support of w. If so, we proceed to Step 5 with that instance, say t, as the proposed sample.

Say the dimension of t is k. Then, we must have $k \leq k_0$ as the instance $t \in \mathbb{T}$ of a k_0-dimensional parameter $x \in \mathbb{X}^{(k_0)}$ must have a dimension that is lower than or equals to k_0. Hence, the dimension of the proposed sample t is lower than or equals to the current sample t_0.

Propose a sample of a higher dimension Otherwise (i.e. none of the instances of $x \in \mathbb{X}^{(k_0)}$ is in the support of w) Step 4 extends the initial state $(x_0, v_0) \in \mathbb{X}^{(k_0)} \times \mathbb{Y}^{(k_0)}$ to $(x_0 + y_0, v_0 + u_0) \in \mathbb{X}^{(k_0+1)} \times \mathbb{Y}^{(k_0+1)}$; and computes a new $(k_0 + 1)$-dimensional state $(x + y, v + u) \in \mathbb{X}^{(k_0+1)} \times \mathbb{Y}^{(k_0+1)}$ (via Step 3). This process of incrementing the dimensions of both the initial and new states is repeated until an instance t of the new state, say of dimension n, is in the support of w. At which point, the proposed sample is set to be t.

Say the dimension of t is k. Then, we must have $k > k_0$ as t is *not* an instance of the k_0-dimensional parameter $x \in \mathbb{X}^{(k_0)}$ but one of $x + y \in \mathbb{X}^{(n)}$. Hence, the dimension of the proposed sample t is higher than the current sample t_0.

Accept or reject the proposed sample Say the proposed sample t is of dimension k. With the probability given in Equation (1), Step 5 accepts t as the next sample and Hybrid NP-iMCMC updates the current sample t_0 of dimension k_0 to a sample t of dimension k. Otherwise, the current sample t_0 is repeated and the dimension remains unchanged.

B.3.2. HYBRID NP-iMCMC IS A GENERALISATION OF NP-iMCMC

Given a target density w on $\bigcup_{n \in \mathbb{N}} \mathbb{R}^n$, we can set the entropy space $\mathbb{E}$ to be $\mathbb{R}$ and the index maps $\iota_{\mathbb{X}}$ and $\iota_{\mathbb{Y}}$ to be identities. Then, the n-dimensional parameter space $\mathbb{X}^{(n)} := \mathbb{R}^n$, the n-dimensional auxiliary space $\mathbb{Y}^{(n)} := \mathbb{R}^n$ and the state space $\mathbb{S} := \bigcup_{n \in \mathbb{N}} (\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}) = \bigcup_{n \in \mathbb{N}} (\mathbb{R}^n \times \mathbb{R}^n)$ of the Hybrid NP-iMCMC sampler matches with those given in Sec. 3.2 for the NP-iMCMC sampler. An instance t is then a prefix $x^{1..k}$ of a parameter variable x. Moreover, the assumptions H1 to 3 on the inputs of Hybrid NP-iMCMC are identical to those V1 to 3 on the inputs of NP-iMCMC. Hence the Hybrid NP-iMCMC algorithm (App. B.3) is a generalisation of the NP-iMCMC sampler (Fig. 2).

B.3.3. PSEUDOCODE OF HYBRID NP-iMCMC ALGORITHM

We implement the Hybrid NP-iMCMC algorithm in the flexible and expressive SPCF language explored in App. A.

The `NPiMCMC` function in Listing 2 is an implementation of the Hybrid NP-iMCMC algorithm in SPCF. We assume that the following SPCF types and terms exist. For each $n \in \mathbb{N}$, the SPCF types `T`, `X[n]` and `Y[n]` implements $\mathbb{T}$, $\mathbb{X}^{(n)}$ and $\mathbb{Y}^{(n)}$ respectively; the SPCF term `w` of type `T -> R` implements the target density w; for each $n \in \mathbb{N}$, the SPCF terms `auxkernel[n]` of type `X[n] -> Y[n]` implements the auxiliary kernel $K^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$; `pdfauxkernel[n]` of type `X[n]*Y[n] -> R` implements the probability density function $\text{pdf} K^{(n)} : \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)} \to \mathbb{R}$ of the auxiliary kernel; `involution[n]` of type `X[n]*Y[n] -> X[n]*Y[n]` implements the involution $\Phi^{(n)}$ on $\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$; and

Listing 2. Pseudocode of the Hybrid NP-iMCMC algorithm

```
1   def NPiMCMC(t0):
2     k0 = dim(t0)                                              # initialisation step
3     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4     v0 = auxkernel[k0](x0)                                    # stochastic step
5     (x,v) = involution[k0](x0,v0)                             # deterministic step
6     n = k0                                                    # extend step
7     while not intersect(instance(x),support(w)):
8       x0 = x0 + [(normal, coin)]*(indexX(n+1)-indexX(n))
9       v0 = v0 + [(normal, coin)]*(indexY(n+1)-indexY(n))
10      n = n + 1
11      (x,v) = involution[n](x0,v0)
12    t = intersect(instance(x),support(w))[0]                 # accept/reject step
13    k = dim(t)
14    return t if uniform < min{1, w(t)/w(t0) * pdfauxkernel[k](proj((x,v),k))/
15                                      pdfauxkernel[k0](proj((x0,v0),k0)) *
16                                  pdfpar[n](x)/pdfpar[n](x0) *
17                                  pdfaux[n](v)/pdfaux[n](v0) *
18                                  absdetjacinv[n](x0,v0)}
19                 else t0
```

`absdetjacinv[n]` of type `X[n]*Y[n] -> R` implements the absolute value of the Jacobian determinant of $\Phi^{(n)}$.

We further assume that the following primitives are implemented: `dim` returns the dimension of a given trace; `indexX` and `indexY` implements the maps $\iota_{\mathbb{Y}}$ and $\iota_{\mathbb{X}}$ respectively; `pdfpar[n]` implements the derivative $\varphi_{\mathbb{X}^{(n)}}$ of the n-dimensional parameter space $\mathbb{X}^{(n)}$; `pdfaux[n]` implements the derivative $\varphi_{\mathbb{Y}^{(n)}}$ of the n-dimensional auxiliary space $\mathbb{Y}^{(n)}$; `instance` returns a list of all instances of a given entropy vector; `support` returns a list of traces in the support of a given function; and `proj` implements the projection function where `proj((x,v),k)=(x[:indexX(k)],v[:indexY(k)])`.

B.4. Correctness

The Hyrbid Nonparametric Involutive Markov chain Monte Carlo (Hyrbid NP-iMCMC) algorithm is presented in App. B.3 for the simulation of probabilistic models specified by probabilistic programs.

We justify this by proving that the Markov chain generated by iterating the Hybrid NP-iMCMC algorithm preserves the target distribution, specified by

$$\nu : \Sigma_{\mathbb{T}} \longrightarrow [0, \infty)$$
$$U \longmapsto \frac{1}{Z} \int_U w \, d\mu_{\mathbb{T}} \qquad \text{where } Z := \int_{\mathbb{T}} w \, d\mu_{\mathbb{T}},$$

as long as the target density function w (given by the weight function of the probabilistic program) is integrable (H1) and almost surely terminating (H2); with a probability kernel $K^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$ and an endofunction $\Phi^{(n)}$ on $\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$ that is involutive and differentiable almost everywhere for each $n \in \mathbb{N}$ such that $\{\Phi^{(n)}\}_n$ satisfies the projection commutation property (H3).

Throughout this chapter, we assume the assumptions stated above, and prove the followings.

1. The Hybrid NP-iMCMC sampler almost surely returns a sample for the simulation (Lem. B.4).

2. The state movement in the Hybrid NP-iMCMC sampler preserves a distribution on the states (Lem. B.14).

3. The marginalisation of the state distribution which the state movement of Hybrid NP-iMCMC preserves coincides with the target distribution (Lem. B.17).

B.4.1. Almost Sure Termination

In Rem. B.2, we asserted that the almost surely terminating assumption (H2) on the target density guarantees that the Hybrid NP-iMCMC algorithm (App. B.3) almost surely terminates. We justify this claim here.

Step 3 in the Hybrid NP-iMCMC algorithm (App. B.3) repeats itself if the sample-component x of the new state (x, v) (computed by applying the involution $\Phi^{(n)}$ on the extended initial state (x_0, v_0)) does not have an instance in the support of w. This loop halts almost surely if the measure of

$$\{(x_0, v_0) \in \mathbb{S} \mid (x, v) = \Phi^{(n)}(x_0, v_0) \text{ and } \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\}$$

tends to zero as the dimension n tends to infinity. Since $\Phi^{(n)}$ is invertible and $|\det \nabla \Phi^{(n)}(x_0, v_0)| > 0$ for all $n \in \mathbb{N}$ and $(x_0, v_0) \in \mathbb{S}$,

$$\mu_{\mathbb{S}}(\{(x_0, v_0) \in \mathbb{S} \mid (x, v) = \Phi^{(n)}(x_0, v_0) \text{ and } \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\})$$
$$= \Phi^{(n)}{}_*\mu_{\mathbb{S}}(\{(x, v) \in \mathbb{S} \mid \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\})$$
$$< \mu_{\mathbb{S}}(\{(x, v) \in \mathbb{S} \mid \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\})$$
$$= \mu_{\mathbb{X}^{(n)}}(\{x \in \mathbb{X}^{(n)} \mid \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\}).$$

Thus it is enough to show that the measure of a n-dimensional parameter variable not having any instances in the support of w tends to zero as the dimension n tends to infinity, i.e.

$$\mu_{\mathbb{X}^{(n)}}(\{x \in \mathbb{X}^{(n)} \mid \mathsf{instance}(x) \cap \mathrm{Supp}(w) = \varnothing\}) \to 0 \qquad \text{as} \qquad n \to \infty.$$

We start with the following proposition which shows that the chance of a n-dimensional parameter variable having some instances in the support of w is the same as the chance of w terminating before n reduction steps.

Proposition B.3. $\mu_{\mathbb{X}^{(n)}}(\{x \in \mathbb{X}^{(n)} \mid \mathsf{instance}(x) \cap \mathit{Supp}(w) \neq \varnothing\}) = \mu_{\mathbb{T}}(\bigcup_{i=1}^{n} \mathit{Supp}^{\iota_{\mathbb{X}}(i)}(w))$ *for all $n \in \mathbb{N}$ and all tree representable function w.*

Proof. Let $n \in \mathbb{N}$ and w be a tree representable function.

For each $i \leq n$, we unpack the set $\{x \in \mathbb{X}^{(i)} \mid \mathsf{instance}(x) \cap A \neq \varnothing\}$ of i-dimensional parameter variables that has an instance in the set $A \in \Sigma_{\mathbb{N}^{\iota_{\mathbb{X}}(i)}}$ of traces of length $\iota_{\mathbb{X}}(i)$ where $\mathbb{N} := \mathbb{R} \cup \mathbb{2}$. Write $\pi : \{1, \dots, \iota_{\mathbb{X}}(i)\} \to \{\mathbb{R}, \mathbb{2}\}$ for the measurable space $\pi(1) \times \pi(2) \times \cdots \times \pi(\iota_{\mathbb{X}}(i))$ with a probability measure $\mu_{\pi} := \mu_{\mathbb{N}^{\iota_{\mathbb{X}}(i)}}$ on π; π^{-1} for the "inverse" measurable space of π, i.e. $\pi^{-1}(j) := \mathbb{N} \setminus \pi(j)$ for all $j \leq \iota_{\mathbb{X}}(i)$; and S for the set of all such measurable spaces. Then, for any i-dimensional parameter variable x, $t \in \mathsf{instance}(x) \cap A$ if and only if there is some $\pi \in S$ where $t \in A \cap \pi$ and $x \in \mathsf{zip}(A \cap \pi, \pi^{-1})$. Hence, $\{x \in \mathbb{X}^{(i)} \mid \mathsf{instance}(x) \cap A \neq \varnothing\}$ can be written as $\bigcup_{\pi \in S} \mathsf{zip}(A \cap \pi, \pi^{-1})$. Moreover $\mu_{\mathbb{X}^{(i)}}(\mathsf{zip}(A \cap \pi, \pi^{-1})) = \mu_{\pi}(A \cap \pi) \cdot \mu_{\pi^{-1}}(\pi^{-1}) = \mu_{\pi}(A \cap \pi)$.

Consider the case where $A := \mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w)$. Then, we have

$$\{x \in \mathbb{X}^{(i)} \mid \mathsf{instance}(x) \cap \mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \neq \varnothing\} = \bigcup_{\pi \in S} \mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi, \pi^{-1}).$$

We first show that this is a disjoint union, i.e. for all $\pi \in S$, $\mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi, \pi^{-1})$ are disjoint. Let $x \in \mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi_1, \pi_1^{-1}) \cap \mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi_2, \pi_2^{-1})$ where $\pi_1, \pi_2 \in S$. Then, at least one instance t_1 of x is in $\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi_1$ and similarly at least one instance t_2 of x is in $\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi_2$. By Prop. B.1, $t_1 = t_2$ and hence $\pi_1 = \pi_2$.

Since $\mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi, \pi^{-1})$ are disjoint for all $\pi \in S$, we have

$$\mu_{\mathbb{X}^{(i)}}(\{x \in \mathbb{X}^{(i)} \mid \mathsf{instance}(x) \cap \mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \neq \varnothing\}) = \mu_{\mathbb{X}^{(i)}}(\bigcup_{\pi \in S} \mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi, \pi^{-1}))$$

$$= \sum_{\pi \in S} \mu_{\mathbb{X}^{(i)}}(\mathsf{zip}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi, \pi^{-1})) = \sum_{\pi \in S} \mu_{\pi}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi)$$

$$= \sum_{\pi \in S} \mu_{\mathbb{N}^{\iota_{\mathbb{X}}(i)}}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w) \cap \pi) = \mu_{\mathbb{N}^{\iota_{\mathbb{X}}(i)}}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w)) = \mu_{\mathbb{T}}(\mathrm{Supp}^{\iota_{\mathbb{X}}(i)}(w)).$$

Finally, $\{\boldsymbol{x} \in \mathbb{X}^{(n)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing\}$ is equal to $\bigcup_{i=1}^{n} \{\boldsymbol{x} \in \mathbb{X}^{(i)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}^{\iota_\mathbb{X}(i)}(w) \neq \varnothing\} \times \mathbb{E}^{\iota_\mathbb{X}(n) - \iota_\mathbb{X}(i)}$ and hence

$$\mu_{\mathbb{X}^{(n)}}(\{\boldsymbol{x} \in \mathbb{X}^{(n)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing\})$$

$$= \mu_{\mathbb{X}^{(n)}}(\bigcup_{i=1}^{n} \{\boldsymbol{x} \in \mathbb{X}^{(i)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}^{\iota_\mathbb{X}(i)}(w) \neq \varnothing\} \times \mathbb{E}^{\iota_\mathbb{X}(n) - \iota_\mathbb{X}(i)})$$

$$= \sum_{i=1}^{n} \mu_{\mathbb{X}^{(i)}}(\{\boldsymbol{x} \in \mathbb{X}^{(i)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}^{\iota_\mathbb{X}(i)}(w) \neq \varnothing\})$$

$$= \sum_{i=1}^{n} \mu_\mathbb{T}(\mathsf{Supp}^{\iota_\mathbb{X}(i)}(w))$$

$$= \mu_\mathbb{T}(\bigcup_{i=1}^{n} \mathsf{Supp}^{\iota_\mathbb{X}(i)}(w))$$

$\square$

Prop. B.3 links the termination of the Hybrid NP-iMCMC sampler with that of the target density function w. Hence by assuming that w terminates almost surely (H2), we can deduce that the Hybrid NP-iMCMC algorithm (App. B.3) almost surely terminates.

Lemma B.4 (Almost Sure Termination). *Assuming H2, the Hybrid NP-iMCMC algorithm (App. B.3) almost surely terminates.*

Proof. Since $\Phi^{(n)}$ is invertible for all $n \in \mathbb{N}$, and w almost surely terminates (H2), i.e. $\lim_{m \to \infty} \mu_\mathbb{T}(\bigcup_{j=1}^{m} \mathsf{Supp}^j(w)) = 1$, we deduce from Prop. B.3 that

$$\mu_\mathbb{S}(\{(\boldsymbol{x}_0, \boldsymbol{v}_0) \in \mathbb{S} \mid (\boldsymbol{x}, \boldsymbol{v}) = \Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0) \text{ and } \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) = \varnothing\})$$

$$< \mu_{\mathbb{X}^{(n)}}(\{\boldsymbol{x} \in \mathbb{X}^{(n)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) = \varnothing\})$$

$$= \mu_{\mathbb{X}^{(n)}}(\mathbb{X}^{(n)} \setminus \{\boldsymbol{x} \in \mathbb{X}^{(n)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing\})$$

$$= 1 - \mu_{\mathbb{X}^{(n)}}(\{\boldsymbol{x} \in \mathbb{X}^{(n)} \mid \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing\})$$

$$= 1 - \mu_\mathbb{T}(\bigcup_{i=1}^{n} \mathsf{Supp}^{\iota_\mathbb{X}(i)}(w)) \qquad\qquad\qquad\qquad \text{(Prop. B.3)}$$

$$\to 1 - 1 = 0 \quad \text{as } n \to \infty. \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{(H2)}$$

So the probability of satisfying the condition of the loop in Step 3 of Hybrid NP-iMCMC sampler tends to zero as the dimension n tends to infinity, making the Hybrid NP-iMCMC sampler (App. B.3) almost surely terminating. $\square$

B.4.2. INVARIANT STATE DISTRIBUTION

After ensuring the Hybrid NP-iMCMC sampler (App. B.3) almost always returns a sample (Lem. B.4), we identify the distribution on the states and show that it is invariant against the movement between states of varying dimensions in Hybrid NP-iMCMC.

State Distribution   Recall a state is an equal dimension parameter-auxiliary pair. We define the *state distribution* π on the state space $\mathbb{S} := \bigcup_{n \in \mathbb{N}}(\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)})$ to be a distribution with density ζ (with respect to $\mu_\mathbb{S}$) given by

$$\zeta(\boldsymbol{x}, \boldsymbol{v}) := \begin{cases} \dfrac{1}{Z} \cdot w(\boldsymbol{t}) \cdot \mathsf{pdf} K^{(k)}(\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v})) \\ \qquad \text{if } (\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}^{\mathrm{valid}} \text{ and } \boldsymbol{t} \in \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \text{ has dimension } k \\ 0 \qquad \text{otherwise} \end{cases}$$

where $Z := \int_\mathbb{T} w \, \mathrm{d}\mu_\mathbb{T}$ (which exists by H1) and $\mathbb{S}^{\mathrm{valid}}$ is the subset of $\mathbb{S}$ consisting of all *valid states*.

Remark B.5. If there is some trace in $\mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w)$ for a parameter variable $\boldsymbol{x}$, by Prop. B.1 this trace $\boldsymbol{t}$ is unique and hence $\boldsymbol{x}$ represents a sample of the target distribution.

We say a n-dimensional state $(\boldsymbol{x}, \boldsymbol{v})$ is **valid** if

(i) $\mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing$, and

(ii) $(\boldsymbol{y}, \boldsymbol{u}) = \Phi^{(n)}(\boldsymbol{x}, \boldsymbol{v})$ implies $\mathsf{instance}(\boldsymbol{y}) \cap \mathsf{Supp}(w) \neq \varnothing$, and

(iii) $\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v}) \notin \mathbb{S}^{\mathrm{valid}}$ for all $k < n$.

Intuitively, valid states are the states which, when transformed by the involution $\Phi^{(n)}$, the instance of the parameter-component of which does not "fall beyond" the support of w.

We write $\mathbb{S}_n^{\mathrm{valid}} := \mathbb{S}^{\mathrm{valid}} \cap (\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)})$ to denote the the set of all n-dimensional valid states. The following proposition shows that involutions preserve the validity of states.

Proposition B.6. *Assuming H 3, the involution $\Phi^{(n)}$ sends $\mathbb{S}_n^{valid}$ to $\mathbb{S}_n^{valid}$ for all $n \in \mathbb{N}$. i.e. If $(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}_n^{valid}$, then $(\boldsymbol{y}, \boldsymbol{u}) = \Phi^{(n)}(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}_n^{valid}$.*

Proof. Let $(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}_n^{\mathrm{valid}}$ and $(\boldsymbol{y}, \boldsymbol{u}) = \Phi^{(n)}(\boldsymbol{x}, \boldsymbol{v})$. We prove $(\boldsymbol{y}, \boldsymbol{u}) \in \mathbb{S}_n^{\mathrm{valid}}$ by induction on $n \in \mathbb{N}$.

- Let $n = 1$. As $\Phi^{(1)}$ is involutive and $(\boldsymbol{x}, \boldsymbol{v})$ is a valid state,

  (i) $\mathsf{instance}(\boldsymbol{y}) \cap \mathsf{Supp}(w) \neq \varnothing$ and
  (ii) $(\boldsymbol{x}, \boldsymbol{v}) = \Phi^{(1)}(\boldsymbol{y}, \boldsymbol{u})$ and $\mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \neq \varnothing$.
  (iii) holds trivially

  and hence $(\boldsymbol{y}, \boldsymbol{u}) \in \mathbb{S}_1^{\mathrm{valid}}$.

- Assume for all $m < n$, $(\boldsymbol{z}, \boldsymbol{w}) \in \mathbb{S}_m^{\mathrm{valid}}$ implies $(\boldsymbol{z}', \boldsymbol{w}') = \Phi^{(m)}(\boldsymbol{z}, \boldsymbol{w}) \in \mathbb{S}_m^{\mathrm{valid}}$. Similar to the base case, (i) and (ii) hold as $\Phi^{(n)}$ is involutive and $(\boldsymbol{x}, \boldsymbol{v})$ is a valid state. Assume for contradiction that (iii) does not hold, i.e. there is $k < n$ where $\mathsf{take}_k(\boldsymbol{y}, \boldsymbol{u}) \in \mathbb{S}_k^{\mathrm{valid}}$. As $\mathsf{instance}(\mathsf{take}_k(\boldsymbol{y})) \cap \mathsf{Supp}(w) \neq \varnothing$, by H3 and the inductive hypothesis,

$$\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v}) = \mathsf{take}_k(\Phi^{(n)}(\boldsymbol{y}, \boldsymbol{u})) = \Phi^{(k)}(\mathsf{take}_k(\boldsymbol{y}, \boldsymbol{u})) \in \mathbb{S}_k^{\mathrm{valid}}$$

  which contradicts with the fact that $(\boldsymbol{x}, \boldsymbol{v})$ is a valid state.

$\square$

We can partition the set $\mathbb{S}^{\mathrm{valid}}$ of valid states. Let $(\boldsymbol{x}, \boldsymbol{v})$ be a m-dimensional valid state. The parameter variable $\boldsymbol{x}$ can be written as $\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \+\!\+ \boldsymbol{y}$ where $\boldsymbol{t}_1 \in \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w)$ is of dimension k_0, $\boldsymbol{t}_2$ is a trace where $\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) = \mathsf{take}_{k_0}(\boldsymbol{x})$, and $\boldsymbol{y} := \mathsf{drop}_{k_0}(\boldsymbol{x})$ where drop_k drops the first $\iota_{\mathbb{X}}(k)$ components of the input parameter. Similarly, the auxiliary variable $\boldsymbol{v}$ can be written as $\boldsymbol{v}_1 \+\!\+ \boldsymbol{v}_2$ where $\boldsymbol{v}_1 := \mathsf{take}_{k_0}(\boldsymbol{v})$ and $\boldsymbol{v}_2 := \mathsf{drop}_{k_0}(\boldsymbol{v})$ where drop_k drops the first $\iota_{\mathbb{Y}}(k)$ components of the input parameter. Hence, we have

$$\mathbb{S}^{\mathrm{valid}} = \bigcup_{k_0=1}^{\infty} \bigcup_{m=1}^{\infty} \{(\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \+\!\+ \boldsymbol{y}, \boldsymbol{v}_1 \+\!\+ \boldsymbol{v}_2) \in \mathbb{S}_m^{\mathrm{valid}} \mid$$

$$\boldsymbol{t}_1 \in \mathsf{Supp}^{\iota_{\mathbb{X}}(k_0)}(w), \boldsymbol{t}_2 \in \mathbb{T}, \boldsymbol{y} \in \mathbb{E}^{\iota_{\mathbb{X}}(m) - \iota_{\mathbb{X}}(k_0)}, \boldsymbol{v}_1 \in \mathbb{Y}^{(k_0)}, \boldsymbol{v}_2 \in \mathbb{E}^{\iota_{\mathbb{Y}}(m) - \iota_{\mathbb{Y}}(k_0)}\}$$

and the state distribution π on the measurable set $S \in \Sigma_{\mathbb{S}}$ can be written as

$$\pi(S) = \sum_{k_0=1}^{\infty} \sum_{m=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{T}} \int_{\mathsf{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)}$$

$$[(\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \+\!\+ \boldsymbol{y}, \boldsymbol{v}_1 \+\!\+ \boldsymbol{v}_2) \in S \cap \mathbb{S}_m^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathsf{pdf} K^{(k_0)}(\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2), \boldsymbol{v}_1)$$

$$\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_1) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_2) \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}) \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1) \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{v}_2)$$

We can now show that the state distribution π is indeed a probability measure and the set of valid states almost surely covers all states w.r.t. the state distribution.

Proposition B.7. *Assuming H1,*

1. $\pi(\mathbb{S}) = 1$*; and*

2. $\pi(\mathbb{S} \setminus \bigcup_{k=1}^{n} \mathbb{S}_k^{valid}) \to 0$ *as* $n \to \infty$.

Proof.   1. Consider the set $\mathbb{S}^{\text{valid}}$ with the partition discussed above.

$$
\pi(\mathbb{S})
$$
$$
= \pi(\mathbb{S}^{\text{valid}})
$$
$$
= \sum_{k_0=1}^{\infty} \sum_{m=1}^{\infty} \int_{\mathbb{E}^{\iota_\mathbb{Y}(m)-\iota_\mathbb{Y}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_\mathbb{X}(m)-\iota_\mathbb{X}(k_0)}} \int_\mathbb{T} \int_{\text{Supp}^{\iota_\mathbb{X}(k_0)}(w)}
$$
$$
[(\text{zip}(\boldsymbol{t}_1,\boldsymbol{t}_2) + \boldsymbol{y}, \boldsymbol{v}_1 + \boldsymbol{v}_2) \in \mathbb{S}_m^{\text{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \text{pdf} K^{(k_0)}(\text{zip}(\boldsymbol{t}_1,\boldsymbol{t}_2), \boldsymbol{v}_1)
$$
$$
\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_2)\,\mu_{\mathbb{E}^{\iota_\mathbb{X}(m)-\iota_\mathbb{X}(k_0)}}(\mathrm{d}\boldsymbol{y})\,\mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1)\,\mu_{\mathbb{E}^{\iota_\mathbb{Y}(m)-\iota_\mathbb{Y}(k_0)}}(\mathrm{d}\boldsymbol{v}_2)
$$
$$
= \sum_{k_0=1}^{\infty} \int_{\mathbb{Y}^{(k_0)}} \int_\mathbb{T} \int_{\text{Supp}^{\iota_\mathbb{X}(k_0)}(w)} \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \text{pdf} K^{(k_0)}(\text{zip}(\boldsymbol{t}_1,\boldsymbol{t}_2), \boldsymbol{v}_1) \cdot
$$
$$
\left(\bigcup_{\ell_1=1}^{\infty} \bigcup_{\ell_2=1}^{\infty} \int_{\mathbb{E}^{\ell_1}} \int_{\mathbb{E}^{\ell_2}} [(\text{zip}(\boldsymbol{t}_1,\boldsymbol{t}_2) + \boldsymbol{y}, \boldsymbol{v}_1 + \boldsymbol{v}_2) \in \mathbb{S}^{\text{valid}}]\,\mu_{\mathbb{E}^{\ell_2}}(\mathrm{d}\boldsymbol{y})\,\mu_{\mathbb{E}^{\ell_1}}(\mathrm{d}\boldsymbol{v}_2) \right)
$$
$$
\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_2)\,\mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1)
$$
$$
= \sum_{k_0=1}^{\infty} \int_\mathbb{T} \int_{\text{Supp}^{\iota_\mathbb{X}(k_0)}(w)} \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \left(\int_{\mathbb{Y}^{(k_0)}} \text{pdf} K^{(k_0)}(\text{zip}(\boldsymbol{t}_1,\boldsymbol{t}_2), \boldsymbol{v}_1)\,\mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1) \right)
$$
$$
\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_2)
$$
$$
= \sum_{k_0=1}^{\infty} \int_\mathbb{T} \int_{\text{Supp}^{\iota_\mathbb{X}(k_0)}(w)} \frac{1}{Z} w(\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_2)
$$
$$
= \int_{\text{Supp}(w)} \frac{1}{Z} w(\boldsymbol{t}_1)\,\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1) \ = 1
$$

2. Since π is a probability distribution and $\pi(\mathbb{S} \setminus \mathbb{S}^{\text{valid}}) = 0$, the series $\sum_{n=1}^{\infty} \pi(\mathbb{S}_n^{\text{valid}})$ which equals $\pi(\bigcup_{n=1}^{\infty} \mathbb{S}_n^{\text{valid}}) = \pi(\mathbb{S}^{\text{valid}}) = 1$ must converge. Hence $\pi(\mathbb{S} \setminus \bigcup_{k=1}^{n} \mathbb{S}_k^{\text{valid}}) = \pi(\mathbb{S}^{\text{valid}} \setminus \bigcup_{k=1}^{n} \mathbb{S}_k^{\text{valid}}) = \sum_{i=n+1}^{\infty} \pi(\mathbb{S}_i^{\text{valid}}) \to 0$ as $n \to \infty$. $\qquad\square$

Equivalent Program   Though the Hybrid NP-iMCMC algorithm (App. B.3) traverses state, it takes and returns *samples* on the trace space $\mathbb{T}$. Hence instead of asking whether the state distribution π is invariant against the Hybrid NP-iMCMC sampler directly, we consider a similar program which takes and returns *states* and prove the state distribution π is *invariant* w.r.t. this program.

Consider the program `eNPiMCMC` in Listing 3. It is similar to `NPiMCMC` (Listing 2) syntactically except it takes and returns states instead of traces, and has two additional lines (Lines 2 and 13). Hence, it is easy to deduce from Lem. B.4 that `eNPiMCMC` almost surely terminates.

In `eNPiMCMC`, we group the commands differently and into two groups:

Line 2-12 An initial valid state `(x0,v0)` is constructed so that `x0` and `x*` have the same instance in the support of `w`.

Line 14-22 A proposed state `(x,v)` is computed and accepted/rejected.

Listing 3. Code for the equivalent program

```
1    def eNPiMCMC(x*,v*):
2      t0 = intersect(instance(x*),support(w))[0]    # find a valid state
3      k0 = dim(t0)
4      x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
5      v0 = auxkernel[k0](x0)
6      (x,v) = involution[k0](x0,v0)
7      n = k0
8      while not intersect(instance(x),support(w)):
9        x0 = x0 + [(normal, coin)]*(indexX(n+1)-indexX(n))
10       v0 = v0 + [(normal, coin)]*(indexY(n+1)-indexY(n))
11       n = n + 1
12       (x,v) = involution[n](x0,v0)
13     (x,v) = involution[n](x0,v0)                   # accept/reject proposed state
14     t = intersect(instance(x),support(w))[0]
15     k = dim(t)
16     return (x,v) if uniform < min{1, w(t)/w(t0) *
17                           pdfauxkernel[k](proj((x,v),k))/
18                            pdfauxkernel[k0](proj((x0,v0),k0)) *
19                           pdfpar[n](x)/pdfpar[n](x0) *
20                           pdfaux[n](v)/pdfaux[n](v0) *
21                           absdetjacinv[n](x0,v0)}
22                else (x0,v0)
```

Invariant Distribution   Take a SPCF program `M` of type `List(X*Y) -> List(X*Y)` where the SPCF types `X` and `Y` implements the parameter space $\mathbb{X}$ and auxiliary space $\mathbb{Y}$ respectively. We define the ***transition kernel*** of `M` to be the kernel $T_{\texttt{M}} : \mathbb{S} \rightsquigarrow \mathbb{S}$ where

$$T_{\texttt{M}}(\boldsymbol{s}, S) := \int_{\text{value}_{\texttt{M(s)}}^{-1}(S')} \text{weight}_{\texttt{M(s)}} \, \mathrm{d}\mu_{\mathbb{T}} = \langle\!\langle\, \texttt{M(s)}\,\rangle\!\rangle(S')$$

where `s` implements the state $\boldsymbol{s}$ and S' is the set consisting of SPCF terms that implements states in S. Intuitively, $T_{\texttt{M}}(\boldsymbol{s}, S)$ gives the probability that the term `M` returns a state in S given the current state $\boldsymbol{s}$.

Proposition B.8. *Let* `M` *be a SPCF term of type* `List(X*Y) -> List(X*Y)`*. If* `M(s)` *does not contain any scoring subterm and almost surely terminates for all SPCF terms* `s`*, then its transition kernel* $T_{\texttt{M}}$ *is probabilistic.*

Proof. Since the term `M(s)` does not contain score($\cdot$) and terminates almost surely, by Prop. A.11 its value measure must be probabilistic. Hence $T_{\texttt{M}}(\,\texttt{s}\,, \mathbb{S}) = \langle\!\langle\,\texttt{M(s)}\,\rangle\!\rangle(\mathbb{S}') = \langle\!\langle\,\texttt{M(s)}\,\rangle\!\rangle(\Lambda_v^0) = 1.$ □

We say a distribution μ on states $\mathbb{S}$ is ***invariant*** w.r.t. a almost surely terminating SPCF program `M` of type `List(X*Y) -> List(X*Y)` if μ is not altered after applying `M`, formally $\int_{\mathbb{S}} T_{\texttt{M}}(\boldsymbol{s}, S)\, \mu(\mathrm{d}\boldsymbol{s}) = \mu(S)$.

We now prove that `eNPiMCMC` preserves the state distribution π stated in App. B.4.2 by considering the transition kernels given by the two steps in `eNPiMCMC` given in App. B.4.2: find a valid state (Lines 2-12) and accept/reject the computed proposed state (Lines 13-22).

Finding a Valid State   Assuming the initial state `(x*,v*)` is valid, `eNPiMCMC` (Lines 2-12) aims to construct a *valid* state `(x0,v0)` where `x*` and `x0` share the same instance `t0` that is in the support of the density `w`.

To do this, it first finds the instance `t0` of `x*` which is in the support of `w` (Line 2). Say the dimension of `t0` is `k0` (Line 3). It then forms a `k0`-dimensional state `(x0,v0)` by sampling partners `t` for each value in the trace `t0`

to form a `k0`-dimensional parameter variable `x0` (Line 4); and drawing a `k0`-dimensional auxiliary variable from `auxkernel[k0](x0)` (Line 5). Say `v` is the auxiliary value drawn. Then, the `k0`-dimensional state can be written as `(zip(t0,t),v)`.

Note that the `k0`-dimensional state `(zip(t0,t),v)` might *not* be valid. In which case, it repeatedly appends `zip(t0,t)` and `v` with entropies (`normal`, `coin`) until the resulting state is valid (Lines 6-12). Say `y` and `u` are the entropy vectors drawn for the parameter and auxiliary variables respectively. Then the resulting state can be written as `(zip(t0,t)+y, v+u)`.

The transition kernel of Lines 2-12 can be expressed

$$T_1((\boldsymbol{x}^*, \boldsymbol{v}^*), S)$$

$$:= \sum_{n=1}^{\infty} \int_{\mathbb{E}^{\iota_\mathbb{Y}(n)-\iota_\mathbb{Y}(k_0)}} \int_{\mathbb{E}^{\iota_\mathbb{X}(n)-\iota_\mathbb{X}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{T}} [(\mathsf{zip}(\boldsymbol{t}_0, \boldsymbol{t}) \mathbin{+\!\!+} \boldsymbol{y}, \boldsymbol{v} \mathbin{+\!\!+} \boldsymbol{u}) \in S \cap \mathbb{S}_n^{\mathrm{valid}}] \cdot \mathsf{pdf} K^{(k_0)}(\mathsf{zip}(\boldsymbol{t}_0, \boldsymbol{t}), \boldsymbol{v})$$

$$\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}) \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}) \, \mu_{\mathbb{E}^{\iota_\mathbb{X}(n)-\iota_\mathbb{X}(k_0)}}(\mathrm{d}\boldsymbol{y}) \, \mu_{\mathbb{E}^{\iota_\mathbb{Y}(n)-\iota_\mathbb{Y}(k_0)}}(\mathrm{d}\boldsymbol{u})$$

if $(\boldsymbol{x}^*, \boldsymbol{v}^*) \in \mathbb{S}^{\mathrm{valid}}$ and $\boldsymbol{t}_0 \in \mathsf{instance}(\boldsymbol{x}^*) \cap \mathsf{Supp}(w)$ has some dimension $k_0 \in \mathbb{N}$; and 0 otherwise.

Remark B.9. Recall $\mathsf{zip}(\ell_1, \ell_2) := [(\ell_1{}^1, \ell_2{}^1), (\ell_1{}^2, \ell_2{}^2), \ldots, (\ell_1{}^n, \ell_2{}^n)] \in (L_1 \times L_2)^n$ for any vectors $\ell_1 \in L_1^{n_1}$ and $\ell_2 \in L_2^{n_2}$ with $n := \min\{n_1, n_2\}$. Here we extend the definition to lists $\ell_1, \ell_2 \in (L_1 \cup L_2)^n$ such that either $(\ell_1{}^i, \ell_2{}^i)$ or $(\ell_2{}^i, \ell_1{}^i)$ is in $L_1 \times L_2$ for all $i = 1, \ldots, n$. Then, we write $\mathsf{zip}(\ell_1, \ell_2)$ for the list of pairs in $L_1 \times L_2$.

Proposition B.10. *Assuming H2, $T_1((\boldsymbol{x}_0, \boldsymbol{v}_0), \mathbb{S}^{valid}) = 1$ for all $(\boldsymbol{x}_0, \boldsymbol{v}_0) \in \mathbb{S}^{valid}$.*

Proof. Since Lines 2-12 in `eNPiMCMC` can be described by a closed SPCF term that does not contain $\mathsf{score}(\cdot)$ and terminates almost surely. By Prop. B.8, its transition kernel is probabilistic. Moreover, as this term always return a valid state, we have $T_1((\boldsymbol{x}_0, \boldsymbol{v}_0), \mathbb{S}^{\mathrm{valid}}) = T_1((\boldsymbol{x}_0, \boldsymbol{v}_0), \mathbb{S}) = 1$. □

Proposition B.11. *Assuming H1 and 2, the state distribution π is invariant against Lines 2-12 in* `eNPiMCMC`.

Proof. We aim to show: $\int_\mathbb{S} T_1((\boldsymbol{x}^*, \boldsymbol{v}^*), S) \, \pi(\mathrm{d}(\boldsymbol{x}^*, \boldsymbol{v}^*)) = \pi(S)$ for any measurable set $S \in \Sigma_\mathbb{S}$. (Changes are highlighted for readability.)

$$\int_\mathbb{S} T_1((\boldsymbol{x}^*, \boldsymbol{v}^*), S) \, \pi(\mathrm{d}(\boldsymbol{x}^*, \boldsymbol{v}^*))$$

$$= \quad \{ \ T_1((\boldsymbol{x}^*, \boldsymbol{v}^*), S) = 0 \text{ for all } (\boldsymbol{x}^*, \boldsymbol{v}^*) \notin \mathbb{S}^{\mathrm{valid}} \ \}$$

$$\int_{\mathbb{S}^{\mathrm{valid}}} T_1(\, (\boldsymbol{x}^*, \boldsymbol{v}^*) \,, S) \, \pi(\mathrm{d}(\boldsymbol{x}^*, \boldsymbol{v}^*))$$

$$= \quad \{ \ \text{Writing } (\boldsymbol{x}^*, \boldsymbol{v}^*) \text{ as } (\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \mathbin{+\!\!+} \boldsymbol{y}, \boldsymbol{v}_1 \mathbin{+\!\!+} \boldsymbol{v}_2) \in \mathbb{S}_m^{\mathrm{valid}} \text{ where}$$
$$\boldsymbol{t}_1 \in \mathsf{Supp}^{\iota_\mathbb{X}(k_0)}(w), \boldsymbol{t}_2 \in \mathbb{T}, \boldsymbol{y} \in \mathbb{E}^{\iota_\mathbb{X}(m)-\iota_\mathbb{X}(k_0)}, \boldsymbol{v}_1 \in \mathbb{Y}^{(k_0)}, \boldsymbol{v}_2 \in \mathbb{E}^{\iota_\mathbb{Y}(m)-\iota_\mathbb{Y}(k_0)}, m, k_0 \in \mathbb{N} \ \}$$

$$\sum_{k_0=1}^{\infty} \sum_{m=1}^{\infty} \int_{\mathbb{E}^{\iota_\mathbb{Y}(m)-\iota_\mathbb{Y}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_\mathbb{X}(m)-\iota_\mathbb{X}(k_0)}} \int_{\mathbb{T}} \int_{\mathsf{Supp}^{\iota_\mathbb{X}(k_0)}(w)} T_1((\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \mathbin{+\!\!+} \boldsymbol{y}, \boldsymbol{v}_1 \mathbin{+\!\!+} \boldsymbol{v}_2), S)$$

$$[(\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \mathbin{+\!\!+} \boldsymbol{y}, \boldsymbol{v}_1 \mathbin{+\!\!+} \boldsymbol{v}_2) \in \mathbb{S}_m^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathsf{pdf} K^{(k_0)}(\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2), \boldsymbol{v}_1)$$

$$\mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_1) \, \mu_\mathbb{T}(\mathrm{d}\boldsymbol{t}_2) \, \mu_{\mathbb{E}^{\iota_\mathbb{X}(m)-\iota_\mathbb{X}(k_0)}}(\mathrm{d}\boldsymbol{y}) \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1) \, \mu_{\mathbb{E}^{\iota_\mathbb{Y}(m)-\iota_\mathbb{Y}(k_0)}}(\mathrm{d}\boldsymbol{v}_2)$$

$$= \quad \{ \ \text{Definition of } T_1 \text{ on } (\mathsf{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) \mathbin{+\!\!+} \boldsymbol{y}, \boldsymbol{v}_1 \mathbin{+\!\!+} \boldsymbol{v}_2) \in \mathbb{S}^{\mathrm{valid}} \text{ where } \boldsymbol{t}_1 \in \mathsf{Supp}^{\iota_\mathbb{X}(k_0)}(w) \ \}$$

$$\sum_{k_0=1}^{\infty} \sum_{m=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{T}} \int_{\mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)}$$

$$\left(\sum_{n=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{T}} \right.$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}') +\!\!+ \boldsymbol{y}', \boldsymbol{v}' +\!\!+ \boldsymbol{u}') \in S \cap \mathbb{S}_n^{\mathrm{valid}}] \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}'), \boldsymbol{v}')$$

$$\left. \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}') \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{u}') \right)$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) +\!\!+ \boldsymbol{y}, \boldsymbol{v}_1 +\!\!+ \boldsymbol{v}_2) \in \mathbb{S}_m^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2), \boldsymbol{v}_1)$$

$$\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_1) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_2) \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}) \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1) \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{v}_2)$$

$=$ { Tonelli's theorem as all measurable functions are non-negative }

$$\sum_{k_0=1}^{\infty} \sum_{n=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{T}} \int_{\mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)}$$

$$\left(\sum_{m=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{T}} \right.$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) +\!\!+ \boldsymbol{y}, \boldsymbol{v}_1 +\!\!+ \boldsymbol{v}_2) \in \mathbb{S}_m^{\mathrm{valid}}] \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2), \boldsymbol{v}_1)$$

$$\left. \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_2) \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}_1) \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}) \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{v}_2) \right)$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}') +\!\!+ \boldsymbol{y}', \boldsymbol{v}' +\!\!+ \boldsymbol{u}') \in S \cap \mathbb{S}_n^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}'), \boldsymbol{v}')$$

$$\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_1) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}') \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{u}')$$

$=$ { Definition of T_1 on $(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) +\!\!+ \boldsymbol{y}, \boldsymbol{v}_1 +\!\!+ \boldsymbol{v}_2) \in \mathbb{S}^{\mathrm{valid}}$ where $\boldsymbol{t}_1 \in \mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)$ }

$$\sum_{k_0=1}^{\infty} \sum_{n=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{T}} \int_{\mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)} T_1((\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) +\!\!+ \boldsymbol{y}, \boldsymbol{v}_1 +\!\!+ \boldsymbol{v}_2), \mathbb{S}^{\mathrm{valid}})$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}') +\!\!+ \boldsymbol{y}', \boldsymbol{v}' +\!\!+ \boldsymbol{u}') \in S \cap \mathbb{S}_n^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}'), \boldsymbol{v}')$$

$$\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_1) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}') \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{u}')$$

$=$ { By Prop. B.10, $T_1((\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}_2) +\!\!+ \boldsymbol{y}, \boldsymbol{v}_1 +\!\!+ \boldsymbol{v}_2), \mathbb{S}^{\mathrm{valid}}) = 1$ }

$$\sum_{k_0=1}^{\infty} \sum_{n=1}^{\infty} \int_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}} \int_{\mathbb{Y}^{(k_0)}} \int_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}} \int_{\mathbb{T}} \int_{\mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w)}$$

$$[(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}') +\!\!+ \boldsymbol{y}', \boldsymbol{v}' +\!\!+ \boldsymbol{u}') \in S \cap \mathbb{S}_n^{\mathrm{valid}}] \cdot \frac{1}{Z} w(\boldsymbol{t}_1) \cdot \mathrm{pdf}\, K^{(k_0)}(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}'), \boldsymbol{v}')$$

$$\mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}_1) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{X}}(n)-\iota_{\mathbb{X}}(k_0)}}(\mathrm{d}\boldsymbol{y}') \, \mu_{\mathbb{Y}^{(k_0)}}(\mathrm{d}\boldsymbol{v}') \, \mu_{\mathbb{E}^{\iota_{\mathbb{Y}}(n)-\iota_{\mathbb{Y}}(k_0)}}(\mathrm{d}\boldsymbol{u}')$$

$=$ { Writing $(\boldsymbol{x}^*, \boldsymbol{v}^*) \in S \cap \mathbb{S}_n^{\mathrm{valid}}$ as $(\mathrm{zip}(\boldsymbol{t}_1, \boldsymbol{t}') +\!\!+ \boldsymbol{y}', \boldsymbol{v}' +\!\!+ \boldsymbol{u}')$ where
$\boldsymbol{t}_1 \in \mathrm{Supp}^{\iota_{\mathbb{X}}(k_0)}(w), \boldsymbol{t}' \in \mathbb{T}, \boldsymbol{y}' \in \mathbb{E}^{\iota_{\mathbb{X}}(m)-\iota_{\mathbb{X}}(k_0)}, \boldsymbol{v}' \in \mathbb{Y}^{(k_0)}, \boldsymbol{u}' \in \mathbb{E}^{\iota_{\mathbb{Y}}(m)-\iota_{\mathbb{Y}}(k_0)}, n, k_0 \in \mathbb{N}$ }

$\pi(S)$

$\square$

Accept/Reject Proposed State After constructing a valid state `(x0,v0)`, say of dimension `n`, `eNPiMCMC` traverses the state space via `involution[n]` to obtain a proposal state `(x,v)` (Line 13). By Prop. B.6, `(x,v)` must also be a `n`-dimensional valid state. Say it has an instance `t` of dimension `k` in the support of `w`, then (Line 14-22) `(x,v)` is

accepted with probability

$$\alpha(\boldsymbol{x}_0, \boldsymbol{v}_0) := \min\left\{1, \frac{w(\boldsymbol{t}) \cdot \mathsf{pdf}\, K^{(k)}(\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v})) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v})}{w(\boldsymbol{t}_0) \cdot \mathsf{pdf}\, K^{(k_0)}(\mathsf{take}_{k_0}(\boldsymbol{x}_0, \boldsymbol{v}_0)) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot \left|\det\left(\nabla\Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0)\right)\right|\right\}$$

$$= \min\left\{1, \frac{\zeta(\boldsymbol{x}, \boldsymbol{v}) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v})}{\zeta(\boldsymbol{x}_0, \boldsymbol{v}_0) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot \left|\det\left(\nabla\Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0)\right)\right|\right\}.$$

The transition kernel for Line 13-22 can be expressed as

$$T_2(\boldsymbol{s}, S) := \alpha(\boldsymbol{s}) \cdot [\Phi^{(n)}(\boldsymbol{s}) \in S] + (1 - \alpha(\boldsymbol{s})) \cdot [\boldsymbol{s} \in S]$$

if $\boldsymbol{s} \in \mathbb{S}_n^{\mathrm{valid}}$ for some $n \in \mathbb{N}$; and 0 otherwise.

To show that the state distribution π is invariant against T_2, we consider a partition of the set of valid states. Let $S_{ij}^{(n)}$ be the set of n-dimensional valid states where $\boldsymbol{i}$ is the list of boolean values in all $\boldsymbol{s} \in S_{ij}^{(n)}$ and $\Phi^{(n)}$ maps $\boldsymbol{s}$ to a (valid) state with boolean values given by the list $\boldsymbol{j}$. Note that both lists $\boldsymbol{i}, \boldsymbol{j}$ of booleans must be of length $\tilde{n} := \iota_{\mathbb{X}}(n) + \iota_{\mathbb{Y}}(n)$. Formally,

$$S_{ij}^{(n)} := \{\boldsymbol{s} \in \mathbb{S}_n^{\mathrm{valid}} \mid \boldsymbol{s} = \mathsf{zip}(\boldsymbol{r}, \boldsymbol{i}) \text{ and } \Phi^{(n)}(\boldsymbol{s}) = \boldsymbol{s}' = \mathsf{zip}(\boldsymbol{q}, \boldsymbol{j}) \text{ for some } \boldsymbol{r}, \boldsymbol{q} \in \mathbb{R}^{\tilde{n}}\}.$$

Then, the set $\mathbb{S}^{\mathrm{valid}}$ of valid states can written as $\bigcup\{S_{ij}^{(n)} \mid \boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}} \text{ and } n \in \mathbb{N}\}$.

Proposition B.12. *Assuming H1 to 3, for $n \in \mathbb{N}$, $\boldsymbol{s} \in \mathbb{S}_n^{\mathrm{valid}}$ and $\boldsymbol{s}' = \Phi^{(n)}(\boldsymbol{s})$, we have*

$$\alpha(\boldsymbol{s}') \cdot \zeta'(\boldsymbol{s}') \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)| = \alpha(\boldsymbol{s}) \cdot \zeta'(\boldsymbol{s})$$

where $\zeta'(\boldsymbol{z}, \boldsymbol{w}) := \zeta(\boldsymbol{z}, \boldsymbol{w}) \cdot \varphi_{\mathbb{X}^{(m)}}(\boldsymbol{z}) \cdot \varphi_{\mathbb{Y}^{(m)}}(\boldsymbol{w})$ for any $(\boldsymbol{z}, \boldsymbol{w}) \in \mathbb{S}_m$.

Proof. Let $\boldsymbol{s} \in S_{ij}^{(n)}$ where there are $\boldsymbol{r}, \boldsymbol{q} \in \mathbb{R}^{\tilde{n}}$, $\boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}}$ such that $\boldsymbol{s} = \mathsf{zip}(\boldsymbol{r}, \boldsymbol{i})$ and $\boldsymbol{s}' := \Phi^{(n)}(\boldsymbol{s}) = \mathsf{zip}(\boldsymbol{q}, \boldsymbol{j})$. Hence, taking the Jacobian determinant on both sides of the equation $\Phi_{ji}^{(n)} \circ \Phi_{ij}^{(n)} = \mathrm{id}$ gives us

$$|\det\left(\nabla\Phi^{(n)}(\boldsymbol{s}')\right)| = |\det\left(\nabla\Phi_{ji}^{(n)}(\boldsymbol{q})\right)| = \frac{1}{|\det\left(\nabla\Phi_{ij}^{(n)}(\boldsymbol{r})\right)|} = \frac{1}{|\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)|}. \tag{2}$$

Moreover we can write the acceptance ratio in terms of ζ' as

$$\alpha(\boldsymbol{s}'') = \min\{1, \frac{\zeta'(\Phi^{(n)}(\boldsymbol{s}''))}{\zeta'(\boldsymbol{s}'')} \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s}'')\right)|\} \text{ for any } \boldsymbol{s}'' \in \mathbb{S}_m.$$

Hence given $\boldsymbol{s}' = \Phi^{(n)}(\boldsymbol{s})$, we have

$$\alpha(\boldsymbol{s}') \cdot \zeta'(\boldsymbol{s}') \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)|$$

$$= \begin{cases} \dfrac{\zeta'(\boldsymbol{s})}{\zeta'(\boldsymbol{s}')} \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s}')\right)| \cdot \zeta'(\boldsymbol{s}') \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)| & \text{if } \dfrac{\zeta'(\boldsymbol{s})}{\zeta'(\boldsymbol{s}')} \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s}')\right)| < 1 \\ \zeta'(\boldsymbol{s}') \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)| & \text{otherwise} \end{cases} \quad (\boldsymbol{s} = \Phi^{(n)}(\boldsymbol{s}'))$$

$$= \begin{cases} \zeta'(\boldsymbol{s}) & \text{if } \dfrac{\zeta'(\boldsymbol{s}')}{\zeta'(\boldsymbol{s})} \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)| > 1 \\ \dfrac{\zeta'(\boldsymbol{s}')}{\zeta'(\boldsymbol{s})} \cdot |\det\left(\nabla\Phi^{(n)}(\boldsymbol{s})\right)| \cdot \zeta'(\boldsymbol{s}) & \text{otherwise} \end{cases} \quad \text{(By Equation (2))}$$

$$= \alpha(\boldsymbol{s}) \cdot \zeta'(\boldsymbol{s})$$

$\square$

Proposition B.13. *Assuming H1 to 3, the state distribution π is invariant against Line 13-22 in* `eNPiMCMC` *.*

Proof. We aim to show: $\int_{\mathbb{S}} T_2(\boldsymbol{s}, S)\, \pi(\mathrm{d}\boldsymbol{s}) = \pi(S)$ for all $S \in \Sigma_{\mathbb{S}}$.

Let $\boldsymbol{s}$ be a n-dimensional valid state and $S \in \Sigma_{\mathbb{S}}$. Then we can write $T_2(\boldsymbol{s}, S)$ as $[\boldsymbol{s} \in S] + [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s}) - [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s})$. Hence, it is enough to show that the integral of the second and third terms over all valid states are the same, i.e.

$$\int_{\mathbb{S}^{\mathrm{valid}}} [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s}) = \int_{\mathbb{S}^{\mathrm{valid}}} [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s})$$

First we consider the valid states in $S_{\boldsymbol{ij}}^{(n)}$ where $n \in \mathbb{N}$, $\boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}}$ and $\tilde{n} := \iota_{\mathbb{X}}(n) + \iota_{\mathbb{Y}}(n)$. These are n-dimensional valid states with boolean values given by $\boldsymbol{i}$ and are mapped by $\Phi^{(n)}$ to valid states with boolean values given by $\boldsymbol{j}$. Then we have $\mathrm{zip}(\cdot, \boldsymbol{j})^{-1}(S_{\boldsymbol{ji}}^{(n)}) = \Phi_{\boldsymbol{ij}}^{(n)}(\mathrm{zip}(\cdot, \boldsymbol{i})^{-1}(S_{\boldsymbol{ij}}^{(n)}))$ where $\mathrm{zip}(\cdot, \boldsymbol{j}) : \mathbb{R}^{\tilde{n}} \to \mathbb{E}^{\tilde{n}}$ is a measurable function. Writing $\zeta'(\boldsymbol{z}, \boldsymbol{w})$ for $\zeta(\boldsymbol{z}, \boldsymbol{w}) \cdot \varphi_{\mathbb{X}(m)}(\boldsymbol{z}) \cdot \varphi_{\mathbb{Y}(m)}(\boldsymbol{w})$ for any $(\boldsymbol{z}, \boldsymbol{w}) \in \mathbb{S}_m$, we have

$$\int_{S_{\boldsymbol{ji}}^{(n)}} [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s})$$

$$= \int_{S_{\boldsymbol{ji}}^{(n)}} [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s}) \cdot \zeta'(\boldsymbol{s})\, \mu_{\mathbb{E}^{\tilde{n}}}(\mathrm{d}\boldsymbol{s}) \qquad\qquad\qquad (\text{Definition of } \pi)$$

$$= \int_{\mathrm{zip}(\cdot, \boldsymbol{j})^{-1}(S_{\boldsymbol{ji}}^{(n)})} [\mathrm{zip}(\boldsymbol{r}, \boldsymbol{j}) \in S] \cdot \alpha(\mathrm{zip}(\boldsymbol{r}, \boldsymbol{j})) \cdot \zeta'(\mathrm{zip}(\boldsymbol{r}, \boldsymbol{j}))\, \mu_{\mathbb{R}^{\tilde{n}}}(\mathrm{d}\boldsymbol{r}) \qquad (\mathrm{zip}(\cdot, \boldsymbol{j})_* \mu_{\mathbb{R}^{\tilde{n}}} = \mu_{\mathbb{E}^{\tilde{n}}} \text{ on } S_{\boldsymbol{ji}}^{(n)})$$

$$= \int_{\mathrm{zip}(\cdot, \boldsymbol{i})^{-1}(S_{\boldsymbol{ij}}^{(n)})} [\mathrm{zip}(\Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q}), \boldsymbol{j}) \in S] \cdot \alpha(\mathrm{zip}(\Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q}), \boldsymbol{j})) \cdot \zeta'(\mathrm{zip}(\Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q}), \boldsymbol{j})) \cdot |\det \nabla \Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q})|\, \mu_{\mathbb{R}^{\tilde{n}}}(\mathrm{d}\boldsymbol{q})$$

$$(\text{Change of variable where } \boldsymbol{r} = \Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q}))$$

$$= \int_{\mathrm{zip}(\cdot, \boldsymbol{i})^{-1}(S_{\boldsymbol{ij}}^{(n)})} [\Phi^{(n)}(\mathrm{zip}(\boldsymbol{q}, \boldsymbol{i})) \in S] \cdot \alpha(\mathrm{zip}(\boldsymbol{q}, \boldsymbol{i})) \cdot \zeta'(\mathrm{zip}(\boldsymbol{q}, \boldsymbol{i}))\, \mu_{\mathbb{R}^{\tilde{n}}}(\mathrm{d}\boldsymbol{q})$$

$$(\text{Prop. B.12 as } \Phi^{(n)}(\mathrm{zip}(\boldsymbol{q}, \boldsymbol{i})) = \mathrm{zip}(\Phi_{\boldsymbol{ij}}^{(n)}(\boldsymbol{q}), \boldsymbol{j}) \text{ for } (\mathrm{zip}(\boldsymbol{q}, \boldsymbol{i})) \in S_{\boldsymbol{ij}}^{(n)})$$

$$= \int_{S_{\boldsymbol{ij}}^{(n)}} [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s}) \cdot \zeta'(\boldsymbol{s})\, \mu_{\mathbb{E}^{\tilde{n}}}(\mathrm{d}\boldsymbol{s}) \qquad\qquad (\mathrm{zip}(\cdot, \boldsymbol{i})_* \mu_{\mathbb{R}^{\tilde{n}}} = \mu_{\mathbb{E}^{\tilde{n}}} \text{ on } S_{\boldsymbol{ij}}^{(n)})$$

$$= \int_{S_{\boldsymbol{ij}}^{(n)}} [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s})$$

Recall the set $\mathbb{S}^{\mathrm{valid}}$ of all valid states can be written as $\bigcup \{S_{\boldsymbol{ij}}^{(n)} \mid \boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}} \text{ and } n \in \mathbb{N}\}$. Hence, we conclude our proof with

$$\int_{\mathbb{S}^{\mathrm{valid}}} [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s}) = \sum_{n=1}^{\infty} \sum_{\boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}}} \int_{S_{\boldsymbol{ij}}^{(n)}} [\Phi^{(n)}(\boldsymbol{s}) \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s})$$

$$= \sum_{n=1}^{\infty} \sum_{\boldsymbol{i}, \boldsymbol{j} \in 2^{\tilde{n}}} \int_{S_{\boldsymbol{ji}}^{(n)}} [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s}) = \int_{\mathbb{S}^{\mathrm{valid}}} [\boldsymbol{s} \in S] \cdot \alpha(\boldsymbol{s})\, \pi(\mathrm{d}\boldsymbol{s}).$$

$\square$

Since the transition kernel of `eNPiMCMC` is the composition of T_1 and T_2 and both T_1 and T_2 are invariant against π (Propositions B.11 and B.13), we deduce that `eNPiMCMC` preserves the state distribution π.

Lemma B.14 (State Invariant). *π is the invariant distribution of the Markov chain generated by iterating* `eNPiMCMC`.

B.4.3. MARGINALISED MARKOV CHAINS

As discussed above, the Markov chain $\{\boldsymbol{s}_i\}_{i \in \mathbb{N}}$ generated by iterating `eNPiMCMC` (which has invariant distribution π (Lem. B.14)) has elements on the state space $\mathbb{S}$ and *not* the trace space $\mathbb{T}$. The chain we are in fact interested in is the *marginalised* chain $\{\mathrm{m}(\boldsymbol{s}_i)\}_{i \in \mathbb{N}}$ where the measurable function $\mathrm{m} : \mathbb{S}^{\mathrm{valid}} \to \mathbb{T}$ takes a valid state $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{v})$ and returns the instance of the parameter variable $\boldsymbol{x}$ that is in the support of the target density function w.

In this section we show that this marginalised chain simulates the target distribution ν. Let $T_{\texttt{NPiMCMC}} : \mathbb{T} \rightsquigarrow \mathbb{T}$ be a kernel such that

$$T_{\texttt{NPiMCMC}}(\boldsymbol{t}, A) := \begin{cases} T_{\texttt{eNPiMCMC}}(\boldsymbol{s}, \mathfrak{m}^{-1}(A)) & \text{if } \boldsymbol{t} \in \mathrm{Supp}(w) \text{ and } \boldsymbol{s} \in \mathfrak{m}^{-1}(\{\boldsymbol{t}\}) \\ 0 & \text{otherwise.} \end{cases}$$

Comparing the commands of $\texttt{NPiMCMC}$ and $\texttt{eNPiMCMC}$ in Listings 2 and 3, we claim that $T_{\texttt{NPiMCMC}}$ is *the* transition kernel of $\texttt{NPiMCMC}$.

Proposition B.15. *We consider some basic properties of $T_{\texttt{NPiMCMC}}$.*

1. *$T_{\texttt{NPiMCMC}}$ is well-defined.*

2. *$T_{\texttt{eNPiMCMC}}(\boldsymbol{s}, \mathfrak{m}^{-1}(A)) = T_{\texttt{NPiMCMC}}(\mathfrak{m}(\boldsymbol{s}), A)$ for all $\boldsymbol{s} \in \mathbb{S}^{valid}$ and $A \in \Sigma_{\mathbb{T}}$.*

Proof. 1. Let $\boldsymbol{t} \in \mathrm{Supp}(w)$ and $A \in \Sigma_{\mathbb{T}}$. Say $\boldsymbol{s}, \boldsymbol{s}' \in \mathfrak{m}^{-1}(\{\boldsymbol{t}\})$. Since only the instance of the input state matters in $\texttt{eNPiMCMC}$ (Listing 3), the value of $T_{\texttt{NPiMCMC}}(\boldsymbol{t}, A)$ given by $\boldsymbol{s}$ and $\boldsymbol{s}'$ are the same, i.e. $T_{\texttt{eNPiMCMC}}(\boldsymbol{s}, \mathfrak{m}^{-1}(A)) = T_{\texttt{eNPiMCMC}}(\boldsymbol{s}', \mathfrak{m}^{-1}(A))$.

2. Let $\boldsymbol{s} \in \mathbb{S}^{valid}$ and $A \in \Sigma_{\mathbb{T}}$. Then, $T_{\texttt{NPiMCMC}}(\mathfrak{m}(\boldsymbol{s}), A) = T_{\texttt{eNPiMCMC}}(\boldsymbol{s}', \mathfrak{m}^{-1}(A))$ for some $\boldsymbol{s}' \in \mathfrak{m}^{-1}(\{\mathfrak{m}(\boldsymbol{s})\})$. Since $\boldsymbol{s} \in \mathfrak{m}^{-1}(\{\mathfrak{m}(\boldsymbol{s})\})$, we have $T_{\texttt{eNPiMCMC}}(\boldsymbol{s}', \mathfrak{m}^{-1}(A)) = T_{\texttt{eNPiMCMC}}(\boldsymbol{s}, \mathfrak{m}^{-1}(A))$.

$\square$

To show $T_{\texttt{NPiMCMC}}$ preserves the target distribution, we consider a distribution π_n on each of the n-dimensional state space $\mathbb{S}^{(n)} := \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$ with density ζ_n (w.r.t. $\mu_{\mathbb{S}^{(n)}}$) given by

$$\zeta_n(\boldsymbol{x}, \boldsymbol{v}) := \begin{cases} \dfrac{1}{Z_n} \cdot w(\boldsymbol{t}) \cdot \mathrm{pdf} K^{(k)}(\mathrm{take}_k(\boldsymbol{x}, \boldsymbol{v})) & \text{if } \boldsymbol{t} \in \mathrm{instance}(\boldsymbol{x}) \cap \mathrm{Supp}(w) \text{ has dimension } k \leq n \\ 0 & \text{otherwise} \end{cases}$$

where $Z_n := \int_{\mathbb{T}} [|\boldsymbol{t}| \leq \iota_{\mathbb{X}}(n)] \cdot w(\boldsymbol{t})\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t})$. Notice that $Z_n \cdot \zeta_n$ and $Z \cdot \zeta$ are the same, except on non-valid states. The following proposition shows how the state distribution π can be represented using π_n.

Proposition B.16. *Let $n \in \mathbb{N}$.*

1. *π_n is a probability measure.*

2. *For $k \leq n$, $Z_k \cdot \pi_k = Z_n \cdot \mathrm{take}_{k*}\pi_n$ on $\mathbb{S}_k^{valid}$.*

3. *Let $g^{(n)} : \mathbb{S}^{(n)} \rightharpoonup \bigcup_{k=1}^n \mathbb{S}_k^{valid}$ be the partial measurable function that returns the projection of the input state that is valid, if it exists. Formally, $g^{(n)}(\boldsymbol{s}) = \mathrm{take}_k(\boldsymbol{s})$ if $\mathrm{take}_k(\boldsymbol{s}) \in \mathbb{S}_k^{valid}$. Then $Z \cdot \pi = Z_n \cdot g^{(n)}_* \pi_n$ on $\bigcup_{k=1}^n \mathbb{S}_k^{valid}$.*

Proof. 1. Consider $\pi_n(\mathbb{S}^{(n)})$,

$$\pi_n(\mathbb{S}^{(n)}) = \sum_{k=1}^n \int_{\mathbb{S}^{(n)}} [\boldsymbol{t} \in \mathrm{instance}(\boldsymbol{x})] \cdot [|\boldsymbol{t}| = \iota_{\mathbb{X}}(k)] \cdot \frac{1}{Z_n} \cdot w(\boldsymbol{t}) \cdot \mathrm{pdf} K^{(k)}(\mathrm{take}_k(\boldsymbol{x}, \boldsymbol{v}))\, \mu_{\mathbb{S}^{(n)}}(\mathrm{d}(\boldsymbol{x}, \boldsymbol{v}))$$

$$= \sum_{k=1}^n \int_{\mathrm{Supp}\, \iota_{\mathbb{X}}^{(k)}(w)} \int_{\mathbb{T}} \int_{\mathbb{Y}^{(k)}} \frac{1}{Z_n} \cdot w(\boldsymbol{t}) \cdot \mathrm{pdf} K^{(k)}(\mathrm{zip}(\boldsymbol{t}, \boldsymbol{t}'), \boldsymbol{v}'))\, \mu_{\mathbb{Y}^{(k)}}(\mathrm{d}\boldsymbol{v}')\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}')\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t})$$

$$= \sum_{k=1}^n \int_{\mathrm{Supp}\, \iota_{\mathbb{X}}^{(k)}(w)} \int_{\mathbb{T}} \frac{1}{Z_n} \cdot w(\boldsymbol{t})\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}')\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}) \qquad (K^{(k)} \text{ is a probability kernel})$$

$$= \int_{\mathbb{T}} [|\boldsymbol{t}| \leq \iota_{\mathbb{X}}(n)] \cdot \frac{1}{Z_n} \cdot w(\boldsymbol{t})\, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}) = 1$$

2. Let $S \in \Sigma_{\mathbb{S}}$ where $S \subseteq \mathbb{S}_k^{\text{valid}}$. Hence $Z_k \cdot \rho_k(s) = Z_k \cdot \rho_k(s')$ if $s \in S$ and $s = \text{take}_k(s')$. Then,

$$
\begin{aligned}
& Z_n \cdot (\text{take}_{k*}\pi_n)(S) \\
&= Z_n \cdot \pi_n(\text{take}_k^{-1}(S)) \\
&= \int_{\mathbb{S}^{(n)}} [\text{take}_k(s') \in S] \cdot Z_n \cdot \zeta_n(s') \, \mu_{\mathbb{S}^{(n)}}(\mathrm{d}(s')) \\
&= \int_{\mathbb{S}^{(k)}} [(s) \in S] \cdot Z_k \cdot \zeta_k(s) \cdot \mu_{\mathbb{S}^{(k)}}(\mathrm{d}(s)) \\
&= Z_k \cdot \pi_k(S)
\end{aligned}
$$

3. Let $S \in \Sigma_{\mathbb{S}}$ where $S \subseteq \bigcup_{k=1}^{n} \mathbb{S}_k^{\text{valid}}$. Then, $Z \cdot \rho(s) = Z_k \cdot \rho_k(s)$ for all $s \in S \cap \mathbb{S}_k^{\text{valid}}$.

$$
\begin{aligned}
Z \cdot \pi(S) &= \int_S [s \in \mathbb{S}^{\text{valid}}] \cdot Z \cdot \zeta(s) \, \mu_{\mathbb{S}}(\mathrm{d}s) \\
&= \sum_{k=1}^{n} \int_S [s \in \mathbb{S}_k^{\text{valid}}] \cdot Z \cdot \zeta(s) \, \mu_{\mathbb{S}^{(k)}}(\mathrm{d}s) \\
&= \sum_{k=1}^{n} \int_S [s \in \mathbb{S}_k^{\text{valid}}] \cdot Z_k \cdot \zeta_k(s) \, \mu_{\mathbb{S}^{(k)}}(\mathrm{d}s) \\
&= \sum_{k=1}^{n} Z_k \cdot \pi_k(S \cap \mathbb{S}_k^{\text{valid}}) \\
&= Z_n \sum_{k=1}^{n} \text{take}_{k*}\pi_n(S \cap \mathbb{S}_k^{\text{valid}}) && \text{(i)} \\
&= Z_n \cdot \pi_n\Big(\bigcup_{k=1}^{n} \{s \in \mathbb{S}^{(n)} \mid \text{take}_k(s) \in S \cap \mathbb{S}_k^{\text{valid}}\}\Big) \\
&= Z_n \cdot g_*^{(n)} \pi_n(S).
\end{aligned}
$$

$\square$

Lemma B.17 (Invariant). *Assuming H1 to 3, ν is the invariant distribution of the Markov chain generated by iterating the Hyrbid NP-iMCMC algorithm (App. B.3).*

Proof. Assuming (1) $\nu = \mathfrak{m}_*\pi$ on $\mathbb{T}$ and (2) $\mu_{\mathbb{T}} = \mathfrak{m}_*\mu_{\mathbb{S}}$ on $\text{Supp}(w)$, we have for any $A \in \Sigma_{\mathbb{T}}$,

$$
\begin{aligned}
\nu(A) &= \mathfrak{m}_*\pi(A) && \text{(1)} \\
&= \int_{\mathbb{S}} T_{\texttt{eNPiMCMC}}(s, \mathfrak{m}^{-1}(A)) \, \mu_{\mathbb{S}}(\mathrm{d}s) && \text{(Lem. B.14)} \\
&= \int_{\mathbb{S}^{\text{valid}}} T_{\texttt{eNPiMCMC}}(s, \mathfrak{m}^{-1}(A)) \, \mu_{\mathbb{S}}(\mathrm{d}s) \\
&= \int_{\mathbb{S}^{\text{valid}}} T_{\texttt{NPiMCMC}}(\mathfrak{m}(s), A) \, \mu_{\mathbb{S}}(\mathrm{d}s) && \text{(Prop. B.15.ii)} \\
&= \int_{\text{Supp}(w)} T_{\texttt{NPiMCMC}}(t, A) \, \mathfrak{m}_*\mu_{\mathbb{S}}(\mathrm{d}t) \\
&= \int_{\text{Supp}(w)} T_{\texttt{NPiMCMC}}(t, A) \, \mu_{\mathbb{T}}(\mathrm{d}t) && \text{(2)} \\
&= \int_{\mathbb{T}} T_{\texttt{NPiMCMC}}(t, A) \, \mu_{\mathbb{T}}(\mathrm{d}t).
\end{aligned}
$$

It is enough to show (1) and (2).

1. Let $A \in \Sigma_{\mathbb{T}}$ where $A \subseteq \text{Supp}^{\iota_{\times}(n)}(w)$ and $\delta > 0$. Then partitioning $\mathfrak{m}^{-1}(A)$ using $\mathbb{S}_k^{\text{valid}}$, we have for sufficiently large m,

$$
\mathfrak{m}_* \pi(A)
$$

$$
= \pi \left(\bigcup_{k=1}^m \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}} \right) + \pi \left(\bigcup_{k=m+1}^\infty \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}} \right)
$$

$$
< \frac{Z_m}{Z} \cdot g_*^{(m)} \pi_m \left(\bigcup_{k=1}^m \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}} \right) + \delta \qquad \text{(Prop. B.15.iii, Prop. B.16.ii)}
$$

$$
\leq \frac{Z_m}{Z} \cdot \pi_m(\{(\text{zip}(\boldsymbol{t}, \boldsymbol{t}') + \boldsymbol{y}, \boldsymbol{v}) \mid \boldsymbol{t} \in A, \boldsymbol{t}' \in \mathbb{T}, \boldsymbol{y} \in \mathbb{E}^{\iota_{\times}(m) - \iota_{\times}(n)}, \boldsymbol{v} \in \mathbb{Y}^{(m)}\}) + \delta
$$

$$
= \frac{1}{Z} \int_A \int_{\mathbb{T}} \int_{\mathbb{E}^{\iota_{\times}(m) - \iota_{\times}(n)}} w(\boldsymbol{t}) \cdot
$$

$$
\left(\int_{\mathbb{Y}^{(m)}} \text{pdf} K^{(n)} (\text{take}_n(\text{zip}(\boldsymbol{t}, \boldsymbol{t}') + \boldsymbol{y}, \boldsymbol{v})) \, \mu_{\mathbb{Y}^{(m)}}(\mathrm{d}\boldsymbol{v}) \right)
$$

$$
\mu_{\mathbb{E}^{\iota_{\times}(m) - \iota_{\times}(n)}}(\mathrm{d}\boldsymbol{y}) \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}') \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}) + \delta
$$

$$
= \frac{1}{Z} \int_A w(\boldsymbol{t}) \, \mu_{\mathbb{T}}(\mathrm{d}\boldsymbol{t}) + \delta \qquad \text{($K^{(n)}$ is a probability kernel)}
$$

$$
= \nu(A) + \delta.
$$

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

2. Similarly, let $A \in \Sigma_{\mathbb{T}}$ where $A \subseteq \text{Supp}^{\iota_{\times}(n)}(w)$ and $\delta > 0$. Then by Prop. B.15.iii, 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,

$$
\mathfrak{m}_* \mu_{\mathbb{S}}(A)
$$

$$
= \mu_{\mathbb{S}} \left(\bigcup_{k=1}^m \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}} \right) + \mu_{\mathbb{S}} \left(\bigcup_{k=m+1}^\infty \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}} \right)
$$

$$
< \sum_{k=1}^m \mu_{\mathbb{S}^{(k)}} (\mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}) + \delta
$$

$$
= \sum_{k=1}^m \mu_{\mathbb{S}^{(m)}} (\{(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}^{(m)} \mid \text{take}_k(\boldsymbol{x}, \boldsymbol{v}) \in \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\}) + \delta
$$

$$
= \mu_{\mathbb{S}^{(m)}} (\bigcup_{k=1}^m \{(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{S}^{(m)} \mid \text{take}_k(\boldsymbol{x}, \boldsymbol{v}) \in \mathfrak{m}^{-1}(A) \cap \mathbb{S}_k^{\text{valid}}\}) + \delta
$$

$$
\leq \mu_{\mathbb{S}^{(m)}} (\{(\text{zip}(\boldsymbol{t}, \boldsymbol{t}') + \boldsymbol{y}, \boldsymbol{v}) \mid \boldsymbol{t} \in A, \boldsymbol{t}' \in \mathbb{T}, \boldsymbol{y} \in \mathbb{E}^{\iota_{\times}(m) - \iota_{\times}(n)}, \boldsymbol{v} \in \mathbb{Y}^{(m)}\}) + \delta
$$

$$
= \mu_{\mathbb{T}}(A) + \delta.
$$

   Then the proof proceeds as in (1). Note that since w almost surely terminating (H 2), $\mathfrak{m}_* \mu_{\mathbb{S}}(\text{Supp}(w)) = \mu_{\mathbb{T}}(\text{Supp}(w)) = 1$

$\square$

B.4.4. CORRECTNESS OF NP-IMCMC

The correctness of the NP-iMCMC sampler (Fig. 2) can be deduce from Lem. B.17 and the fact that Hyrbid NP-iMCMC is a generalisation of NP-iMCMC

Corollary B.18 (Invariant). *If all inputs satisfy V 1 to 3 then ν is the invariant distribution of the Markov chain generated by iterating the algorithm described in Fig. 2.*

C. Transforming Nonparametric Involutive MCMC

In this section, we discuss how the techniques discussed in (Neklyudov et al., 2020) can be applied to the Hybrid NP-iMCMC sampler presented in App. B. Hence instances of the Hybrid NP-iMCMC sampler, such as NP-MH and NP-HMC, can be extended using these techniques to become more flexible and efficient.

We assume the input target density function $w : \mathbb{T} \to [0, \infty)$ is tree representable, integrable (H1) and almost surely terminating (H2).

C.1. State-dependent Hybrid NP-iMCMC Mixture

Say we want to use multiple Hybrid NP-iMCMC samplers to simulate the posterior given by the target density function w. The following technique allows us to 'mix' Hybrid NP-iMCMC samplers in such a way that the resulting sampler still preserves the posterior.

Given a collection of Hybrid NP-iMCMC samplers, indexed by $m \in \mathbb{E}^\alpha$, for some $\alpha \in \mathbb{N}$, each with auxiliary kernels $\{K_m^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}\}_{n \in \mathbb{N}}$ and involutions $\{\Phi^{(n)}{}_m : \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)} \to \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}\}_{n \in \mathbb{N}}$ satisfying the projection commutation property (H3), the **State-dependent Hybrid NP-iMCMC Mixture** sampler determines which Hybrid NP-iMCMC sampler to use by drawing an indicator $m \in \mathbb{E}^\alpha$ from a probability measure $K_M(\boldsymbol{x}_0, \cdot)$ where $K_M : \bigcup_{n \in \mathbb{N}} \mathbb{X}^{(n)} \rightsquigarrow \mathbb{E}^m$ is a probability kernel and $\boldsymbol{x}_0$ is the entropy vector constructed from the current sample $\boldsymbol{t}_0$ at the initialisation step (Step 1 of Hybrid NP-iMCMC). Then, using the m-indexed Hybrid NP-iMCMC sampler, a proposal $\boldsymbol{t}$ is generated and accepted with a modified probability that includes the probability of picking m, namely

$$\min\left\{1; \frac{w(\boldsymbol{t}) \cdot \mathsf{pdf}\,K^{(k)}{}_m(\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v})) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v})}{w(\boldsymbol{t}_0) \cdot \mathsf{pdf}\,K^{(k_0)}{}_m(\mathsf{take}_{k_0}(\boldsymbol{x}_0, \boldsymbol{v}_0)) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot \frac{\mathsf{pdf}\,K_M(\boldsymbol{x}_0{}^{1..k_0}, m)}{\mathsf{pdf}\,K_M(\boldsymbol{x}^{1..k}, m)} \cdot |\det(\nabla \Phi^{(n)}{}_m(\boldsymbol{x}_0, \boldsymbol{v}_0))|\right\}$$

where $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ is the (possibly extended) initial state, $(\boldsymbol{x}, \boldsymbol{v})$ is the new state, $n = \dim(\boldsymbol{x}_0) = \dim(\boldsymbol{v}_0)$, k_0 is the dimension of $\boldsymbol{t}_0$ (i.e. $|\boldsymbol{t}_0| = \iota_{\mathbb{X}}(k_0)$) and k is the dimension of $\boldsymbol{t}$ (i.e. $|\boldsymbol{t}| = \iota_{\mathbb{X}}(k)$).

Pseudocode This sampler can be implemented in SPCF as the `MixtureNPiMCMC` function in Listing 4. (Terms specific to this technique are highlighted.) We assume the following SPCF terms exists: `mixkernel` of type `List(X) -> (R*B)^l` implements the mixture kernel $K_M : \bigcup_{n \in \mathbb{N}} \mathbb{X}^{(n)} \rightsquigarrow \mathbb{E}^\alpha$; `pdfmixkernel` of type `List(X) * (R*B)^l -> R` implements the probability density function $\mathsf{pdf}\,K_M : \bigcup_{n \in \mathbb{N}} \mathbb{X}^{(n)} \times \mathbb{E}^\alpha \to \mathbb{R}$; and for each $m \in \mathbb{E}^\alpha$ and $n \in \mathbb{N}$, `auxkernel[n][m]` implements the auxiliary kernel $K_m^{(n)}$; `pdfauxkernel[n][m]` and implements the pdf $\mathsf{pdf}\,K^{(n)}{}_m$; `involution[n][m]` implements the involution $\Phi^{(n)}{}_m$; and `absdetjacinv[n][m]` implements the absolute value of the Jacobian determinant of $\Phi^{(n)}{}_m$.

Correctness Similar to the correctness arguments in (Neklyudov et al., 2020), we show that the State-dependent Hybrid NP-iMCMC Mixture sampler is correct by formulating `MixtureNPiMCMC` as an instance of `NPiMCMC` (Listing 2). This means specifying `auxkernel[n]` and `involution[n]` in `NPiMCMC` and arguing that the resulting `NPiMCMC` function is equivalent to `MixtureNPiMCMC`.

The SPCF terms `mixauxkernel[n]` and `mixinvolution[n]` given in Listing 5 should suffice. The auxiliary space is expanded to embed the indicator `m` in such a way that the auxiliary variable `mixv` is in the space $\mathbb{E}^\alpha \times \mathbb{Y}^{(n)}$ where its first ℓ-th components `mixv[:l]` gives `m` and the rest `mixv[l:]` gives `v`. Since the auxiliary space is expanded to include the indicator, the maps `mixindexX` and `mixindexY` and the projection `mixproj` are modified accordingly.

To see how the `NPiMCMC` function with `auxkernel[n]` replaced by `mixauxkernel[n]` and `involution[n]` replaced by `mixinvolution[n]` is equivalent to `MixtureNPiMCMC`, we onyl need to consider the probability density of `mixauxkernel[k]` at `mixproj((x,mixv),k)`.

```
  pdfmixauxkernel[k](x[:mixindexX(k)], mixv[:mixindexY(k)])
= pdfmixauxkernel[k](x[:indexX(k)], mixv[:l+indexY(k)])
= pdfmixkernel(x[:indexX(k)], mixv[:l]) * pdfauxkernel[k][mixv[:l]](x[:indexX(k)],
    mixv[l:l+indexY(k)])
```

Listing 4. Pseudocode of the State-dependent Hybrid NP-iMCMC Mixture algorithm

```
1    def MixtureNPiMCMC(t0):
2      k0 = dim(t0)                                    # initialisation step
3      x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4      m = mixkernel(x0)                               # mixture step
5      v0 = auxkernel[k0][m](x0)                        # stochastic step
6      (x,v) = involution[k0][m](x0,v0)                # deterministic step
7      n = k0                                          # extend step
8      while not intersect(instance(x),support(w)):
9        x0 = x0 + [(normal, coin)]*(indexX(n+1)-indexX(n))
10       v0 = v0 + [(normal, coin)]*(indexY(n+1)-indexY(n))
11       n = n + 1
12       (x,v) = involution[n][m](x0,v0)
13     t = intersect(instance(x),support(w))[0]        # accept/reject step
14     k = dim(t)
15     return t if uniform < min{1, w(t)/w(t0) *
16                               pdfauxkernel[k][m](proj((x,v),k))/
17                                 pdfauxkernel[k0][m](proj((x0,v0),k0)) *
18                               pdfpar[n](x)/pdfpar[n](x0) *
19                               pdfaux[n](v)/pdfaux[n](v0) *
20                               pdfmixkernel(proj(x,k),m)/
21                                 pdfmixkernel(proj(x0,k0),m) *
22                               absdetjacinv[n][m](x0,v0)}
23             else t0
```

Listing 5. Pseudocode for the correctness of the State-dependent Hybrid NP-iMCMC Mixture algorithm

```
1    def mixauxkernel[n](x0)
2      m = mixkernel(x0)
3      v0 = auxkernel[n][m](x0)
4      return m + v0
5
6    def mixinvolution[n](x0,mixv0)
7      m = mixv0[:l]
8      v0 = mixv0[l:]
9      (x,v) = involution[n][m](x0,v0)
10     return (x,m + v)
11
12   def mixindexX(n): return indexX(n)
13   def mixindexY(n): return l + indexY(n)
14   def mixproj((x,v),k): return (x[:mixindexX(k)],v[:mixindexY(k)])
```

```
    = pdfmixkernel(x[:indexX(k)], m) * pdfauxkernel[k][m](x[:indexX(k)], v[:indexY(k)])
    = pdfmixkernel(proj(x,k),m) * pdfauxkernel[k][m](proj((x,v),k))
```

where `m = mixv[:l]` and `v = mixv[l:]`. This shows that the acceptance probability in `NPiMCMC` is identical to that in `MixtureNPiMCMC` and hence the two algorithms are equivalent.

C.2. Direction Hybrid NP-iMCMC Algorithm

Sometimes it is difficult to specify involutions that explores the model fully. The following technique tells us that bijections are good enough.

Given endofunctions $f^{(n)}$ on $\mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}$ that are differentiable almost everywhere and bijective for each $n \in \mathbb{N}$ such that the sets $\{f^{(n)}\}_n$ and $\{f^{(n)}{}^{-1}\}_n$ satisfy the projection commutative property (H3), the ***Direction Hybrid NP-iMCMC*** algorithm randomly use either $f^{(n)}$ or $f^{(n)}{}^{-1}$ to move around the state space and proposes a new sample.

Pseudocode This sampler can be expressed in SPCF as the `DirectionNPiMCMC` function in Listing 6. (Terms specific to this technique are highlighted.) We assume for each $n \in \mathbb{N}$ and $d \in 2$, there is a SPCF term `bijection[n][d]` where `bijection[n][True]` implements the bijection $f^{(n)}$ and `bijection[n][False]` implements the inverse $f^{(n)}{}^{-1}$ and SPCF term `absdetjacbij[n][d]` that implements the absolute value of the Jacobian determinant of $f^{(n)}$ if `d = True` and the inverse $f^{(n)}{}^{-1}$ otherwise.

Correctness We show that `DirectionNPiMCMC` can be formulated as an instance of `NPiMCMC` (Listing 2) with a specification of `auxkernel[n]` and `involution[n]`.

The SPCF terms `dirauxkernel[n]` and `dirinvolution[n]` in Listing 7 would work. The auxiliary space is expanded to include the direction variable `d0` so that the auxiliary variable `dirv0` is in the space $\mathbb{E} \times \mathbb{Y}^{(n)}$ where the Boolean-component `dirv0[0][1]` of its first coordinate gives `d0` and the second to last coordinates `dirv0[1:]` gives `v0`. (Note the value of `dirv0[0][0]` is redundant and is only used to make `dirv0[0]` an entropy.) Since the auxiliary space is expanded, the maps `dirindexX` and `dirindexY` and the projection `dirproj` are modified accordingly.

To see how the `NPiMCMC` function with `auxkernel[n]` replaced by `dirauxkernel[n]` and `involution[n]` replaced by `dirinvolution[n]` is equivalent to `DirectionNPiMCMC`, we first consider the density of `dirauxkernel[k0]` at `dirproj((x0,dirv0),k0)`.

```
  pdfdirauxkernel[k0](x0[:dirindexX(k0)], dirv0[:dirindexY(k0)])
= pdfdirauxkernel[k0](x0[:indexX(k0)], dirv0[:1+indexY(k0)])
= pdfcoin(dirv0[0][1]) * pdfnormal(dirv0[0][0]) * pdfauxkernel[k0](x0[:indexX(k0)],
    dirv0[1:1+indexY(k0)])
= 0.5 * pdfnormal(dirv0[0][0]) * pdfauxkernel[k0](proj((x0,v0),k0))
```

where `v0 = dirv0[1:]`. A similar argument can be made for `pdfdirauxkernel[k](dirproj((x,dirv),k))`, which makes the acceptance probability in `NPiMCMC` identical to that in `DirectionNPiMCMC`. Moreover, writing `d0` for `dirv0[0][1]`, the absolute value of the Jacobian determinant of `dirinvolution[n]` at `(x0,dirv0)` is `absdetjacbij[n][d0](x0,v0)`. Most importantly, `dirinvolution[n]` is now involutive. Hence, `NPiMCMC` is the same as `DirectionNPiMCMC`.

C.3. Persistent Hybrid NP-iMCMC Algorithm

It is known that irreversible transition kernels (those that does not satisfy detailed balance) have better mixing times, i.e. converges more quickly to the target distribution, compared to reversible ones. The following technique gives us a method to transform Hybrid NP-iMCMC algorithms to irreversible ones that still preserves the target distribution. The key is to compose the Hybrid NP-iMCMC sampler with a transition kernel so that the resulting algorithm does not satisfy detailed

Listing 6. Pseudocode of the Direction Hybrid NP-iMCMC algorithm

```
1   def DirectionNPiMCMC(t0):
2     d0 = coin                                    # direction step
3     k0 = dim(t0)                                 # initialisation step
4     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
5     v0 = auxkernel[k0](x0)                       # stochastic step
6     (x,v) = bijection[k0][d0](x0,v0)             # deterministic step
7     n = k0                                       # extend step
8     while not intersect(instance(x),support(w)):
9       x0 = x0 + [(normal, coin)]*(indexX(n+1)-indexX(n))
10      v0 = v0 + [(normal, coin)]*(indexY(n+1)-indexY(n))
11      n = n + 1
12      (x,v) = bijection[n][d0](x0,v0)
13    t = intersect(instance(x),support(w))[0]     # accept/reject step
14    k = dim(t)
15    return t if uniform < min{1, w(t)/w(t0) *
16                                pdfauxkernel[k](proj((x,v),k))/
17                                  pdfauxkernel[k0](proj((x0,v0),k0)) *
18                                pdfpar[n](x)/pdfpar[n](x0) *
19                                pdfaux[n](v)/pdfaux[n](v0) *
20                                absdetjacbij[n][d0](x0,v0)}
21            else t0
```

Listing 7. Pseudocode for `dirauxkernel` and `dirinvolution`

```
1   def dirauxkernel[n](x0)
2     d0 = coin
3     v0 = auxkernel[n](x0)
4     return [(normal, d0)] + v0
5
6   def dirinvolution[n](x0,dirv0)
7     d0 = dirv0[0][1]
8     v0 = dirv0[1:]
9     (x,v) = bijection[n][d0](x0,v0)
10    d = not d0
11    return (x, [(dirv0[0][0],d)] + v)
12
13  def dirindexX(n): return indexX(n)
14  def dirindexY(n): return 1+indexY(n)
15  def dirproj((x,v),k): return (x[:dirindexX(k)], v[:dirindexY(k)])
```

balance.

The ***Persistent Hybrid NP-iMCMC*** algorithm is a MCMC algorithm similar to the Direction Hybrid NP-iMCMC sampler in which the direction variable is used to determine auxiliary kernels ($\{K_1^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}\}_n$ or $\{K_2^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}\}_n$) and bijections ($\{f^{(n)} : \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)} \to \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}\}_n$ or $\{f^{(n)^{-1}} : \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)} \to \mathbb{X}^{(n)} \times \mathbb{Y}^{(n)}\}_n$) being used. The difference is that Persistent Hybrid NP-iMCMC keeps track of the direction (instead of sampling a fresh one in each iteration) and flips it strategically to make the resulting algorithm irreversible.

Pseudocode This sampler can be expressed in SPCF as `PersistentNPiMCMC` in Listing 8. (Terms specific to this technique are highlighted.) In addition to the SPCF terms in `DirectionNPiMCMC`, we assume there is a SPCF term `auxkernel[n][d]` such that `auxkernel[n][True]` implements the auxiliary kernel $K_1^{(n)}$: $\mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$ and `pdfauxkernel[n][True]` its pdf $\mathrm{pdf}K^{(n)}_1$ and similarly for `auxkernel[n][False]` and `pdfauxkernel[n][False]`. Note that `PersistentNPiMCMC` updates samples on the space $\mathbb{X}^{(n)} \times 2$, which can easily be marginalised to $\mathbb{X}^{(n)}$ by taking the first $\iota_{\mathbb{X}}(n)$ components.

Correctness We show that `PersistentNPiMCMC` can be formulated as a composition of two instances of `NPiMCMC` (Listing 2).

Consider the `NPiMCMC` with auxiliary kernel `perauxkernel[n]` and involution `perinvolution[n]` in Listing 9. In this case, the parameter space is expanded to include the direction variable so that a parameter variable `perx` is on the space $\mathbb{E} \times \mathbb{X}^{(n)}$ where `perx[0][1]` gives `d` and `perx[1:]` gives `x`. Since the parameter space is expanded, the maps `perindexX` and `perindexY` and projection `perproj` are modified accordingly.

Again, we first consider the density of `perauxkernel[k0]` at `perproj((perx0,v0),k0)`.

```
  pdfperauxkernel[k0](perx0[:perindexX(k0)], v0[:perindexY(k0)])
= pdfauxkernel[k0][perx[0][1]](perx0[1:1+indexX(k0)], v0[:indexY(k0)])
= pdfauxkernel[k0][d0](x0[:indexX(k0)], v0[:indexY(k0)])
= pdfauxkernel[k0][d0](proj((x0,v0),k0))
```

where `d0 = perx0[0][1]` and `x0 = perx0[1:]`. A similar argument can be made for `pdfperauxkernel[k](perproj((perx,v),k))`. Moreover, the absolute value of the Jacobian determinant of `perinvolution[n]` at `(perx0,v0)` is `absdetjacbij[n][d0](x0,v0)`. Hence, the acceptance probability in `NPiMCMC` is identical to that in `PersistentNPiMCMC`.

The `NPiMCMC` function with `auxkernel[n]` replaced by `perauxkernel[n]` and `involution[n]` replaced by `perinvolution[n]` is *almost* equivalent to `PersistentNPiMCMC`, except `NPiMCMC` induces a transition kernel on $\mathbb{E} \times \mathbb{X}^{(n)}$ whereas `PersistentNPiMCMC` induces a transition kernel on $2 \times \mathbb{X}^{(n)}$; and when the proposal `t` is accepted, `NPiMCMC` returns `d` whereas `PersistentNPiMCMC` returns `not d`.

These differences can be reconciled by composing `NPiMCMC` with `flipdir`, which is an instance of `NPiMCMC` which skips the stochastic step and has an involution that flips the direction variable stored in `perx0[0][1]`. The composition generates a Markov chain on $\mathbb{E} \times \mathbb{X}^{(n)}$ and marginalising it to a Markov chain on $2 \times \mathbb{X}^{(n)}$ gives us the same result as `PersistentNPiMCMC`.

Listing 8. Pseudocode of the Persistent Hybrid NP-iMCMC algorithm

```
1    def PersistentNPiMCMC(t0,d0):
2      k0 = dim(t0)                                  # initialisation step
3      x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4      v0 = auxkernel[k0][d0](x0)                    # stochastic step
5      (x,v) = bijection[k0][d0](x0,v0)              # deterministic step
6      n = k0                                        # extend step
7      while not intersect(instance(x),support(w)):
8        x0 = x0 + [(normal, coin)]*(indexX(n+1)-indexX(n))
9        v0 = v0 + [(normal, coin)]*(indexY(n+1)-indexY(n))
10       n = n + 1
11       (x,v) = bijection[n][d0](x0,v0)
12     d = not d0
13     t = intersect(instance(x),support(w))[0]     # accept/reject step
14     k = dim(t)
15     return (t, not d) if uniform < min{1, w(t)/w(t0) *
16                            pdfauxkernel[k][d](proj((x,v),k))/
17                             pdfauxkernel[k0][d0](proj((x0,v0),k0)) *
18                            pdfpar[n](x)/pdfpar[n](x0) *
19                            pdfaux[n](v)/pdfaux[n](v0) *
20                            absdetjacbij[n][d0](x0,v0)}
21              else (t0, d)
```

Listing 9. Pseudocode for `perauxkernel` and `perinvolution`

```
1    def perauxkernel[n](perx0)
2      d0 = perx0[0][1]
3      x0 = perx0[1:]
4      v0 = auxkernel[n][d0](x0)
5      return v0
6
7    def perinvolution[n](perx0,v0)
8      d0 = perx0[0][1]
9      x0 = perx0[1:]
10     (x,v) = bijection[n][d0](x0,v0)
11     d = not d0
12     return ([(perx0[0][0],d)] + x, v)
13
14   def perindexX(n): return 1+indexX(n)
15   def perindexY(n): return indexY(n)
16   def perproj((x,v),k): return (x[:perindexX(k)],v[:perindexY(k)])
17
18   def flipdir(perx0):
19     d0 = perx0[0][1]
20     perx0[0][1] = not d0
21     return perx0
```

D. Multiple Step Nonparametric Involutive MCMC

In this section, we study the **Multiple Step NP-iMCMC** sampler, a generalisation of the Hybrid NP-iMCMC sampler (App. B.3) (and also of NP-iMCMC (Fig. 2)), where the involution is applied multiple times to generate a proposed state.

D.1. Motivation

Step 4 in the Hyrbid NP-iMCMC sampler may seem inefficient. While it terminates almost surely (thanks to H2), This is because whenever the dimension of the state is changed, the algorithm has to "re-run" the involution again (Step 4.ii). This means the expected number of iterations may be infinite.

To remedy this problem, we introduce two new concepts:

- The *slice function* which might make "re-runs" (Step 4.ii) quicker.

- The *Multiple Step NP-iMCMC* sampler, a generaliation of Hyrbid NP-iMCMC, which uses a list of bijections to move around the state space.

D.2. Slice function

For each dimension $n \in \mathbb{N}$, we call the measurable function $s^{(n)} : \mathbb{S}^{(n)} \to \mathbb{E}^{\iota_{\mathbb{X}}(n) - \iota_{\mathbb{X}}(n-1)} \times \mathbb{E}^{\iota_{\mathbb{Y}}(n) - \iota_{\mathbb{Y}}(n-1)}$ a **slice** of the endofunction $\Phi^{(n)}$ on $\mathbb{S}^{(n)}$ if it captures the movement of the n-th dimensional states with an instance of dimension *lower than n*. Formally, this means

$$s^{(n)}(\boldsymbol{x}, \boldsymbol{v}) = (\mathsf{drop}_{n-1} \circ \Phi^{(n)})(\boldsymbol{x}, \boldsymbol{v}) \quad \text{if } \boldsymbol{t} \in \mathsf{instance}(\boldsymbol{x}) \cap \mathsf{Supp}(w) \text{ and } |\boldsymbol{t}| < \iota_{\mathbb{X}}(n).$$

Note we can always define a slice of $\Phi^{(n)}$ by setting $s^{(n)} := \mathsf{drop}_{n-1} \circ \Phi^{(n)}$.

With the slice function $s^{(n)}$ defined for each involution $\Phi^{(n)}$, Step 4.ii in the Hyrbid NP-iMCMC algorithm (App. B.3):

(Step 4.ii) Move around the $n + 1$-dimensional state space $\mathbb{X}^{(n+1)} \times \mathbb{Y}^{(n+1)}$ and compute the new state by applying the involution $\Phi^{(n+1)}$ to the initial state $(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$;

can be replaced by the following Step 4.ii':

(Step 4.ii') Replace and extend the n-dimensional new state from $(\boldsymbol{x}, \boldsymbol{v})$ to a state $(\boldsymbol{x} + \boldsymbol{y}, \boldsymbol{v} + \boldsymbol{u})$ of dimension $n + 1$ where $(\boldsymbol{y}, \boldsymbol{u})$ is the result of $s^{(n+1)}(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$.

By H3, the first n components of the new $n + 1$-dimensional state $\Phi^{(n+1)}(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$ is

$$\mathsf{take}_n(\Phi^{(n+1)}(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)) = \Phi^{(n)}(\mathsf{take}_n(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)) = \Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0) = (\boldsymbol{x}, \boldsymbol{v})$$

and by the definition of slice the $(n + 1)$-th component of the new state is

$$\mathsf{drop}_n(\Phi^{(n+1)}(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)) = s^{(n+1)}(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0).$$

Hence the new state computed by Step 4.ii and Step 4.ii' are the same.

The slice function $s^{(n)}$ is useful when the involution is computationally expensive but has a light slice function. After Step 4.ii is replaced by Step 4.ii', the Hyrbid NP-iMCMC sampler need only to run the involution once (Step 3) and any subsequent "re-runs" (Step 4) can be performed by the slice function.

If the slice function $s^{(n)}$ is implemented as `slice[n]` in SPCF, Line 11 in `NPiMCMC` can be changed from `(x,v) = involution[n](x0,v0)` to

```
(x',v') = slice[n](x0,v0); (x,v) = (x + x', v + v')
```

D.2.1. EXAMPLE (HMC)

Momentum update is the most computationally heavy component in the HMC sampler. Hence it would be useful if it has a lightweight slice function.

In the setting of Hyrbid NP-iMCMC, we assume the trace space $\mathbb{T}$ is a list measurable space of the Real measurable space $\mathbb{R}$. Then, the n-dimensional momentum update ϕ_k^M is an endofunction on $\mathbb{R}^n \times \mathbb{R}^n$ defined as

$$\phi_k^M(\boldsymbol{q}, \boldsymbol{p}) := (\boldsymbol{q}, \boldsymbol{p} - k\nabla U(\boldsymbol{q}))$$

where $U(\boldsymbol{q}) := -\log \max\{w(\boldsymbol{t}) \mid \boldsymbol{t} \in \mathsf{instance}(\boldsymbol{q})\}$. is the n-dimensional potential energy.

Given a n-dimensional state $(\boldsymbol{q}, \boldsymbol{p})$ where $\boldsymbol{t} \in \mathsf{instance}(\boldsymbol{q}) \cap \mathrm{Supp}(w)$ has dimension lower than n, the gradient of the potential energy U at $\boldsymbol{q}$ w.r.t. the n-th coordinate is zero. Hence,

$$(\mathsf{drop}_{n-1} \circ \phi_k^M)(\boldsymbol{q}, \boldsymbol{p}) = \mathsf{drop}_{n-1}(\boldsymbol{q}, \boldsymbol{p} - k\nabla U(\boldsymbol{q})) = (\boldsymbol{q}^n, \boldsymbol{p}^n),$$

and the slice of the momentum update ϕ_k^M is simply the projection $\mathsf{drop}_{n-1}(\boldsymbol{q}, \boldsymbol{p}) := (\boldsymbol{q}^n, \boldsymbol{p}^n)$.

However, not *all* $2L$ momentum updates in the re-runs of the leapfrog function $\boldsymbol{L}$ can be replaced by its slice drop_{n-1}. This is because when the dimension increments to say $n+1$, only the extended initial state $(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$ has the property that it has an instance with dimension lower than $n+1$ and not the intermediate states.

D.3. Multiple Step NP-iMCMC

Say the involution of a Hybrid NP-iMCMC sampler is comprised of a list of bijective endofunctions on $\mathbb{S}^{(n)}$, namely $\Phi^{(n)} := f_L^{(n)} \circ \cdots \circ f_2^{(n)} \circ f_1^{(n)}$. To compute the new state, we can either

- apply the involution $\Phi^{(n)}$ to the initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ in one go and check whether the result $(\boldsymbol{x}, \boldsymbol{v})$ has an instance in the support of w, or

- for each $\ell = 1, \ldots, L$, apply the endofunction $f_\ell^{(n)}$ to $(\boldsymbol{x}_{\ell-1}, \boldsymbol{v}_{\ell-1})$ and (immediately) check whether the intermediate state $(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell)$ has an instance in the support of w.

The Hybrid NP-iMCMC sampler presented in App. B.3 takes the first option as it is conceptually simpler. However, the second option is just as valid and more importantly give us the requirements needed to replace each endofunction by its slice in any subsequent "re-runs".

D.3.1. THE MULTIPLE STEP NP-IMCMC ALGORITHM

Assume the target density w satisfies V1 and 2; and for each $n \in \mathbb{N}$, there is a probability kernel $K^{(n)}$ and a list of L bijective endofunctions $\{f_\ell^{(n)} : \mathbb{S}^{(n)} \to \mathbb{S}^{(n)} \mid \ell = 1, \ldots, L\}_n$ such that for each ℓ, $\{f_\ell^{(n)}\}_n$ satisfies the projection commutation property (V3) and for each $n \in \mathbb{N}$, their composition $f_L^{(n)} \circ \cdots \circ f_1^{(n)}$ is involutive.

Let $s_\ell^{(n)}$ be a slice of the endofunction $f_\ell^{(n)}$. Given a SPCF program M with weight function w on the trace space, the ***Multiple Step NP-iMCMC*** sampler generates a Markov chain as follows. Given a current sample $\boldsymbol{t}_0$ of dimension k_0,

1. (Initialisation Step) Form a k_0-dimensional parameter variable $\boldsymbol{x}_0 \in \mathbb{X}^{(k_0)}$ by pairing each value $\boldsymbol{t}_0{}^i$ in $\boldsymbol{t}_0$ with a randomly drawn value t of the other type to make a pair $(\boldsymbol{t}_0{}^i, t)$ or $(t, \boldsymbol{t}_0{}^i)$ in the entropy space $\mathbb{E}$.

2. (Stochastic Step) Introduce randomness to the sampler by drawing a k_0-dimensional value $\boldsymbol{v}_0 \in \mathbb{Y}^{(k_0)}$ from the probability measure $K^{(k_0)}(\boldsymbol{x}_0, \cdot)$.

3. (Multiple Step) Initialise $\ell = 1$. If $\ell = L$, proceed to Step 4 with $\boldsymbol{t}$ as the proposed sample; otherwise

    3.1. (Deterministic Step) Compute the ℓ-th state $(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell)$ by applying the endofunction $f_\ell^{(n)}$ to $(\boldsymbol{x}_{\ell-1}, \boldsymbol{v}_{\ell-1})$ where $n = \dim(\boldsymbol{x}_{\ell-1})$.

    3.2. (Extend Step) Test whether any instance $\boldsymbol{t}$ of $\boldsymbol{x}_\ell$ is in the support of w. If so, go to Step 3 with an incremented ℓ; otherwise (none of the instances of $\boldsymbol{x}_\ell$ is in the support of w),

3.2.i Extend and replace the n-dimensional initial state from $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ to a state $(\boldsymbol{x}_0 + \boldsymbol{y}_0, \boldsymbol{v}_0 + \boldsymbol{u}_0)$ of dimension $n+1$ where $\boldsymbol{y}_0$ and $\boldsymbol{u}_0$ are values drawn randomly from $\mu_{\mathbb{E}^{\iota}\mathbb{X}^{(n+1)-\iota}\mathbb{X}^{(n)}}$ and $\mu_{\mathbb{E}^{\iota}\mathbb{Y}^{(n+1)-\iota}\mathbb{Y}^{(n)}}$ respectively.

3.2.ii For each $i = 1, \ldots, \ell$, extend and replace the n-dimensional i-th intermediate state from $(\boldsymbol{x}_i, \boldsymbol{v}_i)$ to a state $(\boldsymbol{x}_i + \boldsymbol{y}_i, \boldsymbol{v}_i + \boldsymbol{u}_i)$ of dimension $n+1$ where $(\boldsymbol{y}_i, \boldsymbol{u}_i)$ is the result of $s_i^{(n+1)}(\boldsymbol{x}_{i-1}, \boldsymbol{v}_{i-1})$.

3.2.iii Go to Step 3.2 with the extended $n+1$-dimensional states $(\boldsymbol{x}_i, \boldsymbol{v}_i)$ for $i = 0, \ldots, \ell$.

4. (Accept/reject Step) Accept the proposed sample $\boldsymbol{t}$ as the next sample with probability

$$\min \left\{ 1; \frac{w(\boldsymbol{t}) \cdot \mathsf{pdf}\,K^{(k)}(\mathsf{take}_k(\boldsymbol{x}_L, \boldsymbol{v}_L)) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_L) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_L)}{w(\boldsymbol{t}_0) \cdot \mathsf{pdf}\,K^{(k_0)}(\mathsf{take}_{k_0}(\boldsymbol{x}_0, \boldsymbol{v}_0)) \cdot \varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0) \cdot \varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot \prod_{\ell=1}^{L} |\det(\nabla f_\ell^{(n)}(\boldsymbol{x}_{\ell-1}, \boldsymbol{v}_{\ell-1}))| \right\}$$

where $n = \dim(\boldsymbol{x}_0) = \dim(\boldsymbol{v}_0)$, k is the dimension of $\boldsymbol{t}$ and k_0 is the dimension of $\boldsymbol{t}_0$; otherwise reject the proposal and repeat $\boldsymbol{t}_0$.

Unlike in Hybrid NP-iMCMC, the Multiple Step NP-iMCMC sampler computes the intermediate states $\{(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell)\}_{\ell=1,\ldots,L}$ one-by-one, making sure in Step 3.2 that each of these state $(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell)$ has an instance in the support of w. Hence when the dimension is incremented from n to $n+1$ we can use the slice functions to extend intermediate states to states of dimension $n+1$.

Remark D.1. The Multiple Step NP-iMCMC sampler can be seen as a generalisation of Hybrid NP-iMCMC (and hence a generalisation of NP-iMCMC (App. B.3.2)) as we can recover Hybrid NP-iMCMC by setting L to one and taking the involution $\Phi^{(n)}$ as the only endofunction in Multiple Step NP-iMCMC.

D.3.2. PSEUDOCODE OF MULTIPLE STEP NP-IMCMC ALGORITHM

Listing 10 gives a SPCF implementation of Multiple Step NP-iMCMC as the function `MultistepNPiMCMC` with target density `w` ; auxiliary kernel `auxkernel[n]` and its density `pdfauxkernel[n]` and `L` number of endofunctions `f[n][l]` (`l` ranges from `1` to `L`) for each dimension `n` with slice `slice[n][l]` and the absolute value of its Jacobian determinant `absdetjacf[n][l]` ; parameter and auxiliary index maps `indexX` and `indexY` and projection `proj` .

D.3.3. CORRECTNESS OF MULTIPLE STEP NP-IMCMC ALGORITHM

The Multiple Step NP-iMCMC sampler cannot be formulated as an instance of Hybrid NP-iMCMC and requires a separate proof. Nonetheless, the arguments are similar.

- Prop. B.3 tells us that as long as w almost surely terminates (V2), the measure of a n-dimensional parameter variable not having any instances in the support of w tends to zero as the dimension n tends to infinity. As $f_\ell^{(n)}$ is bijective (and hence invertible), the Multiple Step NP-iMCMC sampler almost surely satisfies the condition in the loop in Step 3.2 and hence almost surely terminates.

- Next, we identify the state distribution of Multiple Step NP-iMCMC. We say a n-dimensional state $(\boldsymbol{x}, \boldsymbol{v})$ is ***valid*** if

  (i) For all $\ell = 1, \ldots, L$, $\mathsf{instance}(\boldsymbol{x}_\ell) \cap \mathsf{Supp}(w) \neq \varnothing$ where $(\boldsymbol{x}_0, \boldsymbol{v}_0) := (\boldsymbol{x}, \boldsymbol{v})$ and $(\boldsymbol{x}_\ell, \boldsymbol{v}_\ell) := f_\ell^{(n)}(\boldsymbol{x}_{\ell-1}, \boldsymbol{v}_{\ell-1})$; and

  (ii) For all $\ell = 1, \ldots, L$, $\mathsf{instance}(\boldsymbol{y}_\ell) \cap \mathsf{Supp}(w) \neq \varnothing$ where $(\boldsymbol{y}_0, \boldsymbol{u}_0) := (\boldsymbol{x}, \boldsymbol{v})$ and $(\boldsymbol{y}_{L-\ell+1}, \boldsymbol{u}_{L-\ell+1}) := f_\ell^{(n)\,-1}(\boldsymbol{y}_{L-\ell}, \boldsymbol{u}_{L-\ell})$; and

  (iii) For all $k < n$, $\mathsf{take}_k(\boldsymbol{x}, \boldsymbol{v})$ is not a valid state.

  Then, we can define the state distribution and show that the state movement in Multiple Step NP-iMCMC is invariant against this distribution.

- Finally, we conclude by Lem. B.17 that the Multiple Step NP-iMCMC sampler is correct.

D.3.4. TRANSFORMING MULTIPLE STEP NP-IMCMC SAMPLER

Recall we discussed three techniques in App. C, each when applied to the Hybrid NP-iMCMC sampler, improve its flexibility and/or efficiency. We now see how these techniques can be applied to Multiple Step NP-iMCMC.

Listing 10. Code for Multiple Step NP-iMCMC

```
1    def MultistepNPiMCMC(t0):
2      k0 = dim(t0)                                           # initialisation step
3      x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4      v0 = auxkernel[k0](x0)                                 # stochastic step
5
6      # start of multiple step
7      n = k0
8      (x[0],v[0]) = (x0,v0)
9      for l in range(1,L+1):
10       (x[l],v[l]) = f[n][l](x[l-1],v[l-1])                 # deterministic step
11       while not intersect(instance(x[l]),support(w)):      # extend step
12         x[0] = x[0] + [(normal, coin)]*(indexX(n+1)-indexX(n))
13         v[0] = v[0] + [(normal, coin)]*(indexY(n+1)-indexY(n))
14         for i in range(1,l+1):
15           (y,u) = slice[n+1][i](x[i-1],v[i-1])
16           (x[i],v[i]) = (x[i]+y, v[i]+u)
17         n = n + 1
18     (x0,v0) = (x[0],v[0])
19     (x,v) = (x[L],v[L])
20     # end of multiple step
21
22     t = intersect(instance(x),support(w))[0]               # accept/reject step
23     k = dim(t)
24     return t if uniform < min{1,w(t)/w(t0) *
25                         pdfauxkernel[k](proj((x,v),k))/
26                           pdfauxkernel[k0](proj((x0,v0),k0)) *
27                         pdfpar[n](x)/pdfpar[n](x0) *
28                         pdfaux[n](v)/pdfaux[n](v0) *
29                         product([absdetjacf[n][l](x[l-1],v[l-1]) for l in range(1,L+1)])
30              else t0
```

D.3.5. STATE-DEPENDENT MULTIPLE STEP NP-IMCMC MIXTURE

This technique allows us to 'mix' Multiple Step NP-iMCMC samplers in such a way that the resulting sampler still preserves the posterior. Given a collection of Multiple Step NP-iMCMC samplers, indexed by $m \in \mathbb{E}^\alpha$ for some $\alpha \in \mathbb{N}$, the *State-dependent Multiple Step NP-iMCMC Mixture* sampler draws an indicator $m \in \mathbb{E}^\alpha$ from a probability measure $K_M(\boldsymbol{x}_0, \cdot)$ on $\mathbb{E}^\alpha$ where $K_M : \bigcup_{n \in \mathbb{N}} \mathbb{X}^{(n)} \rightsquigarrow \mathbb{E}^\alpha$ is a probability kernel and $\boldsymbol{x}_0$ is the parameter variable constructed from the current sample $\boldsymbol{t}_0$ in Step 1. A proposal $\boldsymbol{t}$ is then generated by running Steps 2 and 3 of the m-indexed Multiple Step NP-iMCMC sampler, and is accepted with a modified probability that includes the probability of picking m.

Pseudocode   Listing 11 gives the SPCF implementation of this sampler as the `MixtureMSNPiMCMC` function. (Terms specific to this technique are highlighted.) We assume the SPCF term `mixkernel` implements the mixture kernel K_M; `pdfmixkernel` implements the probability density function $\text{pdf} K_M$; and for each $m \in \mathbb{E}^\alpha$ and $n \in \mathbb{N}$, `auxkernel[n][m]` implements the auxiliary kernel and `pdfauxkernel[n][m]` implements its density; `f[n][l][m]` implements the endofunction `slice[n][l][m]` implements its slice and `absdetjacf[n][l][m]` implements the absolute value of the Jacobian determinant of the endofunction of the `m`-indexed Multiple Step NP-iMCMC sampler.

Correctness   `MixtureMSNPiMCMC` can be formulated as an instance of `MultistepNPiMCMC` with auxiliary kernel `mixauxkernel[n]` and its density `mixpdfauxkernel[n]` and `L` number of endofunctions `mixf[n][l]` (`l` ranges from `1` to `L`) for each dimension `n` with slice `mixslice[n][l]` and the absolute value of its Jacobian determinant `absdetjacmixf[n][l]`; parameter and auxiliary index maps `mixindexX` and `mixindexY` and projection `mixproj` given in Listing 12.

Listing 11. Pseudocode of the State-dependent Multiple Step NP-iMCMC Mixture algorithm

```
1   def MixtureMSNPiMCMC(t0):
2     k0 = dim(t0)                                    # initialisation step
3     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4     m = mixkernel(x0)                               # mixture step
5     v0 = auxkernel[k0][m](x0)                        # stochastic step
6
7     # start of multiple step
8     n = k0
9     (x[0],v[0]) = (x0,v0)
10    for l in range(1,L+1):
11      (x[l],v[l]) = f[n][l][m](x[l-1],v[l-1])              # deterministic step
12      while not intersect(instance(x[l]),support(w)):    # extend step
13        x[0] = x[0] + [(normal, coin)]*(indexX(n+1)-indexX(n))
14        v[0] = v[0] + [(normal, coin)]*(indexY(n+1)-indexY(n))
15        for i in range(1,l+1):
16          (y,u) = slice[n+1][i][m](x[i-1],v[i-1])
17          (x[i],v[i]) = (x[i]+y, v[i]+u)
18        n = n + 1
19      (x0,v0) = (x[0],v[0])
20      (x,v) = (x[L],v[L])
21    # end of multiple step
22
23    t = intersect(instance(x),support(w))[0]      # accept/reject step
24    k = dim(t)
25    return t if uniform < min{1, w(t)/w(t0) *
26                                pdfauxkernel[k][m](proj((x,v),k))/
27                                  pdfauxkernel[k0][m](proj((x0,v0),k0)) *
28                                pdfpar[n](x)/pdfpar[n](x0) *
29                                pdfaux[n](v)/pdfaux[n](v0) *
30                                pdfmixkernel(proj(x,k),m)/
31                                  pdfmixkernel(proj(x0,k0),m) *
32                                product([absdetjacf[n][m](x[l-1],v[l-1]) for l in range
33                                       (1,L+1)])}
              else t0
```

Listing 12. Pseudocode for the correctness of State-dependent Multiple Step NP-iMCMC Mixture

```
1   def mixauxkernel[n](x0):
2     m = mixkernel(x0)
3     v0 = auxkernel[n][m](x0)
4     return m + v0
5
6   def mixf[n][l](x0,mixv0):
7     m = mixv0[:a]
8     v0 = mixv0[a:]
9     (x,v) = f[n][l][m](x0,v0)
10    return (x,m + v)
11
12  def mixslice[n][l](x0,mixv0):
13    m = mixv0[:a]
14    v0 = mixv0[a:]
15    (y,u) = slice[n][l][m](x0,v0)
16    return (y,u)
17
18  def mixindexX(n): return indexX(n)
19  def mixindexY(n): return a + indexY(n)
20  def mixproj((x,v),k): return (x[:mixindexX(k)],v[:mixindexY(k)])
```

D.3.6. DIRECTION MULTIPLE STEP NP-IMCMC

This technique allows us to relax the assumption that the composition $f_L^{(n)} \circ \cdots \circ f_2^{(n)} \circ f_1^{(n)}$ is involutive. Assume for $\ell = 1, \ldots, L$, both sets $\{f_\ell^{(n)}\}_n$ and $\{f_\ell^{(n)^{-1}}\}_n$ satisfy the projection commutation property (V3), the **_Direction Multiple Step NP-iMCMC_** sampler randomly employ either $f_L^{(n)} \circ \cdots \circ f_2^{(n)} \circ f_1^{(n)}$ or $f_1^{(n)^{-1}} \circ f_2^{(n)^{-1}} \circ \cdots \circ f_L^{(n)^{-1}}$ to move around the n-dimensional state space and proposes a new sample.

Pseudocode Listing 13 gives the SPCF implementation of this sampler as `DirectionMSNPiMCMC` function. (Terms specific to this technique are highlighted.) We assume for each $n \in \mathbb{N}$ and $d \in 2$, the SPCF term `f[n][l][True]` implements the endofunction $f_\ell^{(n)}$ and `f[n][l][False]` implements the inverse $f_{L-\ell+1}^{(n)^{-1}}$; `slice[n][l][True]` implements the slice of $f_\ell^{(n)}$ and `slice[n][l][False]` implements the slice of $f_{L-\ell+1}^{(n)^{-1}}$; and `absdetjacf[n][l][True]` implements the absolute value of the Jacobian determinant of $f_\ell^{(n)}$ and `absdetjacf[n][l][False]` implements that of $f_{L-\ell+1}^{(n)}$.

Correctness `DirectionMSNPiMCMC` can be formulated as an instance of `MultistepNPiMCMC` with auxiliary kernel `dirauxkernel[n]` and its density `pdfdirauxkernel[n]` and `dirL` number of endofunctions `dirf[n][l]` (`l` ranges from `1` to `dirL`) for each dimension `n` with slice `dirslice[n][l]` and the absolute value of its Jacobian determinant `absdetjacf[n][l]` ; parameter and auxiliary index maps `dirindexX` and `dirindexY` and projection `dirproj` given in Listing 14. Note the `dirf[n]` function denotes the composition that flips the direction after applying the endofunctions $f_\ell^{(n)}$ for $\ell = 1, \ldots, L$ with an inverse the flips the direction and then apply the endofunctions $f_{L-\ell+1}^{(n)}$ for $\ell = 1, \ldots, L$.

D.3.7. PERSISTENT MULTIPLE STEP NP-IMCMC ALGORITHM

This technique gives us a method to construct irreversible Multiple Step NP-iMCMC samplers. The key is to persist the direction from a previous iteration.

The **_Persistent Multiple Step NP-iMCMC_** sampler keeps trace of a direction variable $d_0 \in 2$ (instead of sampling a fresh one at the start) and use it to determine the auxiliary kernel ($K_{\mathsf{T}}^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$ or $K_{\mathsf{F}}^{(n)} : \mathbb{X}^{(n)} \rightsquigarrow \mathbb{Y}^{(n)}$) and list of endofunctions ($f_L^{(n)} \circ \cdots \circ f_1^{(n)}$ or $f_1^{(n)^{-1}} \circ \cdots \circ f_L^{(n)^{-1}}$) employed. This direction variable is flipped strategically to make the resulting algorithm irreversible.

Pseudocode Listing 15 gives the SPCF implementation of this sampler as the function `PersistentMSNPiMCMC`. (Terms specific to this technique are highlighted.) In addition to the SPCF terms in `DirectionMSNPiMCMC`, the SPCF term `auxkernel[n][True]` implements the auxiliary kernel $K_{\mathsf{T}}^{(n)}$ and `pdfauxkernel[n][True]` implements its density $\mathrm{pdf}\,K^{(n)}{}_{\mathsf{T}}$ and `auxkernel[n][False]` implements the auxiliary kernel $K_{\mathsf{F}}^{(n)}$ and `pdfauxkernel[n][False]` implements its density $\mathrm{pdf}\,K^{(n)}{}_{\mathsf{F}}$. Note that `PersistentMSNPiMCMC` updates samples on the space $\mathbb{X}^{(n)} \times 2$, which can easily be marginalised to $\mathbb{X}^{(n)}$ by taking the first $\iota_{\mathbb{X}}(n)$ components.

Correctness Consider the `MultistepNPiMCMC` function with auxiliary kernel `perauxkernel[n]` and its density `pdfperauxkernel[n]` and `perL` number of endofunctions `perf[n][l]` (`l` ranges from `1` to `perL`) for each dimension `n` with slice `perslice[n][l]` and the absolute value of its Jacobian determinant `absdetjacperf[n][l]` ; parameter and auxiliary index maps `perindexX` and `perindexY` and projection `perproj` given in Listing 16.

The `MultistepNPiMCMC` function with the primitives indicated in Listing 16 is _almost_ equivalent to `PersistentMSNPiMCMC`, except `MultistepNPiMCMC` induces a transition kernel on $\mathbb{E} \times \mathbb{X}^{(n)}$ whereas `PersistentMSNPiMCMC` induces a transition kernel on $2 \times \mathbb{X}^{(n)}$; and when the proposal `t` is accepted, `MultistepNPiMCMC` returns `d` whereas `PersistentMSNPiMCMC` returns _not_ `d`.

By composing `MultistepNPiMCMC` with `flipdir` which flips the direction and marginalising the Markov chain

Listing 13. Pseudocode of the Direction Multiple Step NP-iMCMC algorithm

```
1   def DirectionMSNPiMCMC(t0):
2     d0 = coin                                    # direction step
3     k0 = dim(t0)                                 # initialisation step
4     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
5     v0 = auxkernel[k0](x0)                       # stochastic step
6     n = k0                                       # multiple step
7     (x[0],v[0]) = (x0,v0)
8     for l in range(1,L+1):
9       (x[l],v[l]) = f[n][l][d0](x[l-1],v[l-1])          # deterministic step
10      while not intersect(instance(x[l]),support(w)):    # extend step
11        x[0] = x[0] + [(normal, coin)]*(indexX(n+1)-indexX(n))
12        v[0] = v[0] + [(normal, coin)]*(indexY(n+1)-indexY(n))
13        for i in range(1,l+1):
14          (y,u) = slice[n+1][i][d0](x[i-1],v[i-1])
15          (x[i],v[i]) = (x[i]+y, v[i]+u)
16        n = n + 1
17    (x0,v0) = (x[0],v[0])
18    (x,v) = (x[L],v[L])
19    d = not d0                                          # flip direction (not used)
20    t = intersect(instance(x),support(w))[0]     # accept/reject step
21    k = dim(t)
22    return t if uniform < min{1, w(t)/w(t0) *
23                              pdfauxkernel[k](proj((x,v),k))/
24                                pdfauxkernel[k0](proj((x0,v0),k0)) *
25                              pdfpar[n](x)/pdfpar[n](x0) *
26                              pdfaux[n](v)/pdfaux[n](v0) *
27                              product([absdetjacf[n][l][d0](x[l-1],v[l-1]) for l in
28                                  range(1,L+1)])}
            else t0
```

Listing 14. Pseudocode for the correctness of Direction Multiple Step NP-iMCMC

```
1   def dirauxkernel[n](x0):
2     d0 = coin
3     v0 = auxkernel[n](x0)
4     return [(normal, d0)] + v0
5
6   def dirf[n][l](x,dirv):
7     d = dirv[0][1]
8     v = dirv[1:]
9     if l == dirL: return (x,[(dirv[0][0],not d)] + v)
10    else:
11      (x,v) = f[n][l][d](x,v)
12      return (x, [(dirv[0][0],d)] + v)
13
14  def dirslice[n][l](x,dirv):
15    d = dirv[0][1]
16    v = dirv[1:]
17    return slice[n][l][d](x,v)
18
19  dirL = L+1
20  def dirindexX(n): return indexX(n)
21  def dirindexY(n): return 1+indexY(n)
22  def dirproj((x,v),k): return (x[:dirindexX(k)], v[:dirindexY(k)])
```

Listing 15. Pseudocode of the Persistent Multiple Step NP-iMCMC algorithm

```
1    def PersistentMSNPiMCMC(t0,d0):
2      k0 = dim(t0)                                      # initialisation step
3      x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4      v0 = auxkernel[k0][d0](x0)                        # stochastic step
5      n = k0                                            # multiple step
6      (x[0],v[0]) = (x0,v0)
7      for l in range(1,L+1):
8        (x[l],v[l]) = f[n][l][d0](x[l-1],v[l-1])             # deterministic step
9        while not intersect(instance(x[l]),support(w)):     # extend step
10         x[0] = x[0] + [(normal, coin)]*(indexX(n+1)-indexX(n))
11         v[0] = v[0] + [(normal, coin)]*(indexY(n+1)-indexY(n))
12         for i in range(1,l+1):
13            (y,u) = slice[n+1][i][d0](x[i-1],v[i-1])
14            (x[i],v[i]) = (x[i]+y, v[i]+u)
15         n = n + 1
16     (x0,v0) = (x[0],v[0])
17     (x,v) = (x[L],v[L])
18     d = not d0                                        # flip direction
19     t = intersect(instance(x),support(w))[0]       # accept/reject step
20     k = dim(t)
21     return (t, not d) if uniform < min{1, w(t)/w(t0) *
22                        pdfauxkernel[k][d](proj((x,v),k))/
23                          pdfauxkernel[k0][d0](proj((x0,v0),k0)) *
24                        pdfpar[n](x)/pdfpar[n](x0) *
25                        pdfaux[n](v)/pdfaux[n](v0) *
26                        product([absdetjacf[n][l][d0](x[l-1],v[l-1]) for l in range(1,L
27                          +1)])}
                  else (t0, d)
```

Listing 16. Pseudocode for the correctness of Persistent Multiple Step NP-iMCMC

```
1    def perauxkernel[n](perx0)
2      d0 = perx0[0][1]; x0 = perx0[1:]
3      v0 = auxkernel[n][d0](x0)
4      return v0
5
6    def perf[n][l](perx,v)
7      d = perx[0][1]; x = perx[1:]
8      if l == perL: return ([(perx[0][0],not d)] + x, v)
9      else:
10       (x,v) = f[n][l][d](x,v)
11       return ([(perx[0][0],d)] + x, v)
12
13   def perslice[n][l](perx,v)
14     d = perx[0][1]; x = perx[1:]
15     return slice[n][l][d](x,v)
16
17   perL = L+1
18   def perindexX(n): return 1+indexX(n)
19   def perindexY(n): return indexY(n)
20   def perproj((x,v),k): return (x[:perindexX(k)],v[:perindexY(k)])
21
22   def flipdir(perx0,v0):
23     perx0[0][1] = not perx0[0][1]
24     return (perx0,v0)
```

generated by the composition from $\mathbb{E} \times \mathbb{X}^{(n)}$ to $\mathcal{2} \times \mathbb{X}^{(n)}$, we get `PersistentMSNPiMCMC` .

E. Examples of Nonparametric Involutive MCMC

In this section, we design *novel* nonparametric samplers using the Hybrid NP-iMCMC method described in App. B or the Multiple Step NP-iMCMC method described in App. D.

We assume the target density function w on the trace space $\mathbb{T}$ is tree representable and satisfies H1 and 2. Specifications of the auxiliary kernels and involutions are given for each sampler.

E.1. Nonparametric Metropolis-Hastings

As discussed in Sec. 2.1, the standard MH sampler can be seen as an instance of the iMCMC sampler with the proposal distribution q as the auxiliary kernel and a swap function as the involution.

Suppose a proposal kernel $q^{(n)} : \mathbb{E}^n \rightsquigarrow \mathbb{E}^n$ exists for each dimension $n \in \mathbb{N}$. Setting both $\iota_{\mathbb{X}}$ and $\iota_{\mathbb{Y}}$ to be identities (which means $\mathbb{X}^{(n)} = \mathbb{Y}^{(n)} = \mathbb{E}^n$ for all $n \in \mathbb{N}$), the Hybrid NP-iMCMC method (App. B.3) gives an nonparametric extension of the MH sampler.

Listing 17. Pseudocode of the NP-MH algorithm

```
1   def NPMH(t0):
2     k0 = dim(t0)                              # initialisation step
3     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4     v0 = q[k0](x0)                            # stochastic step
5     (x,v) = (v0,x0)                           # deterministic step
6     while not intersect(instance(x),support(w)): # extend step
7       x0 = x0 + [(normal, coin)]
8       v0 = v0 + [(normal, coin)]
9       (x,v) = (v0,x0)
10    t = intersect(instance(x),support(w))[0]     # accept/reject step
11    k = dim(t)
12    return t if uniform < min{1, w(t)/w(t0) * pdfq[k](proj((x,v),k))/pdfq[k0](proj((x0,v0)
          ,k0))}
13            else t0
```

The `NPMH` function in Listing 17 is a SPCF implementation of this sampler. It can seen as an instance of the `NPiMCMC` function with `auxkernel[n]` replaced by the proposal distribution `q[n]`, `pdfauxkernel[n]` replaced by the pdf of the proposal distribution `pdfq[n]`, `involution[n]` replaced by a swap function, and `indexX` and `indexY` replaced by identities, alongside a simplified acceptance ratio as $(\boldsymbol{x}, \boldsymbol{v}) = (\boldsymbol{v}_0, \boldsymbol{x}_0)$, $\varphi_{\mathbb{X}^{(n)}} = \varphi_{\mathbb{Y}^{(n)}}$, and

$$\frac{\varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x})}{\varphi_{\mathbb{X}^{(n)}}(\boldsymbol{x}_0)} \cdot \frac{\varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v})}{\varphi_{\mathbb{Y}^{(n)}}(\boldsymbol{v}_0)} \cdot |\det \nabla \Phi^{(n)}(\boldsymbol{x}_0, \boldsymbol{v}_0)| = 1.$$

E.2. Nonparametric Metropolis-Hastings with Persistence

Following the persistent technique described in App. C.3 for Hybrid NP-iMCMC, we can form a nonreversible variant of the NP-MH sampler described in App. E.1. We call the resulting algorithm the Nonparametric Metropolis-Hastings with Persistence (NP-MH-P) sampler.

Suppose a proposal kernel $q^{(n)} : \mathbb{E}^n \rightsquigarrow \mathbb{E}^n$ exists for each dimension $n \in \mathbb{N}$. Similar to NP-MH, both $\iota_{\mathbb{X}}$ and $\iota_{\mathbb{Y}}$ are set to be identities (which means $\mathbb{X}^{(n)} = \mathbb{Y}^{(n)} = \mathbb{E}^n$ for all $n \in \mathbb{N}$). Following (Turitsyn et al., 2011), given a parameter variable $\boldsymbol{x} \in \mathbb{E}^n$, we can partition the auxiliary space $\mathbb{E}^n$ into two sets $U_{\boldsymbol{x},+} := \{\boldsymbol{v} \in \mathbb{E}^n \mid \eta(\boldsymbol{v}) \geq \eta(\boldsymbol{x})\}$ and $U_{\boldsymbol{x},-} := \{\boldsymbol{v} \in \mathbb{E}^n \mid \eta(\boldsymbol{v}) < \boldsymbol{x}(\boldsymbol{x})\}$ where $\eta : \mathbb{E}^n \to \mathbb{R}$ is some measurable function; and form two kernels $K^{(n)}{}_+$ and $K^{(n)}{}_-$ from $q^{(n)}$ defined as

$$K^{(n)}{}_+(\boldsymbol{x}, V) := \frac{q^{(n)}(V \cap U_{\boldsymbol{x},+})}{q^{(n)}(U_{\boldsymbol{x},+})} \qquad \text{and} \qquad K^{(n)}{}_-(\boldsymbol{x}, V) := \frac{q^{(n)}(V \cap U_{\boldsymbol{x},-})}{q^{(n)}(U_{\boldsymbol{x},-})}. \tag{3}$$

Using the Persistent (Hybrid) NP-iMCMC sampler as described in App. C.3, a nonreversible variant of NP-MH can be formed.

Listing 18. Pseudocode of the NP-MH with Persistence algorithm

```
1   def NPMHwP(t0,d0):
2     k0 = dim(t0)                                      # initialisation step
3     x0 = [(e, coin) if Type(e) in R else (normal, e) for e in t0]
4     v0 = auxkernel[k0][d0](x0)                        # stochastic step
5     (x,v) = (v0,x0)                                   # deterministic step
6     n = k0                                            # extend step
7     while not intersect(instance(x),support(w)):
8       x0 = x0 + [(normal, coin)]
9       v0 = v0 + [(normal, coin)]
10      n = n + 1
11      (x,v) = (v0,x0)
12    d = not d0
13    t = intersect(instance(x),support(w))[0]          # accept/reject step
14    k = dim(t)
15    return (t, not d) if uniform < min{1, w(t)/w(t0) *
16                              pdfauxkernel[k][d](proj((x,v),k))/
17                                pdfauxkernel[k0][d0](proj((x0,v0),k0))}
18                  else (t0, d)
```

The `LiftedNPMH` function in Listing 18 is a SPCF implementation of this sampler. It can seen as an instance of the `PersistentNPiMCMC` function (Listing 8) with `auxkernel[n][True]` implementing $K^{(n)}_+$ and `auxkernel[n][False]` implementing $K^{(n)}_-$.

See how the direction `d0` (and hence the family of auxiliary kernels) is persisted if the proposal `t` is accepted.

E.3. Nonparametric Hamiltonian Monte Carlo

The Nonparametric Hamiltonian Monte Carlo (NP-HMC) is a MCMC sampler introduced by (Mak et al., 2021b) for probabilistic programming. Here we show that it is an instance of the Direction Multiple Step NP-iMCMC sampler (App. D.3.6).

Typically, the Hamiltonian Monte Carlo (HMC) sampler takes a target density on $\mathbb{R}^n$ and proposes a new state by simulating L leapfrog steps:
$$\boldsymbol{L} := (\phi^M_{\epsilon/2} \circ \phi^P_\epsilon \circ \phi^M_{\epsilon/2})^L$$
where $\phi^M_\epsilon(\boldsymbol{x}, \boldsymbol{v}) := (\boldsymbol{x}, \boldsymbol{v} - \epsilon \nabla U(\boldsymbol{x}))$ and $\phi^P_\epsilon(\boldsymbol{x}, \boldsymbol{v}) := (\boldsymbol{x} + \epsilon \boldsymbol{v}, \boldsymbol{v})$ are the momentum and position updates with step size ϵ respectively. Notice that that the momentum and position updates satisfy projection commutation property (H3), have inverses $(\phi^M_\epsilon)^{-1} = M \circ \phi^M_\epsilon \circ M$ and $(\phi^P_\epsilon)^{-1} = M \circ \phi^P_\epsilon \circ M$ where $M(\boldsymbol{x}, \boldsymbol{v}) := (\boldsymbol{x}, -\boldsymbol{v})$ and slices drop_{n-1} (for $\phi^M_{\epsilon/2}$, see App. D.2.1 for more details) and $(\boldsymbol{x}, \boldsymbol{v}) \mapsto (\boldsymbol{x}^n + \epsilon \boldsymbol{v}^n, \boldsymbol{v}^n)$ (for ϕ^P_ϵ) respectively. Moreover, the absolute value of the Jacobian determinant of both updates are $|\det \nabla \phi^M_\epsilon(\boldsymbol{x}, \boldsymbol{v})| = |\det \nabla \phi^P_{\epsilon/2}(\boldsymbol{x}, \boldsymbol{v})| = 1$.

Given a target density w on $\bigcup_{n \in \mathbb{N}} \mathbb{R}^n$, the HMC sampler can be extended using the Direction Multiple Step NP-iMCMC sampler. Given an input sample $\boldsymbol{t}_0 \in \mathbb{R}^{k_0}$, a k_0-dimensional initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ is formed where $\boldsymbol{x}_0 := \boldsymbol{t}_0$ and $\boldsymbol{v}_0$ drawn from $K^{(n)}(\boldsymbol{x}, \cdot) := \mathcal{N}_n$. A direction variable d_0 is drawn to determine whether the leapfrog steps $\boldsymbol{L}$ or its inverse $\boldsymbol{L}^{-1}$ is performed on the initial state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$, one update at a time, extending the dimension as required. Say the initial state is extended to a n-dimensional state $(\boldsymbol{x}_0, \boldsymbol{v}_0)$ and is traversed to the n-dimensional new state $(\boldsymbol{x}^*, \boldsymbol{v}^*)$ which has an instance $\boldsymbol{t}$ in the support of w. $\boldsymbol{t}$ is returned with probability

$$\min\left\{1; \frac{w(\boldsymbol{t}) \cdot \varphi_n(\boldsymbol{x}^*) \cdot \varphi_n(\boldsymbol{v}^*)}{w(\boldsymbol{t}_0) \cdot \varphi_n(\boldsymbol{x}_0) \cdot \varphi_n(\boldsymbol{v}_0)}\right\}.$$

Pseudocode of NP-HMC Listing 19 gives the SPCF implementations `leapfrog[n][m]` and `leapfrogslice[n][m]`, where `leapfrog[n][m][True]` and `leapfrogslice[n][m][True]` return the m-th endofunction and its slice in the composition $(\phi^M_{\epsilon/2} \circ \phi^P_\epsilon \circ \phi^M_{\epsilon/2})^L$ of $3L$ updates respectively; and similarly, `leapfrog[n][m][False]` and `leapfrogslice[n][m][False]` return the m-th endofunction and its slice in

Listing 19. Pseudocode of the leapfrog step and its slice in NP-HMC

```
1   def leapfrog[n][m][d0](x,v):
2     if d0:
3       if m % 3 == 0 or m % 3 == 2:
4         return (x, v-ep/2*grad(U)(x))       # half momentum update
5       else: return (x+ep*v, v)              # full position update
6     else:
7       if m % 3 == 0 or m % 3 == 2:
8         return (x, v+ep/2*grad(U)(x))       # inverse of half momentum update
9       else: return (x-ep*v, v)              # inverse of full position update
10
11  def leapfrogslice[n][m][d0](x,v):
12    if d0:
13      if m % 3 == 0 or m % 3 == 2:
14        return (x[-1], v[-1])               # slice of half momentum update
15      else: return (x[-1]+ep*v[-1], v[-1])  # slice of full position update
16    else:
17      if m % 3 == 0 or m % 3 == 2:
18        return (x[-1], v[-1])               # slice of inverse of half momentum
19      else: return (x[-1]-ep*v[-1], v[-1])  # slice of inverse of full position
```

Listing 20. Pseudocode for NP-HMC

```
1   def NPHMC(t0):
2     d0 = coin                                    # direction step
3     k0 = dim(t0)                                 # initialisation step
4     x0 = t0
5     v0 = [normal]*k0                             # stochastic step
6     # start of multiple step
7     n = k0
8     (x[0],v[0]) = (x0,v0)
9     for m in range(1,3L+1):
10      (x[m],v[m]) = leapfrog[n][m][d0](x[m-1],v[m-1]) # deterministic step
11      while not intersect(instance(x[m]),support(w)): # extend step
12        x[0] = x[0] + [normal]
13        v[0] = v[0] + [normal]
14        for i in range(1,m+1):
15          (y,u) = leapfrogslice[n][m][d0](x[m-1],v[m-1])
16          (x[i],v[i]) = (x[i]+y, v[i]+u)
17        n = n + 1
18    (x0,v0) = (x[0],v[0])
19    (x,v) = (x[3L],v[3L])
20    d = d0
21    # end of multiple step
22    t = intersect(instance(x),support(w))[0]     # accept/reject step
23    k = dim(t)
24    return t if uniform < min{1,w(t)/w(t0) * pdfnormal[n](x)/pdfnormal[n](x0) *
25                                     pdfnormal[n](v)/pdfnormal[n](v0)}
26         else t0
```

Listing 21. Pseudocode for the NP-HMC with Persistence algorithm

```
1    NPHMCwPersistent((x0,v0),d0) = PersistMom(CorruptMom((x0,v0),d0))
2
3    def HMCw(x,v): return w(x)
4
5    def CorruptMom((x0,v0),d0):
6      u = [normal(v0[i]*sqrt(1-alpha^2), alpha^2) for i in range(len(v0))]
7      return ((x0,u),d0)
8
9    def PersistMom((x0,v0),d0):
10     k0 = dim(x0)                              # initialisation step
11     # start of multiple step
12     n = k0
13     (x[0],v[0]) = (x0,v0)
14     for m in range(1,3L+1):
15       (x[m],v[m]) = leapfrog[n][m][d0](x[m-1],v[m-1])        # deterministic step
16       while not intersect(instance(x[m],v[m]),support(HMCw)):  # extend step
17         x[0] = x[0] + [normal]
18         v[0] = v[0] + [normal]
19         for i in range(1,m+1):
20            (y,u) = leapfrogslice[n+1][i][d0](x[i-1],v[i-1])
21            (x[i],v[i]) = (x[i]+y, v[i]+u)
22         n = n + 1
23     d = not d0                                # flip direction
24     # end of multiple step
25     (x,v) = intersect(instance(x[3L],v[3L]),support(HMCw))[0]  # accept/reject step
26     return ((x,v), not d) if uniform < min{1, HMCw(x,v)/HMCw(x0,v0) *
27                                   pdfnormal[n](x[3L])/pdfnormal[n](x[0]) *
28                                   pdfnormal[n](v[3L])/pdfnormal[n](v[0]) }
29             else ((x0,v0), d)
```

$$(\phi_{\epsilon/2}^{M}{}^{-1} \circ \phi_{\epsilon}^{P}{}^{-1} \circ \phi_{\epsilon/2}^{M}{}^{-1})^{L}.$$

Listing 20 gives the SPCF implementation `NPHMC` of the NP-HMC sampler as an instance of the Direction Multiple Step NP-iMCMC sampler. Importantly, the expensive `leapfrog[n][m]` function is called once for each `m` ranging from `1` to `3L` the lightweight `leapfrogslice` is called in any subsequent re-runs.

Correctness   Since both $\phi_{\epsilon/2}^{M}$ and ϕ_{ϵ}^{P} are bijective and satisfies the projection commutation property (H3), the correctness of NP-HMC is implied by the correctness of Direction Multiple Step NP-iMCMC sampler.

E.4. Nonparametric Hamiltonian Monte Carlo with Persistence

With the catalogue of techniques explored in App. D.3.4, novel irreversible variants of the NP-HMC algorithm can be formed. Here we focus on the NP-HMC with Persistence algorithm which can be seen as a nonparametric extension of the Generalised HMC algorithm (Horowitz, 1991).

E.4.1. GENERALISED HMC

Horowitz (1991) made two changes to the conventional HMC algorithm in order to generate an irreversible Markov chain on $\mathbb{R}^n \times \mathbb{R}^n$ and improve its performance.

1. A "corrupted" momentum is used to move round the state space.

2. The direction is "persisted" if the proposal is accepted; otherwise it is negated.

The resulting sampler is called the *Generalised HMC* algorithm as it is a generalisation of the typical HMC sampler.

As shown in (Neklyudov et al., 2020), the Generalised HMC algorithm can be presented as a composition of an iMCMC algorithm that "corrupts" the momentum and a Persistent iMCMC algorithm that uses Hamiltonian dynamics to find a new state with a persisting direction. We consider a similar approach in our construction of a nonparametric extension of Generalised HMC.

E.4.2. NP-HMC WITH PERSISTENCE

State Density Let the state $(\boldsymbol{x}, \boldsymbol{v}) \in \mathbb{R}^n \times \mathbb{R}^n$ has density $w'(\boldsymbol{x}, \boldsymbol{v}) := w(\boldsymbol{x})$ w.r.t. the normal distribution $\mathcal{N}_{2n}$. It is clear that this density w' is integrable (H1) and almost surely terminating (H2). By setting the parameter index map to $\iota_{\mathbb{X}}(n) := 2n$ and parameter space $\mathbb{X}^{(n)} := \mathbb{R}^n \times \mathbb{R}^n$, the state $(\boldsymbol{x}, \boldsymbol{v})$ of length $2n$ is a n-dimensional parameter variable. `HMCw` in Listing 21 is a SPCF implementation of w'.

Corrupt Momentum Given the current state $(\boldsymbol{x}_0, \boldsymbol{v}_0) \in \mathbb{R}^n \times \mathbb{R}^n$ with direction $d_0 \in 2$, a new momentum is drawn from the distribution $\mathcal{N}_n(\boldsymbol{v}_0 \sqrt{1 - \alpha^2}, \alpha^2)$ for a hyper-parameter $\alpha \in [0, 1)$.

This can be presented in the NP-iMCMC format with the auxiliary variable $\boldsymbol{u}$ sampled from $\mathcal{N}_n(\boldsymbol{v}_0 \sqrt{1 - \alpha^2}, \alpha^2)$ and the swap $(((\boldsymbol{x}_0, \boldsymbol{v}_0), d_0), \boldsymbol{u}) \mapsto (((\boldsymbol{x}_0, \boldsymbol{u}), d_0), \boldsymbol{v}_0)$ as the involution. Since the new state $(\boldsymbol{x}_0, \boldsymbol{u})$ always have an instance in the support of w', and the acceptance ratio is

$$\min \left\{ 1, \frac{w'(\boldsymbol{x}_0, \boldsymbol{u}) \cdot \varphi_{2n}(\boldsymbol{x}_0, \boldsymbol{u}) \cdot \mathsf{pdf}_2(d_0) \cdot \varphi_n(\boldsymbol{v}_0 \mid \boldsymbol{u}\sqrt{1 - \alpha^2}, \alpha^2)}{w'(\boldsymbol{x}_0, \boldsymbol{v}_0) \cdot \varphi_{2n}(\boldsymbol{x}_0, \boldsymbol{v}_0) \cdot \mathsf{pdf}_2(d_0) \cdot \varphi_n(\boldsymbol{u} \mid \boldsymbol{v}_0\sqrt{1 - \alpha^2}, \alpha^2)} \right\} = 1,$$

the extend step (Step 3) and the accept/reject step (Step 4) of the NP-iMCMC sampler can both be skipped. This results in a sampler that has the SPCF implementation `CorruptMom` in Listing 21.

Persist Momentum We consider the Persistent Multiple Step NP-iMCMC algorithm (App. D.3.7) with the target density w' as follows.

Given a k_0-dimensional parameter $(\boldsymbol{x}_0, \boldsymbol{v}_0) \in \mathbb{X}^{(n)} := \mathbb{R}^n \times \mathbb{R}^n$ and direction $d_0 \in 2$, a *dummy* auxiliary variable $\boldsymbol{u} \in \mathbb{Y}^{(n)} := \mathbb{R}^n$ is sampled from $K^{(n)}((\boldsymbol{x}_0, \boldsymbol{v}_0), \cdot) := \mathcal{N}_n$ to form an initial state $((\boldsymbol{x}_0, \boldsymbol{v}_0), \boldsymbol{u})$. Depending on the direction d_0, either $\left((\phi_{\epsilon/2}^M \times \mathrm{id}_{\mathbb{R}^n}) \circ (\phi_\epsilon^P \times \mathrm{id}_{\mathbb{R}^n}) \circ (\phi_{\epsilon/2}^M \times \mathrm{id}_{\mathbb{R}^n}) \right)^L$ or its inverse is performed on $((\boldsymbol{x}_0, \boldsymbol{v}_0), \boldsymbol{u})$, one update at a time, extending the dimension as required. Say the initial state is extended to a n-dimensional $((\boldsymbol{x}_0^*, \boldsymbol{v}_0^*), \boldsymbol{u}^*)$ and is traversed to the n-dimensional new state $((\boldsymbol{x}^*, \boldsymbol{v}^*), \boldsymbol{u}^*)$. Then, the instance $(\boldsymbol{x}, \boldsymbol{v}) \in \mathrm{Supp}(w')$ of the n-dimensional parameter $(\boldsymbol{x}^*, \boldsymbol{v}^*)$ is returned alongside the direction variable d_0 with probability

$$\min \left\{ 1; \frac{w'(\boldsymbol{x}, \boldsymbol{v}) \cdot \varphi_n(\boldsymbol{x}^*) \cdot \varphi_n(\boldsymbol{v}^*)}{w'(\boldsymbol{x}_0, \boldsymbol{v}_0) \cdot \varphi_n(\boldsymbol{x}_0^*) \cdot \varphi_n(\boldsymbol{v}_0^*)} \right\}.$$

Note that the auxiliary variable $\boldsymbol{u}$ has no effect on the sampler. Hence, Listing 21 gives a SPCF implementation `PersistMom` where the stochastic step (Step 2) is skipped.

NP-HMC with Persistence Composing the samplers which "corrupts" and persists the momentum gives us the ***NP-HMC with Persistence*** algorithm, which is an nonparametric extension of Generalised HMC. Listing 21 gives the SPCF implementation `NPHMCwPersistent` by composing `CorruptMom` and `PersistMom`.

E.5. Nonparametric Look Ahead Hamiltonian Monte Carlo

Last but not least, we extend the Look Ahead HMC algorithm (Sohl-Dickstein et al., 2014), which is equivalent to the Extra Chance Generalised HMC algorithm (Campos & Sanz-Serna, 2015).

E.5.1. LOOK AHEAD HMC

The Look Ahead HMC sampler modifies the Generalised HMC algorithm by performing *extra* leapfrog steps when the proposal state is rejected. This has the effect of increasing the acceptance rate for each proposal.

To see Look Ahead HMC as an instance of Persistent iMCMC, we consider the involution Φ on $\mathbb{R}^n \times \mathbb{R}^n \times [0, 1) \times 2$ given

Listing 22. Pseudocode for the NP Look Ahead HMC algorithm

```
1    NPLookAheadHMC((x0,v0),d0) = ExtraLeapfrog(CorruptMom((x0,v0),d0)))
2
3    def ExtraLeapfrog((x0,v0),d0):
4      k0 = dim(x0)                                    # initialisation step
5      u = uniform                                     # stochastic step
6      # start of multiple step
7      n = k0
8      m = 0
9      (x[m],v[m]) = (x0,v0)
10
11     stop = False
12     while not stop:
13       j = 1
14       M = j*3*L
15       # perform a set of leapfrog steps, i.e. to compute (x[i],v[i]) for i in range(m,M)
16       while m < M+1:
17         (x[m],v[m]) = leapfrog[n][m][d0](x[m-1],v[m-1])        # deterministic step
18         while not intersect(instance(x[m],v[m]),support(HMCw)):  # extend step
19           x[0] = x[0] + [normal]
20           v[0] = v[0] + [normal]
21           for i in range(1,m+1):
22             (y,u) = leapfrogslice[n+1][i][d0](x[i-1],v[i-1])
23             (x[i],v[i]) = (x[i]+y, v[i]+u)
24           n = n + 1
25         m = m + 1
26       (x,v) = intersect(instance(x[M],v[M]),support(HMCw))[0]
27       if u > min{1,HMCw(x,v)/HMCw(x0,v0) *
28                   pdfnormal[n](x[M])/pdfnormal[n](x[0]) *
29                   pdfnormal[n](v[M])/pdfnormal[n](v[0]) }:
30         if j <= J:
31           # perform an extra set of leapfrog steps
32           j = j + 1
33         else:
34           # no leapfrog steps is performed
35           (x,v) = (x0,v0)
36           stop = True
37           d = d0
38       else:
39         # enough leapfrog steps are performed
40         stop = True
41         d = not d0
42     # end of multiple step
43     return ((x,v), not d)
```
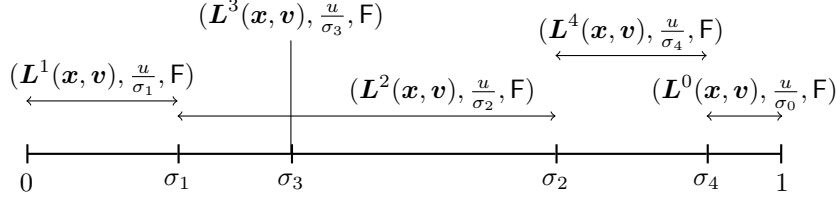
Figure 10. Result of $\Phi(\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{T})$ for varying $u \in [0, 1]$.

by

$$
\Phi(\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{T}) := \begin{cases} \left(\boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v}), \dfrac{u}{\sigma_j}, \mathsf{F}\right) & \text{if } \max\{\sigma_i \mid i < j\} \le u < \min\{1, \sigma_j\} \\ (\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{T}) & \text{if } \max\{\sigma_j \mid j \le J\} \le u \end{cases}
$$

$$
\Phi(\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{F}) := \begin{cases} \left(\boldsymbol{L}^{-j}(\boldsymbol{x}, \boldsymbol{v}), \dfrac{u}{\sigma_j'}, \mathsf{T}\right) & \text{if } \max\{\sigma_i' \mid i < j\} \le u < \min\{1, \sigma_j'\} \\ (\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{F}) & \text{if } \max\{\sigma_j' \mid j \le J\} \le u \end{cases}
$$

where

$$
\sigma_j := \frac{\zeta(\boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v}))}{\zeta(\boldsymbol{x}, \boldsymbol{v})}, \qquad \sigma_j' := \frac{\zeta(\boldsymbol{L}^{-j}(\boldsymbol{x}, \boldsymbol{v}))}{\zeta(\boldsymbol{x}, \boldsymbol{v})}, \qquad \boldsymbol{L}^{-j} := (\boldsymbol{L}^{-1})^j
$$

and ζ is the state density in HMC.

Note that in the involution, u determines how many sets (j) of leapfrog steps are to be performed. Say the direction is T. If the values of $\min\{1, \sigma_j\}$ for $j = 1, \ldots, J$ are marked on the unit interval $[0, 1]$, then the probability that $\boldsymbol{L}^j$ is performed can be represented by the distance between $\min\{1, \sigma_j\}$ and the highest of σ_i for $i < j$, if it is non-negative. App. E.5.1 gives an example of the result of $\Phi(\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{T})$ for varying $u \in [0, 1]$.

The Look Ahead HMC sampler can be formulated as a Persistent iMCMC sampler with the auxiliary kernel $K((\boldsymbol{x}, \boldsymbol{v}), \cdot) := \mathcal{U}[0, 1)$ and above involution Φ. Note that the sampler always accept the proposal since for $u \in [\max\{\sigma_i \mid i < j\}, \min\{1, \sigma_j\})$ with $j \in \{1, \ldots, J\}$, the acceptance ratio is

$$
\min\{1, \frac{\zeta(\boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v}))}{\zeta(\boldsymbol{x}, \boldsymbol{v})} \cdot |\det \nabla \Phi(\boldsymbol{x}, \boldsymbol{v}, u, \mathsf{T})|\} = \min\{1, \sigma_j \cdot |(\det \nabla \boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v})) \cdot \frac{1}{\sigma_j}|\} = 1
$$

and for $u \in [\max\{\sigma_j \mid j \le J\}, 1]$, the acceptance ratio is also 1. A similar argument can be made when the direction is F.

E.5.2. NP LOOK AHEAD HMC

Extra Leapfrog   Similar to the NP-HMC with Persistence, we consider the Persistent Multiple Step iMCMC algorithm (App. D.3.7) that applies a random number of leapfrog function (or its inverse) to the current state with the target density $w'(\boldsymbol{x}, \boldsymbol{v}) := w(\boldsymbol{x})$.

Given a k_0-dimensional parameter $(\boldsymbol{x}_0, \boldsymbol{v}_0) \in \mathbb{R}^n \times \mathbb{R}^n$ and direction $d_0 \in 2$, a random variable $u \in [0, 1)$ and a *dummy* auxiliary variable $\boldsymbol{u}_0 \in \mathbb{R}^n$ are sampled from the uniform distribution $\mathcal{U}(0, 1)$ and $K^{(n)}((\boldsymbol{x}_0, \boldsymbol{v}_0), \cdot) := \mathcal{N}_n$ respectively to form an initial state $((\boldsymbol{x}_0, \boldsymbol{v}_0), (u, \boldsymbol{u}_0))$.

If the direction d_0 is T and $u \in [\max\{\sigma_i \mid i < j\}, \min\{1, \sigma_j\})$ for some $j > 0$ where

$$
\sigma_j := \frac{\max\{w'(\boldsymbol{t}, \boldsymbol{u}) \mid \boldsymbol{t} \in \mathsf{instance}(\boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v}))\} \cdot \varphi_{2n}(\boldsymbol{L}^j(\boldsymbol{x}, \boldsymbol{v}))}{\max\{w'(\boldsymbol{t}, \boldsymbol{u}) \mid \boldsymbol{t} \in \mathsf{instance}((\boldsymbol{x}, \boldsymbol{v}))\} \cdot \varphi_{2n}(\boldsymbol{x}, \boldsymbol{v})},
$$

leapfrog steps $(\mathrm{id}_{\mathbb{R}^n \times \mathbb{R}^n} \times (\frac{1}{\sigma_j}) \times \mathrm{id}_{\mathbb{R}^n}) \circ (\boldsymbol{L}^j \times \mathrm{id}_{[0,1) \times \mathbb{R}^n})$ are performed on $((\boldsymbol{x}_0, \boldsymbol{v}_0), (u, \boldsymbol{u}_0))$, one update at a time, extending the dimension as required with a flipped direction F. Otherwise, $u \ge \max\{\sigma_j \mid j \le J\}$ and no leapfrog steps is performed; $((\boldsymbol{x}_0, \boldsymbol{v}_0), (u, \boldsymbol{u}_0))$ is returned with the direction T remains unchanged. The treatment when d_0 is F is similar.

Say the initial state with direction d_0 is extended to a n-dimensional $((\boldsymbol{x}_0^*, \boldsymbol{v}_0^*), (u, \boldsymbol{u}_0^*))$ and is traversed to the n-dimensional new state $((\boldsymbol{x}^*, \boldsymbol{v}^*), (u^*, \boldsymbol{u}^*))$ with direction d. The instance $(\boldsymbol{x}, \boldsymbol{v}) \in \mathrm{Supp}(w')$ of the n-dimensional parameter $(\boldsymbol{x}^*, \boldsymbol{v}^*)$ is returned alongside a flipped direction not d with probability

$$\min\left\{1; \frac{w'(\boldsymbol{x}, \boldsymbol{v}) \cdot \varphi_{2n}(\boldsymbol{x}^*, \boldsymbol{v}^*) \cdot \mathrm{pdf}_{\mathcal{U}(0,1)}(u^*) \cdot \varphi_n(\boldsymbol{u}^*)}{w'(\boldsymbol{x}_0, \boldsymbol{v}_0) \cdot \varphi_{2n}(\boldsymbol{x}_0^*, \boldsymbol{v}_0^*) \cdot \mathrm{pdf}_{\mathcal{U}(0,1)}(u) \cdot \varphi_n(\boldsymbol{u}_0^*)} \cdot \left|\frac{1}{\sigma_j}\right| \cdot |\det \nabla \boldsymbol{L}^j(\boldsymbol{x}_0, \boldsymbol{v}_0)|\right\} = 1$$

if $j > 0$. Otherwise ($j = 0$), the acceptance ratio is also 1.

Note that the auxiliary variable $\boldsymbol{u}_0$ has no effect on the sampler. Hence, Listing 22 gives a SPCF implementation `ExtraLeapfrog` where the sampling of the auxiliary variable $\boldsymbol{u}_0$ is skipped.

NP Look Ahead HMC Combining `ExtraLeapfrog` with `CorruptMom`, the `NPLookAheadHMC` function in Listing 22 implements the NP Look Ahead HMC sampler.