

# LazyHMC: Hamiltonian Monte Carlo Simulation for Lazy, Infinite Dimensional Probabilistic Programs

MARIA-NICOLETA CRĂCIUN, University of Oxford, UK

C.-H. LUKE ONG, Nanyang Technological University, Singapore

TOM SCHRIJVERS, KU Leuven, Belgium

SAM STATON, University of Oxford, UK

Hamiltonian Monte Carlo (HMC) is a successful generic inference method in probabilistic programming, but in its ordinary formulation it needs gradients and finite-dimensional parameter spaces. In Haskell, lazy evaluation lets probabilistic programs express stochastic processes and other non-parametric Bayesian models over implicit infinite-dimensional spaces. This paper develops new formulations of gradient-based HMC for this infinite-dimensional setting, via lazy evaluation. For automatic differentiation, we provide an analysis based on a new notion of “piecewise analytic under cylindrical analytic partition” (PACAP), to show that even if a program is infinite-dimensional and defined lazily, the gradient of the likelihood function is finitely supported. For the Monte Carlo method itself, we develop several HMC variants and a No-U-Turn Sampler that operate over the infinite-dimensional parameter space but are still productive because of lazy evaluation. Experiments cover Gaussian mixture clustering, random walks, and piecewise-constant regression with Poisson-process changepoints.

CCS Concepts: • **Mathematics of computing** → **Markov-chain Monte Carlo methods**; Nonparametric statistics; Stochastic processes; • **Software and its engineering** → **Functional languages**; • **Theory of computation** → *Denotational semantics*; *Program analysis*.

Additional Key Words and Phrases: probabilistic programming, Hamiltonian Monte Carlo, lazy evaluation, automatic differentiation, measure-theoretic semantics, non-parametric Bayesian inference, No-U-Turn Sampler

## ACM Reference Format:

Maria-Nicoleta Crăciun, C.-H. Luke Ong, Tom Schrijvers, and Sam Staton. 2026. LazyHMC: Hamiltonian Monte Carlo Simulation for Lazy, Infinite Dimensional Probabilistic Programs. *Proc. ACM Program. Lang.* 10, ICFP, Article 298 (August 2026), 24 pages. <https://doi.org/10.1145/3828696>

## 1 Introduction

Probabilistic programming (e.g. [3, 27]) is a method for Bayesian statistical modelling by writing programs. Recall that Bayes’ law specifies how to calculate the posterior probability, in terms of a prior probability and the likelihood of observations. Probabilistic programming is often treated as a declarative programming method: the statistician declares a prior and likelihood by writing a high level program, and a generic inference method (such as a Monte Carlo method) is then used to provide samples from the posterior.

Infinite or unbounded dimensions in a parameter space is often called ‘non-parametric’ statistics (for a broad overview, see e.g. [23]). For example, in a clustering problem we would like to group data points into clusters; in the non-parametric setting we want to explore the number of clusters,

---

Authors’ Contact Information: [Maria-Nicoleta Crăciun](mailto:maria.craciun@cs.ox.ac.uk), University of Oxford, Oxford, UK, [maria.craciun@cs.ox.ac.uk](mailto:maria.craciun@cs.ox.ac.uk); [C.-H. Luke Ong](mailto:luke.ong@ntu.edu.sg), Nanyang Technological University, Singapore, Singapore, [luke.ong@ntu.edu.sg](mailto:luke.ong@ntu.edu.sg); [Tom Schrijvers](mailto:tom.schrijvers@kuleuven.be), KU Leuven, Leuven, Belgium, [tom.schrijvers@kuleuven.be](mailto:tom.schrijvers@kuleuven.be); [Sam Staton](mailto:sam.staton@cs.ox.ac.uk), University of Oxford, Oxford, UK, [sam.staton@cs.ox.ac.uk](mailto:sam.staton@cs.ox.ac.uk).



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2475-1421/2026/8-ART298

<https://doi.org/10.1145/3828696>

and we do not want to fix in advance how many clusters there are. Similarly in many models based on a stochastic process, such as a random walk, the process on the face of it goes on forever, or with infinite resolution. Non-parametric statistics is often well suited to programming, because the programming notations can more clearly express the complex dynamics than informal statistical notations.

*The problem: generic inference over non-parametric models.* Although probabilistic programming provides an elegant declarative framework for specifying statistical models, generic inference over non-parametric models is notoriously difficult. Hamiltonian Monte Carlo (HMC) simulation is arguably the most successful generic inference method, and is widely considered responsible for the uptake of Bayesian statistics in practice. However, HMC fundamentally relies on gradients and finite-dimensional parameter spaces, and so does not directly apply to non-parametric models. We can identify three existing approaches to this challenge:

- *Truncation by hand:* Approximate the infinite dimensional structure by a finite dimensional one. For example, an infinite-dimensional Gaussian process is replaced by a high dimensional multivariate Gaussian. We can then employ well understood generic inference methods for finite dimensions, including HMC. This approach is most widely used in practice. However, the approach is not compositional: the truncation bounds must be carefully adjusted across the whole model.
- *Dynamic dimensions:* As a program runs, keep track of which dimensions are actually needed at any point, adding new dimensions as necessary. This avoids approximation. But the model is less purely declarative, as it explicitly tracks the dimension count and changes to dimensions. Nonetheless, this is a common approach in the probabilistic programming community [19, 20, 25, 30, 32, 33].
- *Lazy use of infinite dimensions:* The statistical model and inference take place in the context of an infinite dimensional parameter space. Of course, no generic inference method can explicitly track infinitely many dimensions. However, lazily, at the last minute, we notice that only a finite subset of dimensions are actually needed for the calculation, and we use that structure to provide samples from the posterior.

This lazy approach has been demonstrated [10] for a simple Metropolis-Hastings simulation, and [7] for discrete programs. However, these methods do not use gradient information, and so they do not benefit from the efficiency that makes HMC so effective in practice.

In summary, the existing approaches either sacrifice compositionality (truncation), declarativeness (dynamic dimensions), or efficiency (gradient-free lazy methods). What is missing is a way to bring the power of gradient-based HMC to the lazy, infinite-dimensional setting.

*Our solution: Lazy Hamiltonian Monte Carlo.* In this paper we demonstrate that HMC can be applied directly to the infinite dimensional parameter spaces where non-parametric statistical models naturally live, by evaluating lazily. The main technical contribution is to structure the Hamiltonian so that the acceptance ratio, ostensibly an infinite product over all dimensions, collapses to a finite product: the gradient has finite support, and the unvisited dimensions cancel. Our lazy HMC method operates over programs in the LazyPPL library (§2), which has an implicit infinite dimensional parameter space and allows explicit types of infinite dimensional stochastic processes.

*Lazy Hamiltonian Monte Carlo in a Nutshell.* At a high level, HMC simulation provides a Markov chain whose stationary distribution is the posterior distribution of the statistical model. It can be regarded as the combination of the following three ideas:

- (1) Gradient descent: the log-likelihood function  $\mathbb{R}^n \rightarrow \mathbb{R}$  or unnormalized density is a function of the parameters; here the dimension of the parameter space is  $n$ . We can optimize this by gradient descent.
- (2) Momentum: gradient descent alone provides a sequence of samples but this does not converge to the posterior and may not explore all modes. To deal with this, we regard a particle following a gradient trajectory and endow it with momentum, moreover, this momentum changes randomly over time. At each step, this Hamiltonian dynamics is typically simulated by a leapfrog integrator.
- (3) Metropolis correction: The discretization error from the leapfrog integrator is corrected for by randomly either accepting or rejecting each proposed sample, according to the Metropolis-Hastings acceptance ratio.

Extending each of these ideas to the infinite-dimensional lazy setting requires new contributions, which we summarize as follows:

- (1) **Infinite-dimensional automatic differentiation** (§3.1, §3.3). In the lazy situation, the log-likelihood function now has an infinite dimensional domain, e.g.  $\mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{R}$ . Although we can lazily maintain an infinite dimensional stream of parameters, we cannot hope to make a gradient step in an infinite dimensional direction. We introduce a smoothness condition, PACAP, guaranteeing that the gradient is non-zero in only a finite subset of the dimensions, and we show that it holds for a core calculus of programs. We find these gradients using automatic differentiation, adapting a recent idea based on sparse-map Nagata numbers to this infinite-dimensional setting.
- (2) **Lazy No-U-Turn Sampler** (§D.1). HMC performance is sensitive to the number of leapfrog steps, which is typically mitigated by switching to the No-U-Turn Sampler (NUTS). We develop lazy versions of this too.
- (3) **Lazy Metropolis-Hastings correction** (§4.5). The Metropolis-Hastings ratio naturally involves an infinite number of terms, one for each dimension. We carefully choose the momentum and structure the Hamiltonian so that all but a finite number of these cancel. Although this requires some care, there are various options, and we propose three different ways of doing this. In §5 we give a detailed worked example, piecewise-constant regression, illustrating how the leapfrog integrator and dimension changes interact.
- (4) **Experimental evaluation** (§6). We implement lazy HMC and lazy NUTS as an extension to the LazyPPL library and evaluate them on non-parametric models from the literature, including geometric distributions, random walks, and clustering. There is no standard benchmark suite for non-parametric statistics, but our experiments provide empirical evidence that the methods work as intended.

## 2 Overview of Lazy Probabilistic Programming

In this section we give an overview of lazy probabilistic programming, illustrating through examples how laziness enables non-parametric statistical models. We then identify where the existing gradient-free inference methods for this setting fall short, and preview how lazy HMC addresses these limitations.

We use the syntax of the LazyPPL library [10] for Haskell. We do not assume very much familiarity with Haskell, but the key idea is that within a `do` block for the probability monad `Prob`, we have, in effect, a domain specific language for probabilistic programming. Haskell's lists are not necessarily finite, and so random lists amount to random processes. Thus the key starting point of non-parametric statistics, that the dimension of the parameter space is unknown or unbounded, is mapped to the programming concept of laziness.

We consider three kinds of example: sequences and geometric distributions (§2.1); random walks (§2.2); and non-parametric clustering (§2.3).

## 2.1 Examples: IID Sequences and Geometric Distributions

We assume a built in primitive `uniform :: Prob RealNum` that produces a random real number uniformly between 0 and 1. We can then write a recursive program that produces an infinite stream of uniform random samples. These will be independent and identically distributed (IID).

```
1 iiduniform :: Prob [RealNum]
2 iiduniform = do { x ← uniform ; xs ← iiduniform ; return (x : xs) }
```

(For now, the reader can assume `RealNum=Double`, but in Section 3.3 we will generalize this for automatic differentiation.) In Haskell, lists can be infinite, and this recursive program describes a random infinite list. (Haskell uses lazy evaluation: values are computed only when needed, so only the elements actually accessed will be generated.) This amounts to what would be written in statistics as “ $x_n \sim U(0, 1)$  iid (for all  $n \in \mathbb{N}$ )”. More generally, we can define a function

```
1 iid p :: Prob a → Prob [a]
2 iid p = do { x ← p ; xs ← iid p ; return (x : xs) }
```

that transforms any probability distribution into an IID sequence, so that `iid uniform = iiduniform`.

This ‘IID sequence’ is idiomatic in statistics. For a first example, for any  $p \in (0, 1)$ , we can consider the Bernoulli distribution

```
1 bernoulli :: RealNum → Prob Bool
2 bernoulli p = do { x ← uniform ; return (x < p) }
```

which gives true or false with probability  $p$ . A sequence of IID Bernoulli trials can then be given by `iid (bernoulli p)`.

The *geometric distribution* with mean  $\frac{1}{p}$  is typically defined as the time of the first success in the random sequence. As a program:

Listing 1. Geometric distribution

```
1 geometric :: RealNum → Prob Int
2 geometric p = do { xs ← iid (bernoulli p) ; let (Just n) = findIndex id xs ;
3                 return (1+n) }
```

As an aside we recall that, as is well known, the geometric distribution can be written without an explicit infinite sequence, as follows:

```
1 geometricStrict p = do x ← bernoulli p
2                       if x then return 1
3                       else do { n ← geometricStrict p ; return (1 + n) }
```

The transformation from `geometric` to `geometricStrict` in effect manually propagates the evaluation of the lazy evaluation of the infinite stream `xs`. The number of random choices remains unknown and unbounded.

## 2.2 Examples: Random Walks

We next consider random walks, which are random sequences of positions. We will work with random walks where the steps are uniformly distributed, for which we first consider a more general uniform distribution:

```

1 uniformRange :: RealNum → RealNum → Prob RealNum
2 uniformRange a b = do { x ← uniform ; return (a + (b-a)*x) }

```

In statistics, this would be written as  $U(a, b)$ . We can then define a random walk by first sampling all the steps IID, and then combining them, together with a starting position in  $[0, 3]$ .

```

1 walk :: Prob [RealNum]
2 walk = do { start ← uniformRange 0 3 ; steps ← iid (uniformRange (-1) 1) ;
3           return (scanl (+) start steps) }

```

Here, recall that `scanl (+)` produces the cumulative sums, i.e., the list of intermediate results. An equivalent way of describing this distribution on sequences adds the steps at each point, as is well known:

```

1 walkFrom :: RealNum → Prob [RealNum]
2 walkFrom start = do { next ← uniformRange (start-1) (start+1) ;
3                     rest ← walkFrom next ; return (next : rest) }

```

Here we will consider a puzzle that also involves tracking the distance travelled (odometer):

```

1 walkOdo :: Prob ([RealNum],[RealNum])
2 walkOdo = do { start ← uniformRange 0 3 ; steps ← iid (uniformRange (-1) 1) ;
3             return (scanl (+) start steps , scanl (+) 0 (map abs steps)) }

```

Notice that this is a random pair of sequences, representing position and distance travelled respectively. But the pair itself is not independent, which is clear from the type.

*Observations and Measures.* We use the distribution `walkOdo` to phrase the puzzle from [19]. The walker will rest when either they reach home (position  $\leq 0$ ) or they have travelled further than `distLim`. When they first rest, their distance from home is roughly 1.1. What is the posterior distribution on starting positions?

Listing 2. Random walk model

```

1 walkModel :: RealNum → Meas RealNum
2 walkModel distLim = do
3   (xs,ds) ← sample walkOdo
4   let (Just(_,finalDistance)) = find (\(x,d) → x ≤ 0 || d ≥ distLim) (zip xs ds) in
5   scoreLog (normalLogPdf 1.1 0.1 finalDistance)
6   return (head xs)

```

In this example, we have included an observation by switching from the probability monad, `Prob`, to the measures monad, `Meas`. There is a coercion `sample :: Prob a → Meas a`, but the measures monad also allows `scoreLog :: RealNum → Meas ()`, which scores (or weights) by the likelihood of the observation. In this case, the rough distance of 1.1 from home is modelled by a normal distribution with standard deviation 0.1. Although `walkOdo` is infinite, `find` forces only a finite prefix: the odometer accumulates i.i.d. positive increments, so almost surely exceeds `distLim` after finitely many steps, where evaluation stops.

As explained in [10], the distinction between `Prob` and `Meas` is helpful because recursion in the `Prob` monad can be lazy, allowing us to build the random infinite sequences that are idiomatic in statistics, lazily ignoring any random elements that are not needed, but recursion in the `Meas` monad is not lazy, because we cannot ignore any observations. This is not really a limitation: an

observation must terminate to contribute a score, so observing an infinite computation is impossible in any PPL, while the laziness LazyPPL adds in the Prob layer is extra expressivity.

### 2.3 Example: Clustering with a Gaussian Mixture

Our final illustration is a clustering model, based on an example from [33, §3]. The number  $k$  of clusters is unknown, but the prior is that it is Poisson distributed.

```

1 clusters :: Prob [RealNum]
2 clusters = do m ← poisson 9
3           let k=m+1 in
4             mapM (\n → uniformRange (20*(n-1)/k) (20*n/k)) [1..k]

```

Here `mapM` is a Haskell library routine that recurses over the given list using the monadic computation. (We elide coercions between integers and real numbers.) Note that the number of points returned is not known nor bounded.

We next define a basic clustering model by treating the points  $xs \leftarrow \text{clusters}$  as the means of Gaussian distributions. This uses a Gaussian mixture density:  $p(y|xs) = \sum_{i=0}^{k-1} \frac{1}{k} f(y, xs!!i)$ , where  $k=(\text{length } xs)$  and  $f(y, x) = \frac{1}{\sqrt{2\pi*1.5^2}} \exp(-\frac{(x-y)^2}{2*1.5^2})$  is the normal density, for fixed standard deviation 1.5.

A standard clustering inference model scores a dataset according to this density function, providing the inferred centres of the clusters:

```

1 clustering :: [RealNum] → Meas [RealNum]
2 clustering dataset = do xs ← sample clusters
3                     mapM (\y → scoreLog (log (p(y|xs)))) dataset
4                     return xs

```

Above, the program phrase  $(\log (p(y|xs)))$  is pseudocode, the actual code takes some care over addition in the log domain, as is standard.

This clustering model is a very simple model based on the leading example in [33], to illustrate the idea. More general models would also infer the variance, and infer different ratios in different clusters. This example is already non-parametric, since the number of clusters is inferred. More advanced non-parametric clustering models include Dirichlet process models, which can still be expressed within this language.

### 2.4 Recap of the Probability and Measures Monads, and the Goal of Monte Carlo

Probability theory typically starts from some underlying probability space or seed space  $\Omega$ , which is equipped with a probability measure  $p$ . A random variable in a space  $a$  is a measurable function  $X : \Omega \rightarrow a$ . A random variable induces a probability measure on  $a$  itself, by pushing forward,  $X^*p$ ; this is the law of  $X$ . Often we are interested in achieving samples from the distribution  $X^*p$ , but the underlying space is still important and useful in Markov Chain Monte Carlo (MCMC) simulation.

**2.4.1 Normalization and MCMC.** Recall that a probability measure on  $\Omega$  is normalized, that is to say,  $p(\Omega) = 1$ . An unnormalized measure  $q$  has  $q(\Omega) \neq 1$ . Provided  $q(\Omega) \notin \{0, \infty\}$ , we can form a normalized, probability measure  $\frac{q(-)}{q(\Omega)}$ . The problem is that the normalization constant  $q(\Omega)$  is very difficult or impossible to calculate exactly in general. Nonetheless, sampling from the normalized form of  $q$  is very important, for example many Bayesian inference problems are of this form.

The starting point for MCMC simulation is to have an *unnormalized* measure  $q$  on  $\Omega$  (and hence an unnormalized pushforward measure on  $a$ ), and to provide a Markov chain that gives samples from the corresponding normalized probability distribution but without first calculating  $q(\Omega)$ .



**2.4.2 Score Functions and Unnormalized Densities.** A typical way to express an unnormalized distribution is as a function

$$l : \Omega \rightarrow [0, \infty] \quad (1)$$

on the underlying probability space. If  $\int l \, d\mathbf{p} = 1$ , then this is the density for a probability measure, and if not, it is merely a density for an unnormalized measure.

Many situations in Bayesian statistics refer to the likelihood  $f(d|x)$  of a datapoint  $d$  as a function of its parameters  $x$ . For fixed parameters, the function  $f(-|x)$  is a normalized density, but in Bayesian inversion the datapoint is fixed and the parameters vary; the function  $f(d|-)$  is an unnormalized density. If  $p$  (or  $X^*p$ ) is the prior belief, and  $l : \Omega \rightarrow [0, \infty]$  determines the likelihood of the data, then the Bayesian posterior is proportional to the unnormalized measure  $q = p_l$  induced by the density  $l$ , where  $\int k(\omega) q(d\omega) = \int k(\omega) l(\omega) p(d\omega)$  for all measurable  $k : \Omega \rightarrow [0, \infty]$ .

**2.4.3 LazyPPL Implementation.** The idea of LazyPPL is to fix a probability space  $\Omega$  with a measure-preserving isomorphism  $\Omega \cong \Omega \times \Omega$ . Thus the source of randomness can always be split in two. The probability monad is then implemented as  $\text{Prob } a = (\Omega \rightarrow a)$ , and the monadic sequencing works by splitting  $\Omega$ , and the correct recursive behaviour for infinite lists follows. A convenient example is  $\Omega = \mathbb{R}^{\mathbb{N}^*}$ , infinite rose trees. Thus the sample space is naturally infinite dimensional, but this is not a problem in practice because it is explored lazily.

The measure monad includes observations via scoring and unnormalized densities, such as the `scoreLog` of `walkModel` and `clustering`, or (1). This monad can be implemented as the writer monad transformer,  $\text{Meas } a = \Omega \rightarrow ([0, \infty], a)$ , providing both the result in  $a$  and the unnormalized density.

The LazyPPL library, then, provides a compositional language for building measures in  $\text{Prob } a$  and  $\text{Meas } a$ . It comes with basic Metropolis-Hastings MCMC simulators that convert type  $\text{Meas } a$  to a stream of samples.

## 2.5 Towards Gradient-Based Inference in the Lazy Setting

The lazy approach to probabilistic programming allows models with infinite-dimensional parameter spaces to be expressed compositionally, as illustrated above. The LazyPPL library includes a lazy Metropolis-Hastings sampler (`lazyLMH`) that operates over these infinite-dimensional spaces without gradient information. As discussed in the introduction, gradient-based methods like HMC can be applied to non-parametric models via truncation or dynamic dimension tracking, but these approaches sacrifice compositionality or declarativeness respectively. The aim of this paper is to show that gradient-based HMC can work directly in the lazy infinite-dimensional setting, preserving the compositional benefits.

This is not straightforward. Even defining what a ‘gradient’ means over an infinite-dimensional space requires care, and the Metropolis-Hastings correction, which involves a product over all dimensions, appears to involve infinitely many terms (in fact it collapses to a finite product, as we explain next).

Consider again the random walk model (Listing 2). The infinite sequence of steps produces an infinite-dimensional state space. As we demonstrate in Sec. 6 (Fig. 3b), our lazy HMC methods successfully sample from the posterior of this model, despite operating over a truly infinite-dimensional space, and gradient information allows for more efficient exploration than the gradient-free `lazyLMH` method.

The key insight enabling this is that, although the parameter space is infinite dimensional, the gradient of any program with a PACAP likelihood (Def. 1) is non-zero in only finitely many dimensions. This means we can perform HMC-style leapfrog steps over only the relevant dimensions, while the infinitely many remaining dimensions are handled lazily. Moreover, on unvisited coordinates the

leapfrog involution acts as a rotation preserving the standard-normal density, so the corresponding factors in the Metropolis-Hastings acceptance ratio cancel, leaving a finite product.

*Structure of the rest of the paper.* Sec. 3 develops the smoothness theory and automatic differentiation needed for gradients over infinite-dimensional seed spaces. Sec. 4 presents the lazy HMC algorithm, with three variants for handling the infinite-dimensional Metropolis-Hastings correction. App. D generalizes this to the No-U-Turn Sampler (lazy NUTS), which adaptively sets the trajectory length. Finally, Sec. 6 evaluates the methods on the models introduced in this section.

### 3 Smoothness and Automatic Differentiation with Infinite Dimensional Seed Spaces

Having motivated the need for gradient-based inference in the lazy setting (§2), we now address the first technical challenge: computing gradients over infinite-dimensional seed spaces. We introduce a smoothness condition, PACAP, and prove compositionally that it holds for a core calculus covering primitive recursion and corecursion; for programs involving unbounded search, such as the geometric distribution of §2, PACAP can also be established by an explicit partition of the seed space.

For Hamiltonian Monte Carlo (Sec. 4) we need the gradient of the unnormalized density  $l : \Omega \rightarrow [0, \infty]$  from (1), where the seed space  $\Omega$  is infinite dimensional. This leads to two tasks:

- We need to establish a useful notion of gradient for  $l$ . Although we want to consider infinite dimensional systems, in finite time we can only ever inspect finitely many dimensions; although we do not expect to bound the number of dimensions, we can still show that ‘locally’ a function only depends on finitely many dimensions (‘cylindrical’), and so locally (in a sense we make precise below) factors through a projection,  $\Omega \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$ .

This local approach is inspired by earlier analysis of the definable functions  $f : \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}$ . These are known to be locally dependent on finitely many dimensions, which amounts to topological continuity in that setting (e.g. [18]); topological continuity is too strong in the real-valued setting.

- If locally the function only uses finitely many dimensions, and yet the number of dimensions used overall is not bounded, there must be discontinuity or non-smooth points in  $\Omega$  where the number of relevant dimensions changes. These might play an important role. For example, even in the geometric distribution (§2.1), the sample from the geometric distribution is the number of dimensions used, and so it is very important that it is allowed to change. These non-smooth points could be problematic for gradient methods, but we show that they form a measure zero set: there is no chance of actually reaching a non-smooth point, in that the gradient exists almost everywhere.

This analysis is subtle: as [15] have shown, ‘almost surely smooth functions’ are not closed under composition. We follow that work by focusing on analytic functions and analytic partitions, but now extended to the infinite dimensional setting.

A final important high level point is that in the course of a finite computation, we will call  $l$  with various arguments, although these might not be known in advance. Locally, each of these function calls will be dependent on finitely many dimensions; there is no harm in over-approximating this to say that from the perspective of all the function calls in one run of the computation, the function  $l$  itself only depends on finitely many dimensions.

#### 3.1 Smoothness Guarantees

*3.1.1 Functions That Are PACAP: Piecewise Analytic Under Cylindrical Analytic Partition.* We introduce a notion of piecewise smoothness for functions on the seed space, so that we can consider derivatives. Because plain notions of piecewise-smooth are not compositional [15], we adopt a more refined notion that is stable under composition, following [13, 15]. Moreover, we guarantee



that locally the functions are cylindrical, in the sense that they are only dependent on finitely many dimensions. This makes implementation easier, and is computationally natural over an infinite dimensional structure – an output only depends on a finite part of its input.

Let  $\mathcal{A}$  be a countable set of ‘addresses’, indexing the coordinates of the seed space. We refer to a coordinate of a particular seed as a ‘site’. For example, we could let  $\mathcal{A} = \mathbb{N}^*$ , the set of finite lists of natural numbers, so that addresses correspond to nodes of a rose tree (a tree with countably infinite branching and infinite depth). We think of  $\mathbb{X} = \mathbb{R}^{\mathcal{A}}$  as a space of seeds, for example with the product normal distribution  $\mathcal{N}(0, 1)^{\otimes \mathcal{A}}$ . So if  $\mathcal{A} = \mathbb{N}^*$ , then  $\mathbb{X}$  comprises rose trees where each node contains a real number. Recall that a function is *analytic* if its Taylor series converges to the function in some neighbourhood of every point; analytic functions are smooth.

**DEFINITION 1.** A subset  $U$  of  $\mathbb{X}$  is an *analytic cylinder* if there exist

- a finite set  $B \subseteq \mathcal{A}$  of addresses, and open  $V \subseteq \mathbb{R}^B$ ;
- finite sequence of analytic functions  $g_1, \dots, g_n: V \rightarrow \mathbb{R}$ , such that

$$U = \{x \in \mathbb{X} \mid x|_B \in V \text{ \& } g_1(x|_B) \leq 0 \text{ \& } \dots \text{ \& } g_n(x|_B) \leq 0\}.$$

We call  $B$  a *support* of  $U$ . Note that strict inequalities  $g(x|_B) < 0$  can be absorbed into the open set  $V$  (replace  $V$  by  $V \cap g^{-1}((-\infty, 0))$ ), so the non-strict form  $g_i \leq 0$  suffices to express both strict and non-strict constraints (cf. [13, Lemma B.3]).

A subset of  $\mathbb{X}$  is *c-analytic* if it is a countable disjoint union of analytic cylinders.

A function  $f: U \rightarrow \mathbb{R}$  with c-analytic domain is defined to be *PACAP* if there exists

- a countable partition  $U = \bigsqcup_{i=1}^{\infty} U_i$  into analytic cylinders, with given supports  $B_i \subseteq \mathcal{A}$ ;
- for each  $i$ , an analytic function  $f_i: V \rightarrow \mathbb{R}$  where  $V \subseteq \mathbb{R}^{B_i}$  is open such that  $U_i \subseteq V \times \mathbb{R}^{\mathcal{A} \setminus B_i}$

and  $f(x) = f_i(x|_{B_i})$  when  $x \in U_i$ .

**Example 1.** Let  $\mathcal{A} = \mathbb{N}$ . A function  $c: \mathbb{R}^n \rightarrow \mathbb{R}$  is *PAP* (piecewise analytic on an analytic partition) [13, 15] if  $\mathbb{R}^n$  can be partitioned into finitely many analytic sets  $P_j = \{y \in V_j \mid h_1(y) \leq 0, \dots\}$  (with  $V_j \subseteq \mathbb{R}^n$  open,  $h_i: V_j \rightarrow \mathbb{R}$  analytic), together with analytic functions  $c_j: V_j \rightarrow \mathbb{R}$  such that  $c = c_j$  on  $P_j$ . Every PAP function  $\mathbb{R}^n \rightarrow \mathbb{R}$  induces a PACAP map  $\mathbb{X} \rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$  by composition.

The function  $f: \mathbb{X} \rightarrow \mathbb{R}$  given by  $f(x_1, \dots, x_n, \dots) = \sum_{i=1}^{\infty} 2^{-i} \arctan(x_i)$  is not PACAP, because its result depends on every input dimension; such functions lie outside our scope, since the likelihood is then not finitely computable.

**3.1.2 A Core Calculus for Establishing PACAP.** To establish PACAP-ness compositionally, we introduce a simple core lambda calculus with a real numbers type, function types, finite product types, and countable sum types.

$$\begin{aligned} e ::= x \mid \lambda x: \tau. e \mid e \mid (e, e) \mid \pi_j e \mid () \mid \text{inj}_i e \ (i \in I) \mid \text{case } e \text{ of } (\text{inj}_i x \Rightarrow e_i)_{i \in I} \\ \tau, v ::= \tau \rightarrow v \mid \tau \times v \mid 1 \mid \sum_{i \in I} \tau_i \mid \text{real} \end{aligned}$$

The typing rules are standard:

$$\begin{array}{c} \frac{\Gamma(x) = \tau}{\Gamma \vdash x: \tau} \quad \frac{\Gamma, x: \tau_1 \vdash e: \tau_2}{\Gamma \vdash \lambda x: \tau_1. e: \tau_1 \rightarrow \tau_2} \quad \frac{\Gamma \vdash e_1: \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2: \tau_1}{\Gamma \vdash e_1 e_2: \tau_2} \quad \frac{}{\Gamma \vdash () : 1} \quad \frac{\Gamma \vdash e: \tau_1 \times \tau_2}{\Gamma \vdash \pi_j e: \tau_j} \\[10pt] \frac{\Gamma \vdash e_1: \tau_1 \quad \Gamma \vdash e_2: \tau_2}{\Gamma \vdash (e_1, e_2): \tau_1 \times \tau_2} \quad \frac{\Gamma \vdash e: \tau_i}{\Gamma \vdash \text{inj}_i e: \sum_{j \in I} \tau_j} \quad \frac{\Gamma \vdash e: \sum_{j \in I} \tau_j \quad \{\Gamma, x: \tau_i \vdash e_i: \tau\}_{i \in I}}{\Gamma \vdash \text{case } e \text{ of } (\text{inj}_i x \Rightarrow e_i)_{i \in I}: \tau} \end{array}$$

We also include standard functions as constants, such as  $\exp$ ,  $\sin$ ,  $\log$ ,  $+$ ,  $\times$ , typed as appropriate,  $\Gamma \vdash c: \text{real} \times \dots \rightarrow \text{real}$ . We assume each constant denotes a total PAP function (Example 1); this covers all analytic functions as well as functions like  $\log$  and  $\text{abs}$  (extended to total functions, e.g.  $\log(x) = 0$  for  $x \leq 0$ ).

We interpret types as sets, in a standard way:

$$\llbracket \text{real} \rrbracket = \mathbb{R} \quad \llbracket 1 \rrbracket = 1 \quad \llbracket \tau_1 \times \tau_2 \rrbracket = \llbracket \tau_1 \rrbracket \times \llbracket \tau_2 \rrbracket \quad \llbracket \sum_{j \in J} \tau_j \rrbracket = \biguplus_{j \in J} \llbracket \tau_j \rrbracket \quad \llbracket \tau_1 \rightarrow \tau_2 \rrbracket = (\llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket).$$

We then interpret typing contexts  $\Gamma = (x_1:\tau_1, \dots, x_n:\tau_n)$  as sets of valuations, i.e. products  $\llbracket \Gamma \rrbracket = \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket$ , and every typed term  $\Gamma \vdash e : \tau$  induces a function  $\llbracket e \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket$  in a standard way (e.g. [24, 31]).

The countable sum types include every countable set  $I$  as a type, via  $\sum_{i \in I} 1$ . In particular we have a type of booleans  $\{\text{tt}, \text{ff}\}$ , a type of natural numbers  $\mathbb{N}$ , and a type of lists of natural numbers  $\mathbb{N}^*$ . We have a type of infinite streams,  $(\mathbb{N} \rightarrow \text{real})$ , and a type of lazy rose trees,  $(\mathbb{N}^* \rightarrow \text{real})$ .

Although countable sum types cannot be directly coded on a computer, countable case analysis gives a simple calculus that subsumes primitive recursion (over  $\mathbb{N}$ ) and corecursion (into streams  $\mathbb{N} \rightarrow \text{real}$ ), and many of our statistical models fit inside this calculus; in practice, Haskell's general recursion is used to describe the resulting infinite terms finitely. Programs involving unbounded search, such as `findIndex` and `find`, go beyond this calculus, but their PACAP property can be verified directly. See App. B for details.

**THEOREM 2.** *If  $f : (\mathcal{A} \rightarrow \text{real}) \rightarrow \text{real}$  is definable then it is PACAP.*

(Here  $\mathcal{A}$  is a type since it is countable, via  $\sum_{a \in \mathcal{A}} 1$ , and  $\llbracket \mathcal{A} \rightarrow \text{real} \rrbracket = \mathbb{R}^{\mathcal{A}}$ .)

**PROOF.** We prove this by infinitary logical relations; the proof is reminiscent of [2, 14]. For each type  $\tau$  and each c-analytic set  $U$  we define a relation  $\mathcal{P}_\tau^U \subseteq \llbracket \tau \rrbracket^U$ , by induction on structure of types:

$$\mathcal{P}_{\text{real}}^U = \{f : U \rightarrow \mathbb{R} \mid f \text{ is PACAP}\}$$

$$\mathcal{P}_1^U = \{() : U \rightarrow 1\} \quad (\text{all})$$

$$\mathcal{P}_{\tau_1 \times \tau_2}^U = \{(f_1, f_2) \mid f_1 \in \mathcal{P}_{\tau_1}^U \text{ \& } f_2 \in \mathcal{P}_{\tau_2}^U\}$$

$$\mathcal{P}_{\sum_{i \in I} \tau_i}^U = \{\biguplus_{i \in I} f_i \mid U = \biguplus_{i \in I} U_i \text{ \& } \forall i \in I. U_i \text{ c-analytic \& } f_i \in \mathcal{P}_{\tau_i}^{U_i}\}$$

$$\mathcal{P}_{\tau_1 \rightarrow \tau_2}^U = \{f : U \rightarrow \llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket \mid \forall \text{ c-analytic } U' \subseteq U, \forall g \in \mathcal{P}_{\tau_1}^{U'} . (\lambda u. f(u)(g(u))) \in \mathcal{P}_{\tau_2}^{U'}\}$$

We prove the following ‘fundamental lemma’, by induction on the structure of typing derivations:

$$\begin{aligned} &\text{If } x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau \text{ and given c-analytic } U \text{ and } f_1 \in \mathcal{P}_{\tau_1}^U, \dots, f_n \in \mathcal{P}_{\tau_n}^U \\ &\text{then we have } \lambda u. \llbracket e \rrbracket(f_1(u), \dots, f_n(u)) \in \mathcal{P}_\tau^U. \end{aligned} \quad (2)$$

The full proof, which uses auxiliary lemmas on support enlargement, intersection, restriction, gluing, and composition, is given in App. A. In the function-type clause the subsets  $U'$  range over c-analytic subsets of  $U$ ; no restriction of  $f$  is involved, since for  $u \in U' \subseteq U$  the value  $f(u) \in \llbracket \tau_1 \rrbracket \rightarrow \llbracket \tau_2 \rrbracket$  is just  $f$  evaluated pointwise. As  $g(u) \in \llbracket \tau_1 \rrbracket$  for  $g \in \mathcal{P}_{\tau_1}^{U'}$ , the map  $\lambda u. f(u)(g(u)) : U' \rightarrow \llbracket \tau_2 \rrbracket$  is well-typed, and the clause asks that it lie in  $\mathcal{P}_{\tau_2}^{U'}$ . This quantification over subsets is a Kripke refinement [2, 14], needed for the restriction and lambda cases.

The main result then follows from this lemma (2). For if  $\vdash e : (\mathcal{A} \rightarrow \text{real}) \rightarrow \text{real}$  then by the special case of the lemma with empty context, the constant function

$$\lambda u. \llbracket e \rrbracket \quad \text{is in} \quad \mathcal{P}_{(\mathcal{A} \rightarrow \text{real}) \rightarrow \text{real}}^{\mathbb{R}^{\mathcal{A}}}. \quad (3)$$

By expanding the definitions, the identity function  $\text{id}_{\mathbb{R}^{\mathcal{A}}}$  is in  $\mathcal{P}_{\mathcal{A} \rightarrow \text{real}}^{\mathbb{R}^{\mathcal{A}}}$ . Therefore, using (3) and expanding the definition of  $\mathcal{P}_{(\mathcal{A} \rightarrow \text{real}) \rightarrow \text{real}}^{\mathbb{R}^{\mathcal{A}}}$ , putting  $f_1$  as the identity function, we have  $\lambda u. \llbracket e \rrbracket(u) \in \mathcal{P}_{\mathbb{R}}^{\mathbb{R}^{\mathcal{A}}}$ , so  $\llbracket e \rrbracket$  is PACAP.  $\square$

### 3.2 Representing the Probability and Measures Monads in the Core Lambda Calculus

We can define the probability monad in our calculus, by

$$\begin{aligned} \text{Prob}(\tau) &= ((\mathbb{N}^* \rightarrow \text{real}) \rightarrow \tau) & \text{return}_{\text{Prob}} &= \lambda x. \lambda t. x : \tau \rightarrow \text{Prob}(\tau) \\ \text{bind}_{\text{Prob}} &= \lambda p. \lambda f. \lambda t. f(p(\lambda j. t(\iota_1 j))) (\lambda j. t(\iota_2 j)) : \text{Prob}(\tau) \rightarrow (\tau \rightarrow \text{Prob}(v)) \rightarrow \text{Prob}(v) \end{aligned}$$

where  $\iota_1, \iota_2 : \mathbb{N}^* \rightarrow \mathbb{N}^*$  are two maps with disjoint range. In this way, sequencing splits the random seed space.

As in LazyPPL [10], this monad does not immediately satisfy the identity and associativity laws for monads in the usual theory of equality for the calculus here. Rather, it satisfies the associativity laws in a suitable semantics where we identify random variables by their law (e.g. [10, 11, 13]). Since we do not need to fully formalize a denotational semantics of the language for what follows, we omit the details here.

We can also define the measure monad, which accumulates the unnormalized density, using the standard writer monad transformer:

$$\begin{aligned} \text{Meas}(\tau) &= (\text{Prob}(\text{real} \times \tau)) & \text{return}_{\text{Meas}} &= \lambda x. \text{return}_{\text{Prob}}(1, x) : \tau \rightarrow \text{Meas}(\tau) \\ \text{bind}_{\text{Meas}} &= \lambda p. \lambda f. \text{bind}_{\text{Prob}} p (\lambda q. \text{bind}_{\text{Prob}} (f(\pi_2 q)) (\lambda r. \text{return}_{\text{Prob}}((\pi_1 q) * (\pi_1 r), \pi_2 r))) \\ &: \text{Meas}(\tau) \rightarrow (\tau \rightarrow \text{Meas}(v)) \rightarrow \text{Meas}(v) \end{aligned}$$

The point is that PACAP is a sufficient condition for lazy HMC. The clustering, step regression, and polynomial regression models are all definable in the core calculus, and hence PACAP by Thm. 2; the geometric and random walk models involve unbounded search, but their PACAP property can be verified directly (App. B). Once we know that the likelihood is PACAP, it follows that at almost every seed the likelihood is locally analytic and depends on only finitely many coordinates, so its gradient is well-defined and finitely supported (the boundaries between analytic pieces form a measure-zero set). This is exactly the property exploited by the HMC methods of Sec. 4.

### 3.3 Implementation of Automatic Differentiation in Haskell

We automatically propagate the derivatives of functions by automatic differentiation, using the sparse map version of the Nagata numbers from [28]. This works even for infinite dimensional spaces, as we now explain.

Recall that a naive approach to dual number forward-mode AD (e.g. [14, 26]) would replace a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  by its dual number form,  $\tilde{f} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^2$  and the partial derivative  $\frac{\partial f(x_1 \dots x_n)}{\partial x_i}$  is given by  $\pi_2 \circ \tilde{f}(-, e_i) : \mathbb{R}^n \rightarrow \mathbb{R}$  where  $e_i$  is the one-hot vector. We can then find the gradient at a point by forming a vector of all the partial derivatives.

In the infinite dimensional setting, we consider functions of the form  $f : \mathbb{R}^{\mathcal{A}} \rightarrow \mathbb{R}$  (for  $\mathcal{A} = \mathbb{N}^*$  or  $\mathbb{N}$ , the set of all dimensions). From Theorem 2, it is plausible that we can take a step in the direction of the gradient at any given point: the function  $f$  will locally only depend on finitely many dimensions, and so the gradient at any point will only be non-zero at finitely many dimensions. However, a naive forward-mode AD (see e.g. [14, 26]) would work in terms of the dual number form  $f' : \mathbb{R}^{\mathcal{A}} \times \mathbb{R}^{\mathcal{A}} \rightarrow \mathbb{R}^2$ , from which we can again find the gradients by passing in one-hot vectors, but it is not obvious how to actually find out which of the infinitely-many dimensions have non-zero gradient, and so it is not clear how to take a gradient step.

To circumvent this problem, we follow [28] and switch from the ordinary dual numbers view to one in which a number is represented by a pair  $(x, m)$  where  $x \in \mathbb{R}$  and  $m : \mathcal{A} \rightarrow \mathbb{R}$  is a map. We write  $\mathcal{R} = \mathbb{R} \times \mathbb{R}^{\mathcal{A}}$  for these ‘Nagata numbers’. Now a function  $f : \mathbb{R}^{\mathcal{A}} \rightarrow \mathbb{R}$  is transformed by automatic differentiation into a function  $f' : \mathcal{R}^{\mathcal{A}} \rightarrow \mathcal{R}$ . For any  $\vec{x} \in \mathbb{R}^{\mathcal{A}}$ , we have  $f'(\lambda i. (x_i, e_i)) \in \mathcal{R}$ ,

for which the first component is  $f(\vec{x})$ , and the second component is a map  $\mathcal{A} \rightarrow \mathbb{R}$ , assigning the gradient at each address.

The source-to-source transformation from  $f : \mathbb{R}^{\mathcal{A}} \rightarrow \mathbb{R}$  to  $\tilde{f} : \mathcal{R}^{\mathcal{A}} \rightarrow \mathcal{R}$  is totally automatic in Haskell, simply by defining  $\mathcal{R}$  as an instance of the `Floating` type class.

The final trick is to note that, for all relevant  $(x, m) \in \mathcal{R}$ , the map  $m$  is sparse, i.e. zero except at finitely many points. For this reason we can use a sparse map datatype for this, and this sparse map datatype allows us to inspect its domain of definition. We can then discover the dimensions in  $\mathcal{A}$  that are relevant to the gradient of  $f$  at  $\vec{x} \in \mathbb{R}^{\mathcal{A}}$  by inspecting the domain of the sparse map arising from  $\tilde{f}(\lambda i. (x_i, e_i))$ . Even though this question requires infinitely many  $x_i$ 's and  $e_i$ 's, Haskell's lazy evaluation will ensure that only the relevant dimensions are inspected.

This sparse map approach with Nagata numbers was proposed by [28] to connect to the efficiencies of reverse mode automatic differentiation in a purely functional setting: rather than computing one directional derivative at a time (as in standard forward-mode AD), the full gradient is computed in a single pass, which in finite dimensions is the key advantage of reverse mode. In the infinite dimensional setting, this is not merely an efficiency gain but *essential*: standard forward-mode AD would require infinitely many passes (one per dimension) to recover the gradient, whereas the Nagata approach discovers the finitely many relevant dimensions automatically. We use this throughout the HMC methods of Sec. 4.

#### 4 Lazy HMC: Lazy Hamiltonian Monte Carlo Simulation

With automatic differentiation over infinite-dimensional structures in hand, via the PACAP analysis and laziness of the previous section, we turn to the Monte Carlo method, providing a correctness framework for HMC-based inference on rose-tree states arising from lazy probabilistic programs.

*The key insight.* The main difficulty in applying HMC to lazy probabilistic programs is that the state space is infinite dimensional. A trace contains an unbounded collection of random seeds, and so a naive HMC update would appear to require both an infinite dimensional gradient and an infinite product in the acceptance ratio used for correction.

The key observation is that neither computation is actually infinite for the programs considered in this paper. By the PACAP analysis of Sec. 3, at almost every trace the likelihood is locally analytic and depends only on finitely many *visited sites*. These visited sites are the coordinates whose values are actually read when evaluating the likelihood function on the given trace. Moreover, laziness ensures that only those sites needed along the leapfrog trajectory (defined in Sec. 4.3) are ever sampled. Together with a carefully modified HMC integrator, the infinite product in the acceptance ratio collapses to a *finite* product over sites visited during the trajectory, and the unvisited dimensions cancel exactly.

Thus each lazy HMC proposal is computed by a finite procedure:

- (1) sample an infinite momentum rose tree lazily, so that momentum values are generated only when demanded
- (2) evaluate the likelihood and gradient only at the finitely many sites needed by the trajectory
- (3) collect the finite set of sites visited during the trajectory
- (4) keep the HMC update on the visited sites that was computed during the trajectory
- (5) choose the update on all unvisited sites so their contributions cancel in the acceptance ratio.

The purpose of this section is to make this argument precise. We use the involutive MCMC framework as the correctness principle: if a proposal is generated by a measurable involution and the acceptance ratio uses the corresponding change-of-measure term, then the resulting Markov kernel preserves the target distribution. Our framework A (Sec. 4.4) does not prescribe a unique proposal involution. It only states sufficient conditions under which the infinite-dimensional acceptance

ratio reduces to a finite product. Our different lazy HMC variants instantiate the framework but satisfy the conditions in different ways:

- **lazyHMC1** (Sec. 4.5): uses a *rotation*-based position update. Because the standard normal is rotationally symmetric, the momentum density at unvisited sites is unchanged by the update, so those dimensions cancel immediately from the acceptance ratio. Only the addresses with non-zero gradient need to be tracked so no full visited-site bookkeeping is required. This is the simplest, most purely functional variant.
- **lazyHMC2** (Sec. 4.6): uses a different Hamiltonian splitting inspired by [4]. The position update is again rotation-based, so it shares lazyHMC1's cancellation. It can be preferable when the prior's geometry interacts poorly with lazyHMC1's step size parameterisation.
- **lazyHMC3** (Sec. 4.7): uses the *standard* HMC leapfrog integrator, whose position update is a *translation*. Translations do not preserve the standard normal, so unvisited sites no longer cancel automatically. To restore cancellation, the involution must apply a corrective measure-preserving map to unvisited sites, which requires knowing the full set of visited sites  $v(\mathbf{q}, \mathbf{p})$  (not just the gradient-support  $u$ ). This requires an extra  $O(L)$  pass to collect visited thunks (a larger constant, same asymptotics).
- **lazyNUTS** (App. D.1): adapts the No-U-Turn sampler to the lazy setting, eliminating the need to hand-tune the trajectory length  $L$ .

Sec. 6 shows that no single variant dominates across all models.

The step regression model of Sec. 5 is the running illustration throughout this section. There, the parameter space contains infinitely many changepoint positions and segment heights, yet any evaluation of the likelihood on a concrete dataset only touches the finitely many segments that overlap the observed  $x$ -range (Fig. 1). A reader who finds the formal development of this section dense is encouraged to read Sec. 5 in parallel.

The section is structured as follows. Sec. 4.1 to 4.3 recall background on Markov kernels and stationarity, the iMCMC framework, and HMC. Sec. 4.4 then presents Framework A as an instance of iMCMC on rose trees, and Sec. 4.5 to 4.7 derive HMC-inspired methods whose correctness follows from it. We also briefly discuss lazyNUTS (App. D.1).

#### 4.1 Markov Kernels and Stationarity

Let  $(\mathbb{X}, \Sigma_{\mathbb{X}}, \mu_{\mathbb{X}})$  be a  $\sigma$ -finite measure space and let  $l : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0}$  be the (unnormalized) density of the target distribution with respect to  $\mu_{\mathbb{X}}$ . The aim is to produce Markov chains  $(\mathbf{q}^{(i)})_{i=0,1,\dots}$  with  $\mathbf{q}^{(i)} \in \mathbb{X}$  which converge to the target distribution, i.e. the draws are eventually distributed according to the target. This is done by constructing a Markov kernel (also known as probability kernel)  $\kappa : \mathbb{X} \times \Sigma_{\mathbb{X}} \rightarrow [0, 1]$  which assigns to each current state a probability distribution over next states. One step of the Markov chain then corresponds to drawing the next state  $\mathbf{q}'$  from the kernel given the current state  $\mathbf{q}$  by sampling from  $\kappa(\mathbf{q}, \cdot)$ .

For the chain to converge, the kernel has to be stationary with respect to the target measure, i.e. it leaves the target measure invariant:  $\int_{\mathbb{X}} \kappa(\mathbf{q}, A) l(\mathbf{q}) \mu_{\mathbb{X}}(d\mathbf{q}) = \int_A l(\mathbf{q}) \mu_{\mathbb{X}}(d\mathbf{q})$ . Intuitively, if the current state is distributed according to the target, then after one step it still is.

Invariance alone does not guarantee convergence; that requires further assumptions such as irreducibility and aperiodicity, which are not typically established in full generality for probabilistic programming systems. Our correctness results therefore focus on proving invariance for the HMC-related kernels we define, and we evaluate mixing empirically in Sec. 6. Convergence rate is a separate question: it depends on the target's geometry, with no single theorem covering all targets [17]. Since laziness changes only *when* dimensions are evaluated, not the target geometry, standard HMC tuning guidance carries over.

## 4.2 Background on Involutive Markov Chain Monte Carlo

Involutive MCMC (iMCMC) [1, 8, 20, 22] is a general framework for constructing valid MCMC kernels with the use of an auxiliary random variable and an involutive function, which is a function whose inverse is itself. In particular, we will focus on the presentation of iMCMC introduced by [8]. Algorithms that can be generalized by iMCMC include MH, HMC, LMH, Reversible jump (see [22]).

Let  $(\mathbb{Y}, \Sigma_{\mathbb{Y}}, \mu_{\mathbb{Y}})$  be the  $\sigma$ -finite measure space of the auxiliary variable  $\mathbf{p}$ . For each  $\mathbf{q} \in \text{Supp}(l)$ , the auxiliary distribution has probability density  $p_{\mathbf{q}} : \mathbb{Y} \rightarrow \mathbb{R}_{\geq 0}$  with respect to  $\mu_{\mathbb{Y}}$ . Consider the state space  $(\mathbb{S}, \Sigma_{\mathbb{S}}, \mu_{\mathbb{S}})$  with  $\mathbb{S} = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{X} \times \mathbb{Y} \mid l(\mathbf{q})p_{\mathbf{q}}(\mathbf{p}) > 0\}$ ,  $\Sigma_{\mathbb{S}} = \{A \cap \mathbb{S} \mid A \in \Sigma_{\mathbb{X}} \otimes \Sigma_{\mathbb{Y}}\}$  and  $\mu_{\mathbb{S}} = \mu_{\mathbb{X}} \times \mu_{\mathbb{Y}}$ . The target distribution on the joint space has (unnormalized) density  $w(\mathbf{q}, \mathbf{p}) = l(\mathbf{q})p_{\mathbf{q}}(\mathbf{p})$ .

We also require an involution  $I : \mathbb{S} \rightarrow \mathbb{S}$ , i.e.,  $I^{-1} = I$ , with the property that the Radon-Nikodym derivative  $d(\mu_{\mathbb{S}} \circ I^{-1})/d\mu_{\mathbb{S}} : \mathbb{S} \rightarrow [0, \infty)$  exists. The involution structure simplifies the construction and verification of a reversible Metropolis–Hastings correction, which guarantees the correct stationary distribution (see [8] for details). One trivial example would be the identity involution, which always proposes the same value  $\mathbf{q}$ . Even though this kernel is stationary it will not produce a chain which converges to the correct distribution.

*iMCMC.* Each iteration starts with a value for  $\mathbf{q}$  and draws a value for  $\mathbf{p}$  from the auxiliary distribution  $p_{\mathbf{q}}(\mathbf{p})$  of  $\mathbf{p}$  given  $\mathbf{q}$ . The involution  $I$  is then applied on  $(\mathbf{q}, \mathbf{p})$  in order to get the proposed state  $(\mathbf{q}', \mathbf{p}')$ . For correctness, an accept/reject step is needed. The ratio compares how likely the proposed state is under the target distribution versus the current state, correcting for any asymmetry in the proposal distribution. With some probability, the proposed  $\mathbf{q}'$  is accepted and returned, otherwise it is rejected and  $\mathbf{q}$  is returned instead.

One iteration of iMCMC given  $\mathbf{q}, p_{\mathbf{q}}, l, \mu_{\mathbb{X}}, \mu_{\mathbb{Y}}, I$  does:

- (1) sample  $\mathbf{p} \sim p_{\mathbf{q}}$
- (2) get proposed state  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$
- (3) with probability  $\tilde{\alpha}(\mathbf{q}, \mathbf{p}) = \min\{1, \alpha(\mathbf{q}, \mathbf{p})\}$  accept and return  $\mathbf{q}'$ , otherwise return  $\mathbf{q}$ .

The acceptance ratio is:

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{w(\mathbf{q}', \mathbf{p}')}{w(\mathbf{q}, \mathbf{p})} \cdot \left( \frac{d(\mu_{\mathbb{S}} \circ I^{-1})}{d\mu_{\mathbb{S}}} \right)(\mathbf{q}, \mathbf{p}).$$

Let  $\kappa : \mathbb{X} \times \Sigma_{\mathbb{X}} \rightarrow [0, 1]$  (see App. C.1) be the probability kernel resulting from one iteration of iMCMC. Informally,  $\kappa(\mathbf{q}, A)$  gives the probability of starting with a position  $\mathbf{q}$  and after one iMCMC iteration returning a position in  $A$ .

**PROPOSITION 1.** (iMCMC stationarity [8]) *The kernel  $\kappa$  is stationary with respect to the target distribution, i.e. for any  $A \in \Sigma_{\mathbb{X}}$  we have  $\int_{\mathbb{X}} \kappa(\mathbf{q}, A) l(\mathbf{q}) \mu_{\mathbb{X}}(d\mathbf{q}) = \int_A l(\mathbf{q}) \mu_{\mathbb{X}}(d\mathbf{q})$ .*

## 4.3 Background on Hamiltonian Monte Carlo

HMC is a popular MCMC method that uses Hamiltonian dynamics on an augmented state space to propose the next sample. We give a brief description of HMC here, for more details see [5, 16, 21].

Since we are looking at the standard HMC in this subsection our target (unnormalized) density  $l$  is defined on the finite dimensional space  $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n}, \text{Leb}_n)$ . Suppose  $l$  is continuously differentiable.

*Physical intuition.* Imagine placing a frictionless particle on a surface whose height at position  $\mathbf{q}$  equals the *potential energy*  $U(\mathbf{q}) = -\log l(\mathbf{q})$ . The particle naturally slides toward low-potential regions, i.e. toward regions of high  $l$ , but without momentum it stops at a local mode. To explore the surface better, we periodically give the particle a random momentum kick and let it slide for a while before reading its new position. The momentum kick helps HMC explore the target distribution far more efficiently than random-walk Metropolis. The momentum is usually standard  $n$ -dimensional



normal. This gives the *kinetic energy*  $K : \mathbb{R}^n \rightarrow \mathbb{R}$  with  $K(\mathbf{p}) = -\log \varphi_n(\mathbf{p}) = \sum_{i=1}^n \mathbf{p}_i^2/2$ . The Hamiltonian of the system is given by  $H : \mathbb{R}^{2n} \rightarrow \mathbb{R}$  with  $H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$ . Hamiltonian equations describe how the position  $\mathbf{q}$  and momentum  $\mathbf{p}$  change over time:

$$\frac{d\mathbf{q}}{dt} = \frac{\partial H}{\partial \mathbf{p}} = \mathbf{p}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}} = -\nabla U(\mathbf{q}).$$

where  $\partial K/\partial \mathbf{p} = \mathbf{p}$  since  $K(\mathbf{p}) = \sum_i \mathbf{p}_i^2/2$ .

HMC makes use of the Hamiltonian motion of the particle to propose the next position. This is done by sampling the momentum  $\mathbf{p}$  from  $\varphi_n$  and then simulating the trajectory of the particle with initial position  $\mathbf{q}$ . After some time  $t$ , the particle will have momentum  $\mathbf{p}'$  and position  $\mathbf{q}'$ , which will be the proposed position.

The canonical distribution corresponding to  $H$  on the state space  $(\mathbb{R}^{2n}, \Sigma_{\mathbb{R}^{2n}}, \text{Leb}_{2n})$  is given by  $\zeta(\mathbf{q}, \mathbf{p}) = \exp(-H(\mathbf{q}, \mathbf{p})) = l(\mathbf{q})\varphi_n(\mathbf{p})$ . Hence, having draws  $\{(\mathbf{q}^i, \mathbf{p}^i)\}_{i=1, \dots}$  from  $\zeta$  and discarding the draws for the momentum  $\mathbf{p}^i$ , gives us position draws  $\{\mathbf{q}^i\}_{i=1, \dots}$  from our target distribution  $l$ .

*The leapfrog integrator (kick-drift-kick).* In almost all cases the Hamiltonian equations cannot be solved exactly, so we discretize time with the *leapfrog* integrator. One step of size  $\epsilon$  alternates updating the momentum and the position:

- (1) **Half-kick:** update momentum using the gradient of the potential for half a step,  $\phi_{\epsilon/2}^P(\mathbf{q}, \mathbf{p}) = (\mathbf{q}, \mathbf{p} - \frac{\epsilon}{2}\nabla U(\mathbf{q}))$ .
- (2) **Drift:** update position using the new momentum,  $\phi_{\epsilon}^Q(\mathbf{q}, \mathbf{p}) = (\mathbf{q} + \epsilon\mathbf{p}, \mathbf{p})$ .
- (3) **Half-kick:** update the momentum again for half a step,  $\phi_{\epsilon/2}^P(\mathbf{q}, \mathbf{p}) = (\mathbf{q}, \mathbf{p} - \frac{\epsilon}{2}\nabla U(\mathbf{q}))$ .

One full leapfrog step is  $\psi = \phi_{\epsilon/2}^P \circ \phi_{\epsilon}^Q \circ \phi_{\epsilon/2}^P$ , and  $L$  steps give  $\Psi^{(L)} = F \circ \psi^L$ , where  $F(\mathbf{q}, \mathbf{p}) = (\mathbf{q}, -\mathbf{p})$  is the momentum flip needed for reversibility. We omit step-size  $\epsilon$  and step count  $L$  from the notation when fixed.

The symmetric kick-drift-kick form of the leapfrog integrator is needed for reversibility: after negating the momentum, the same update can be run backwards. This would not be true if a simpler form like kick-drift was used. Reversibility and volume preservation are key properties of Hamiltonian dynamics. They are central to the proof that the HMC transition leaves the target distribution invariant. The leapfrog integrator is used because it preserves these properties:

**PROPOSITION 2.** [21] *The HMC leapfrog integrator is reversible, i.e.  $\Psi^{(L)} = (\Psi^{(L)})^{-1}$  and volume preserving, i.e.  $(\text{Leb}_{2n} \circ (\Psi^{(L)})^{-1})(A) = \text{Leb}_{2n}(A)$  for any measurable  $A \in \Sigma_{\mathbb{R}^{2n}}$ .*

*HMC as part of iMCMC.* We follow the structure of the iMCMC framework from Sec. 4.2 with the target distribution on the space  $(\mathbb{R}^n, \Sigma_{\mathbb{R}^n})$  given by the density  $l$  with respect to the Lebesgue  $\text{Leb}_n$  measure. The momentum variable is the auxiliary variable from the iMCMC framework with the  $n$ -dimensional normal  $\mathcal{N}_n$  as the auxiliary distribution, which has density  $\varphi_n$  with respect to the Lebesgue measure. The target density on the joint state space  $(\mathbb{R}^{2n}, \Sigma_{\mathbb{R}^{2n}})$  is given by the density  $\zeta$  with respect to  $\text{Leb}_{2n}$ . Our involution now is the leapfrog integrator  $\Psi^{(L)}$ .

One iteration of HMC given position  $\mathbf{q}$ ,  $\varphi_n$ ,  $l$ ,  $\Psi^{(L)}$  does:

- (1) sample  $\mathbf{p} \sim \varphi_n$
- (2) get proposed position and momentum  $(\mathbf{q}', \mathbf{p}') = \Psi^{(L)}(\mathbf{q}, \mathbf{p})$
- (3) with probability  $\min\{1, \alpha(\mathbf{q}, \mathbf{p})\}$  accept and return  $\mathbf{q}'$ , otherwise return  $\mathbf{q}$ .

Now we just have to make sure the correct acceptance ratio is used:

PROPOSITION 3. *The probability kernel resulting from HMC is stationary with respect to the target density  $l$ , if the acceptance ratio is given by:*

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{l(\mathbf{q}')\varphi_n(\mathbf{p}')}{l(\mathbf{q})\varphi_n(\mathbf{p})}.$$

PROOF. The proof follows by Prop. 1 and the fact that leapfrog preserves volume (contributing a Radon–Nikodym factor of 1). See App. C.2 for details.  $\square$

The acceptance ratio deduced here corresponds to the usual HMC acceptance ratio (see for example [21]). If we could simulate the trajectory of the particle exactly we would always accept the proposed position. That is because Hamiltonian dynamics keeps  $H$  invariant ( $\frac{dH}{dt} = 0$ ) which would give  $\alpha(\mathbf{q}, \mathbf{p}) = 1$ . So the acceptance ratio is needed to correct for the approximation error introduced by the numerical integrator.

*Example: Harmonic Oscillator.* Consider the case of the Harmonic Oscillator with Hamiltonian  $H(\mathbf{q}, \mathbf{p}) = \frac{\mathbf{q}^2}{2} + \frac{\mathbf{p}^2}{2}$ . The exact flow map is  $\phi_\theta(\mathbf{q}, \mathbf{p}) = (\cos(\theta)\mathbf{q} + \sin(\theta)\mathbf{p}, -\sin(\theta)\mathbf{q} + \cos(\theta)\mathbf{p})$ , so applying it will always preserve the Hamiltonian. We will make use of this in the following sections.

#### 4.4 Framework A: Hamiltonian Monte Carlo on Rose Trees in General

*Section overview.* A *rose tree* is an infinite map  $\mathbf{q} : \mathcal{A} \rightarrow \mathbb{R}$  from a countable address set  $\mathcal{A}$  (e.g.  $\mathcal{A} = \mathbb{N}^*$ , the set of finite lists of natural numbers) to the reals; it represents the full (lazy) parameter state of a probabilistic program. (In step regression, for example,  $\mathbf{q}$  stores the infinite stream of changepoints and the infinite collection of segment heights, indexed by a path in  $\mathcal{A}$ .)

The challenge is that the iMCMC acceptance ratio formally involves a product over all of  $\mathcal{A}$ , which is infinite. Framework A shows how that product collapses to a finite one, provided the involution  $I$  satisfies the three Conditions A1 to A3 below. The key output is Thm. 3. The three lazy HMC variants below are all instances of Framework A: Prop. 5 verifies Conditions A1 to A3 for lazyHMC1, and Sec. 4.6 and 4.7 give the corresponding arguments for lazyHMC2 and lazyHMC3. Readers mainly interested in the algorithmic construction may skip the measure-theoretic details and continue from Sec. 4.5.

*Site measures.* Let  $(X, \Sigma_X, \mu_X)$  and  $(Y, \Sigma_Y, \mu_Y)$  be two measure spaces with  $\mu_X$  and  $\mu_Y$  probability measures. For simplicity we take  $X$  and  $Y$  to be measurable subsets of  $\mathbb{R}$  with Borel  $\sigma$ -algebras  $\Sigma_X = \{A \cap X \mid A \in \Sigma_{\mathbb{R}}\}$  and  $\Sigma_Y = \{A \cap Y \mid A \in \Sigma_{\mathbb{R}}\}$ , where  $\Sigma_{\mathbb{R}}$  is the Borel  $\sigma$ -algebra of  $\mathbb{R}$ . Assume  $\mu_X$  and  $\mu_Y$  are absolutely continuous with respect to the Lebesgue measure and let  $\varphi_X$  and  $\varphi_Y$  denote the corresponding derivatives. Let  $(X^n, \Sigma_{X^n}, \mu_{X^n})$  be the product measure space with  $\Sigma_{X^n} = \Sigma_X^{\otimes n} = \{A \cap X^n \mid A \in \Sigma_{\mathbb{R}^n}\}$  and measure  $\mu_{X^n} = \mu_X^{\otimes n}$ . We similarly define  $(Y^n, \Sigma_{Y^n}, \mu_{Y^n})$ .

*Rose trees measures.* Let  $(\mathbb{X}, \Sigma_{\mathbb{X}})$  be the measurable space of the rose trees where each node contains a real number:  $\mathbb{X} = X^{\mathcal{A}}$ . The  $\sigma$ -algebra  $\Sigma_{\mathbb{X}}$  is the  $\sigma$ -algebra generated by cylindrical sets, i.e. sets of this form:  $\{\mathbf{q} \in \mathbb{X} \mid (\mathbf{q}_{i_1}, \dots, \mathbf{q}_{i_n}) \in A\}$  with  $i_1, \dots, i_n \in \mathcal{A}$  and  $A \in \Sigma_{X^n}$ . The measure  $\mu_{\mathbb{X}}$  on  $\mathbb{X}$  is the countably-infinite product measure of  $(X, \Sigma_X, \mu_X)$ , given by the Kolmogorov extension theorem. This means that  $\mu_{\mathbb{X}}(\{\mathbf{q} \in \mathbb{X} \mid (\mathbf{q}_{i_1}, \dots, \mathbf{q}_{i_n}) \in A\}) = \int_{X^n} [(\mathbf{q}_{i_1}, \dots, \mathbf{q}_{i_n}) \in A] \mu_X(d\mathbf{q}_{i_1}) \dots \mu_X(d\mathbf{q}_{i_n}) = \mu_{X^n}(A)$ , where  $\{\mathbf{q} \in \mathbb{X} \mid (\mathbf{q}_{i_1}, \dots, \mathbf{q}_{i_n}) \in A\}$  are the rose trees for which we restrict the values at nodes  $i_1, \dots, i_n$  to be in the measurable set  $A \in \Sigma_{X^n}$ .  $\mu_{\mathbb{X}}$  is a probability measure. Similarly for the momentum rose trees, we get the measure space  $(\mathbb{Y}, \Sigma_{\mathbb{Y}}, \mu_{\mathbb{Y}})$  with  $\mu_{\mathbb{Y}}$  as the countably-infinite product measure of  $(Y, \Sigma_Y, \mu_Y)$ .

*Target and auxiliary distributions.* Let  $l : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0}$  be the density of the unnormalized target distribution on  $\mathbb{X}$  defined by the program. We assume that  $l$  is PACAP (see Sec. 3). The target

unnormalized measure is then  $\mu_l(A) = \int [\mathbf{q} \in A] l(\mathbf{q}) \mu_{\mathbb{X}}(d\mathbf{q})$ , for any measurable  $A \in \Sigma_{\mathbb{X}}$ . Let the normalized target measure be  $\nu(\cdot) = \frac{\mu_l(\cdot)}{\mu_l(\mathbb{X})}$ . The auxiliary variable  $\mathbf{p}$  has density  $p_{\mathbf{q}}(\mathbf{p}) = 1$  w.r.t.  $\mu_{\mathbb{Y}}$ .

*State space.* Let  $\mathbb{X} \times \mathbb{Y}$  be the state space composed of position and momentum pairs, equipped with the  $\sigma$ -algebra  $\Sigma_{\mathbb{X}} \otimes \Sigma_{\mathbb{Y}}$  and measure  $\mu_{\mathbb{X}} \times \mu_{\mathbb{Y}}$ . We are now considering only the pairs for which  $l(\mathbf{q}) > 0$ , i.e. let  $\mathbb{S} = \text{Supp}(l) \times \mathbb{Y}$  with  $\sigma$ -algebra  $\Sigma_{\mathbb{S}} = \{A \cap \mathbb{S} \mid A \in \Sigma_{\mathbb{X}} \otimes \Sigma_{\mathbb{Y}}\}$  and measure  $\mu_{\mathbb{S}} = \mu_{\mathbb{X}} \times \mu_{\mathbb{Y}}$ , where  $\text{Supp}(l) = \{\mathbf{q} \in \mathbb{X} \mid l(\mathbf{q}) > 0\}$ . Since  $p_{\mathbf{q}}(\mathbf{p}) = 1$ , the target density on the joint space simplifies to  $w(\mathbf{q}, \mathbf{p}) = l(\mathbf{q})$ .

Let  $f : \mathbb{X} \rightarrow Z$  be the result function described by the given probabilistic program. The samples of  $\mathbb{X}$  can be pushed-forward to  $Z$ .

*Framework A.* One iteration given position  $\mathbf{q}$ ,  $\mu_{\mathbb{Y}}$ ,  $\mu_{\mathbb{X}}$ ,  $\mu_{\mathbb{Y}}$ ,  $l$ ,  $I$ ,  $v$  does:

- (1) sample  $\mathbf{p} \sim \mu_{\mathbb{Y}}$  lazily (since only finitely many components of the infinite-dimensional  $\mathbf{p}$  are accessed, we sample each component on-demand)
- (2) get proposed position and momentum  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$
- (3) with probability  $\min\{1, \alpha(\mathbf{q}, \mathbf{p})\}$  accept and return  $\mathbf{q}'$ , otherwise reject and return  $\mathbf{q}$ .

*Involutions conditions.* For the acceptance ratio to exist and be computable, we assume the following sufficient properties. These conditions are abstract here, but Prop. 5 shows that they hold for the lazy HMC construction below. Suppose there exist a measurable function  $v : \mathbb{S} \rightarrow \mathcal{P}_{fin}(\mathcal{A})$  and measure preserving (w.r.t. the Lebesgue measure) involutions  $I' : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  and  $I_T : \mathbb{R}^{2|T|} \rightarrow \mathbb{R}^{2|T|}$  for all  $T \in \mathcal{P}_{fin}(\mathcal{A})$  which satisfy the following conditions:

- (A1) (*Visited set is a cylinder.*) If  $(\mathbf{q}, \mathbf{p}) \in v^{-1}(T)$  then  $\{(\mathbf{x}, \mathbf{y}) \in \mathbb{S} \mid \mathbf{x}_T = \mathbf{q}_T, \mathbf{y}_T = \mathbf{p}_T\} \subset v^{-1}(T)$  (equivalently,  $v^{-1}(T)$  is a cylinder set with base  $C_T \in \Sigma_{\mathbb{R}^{2|T|}}$ ).
- (A2) (*Involutions decomposes along visited/unvisited split.*) For all  $T \in \mathcal{P}_{fin}(\mathcal{A})$  and  $(\mathbf{q}, \mathbf{p}) \in v^{-1}(T)$ :  $(\mathbf{q}'_T, \mathbf{p}'_T) = I_T(\mathbf{q}_T, \mathbf{p}_T)$  and  $(\mathbf{q}'_i, \mathbf{p}'_i) = I'(\mathbf{q}_i, \mathbf{p}_i)$  for all  $i \in \mathcal{A} \setminus T$ , where  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ .
- (A3) (*Unvisited sites preserve density.*)  $\varphi_X(\mathbf{q}')\varphi_Y(\mathbf{p}') = \varphi_X(\mathbf{q})\varphi_Y(\mathbf{p})$ , where  $(\mathbf{q}', \mathbf{p}') = I'(\mathbf{q}, \mathbf{p})$ .

*Step regression example.* In the step regression model of Sec. 5, the address set  $\mathcal{A}$  contains addresses  $a_j, b_j$  for  $j = 1, 2, \dots$ , where  $\mathbf{q}_{a_j}$  is the seed used to generate the  $j$ th segment height and  $\mathbf{q}_{b_j}$  is the seed used to generate the  $j$ th exponential increment of the changepoint process. For a given trace the visited set  $T$  contains the addresses of those sites used by the finitely many segments that overlap the observed  $x$ -range (see Fig. 1). Condition A1 then says that any other rose tree agreeing with  $\mathbf{q}$  on those visited sites visits the same set  $T$ : the values of the unvisited sites cannot change which segments overlap the data. Condition A2 says that the proposed state is obtained by running a finite-dimensional involution on the visited sites, while each unvisited site  $a_k$  or  $b_k$  (for  $k$  beyond the segments that cover the data) is transformed independently by  $I'$ . Finally, Condition A3 says that this independent transformation  $I'$  leaves the prior density on those unvisited sites unchanged, so their contributions cancel in the acceptance ratio.

*Acceptance ratio.* We can reduce the Radon-Nikodym derivative needed in the acceptance ratio to a ratio of finite products (the proof can be found in Prop. 12):

PROPOSITION 4. Let  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ , then  $\mu_{\mathbb{S}}$  - almost everywhere we have:

$$\left( \frac{d(\mu_{\mathbb{S}} \circ I^{-1})}{d\mu_{\mathbb{S}}} \right) (\mathbf{q}, \mathbf{p}) = \frac{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)}.$$

THEOREM 3. Consider the following acceptance ratio in Framework A:

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{w(\mathbf{q}', \mathbf{p}')}{w(\mathbf{q}, \mathbf{p})} \cdot \frac{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)} = \frac{l(\mathbf{q}')}{l(\mathbf{q})} \cdot \frac{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)}.$$

Then the resulting kernel is stationary with respect to the target measure  $\nu$ .

The proof follows by Prop. 4 and Prop. 1.

*Visited sites.* Since  $l$  is PACAP, there exists a countable partition  $\mathbb{X} = \cup_{i=1}^{\infty} \mathbb{X}_i$  into analytic cylinders with given finite supports  $T_i \subset \mathcal{A}$  with the following property. For each  $i$ , there exists an analytic function  $l_i : V \rightarrow \mathbb{R}$ , where  $V \subseteq \mathbb{R}^{T_i}$  is open such that  $\mathbb{X}_i \subseteq V \times \mathbb{R}^{\mathcal{A} \setminus T_i}$  and  $l(\mathbf{q}) = l_i(\mathbf{q}_{T_i})$  when  $\mathbf{q} \in \mathbb{X}_i$ . Let  $v_l : \mathbb{X} \rightarrow \mathcal{P}_{fin}(\mathcal{A})$  be the function given by  $v_l(\mathbf{q}) = T_i$  if  $\mathbf{q} \in \mathbb{X}_i$ . This is measurable because each analytic cylinder  $\mathbb{X}_i$  is a cylinder set in  $\mathbb{X}$ , hence in  $\Sigma_{\mathbb{X}}$ . We call  $v_l$  the *visited sites* function as, given a rose tree  $\mathbf{q}$ , it tells us it is enough to ‘visit’ the sites with addresses  $v_l(\mathbf{q})$  to compute  $l(\mathbf{q})$ , i.e. the values at the sites are needed for the computation. For step regression,  $v_l(\mathbf{q})$  is the set of changepoint and height addresses used by the segments that overlap the data (see Fig. 1).

*The gradient maps.* Moreover, we can define the measurable map  $G : \mathbb{X} \rightarrow \mathbb{R}^{\mathcal{A}}$  such that for  $\mathbf{q} \in \mathbb{X}_i \cap \text{Supp}(l)$  we have  $G(\mathbf{q}) = \mathbf{x}$  with  $x_{T_i} = -\nabla \log l_i(\mathbf{q}_{T_i})$  and  $x_j = 0$  for any  $j \in \mathcal{A} \setminus T_i$ . If  $\mathbf{q} \notin \text{Supp}(l)$  we let  $G(\mathbf{q})$  be the rose tree with all its nodes equal to 0. Let  $T$  be a finite subset of  $\mathcal{A}$  and consider the set  $v_l^{-1}(\mathcal{P}(T))$  of positions for which the visited sites form a subset of  $T$ . Then  $v_l^{-1}(\mathcal{P}(T))$  can be written as the union of the analytic cylinders  $\mathbb{X}_{n_i}$  with supports  $T_{n_i} \subseteq T$ :  $v_l^{-1}(\mathcal{P}(T)) = \cup_{i=1}^{\infty} \mathbb{X}_{n_i}$ . Hence,  $v_l^{-1}(\mathcal{P}(T)) = \{\mathbf{q} \mid \mathbf{q}_T \in D_T\}$  for some measurable set  $D_T \in \Sigma_{\mathcal{X}[T]}$ . Then there exists a measurable map  $G_T : D_T \rightarrow \mathbb{R}^{\mathcal{A}}$  such that for any  $\mathbf{q} \in v_l^{-1}(\mathcal{P}(T))$  we get  $G(\mathbf{q}) = G_T(\mathbf{q}_T)$ .

In the following sections we present different methods inspired by the HMC integrator.

#### 4.5 LazyHMC1: First Instantiation of Framework A

Let  $\mu_X, \mu_Y$  be the probability measures given by the standard normal distribution  $\mathcal{N}(0, 1)$ , so  $\varphi_X$  and  $\varphi_Y$  are the probability density function of  $\mathcal{N}(0, 1)$ . Fix  $\theta \in (0, \pi/2)$  and let  $\epsilon_q = \sin(\theta)$  and  $\epsilon_p = \tan(\theta/2)$ . Consider the following maps:

$$\begin{aligned} \psi &= \phi_{\epsilon_p}^P \circ \phi_{\epsilon_q}^Q \circ \phi_{\epsilon_p}^P, & F(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, -\mathbf{p}) \\ \phi_t^P(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, \mathbf{p} - t\mathbf{q} - tG(\mathbf{q})), & \phi_t^Q(\mathbf{q}, \mathbf{p}) &= (\mathbf{q} + t\mathbf{p}, \mathbf{p}). \end{aligned}$$

The added term  $t\mathbf{q}$  is to account for the gradient of the log prior probability for each site, as each site is sampled from the normal distribution  $\mathcal{N}(0, 1)$ . Let  $\tau : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  be the clockwise rotation by  $\theta$ :  $\tau(q, p) = (\cos(\theta)q + \sin(\theta)p, -\sin(\theta)q + \cos(\theta)p)$ . The constants  $\epsilon_q = \sin \theta$  and  $\epsilon_p = \tan(\theta/2)$  are chosen so that, on any site with  $G(\mathbf{q})_a = 0$ , the leapfrog step  $\psi$  acts on  $(q_a, p_a)$  as  $\tau$  (Prop. 8).

*Visited sites of trajectories.* Let  $v : \mathbb{S} \rightarrow \mathcal{P}_{fin}(\mathcal{A})$  be given by  $v(\mathbf{q}, \mathbf{p}) = \cup_{i=0}^L v_l(\mathbf{q}^{(i)})$ , where  $(\mathbf{q}^{(i)}, \mathbf{p}^{(i)}) = \psi^i(\mathbf{q}, \mathbf{p})$  (for  $i = 0, \dots, L$ ) are the states in the trajectory after  $L$  leapfrog steps starting from  $(\mathbf{q}, \mathbf{p})$ . The function  $v$  tells us which sites need to be visited in both  $\mathbf{q}$  and  $\mathbf{p}$  in order for the weight  $l(\mathbf{q}^{(L)})$  of the final position to be computed. Notice that  $v(\mathbf{q}, \mathbf{p}) = v(F \circ \psi^L(\mathbf{q}, \mathbf{p}))$  since  $L$  leapfrog steps from  $F \circ \psi^L(\mathbf{q}, \mathbf{p})$  give the same positions  $\mathbf{p}^{(i)}$  in the trajectory. For each position  $\mathbf{q}^{(i)}$  we only need to visit sites in  $v_l(\mathbf{q}^{(i)})$  to compute  $l(\mathbf{q}^{(i)})$  and hence  $G(\mathbf{q}^{(i)})$ .

*Constructing the involution on  $\mathbb{S}$ .* It is easy to check that the map  $F \circ \psi^L$  is an involution on  $\mathbb{X} \times \mathbb{Y}$ . However, even if  $(\mathbf{q}, \mathbf{p}) \in \mathbb{S}$ , applying  $F \circ \psi^L$  on it might give us a state  $(\mathbf{q}^{(L)}, \mathbf{p}^{(L)})$  in  $\mathbb{X} \times \mathbb{Y} \setminus \mathbb{S}$ , i.e.  $l(\mathbf{q}^{(L)}) = 0$ . Since we want to have an involution on  $\mathbb{S}$ , i.e. on the states which have non-zero probability, we can define  $I : \mathbb{S} \rightarrow \mathbb{S}$  the following way:

$$I(\mathbf{q}, \mathbf{p}) := \begin{cases} (F \circ \psi^L)(\mathbf{q}, \mathbf{p}) & \text{if } (F \circ \psi^L)(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \\ K(\mathbf{q}, \mathbf{p}) & \text{otherwise,} \end{cases}$$

where  $K(\mathbf{q}, \mathbf{p}) = (\mathbf{q}', \mathbf{p}')$  with  $(\mathbf{q}', \mathbf{p}')_{v(\mathbf{q}, \mathbf{p})} = (\mathbf{q}, \mathbf{p})_{v(\mathbf{q}, \mathbf{p})}$  and  $(\mathbf{q}', \mathbf{p}')_{\mathcal{A} \setminus v(\mathbf{q}, \mathbf{p})} = (F \circ \psi^L)(\mathbf{q}, \mathbf{p})_{\mathcal{A} \setminus v(\mathbf{q}, \mathbf{p})}$ . From the definition of  $v$  and  $v_i$  it follows that if  $l(\mathbf{q}) > 0$  then also  $l(\mathbf{q}') > 0$ , so  $K(\mathbf{q}, \mathbf{p})$  is in  $\mathbb{S}$ . Since  $v(K(\mathbf{q}, \mathbf{p})) = v(\mathbf{q}, \mathbf{p})$  we have that  $K$  is an involution on the measurable set  $\mathbb{S} \setminus (F \circ \psi^L)^{-1}(\mathbb{X} \times \mathbb{Y} \setminus \mathbb{S})$ .

**PROPOSITION 5.** *I satisfies the Conditions A1 to A3.*

**PROOF.** We give the intuition for each condition, the full argument can be found in App. C.4.

**Condition A3.** On any coordinate  $j$  outside  $v(\mathbf{q}, \mathbf{p})$  the gradient  $G(\mathbf{q}^{(i)})_j$  is 0 for every  $i$ , so the leapfrog step  $\psi$  acts on the  $j$ -th coordinate as the rotation  $\tau$ . After  $L$  steps and the momentum flip, the induced map on each unvisited site is  $I' = F' \circ \tau^L$ , which preserves both Lebesgue measure and the normal product density  $\varphi_X \times \varphi_Y$ . Hence the unvisited-site factors cancel in the acceptance ratio.

**Condition A1.** Let  $T$  be a finite subset of  $\mathcal{A}$  and let  $(\mathbf{q}, \mathbf{p}) \in \mathbb{S}$  with  $v(\mathbf{q}, \mathbf{p}) = T$ . By induction on  $i$ , the trajectory coordinates  $(\mathbf{q}_T^{(i)}, \mathbf{p}_T^{(i)})$  depend only on  $(\mathbf{q}_T, \mathbf{p}_T)$ , since the gradient at each step is supported in  $T = v(\mathbf{q}, \mathbf{p})$ . So  $v^{-1}(T)$  is a cylinder set determined by its  $T$ -coordinates.

**Condition A2.** The previous two observations combine to give a clean decomposition: on the visited coordinates  $I$  restricts to the finite-dimensional measure-preserving involution  $I_T = F_T \circ \psi_T^L$  on  $C_T$ , and on each unvisited coordinate  $I$  acts independently as  $I'$ .  $\square$

Let  $u(\mathbf{q}, \mathbf{p}) = \{j \in \mathcal{A} \mid \exists i \in \{0, \dots, L\}, G(\mathbf{q}^{(i)})_j \neq 0\}$  be the function that gives the sites  $j$  for which at least one position  $\mathbf{q}'$  in the trajectory has the property that  $G(\mathbf{q}')_j \neq 0$ . Notice that  $u(\mathbf{q}, \mathbf{p}) \subseteq v(\mathbf{q}, \mathbf{p})$  and  $u(\mathbf{q}, \mathbf{p}) = u(I(\mathbf{q}, \mathbf{p}))$ . For example, we might have  $j \in v(\mathbf{q}, \mathbf{p}) \setminus u(\mathbf{q}, \mathbf{p})$  if site  $j$  corresponds to a variable which has constant  $l$  for different branches, so  $j$  would be visited.

**Acceptance ratio.** By Thm. 3 using the following acceptance ratio

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{l(\mathbf{q}')}{l(\mathbf{q})} \cdot \frac{\prod_{j \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_j) \varphi_Y(\mathbf{p}'_j)}{\prod_{j \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_j) \varphi_Y(\mathbf{p}_j)},$$

where  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$  results in a kernel that is stationary with respect to the target distribution.

If  $j \in v(\mathbf{q}, \mathbf{p}) \setminus u(\mathbf{q}, \mathbf{p})$ , then  $G(\mathbf{q}^{(i)})_j = 0$  for all  $i \in \{0, \dots, L\}$  so  $(\mathbf{q}'_j, \mathbf{p}'_j) = I'(\mathbf{q}_j, \mathbf{p}_j)$ . This means that  $(\varphi_X \times \varphi_Y)(I'(\mathbf{q}_j, \mathbf{p}_j)) = (\varphi_X \times \varphi_Y)(\mathbf{q}_j, \mathbf{p}_j)$ . Hence, the acceptance ratio can be simplified (it is enough to know  $u$  instead of  $v$ ):  $\alpha(\mathbf{q}, \mathbf{p}) = \frac{l(\mathbf{q}')}{l(\mathbf{q})} \cdot \frac{\prod_{j \in u(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_j) \varphi_Y(\mathbf{p}'_j)}{\prod_{j \in u(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_j) \varphi_Y(\mathbf{p}_j)}$ .

**Implementation details.** As discussed in Sec. 3.3 we are able to get the sites on which the gradient of  $\log l$  is not zero and the values of the gradient for those sites. Therefore, for any position  $\mathbf{q}$  we can get the set  $u_l(\mathbf{q})$  which contains the sites for which the gradient of  $\log l(\mathbf{q})$  is not zero. So we can construct  $u(\mathbf{q}, \mathbf{p}) = \cup_{i \in \{0, \dots, L\}} u_l(\mathbf{q}^{(i)})$ . Therefore, there is no need to know  $v$  for this method.

**Note on  $K$ .** If we had proposed a state that is not in  $\mathbb{S}$ , the acceptance ratio would have been 0 so the proposed state would be rejected and the state  $(\mathbf{q}, \mathbf{p})$  would be returned instead. This would be the same as proposing  $(\mathbf{q}, \mathbf{p})$  from the start. Hence, intuitively, we would like  $K$  to be the identity, which is what we do in the implementation. This also makes  $I$  an involution that satisfies the required properties, but we chose the different  $K$  to make the presentation easier to follow.

**Note on  $I$ .** An alternative way of defining  $I$  is to let it be the identity for states  $(\mathbf{q}, \mathbf{p})$  for which any of  $F \circ \psi^i(\mathbf{q}, \mathbf{p})$  is not in  $\mathbb{S}$  for  $i = 1, \dots, L$ . This would correspond to rejecting if along the trajectory we encounter a position that is not in the support of  $l$ .

**Note on the choice of  $I'$ .** One can notice that we could have chosen any  $I'$  that is a Leb<sub>2</sub> preserving involution with  $(\varphi_X \times \varphi_Y)(\mathbf{q}, \mathbf{p}) = (\varphi_X \times \varphi_Y)(I'(\mathbf{q}, \mathbf{p}))$  (see Sec. 4.7) to use on the sites which are not visited. However,  $I'$  might then disagree with  $F \circ \psi^L$  on the sites not visited: we may have

$(F \circ \psi^L)(\mathbf{q}, \mathbf{p})_j \neq I'(\mathbf{q}_j, \mathbf{p}_j)$  for some  $j \in \mathcal{A} \setminus v(\mathbf{q}, \mathbf{p})$ . This would mean that we need to know  $v(\mathbf{q}, \mathbf{p})$  (not only  $u(\mathbf{q}, \mathbf{p})$ ) to apply  $I'$  on these sites. This is the case with the method in Sec. 4.7.

#### 4.6 LazyHMC2: Framework A with a Different Hamiltonian Splitting

We are now considering a different  $\psi$ , inspired by the HMC integrator resulting from a different splitting of the Hamiltonian [4]. Fix  $\theta > 0$ . Then  $\psi = \phi_{\theta/2}^P \circ \phi_{\theta}^{Q,P} \circ \phi_{\theta/2}^P$  with:

$$\begin{aligned}\phi_t^P(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, \mathbf{p} - tG(\mathbf{q})) \\ \phi_t^{Q,P}(\mathbf{q}, \mathbf{p}) &= (\cos(t)\mathbf{q} + \sin(t)\mathbf{p}, -\sin(t)\mathbf{q} + \cos(t)\mathbf{p})\end{aligned}$$

The rest of the setup is the same as in Sec. 4.5. The key difference from lazyHMC1 is that the position–momentum update step  $\phi^{Q,P}$  is a *joint rotation* rather than a pure position drift. Notice that if  $j \notin v(\mathbf{q}, \mathbf{p})$  then  $\psi(\mathbf{q}, \mathbf{p})_j = \tau(\mathbf{q}_j, \mathbf{p}_j)$ , the same rotation  $\tau$  as in Sec. 4.5.

PROPOSITION 6. *I (defined using the new  $\psi$ ) satisfies Conditions A1 to A3.*

PROOF. Both  $\phi_t^P, \phi_t^{Q,P}$  act site-wise: each site's update uses only that site's own  $(\mathbf{q}_j, \mathbf{p}_j)$ , connected to other sites solely through the gradient  $G$ , which is supported on  $T = v(\mathbf{q}, \mathbf{p})$ . This site-wise structure is all that Condition A1 needs, so the cylinder argument of Prop. 5 applies the same here. On an unvisited site  $j$  we have  $G(\mathbf{q}^{(i)})_j = 0$ , so the momentum kick  $\phi^P$  is the identity there and  $\psi$  reduces to its middle factor  $\phi_{\theta}^{Q,P}$ , which is directly the rotation  $\tau$ . Since  $\tau$  is a rotation it preserves  $\varphi_X \times \varphi_Y$ , giving Condition A3 for  $I' = F' \circ \tau^L$ . The map  $\psi_T$  is still volume preserving, so  $I_T = F_T \circ \psi_T^L$  is a measure-preserving involution and  $I$  decomposes as required for Condition A2. The details are in App. C.5.  $\square$

#### 4.7 LazyHMC3: Framework A with the Usual Leapfrog Integrator

The original HMC integrator (§4.3) can also be used for the visited sites. But since the kernel outputs the full rose tree, the unvisited coordinates must be changed carefully: leaving them untouched or copying them arbitrarily would break the Markov chain. We must also know which sites are visited to include them in the acceptance ratio, unlike the methods in Sec. 4.5 and 4.6, where the addresses of non-zero-gradient sites along the trajectory suffice.

Fix  $\epsilon > 0$  and let one step be  $\psi = \phi_{\epsilon/2}^P \circ \phi_{\epsilon}^Q \circ \phi_{\epsilon/2}^P$ , where  $\phi_t^P, \phi_t^Q$  are defined as in Sec. 4.5. Again,  $X, Y, \mu_X, \mu_Y, v$  are defined the same as in Sec. 4.5. Notice that if  $j \in v(\mathbf{q}, \mathbf{p})$  and  $G(\mathbf{q}^{(i)})_j = 0$  for all  $i \in \{0, \dots, L\}$  we cannot guarantee that  $(\mu_X \times \mu_Y)(I(\mathbf{q}, \mathbf{p})) = (\mu_X \times \mu_Y)(\mathbf{q}, \mathbf{p})$ . Therefore, we need to know  $v(\mathbf{q}, \mathbf{p})$  not just  $u(\mathbf{q}, \mathbf{p})$ . Moreover, we need to let  $I(\mathbf{q}, \mathbf{p})_i = I'(\mathbf{q}_i, \mathbf{p}_i)$  for any  $i \in \mathcal{A} \setminus v(\mathbf{q}, \mathbf{p})$ , where  $I' : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  is some Leb<sub>2</sub> preserving involution with  $(\mu_X \times \mu_Y)(I'(\mathbf{q}, \mathbf{p})) = (\mu_X \times \mu_Y)(\mathbf{q}, \mathbf{p})$  (e.g.  $I' = F' \circ \tau$ , or the identity, which is what we use in our implementation). Because  $I'$  preserves the site density  $\varphi_X \times \varphi_Y$ , every unvisited coordinate contributes a factor 1 to the acceptance ratio, so the otherwise infinite product collapses to the finite product over the visited sites  $v(\mathbf{q}, \mathbf{p})$ .

PROPOSITION 7. *I satisfies Conditions A1 to A3 (see App. C.6 for the proof).*

*Implementation details.* To compute  $v(\mathbf{q}, \mathbf{p})$  we need to first apply  $F \circ \psi^L$  to get the states in the trajectory  $(\mathbf{q}^{(i)}, \mathbf{p}^{(i)})$ . Then for each  $\mathbf{q}^{(i)}$  we can determine  $v_I(\mathbf{q}^{(i)})$  by checking which thunks in the seed have been evaluated, as done for the single-site proposal kernel in [10]. For the sites  $j$  not in  $v(\mathbf{q}, \mathbf{p})$  we then need to change  $\mathbf{q}_j^{(L)}, \mathbf{p}_j^{(L)}$  to  $I'(\mathbf{q}_j, \mathbf{p}_j)$  before the accept/reject step.

#### 4.8 LazyNUTS

HMC's performance depends on the number of leapfrog steps  $L$ , which is hard to tune. NUTS addresses this by choosing  $L$  adaptively via a doubling procedure with a no-U-turn stopping rule.



We develop a lazy version, lazyNUTS [12], that keeps the doubling procedure and no-U-turn rule but still evaluates only finitely many sites per iteration, using the leapfrog integrator from lazyHMC1 (Sec. 4.5). We give a correctness argument in App. D.1: the acceptance ratio is 1 provided the stopping rule yields a start-independent set of proposed states ( $C_s = C_{s'}$ ); as with the maximum tree depth in standard NUTS, our cap  $M$  on the number of proposed states is a practical termination we do not show to preserve this. We also derive Framework B (App. D), generalizing all lazy HMC and lazy NUTS versions, in the appendix.

## 5 Example: Piecewise-Constant Regression with Poisson-Process Changepoints

To make the preceding frameworks concrete, we now trace the behaviour of lazy HMC on a single model in detail: a piecewise-constant regression model with Poisson-process changepoints from [10]. For simplicity, we refer to it as the step regression model.

Consider a set of synthetic two-dimensional datapoints as shown in Fig. 1. The task is to fit a piecewise-constant function to this dataset. The model first draws a function  $f$  from a prior, then observes the likelihood of the dataset being generated by  $f$ , after introducing some Gaussian noise.

Listing 3. Step regression model

```
1 stepReg :: [(Double, Double)] → Meas (Double → Double)
2 stepReg dataset = do
3   f ← sample (splice (poissonPP 0 0.2) randConst)
4   forM_ dataset \(x, y) → scoreLog (normalLogPdf (f x) 0.1 y))
5   return f
```

Now consider the prior. The splice function builds a piecewise function from a point process generating changepoints and a prior randomFun over the behaviour on each segment. It samples an infinite stream of changepoints  $xs$  and, for each, an independent function from randomFun. An input is then evaluated by selecting the segment containing it.

```
1 splice :: Prob [Double] → Prob (Double → Double) → Prob (Double → Double)
2 splice pointProcess randomFun = do
3   xs ← pointProcess
4   fs ← mapM (const randomFun) xs
5   default_f ← randomFun
6   let h [] x = default_f x
7       h ((a,f):rest) x
8         | x ≤ a      = f x
9         | otherwise = h rest x
10  return (h (zip xs fs))
```

Since each segment is constant, we use a base prior sampling constant functions:

```
1 randConst :: Prob (Double → Double)
2 randConst = do
3   b ← normal 0 3
4   return (const b)
```

The point process is a Poisson process with rate 0.2 and starting from 0:

```
1 poissonPP :: Double → Double → Prob [Double]
2 poissonPP lower rate = do
3   step ← exponential rate
4   let x = lower + step
```

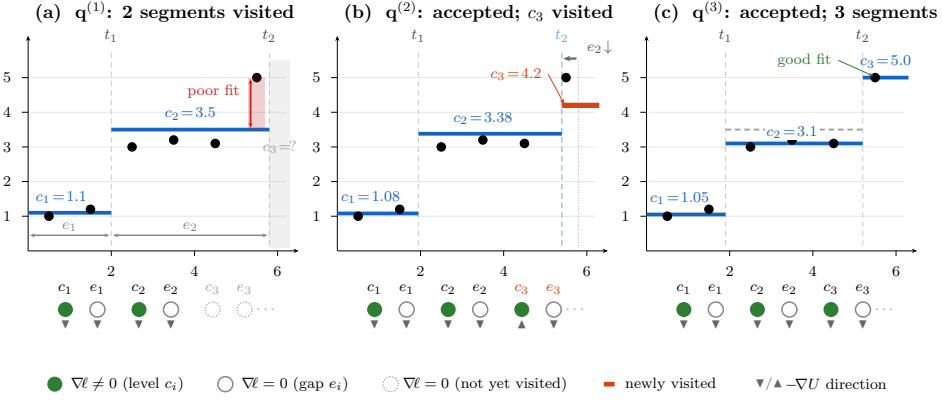


Fig. 1. Dimension change during lazy HMC on the step regression model (§5). (a) Initial state  $q^{(1)}$ , where the second changepoint lies to the right of all datapoints. Only two segments are evaluated; later segments are not visited due to lazy evaluation. (b) After five leapfrog steps (shown as a trajectory in  $(e_2, c_2)$ -space in Fig. 2a), the second changepoint  $t_2$  moves left of the last datapoint, activating a third segment. The previously unused sites  $c_3$  and  $e_3$  become visited, increasing the dimensionality of the evaluated state. (c) After a further five leapfrog steps, the segments adjust using the newly available gradient information, resulting in a better fit. Both transitions shown are accepted. See Fig. 2b for additional posterior samples.

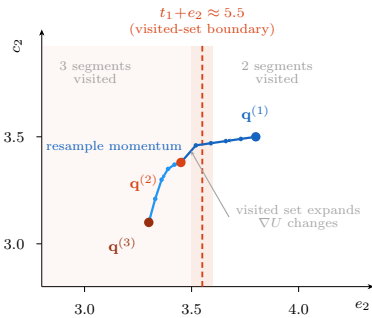
```

5  rest ← poissonPP x rate
6  return (x : rest)

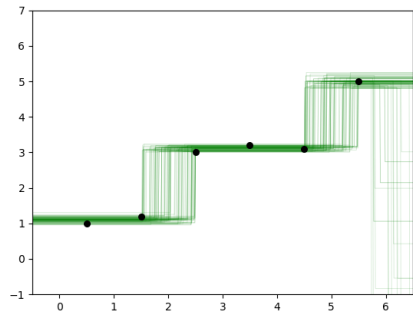
```

Note that a draw  $f$  from `splice` contains infinitely many segments. Consequently, the function returned by `stepReg` (Listing 3) also has infinitely many segments.

This does not cause any practical issues, since evaluation is lazy. Only those segments that are required for computing the likelihood or for producing the plots are ever evaluated. For instance, in Fig. 2b, only segments corresponding to input values in the range  $[-0.5, 6.5]$  are actually used.



(a) Illustrative example of lazy HMC trajectory on the step regression model. The values in the  $(e_2, c_2)$ -space correspond to the positions  $q^{(1)}, q^{(2)}, q^{(3)}$  from Fig. 1, with 5 leapfrog steps between successive positions.



(b) 100 posterior samples from running lazyHMC1( $\epsilon = 0.01, L = 30$ ) on the step regression model thinned every 10 samples after  $10^3$  burn-in samples.

Fig. 2. Lazy HMC on the step regression model: (a) an illustrative trajectory, and (b) actual posterior samples.

*Lazy HMC on the step regression model.* Consider the rose tree  $\mathbf{q}$  corresponding to the infinite stream of seeds used in a run of the step regression model. We distinguish between the seeds  $\mathbf{q}_{a_j}$  used to generate the heights of the constant functions and the seeds  $\mathbf{q}_{b_j}$  used to generate the exponential increments in the Poisson process for  $j = 1, \dots$ . Therefore the height of the  $j$ th constant function segment is  $c_j(\mathbf{q}) = 3\mathbf{q}_{a_j}$  and the  $j$ th exponential draw is  $e_j(\mathbf{q}) = -\log \text{cdf}_{\mathcal{N}(0,1)}(\mathbf{q}_{b_j})/0.2$  where  $\text{cdf}_{\mathcal{N}(0,1)}$  is the cumulative distribution function of  $\mathcal{N}(0, 1)$ . Hence, the  $j$ th changepoint is given by  $t_j(\mathbf{q}) = \sum_{i=1}^j e_i(\mathbf{q})$  and we can consider  $t_0(\mathbf{q}) = 0$ .

Suppose we start from a rose tree  $\mathbf{q}^{(1)}$  for which the second changepoint  $t_2$  is greater than the  $x$ -value of all datapoints, as shown in Fig. 1(a). In this case, the likelihood  $l$  is given by:

$$l(\mathbf{q}^{(1)}) = \prod_{i=1}^2 \varphi(y_i \mid c_1(\mathbf{q}^{(1)}), 0.1) \prod_{i=3}^6 \varphi(y_i \mid c_2(\mathbf{q}^{(1)}), 0.1) = \prod_{i=1}^2 \varphi(y_i \mid 1.1, 0.1) \prod_{i=3}^6 \varphi(y_i \mid 3.5, 0.1).$$

where  $(x_1, y_1) \dots (x_6, y_6)$  are the datapoints in increasing  $x$ -order and  $\varphi(\cdot \mid \mu, s)$  denotes the density of  $\mathcal{N}(\mu, s^2)$ . There exists an open set  $A_1$  containing  $\mathbf{q}^{(1)}$  such that

$$l(\mathbf{q}) = \prod_{i=1}^2 \varphi(y_i \mid c_1(\mathbf{q}), 0.1) \prod_{i=3}^6 \varphi(y_i \mid c_2(\mathbf{q}), 0.1)$$

for all  $\mathbf{q} \in A_1$ . Thus, two segments are enough to cover the dataset (with respect to the  $x$ -axis), and the sites  $a_i, b_i$  with  $i \geq 3$  are not visited. Consequently, the gradient at those sites is zero. Moreover,  $l$  restricted to the set  $A_1$  depends only on the sites  $a_1$  and  $a_2$ . Hence, the gradient at the visited sites  $b_1$  and  $b_2$  is also zero. The second segment in Fig. 1(a) is not a good fit for the 6th datapoint.

Fig. 2a shows the trajectory of the pair consisting of the second exponential increment and the second constant height after five leapfrog steps, starting from position  $\mathbf{q}^{(1)}$  with momentum  $\mathbf{p}^{(1)}$ .

After the fourth leapfrog step, the second changepoint becomes smaller than  $x_6$ . As a result, a third constant function is required in order to evaluate the likelihood contribution of the last datapoint  $(x_6, y_6)$ . After the fifth leapfrog step is performed, the new position  $\mathbf{q}^{(2)}$  (corresponding to Fig. 1(b)) is accepted. The sites that were visited in the intermediary positions computed during the 5 leapfrog steps are the ones corresponding to the first 3 constant function segments, so  $v(\mathbf{q}^{(1)}, \mathbf{p}^{(1)}) = \{a_1, a_2, a_3, b_1, b_2, b_3\}$ . However, if we use lazyHMC1 from Sec. 4.5 or lazyHMC2 from Sec. 4.6 it is enough to only include the sites  $\{a_1, a_2, a_3\}$  in the acceptance ratio as they are the only ones for which the gradient of  $l$  is non-zero.

The heights of the first two constant functions have decreased due to the gradient information at sites  $a_1$  and  $a_2$ , resulting in a better fit.

After the changepoint moves left of  $x_6$ , only datapoints 3, 4, 5 remain in segment 2. Therefore  $l$  is:

$$l(\mathbf{q}) = \left( \prod_{i=1}^2 \varphi(y_i \mid c_1(\mathbf{q}), 0.1) \right) \left( \prod_{i=3}^5 \varphi(y_i \mid c_2(\mathbf{q}), 0.1) \right) \varphi(y_6 \mid c_3(\mathbf{q}), 0.1)$$

for  $\mathbf{q}$  in some open set containing  $\mathbf{q}^{(2)}$ . In this region, the sites  $a_3$  and  $b_3$  become visited, and  $a_3$  now has a non-zero gradient. After resampling the momentum and performing five additional leapfrog steps, we reach position  $\mathbf{q}^{(3)}$  corresponding to Fig. 1(c). The segments now provide a better fit, as gradient information from  $a_1, a_2$ , and  $a_3$  has been incorporated by the leapfrog integrator. Fig. 2b shows 100 posterior samples obtained by lazyHMC1 on the step regression model.

$l$  is PACAP. Let  $T = \{(n_1, \dots, n_6) \in \mathbb{Z}_{>0}^6 \mid n_1 \leq n_2 \leq \dots \leq n_6\}$  be the set of weakly increasing 6-tuples of positive integers. Consider the following countable partition into analytic cylinders of the domain  $\mathbb{X} = \mathbb{R}^{\mathcal{A}}$  of  $l$ :  $\mathbb{X} = \cup_{t \in T} U_t$  where

$$U_{(n_1, n_2, \dots, n_6)} = \{\mathbf{q} \in \mathbb{X} \mid t_{n_i-1}(\mathbf{q}) < x_i \leq t_{n_i}(\mathbf{q}) \text{ for } 1 \leq i \leq 6\}$$

for any  $(n_1, n_2, \dots, n_6) \in T$ . The set  $U_{(n_1, n_2, \dots, n_6)}$  corresponds to all the rose trees for which the  $i$ th datapoint  $(x_i, y_i)$  is going to be on the  $n_i$  segment for all  $1 \leq i \leq 6$ . For each  $\mathbf{q}$ , each datapoint lies in exactly one segment. Therefore the union  $\mathbb{X} = \cup_{t \in T} U_t$  is disjoint. Note that  $t_{n_i}(\mathbf{q})$  can be computed using only the sites  $b_1, b_2, \dots, b_{n_i}$ . We let the support of  $U_{(n_1, n_2, \dots, n_6)}$  be  $B = \{b_1, b_2, \dots, b_{n_6}\} \cup \{a_1, a_2, \dots, a_{n_6}\}$ . To see why  $U_{(n_1, n_2, \dots, n_6)}$  is an analytic cylinder (see Def. 1), consider the analytic functions  $h_{n_i-1} : \mathbb{R}^B \rightarrow \mathbb{R}, g_{n_i} : \mathbb{R}^B \rightarrow \mathbb{R}$  given by  $h_{n_i-1}(\mathbf{q}_B) = t_{n_i-1}(\mathbf{q}) - x_i$  and  $g_{n_i}(\mathbf{q}_B) = x_i - t_{n_i}(\mathbf{q})$  for  $1 \leq i \leq 6$ . Now let the set  $V = \cap_{i=1}^6 h_{n_i-1}^{-1}((-\infty, 0))$  be the open set encoding the strict inequalities:  $V = \{\mathbf{q}_B \in \mathbb{R}^B \mid h_{n_i-1}(\mathbf{q}_B) < 0 \text{ for } 1 \leq i \leq 6\}$ . Hence, we can write  $U_{(n_1, n_2, \dots, n_6)}$  as:

$$U_{(n_1, n_2, \dots, n_6)} = \{\mathbf{q} \in \mathbb{X} \mid \mathbf{q}_B \in V \text{ \& } g_{n_i}(\mathbf{q}_B) \leq 0 \text{ for } 1 \leq i \leq 6\}.$$

Moreover, on  $U_{(n_1, n_2, \dots, n_6)}$  we have:  $l(\mathbf{q}) = \prod_{i=1}^6 \varphi(y_i \mid c_{n_i}(\mathbf{q}), 0.1)$ . Considering that  $c_j$  only needs information from the site  $a_j$  there exists an analytic function  $f : V \rightarrow \mathbb{R}$  given by  $f(\mathbf{q}_B) = \prod_{i=1}^6 \varphi(y_i \mid c_{n_i}(\mathbf{q}_B), 0.1)$ . Hence,  $l(\mathbf{q}) = f(\mathbf{q}_B)$  when  $\mathbf{q} \in U_{(n_1, n_2, \dots, n_6)}$ .

## 6 Experiments

We implemented the lazy HMC methods (§4.5, §4.6 and §4.7) and lazyNUTS methods (§D.1) as an extension to the LazyPPL library, by also using the automatic differentiation method (§3.3). Our implementation is available at <https://github.com/lazypppl-team/lazyhmc>.

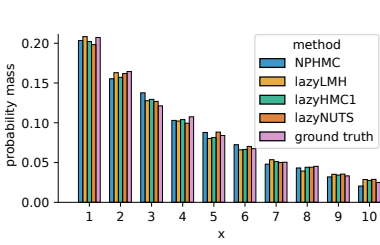
We include the following illustrative experiments that emphasize the non-parametric side of lazy HMC, testing the aspects that standard HMC methods do not typically address. To be clear, we are not evaluating against established benchmarks, since there aren't really any in the non-parametric setting. Although the experiments here are simple, they nonetheless provide empirical evidence that lazy HMC performs as intended for non-parametric and infinite-dimensional models.

In what follows, for the specified step size  $\epsilon$ , we have used a fresh value sampled uniformly at each iteration from  $[\epsilon/2, 3 \cdot \epsilon/2]$ . The maximum number of samples generated in a single iteration of lazyNUTS is denoted by  $M$ . Imposing a bound  $M$  on the number of proposed states may seem to go against the adaptive design of NUTS, but in practice NUTS implementations (e.g. Stan) always cap the trajectory via a maximum tree depth. We set  $M$  mainly for comparability: large enough not to artificially cap the trajectory lengths lazyNUTS explores, while keeping them of a similar order of magnitude to those in the lazy HMC variants. For comparison, we also ran the lazy lightweight Metropolis-Hastings (lazyLMH) method from [10]; this simple method proposes a new rose tree by resampling the value at each site with probability  $b$ , and does not use gradients at all.

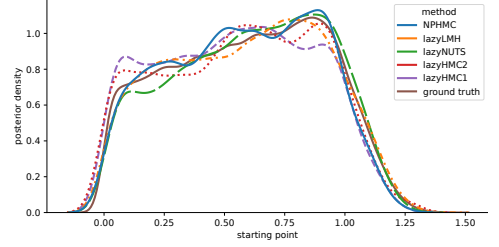
We have also compared lazy HMC performance with NP-HMC (the variant presented in [19] which does not include truncation or discontinuous HMC). Posteriors were comparable to our methods. Lazy HMC variants were generally faster, although we observed slower performance on the GMM model, where the bottleneck appears to be the current lack of optimization in LazyPPL's automatic differentiation implementation. These timings are not perfectly comparable: NP-HMC is implemented in Python, while our methods are implemented in Haskell and LazyPPL's use of laziness means that models with infinite structure cannot always be expressed identically in both systems. Compare e.g. geometric (natural for lazy HMC) and geometricStrict (suited to NP-HMC).

*Experiment: Geometric distribution.* We first sample from the geometric distribution described in Listing 1 with  $p = 0.2$ . This example shows the methods can handle dimension changes. The methods appear to sample correctly from the distribution, as shown in Fig. 3a, which compares the probability mass estimated from the samples with the ground truth distribution. The total variation distance (TVD) provides a scalar measure of the discrepancy from the ground truth, with values closer to zero indicating better agreement. The TVDs are 0.0146 (lazyLMH), 0.034 (NP-HMC), 0.0198 (lazyNUTS), 0.0115 (lazyHMC1), 0.0117 (lazyHMC2), and 0.0132 (lazyHMC3). The total times were

16.6 (NP-HMC), 0.6 (lazyHMC1), 0.6 (lazyHMC2), 0.7 (lazyHMC3) and 0.8 (lazyNUTS) minutes; NP-HMC is at least ten times slower than the lazy variants.



(a) Estimated posterior mass for the Geometric distribution, averaged over 10 runs. Sample sizes per run: 1,300 for lazyHMC1, NP-HMC ( $\epsilon = 0.1$ ,  $L = 15$ ), and lazyNUTS ( $\epsilon = 0.1$ ,  $M = 2^5$ );  $10^4$  for lazyLMH ( $b = 0.5$ ).



(b) Kernel density estimate for the random walk model, averaged over 10 runs. Sample sizes per run: 1,300 for lazyHMC1, lazyHMC2, NP-HMC ( $\epsilon = 0.1$ ,  $L = 5$ ), and lazyNUTS ( $\epsilon = 0.1$ ,  $M = 2^5$ );  $2 \times 10^4$  for lazyLMH ( $b = 0.5$ ).

Fig. 3. Sampler comparison on the Geometric (left) and random walk (right) models.

*Experiment: Random Walk.* We consider the random walk model from [19]. The model is presented in Listing 2. The posterior cannot be computed exactly, so we will use  $10^6$  importance samples instead of the ground truth. For the lazy HMC, lazyNUTS, and NP-HMC methods, we used a heuristic initialisation procedure to find a starting state closer to the typical set. Specifically, we first ran 100 iterations with the momentum excluded from the acceptance ratio. These iterations were used only for initialisation and were discarded. We then ran the correct kernel for 1300 iterations and discarded the first 20% samples as burn-in. We then ran the correct kernel for the number of iterations specified in Fig. 3b, discarding the first 20% of samples as burn-in for each method as well as a thinning for lazyLMH. The resulting kernel density estimates are shown in Fig. 3b.

The upper bound on the number of samples considered per iteration is much larger for lazyNUTS than for the other methods, which makes it slower: lazyNUTS takes 1.6 minutes (for all chains), compared with 0.2 minutes for lazyHMC1 and lazyHMC2 and 0.3 minutes for lazyHMC3. However, lazyNUTS also achieves a much larger effective sample size (ESS), with an ESS of 732.2 compared with 189.6 for lazyHMC1, 175.6 for lazyHMC2 and 231.9 for lazyHMC3. When computational cost is taken into account, the lazy HMC variants are more efficient, achieving 20.09, 18.61 and 15.09 ESS/s respectively, compared with 7.73 ESS/s for lazyNUTS. In comparison, NP-HMC is slower than the lazy HMC variants, at 19 total minutes, and has a similar ESS of 213.2, leading to a much lower efficiency of 0.18 ESS/s.

We emphasise that this model demonstrates some of the core ideas of compositional modelling that are enabled by this lazy approach. Recall that line 3 of the model (Listing 2) produces random infinite sequences: the walk continues forever. This infinite stochastic process determines the infinite-dimensional state space that lazy HMC operates over. Our only observation (line 5) is about the first point the stopping criterion is met. Lazy HMC produces results, despite operating over an infinite dimensional space, because it focuses lazily on the dimensions that are actually needed.

*Experiment: Gaussian mixture model (GMM) with unbounded number of components.* We consider the Gaussian Mixture model inspired by [33] in which the number of components is unbounded:

$$K \sim \text{Poisson}(9) + 1, \quad \mu_k \sim \text{Uniform}\left(\frac{20(k-1)}{K}, \frac{20k}{K}\right), \quad y_n \sim \frac{1}{K} \sum_{k=1}^K \mathcal{N}(\mu_k, 1.5^2).$$

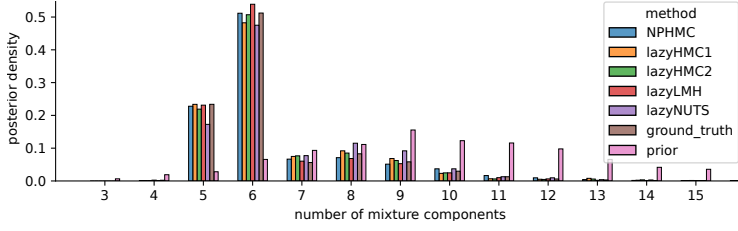
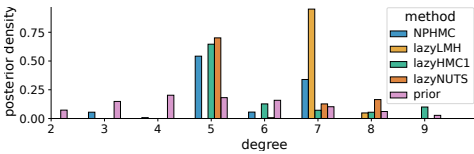


Fig. 4. Histogram of the number of components for the GMM, averaged over 10 runs. Sample sizes per run: 2000 for lazyHMC1, lazyHMC2, NP-HMC ( $\epsilon = 0.05$ ,  $L = 20$ ), and lazyNUTS ( $\epsilon = 0.1$ ,  $M = 2^5$ );  $4 \times 10^4$  for lazyLMH ( $b = 0.5$ ). We discard the first 20% of the samples for each chain.

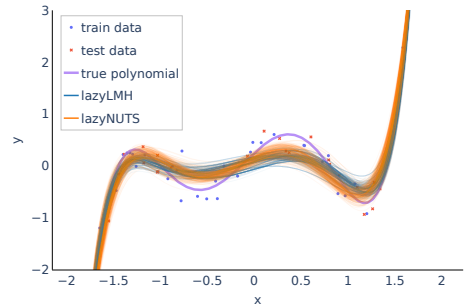
for  $k = 1, \dots, K$ . The model is presented in Sec. 2.3.

We fix the true number of components to  $K^* = 5$  and sample the true components means as  $\mu_k^* \sim \text{Uniform}\left(\frac{20(k-1)}{5}, \frac{20k}{5}\right)$  for  $k = 1 \dots 5$ . Each point in our synthetic dataset is drawn from the Gaussian mixture with  $K^* = 5$  components with means  $\mu_1^*, \dots, \mu_5^*$ . Concretely, we first sample one component index  $k$  uniformly from  $\{1, \dots, 5\}$ , and then draw the datapoint from  $\mathcal{N}(\mu_k^*, 1.5^2)$ . We split the dataset into 200 points for training and 50 points for testing.

The prior (Poisson(9) + 1) on the number of components and the distribution inferred from the posterior samples can be seen in Fig. 4. As the posterior cannot be computed exactly, we again use  $10^6$  importance samples instead of the ground truth. We also approximate the log pointwise predictive density (LPPD, [29]) for the test dataset  $\{y_1, \dots, y_n\}$  by  $\sum_{i=1}^n \log \frac{1}{N} \sum_{j=1}^N p(y_i | \theta_j)$ , where  $\theta_j$  are the samples from the target posterior. The true LPPD is  $-143.73$  and the mean and one standard deviation LPPD over the 10 chains for each method is:  $-147.74 \pm 0.28$  (lazyHMC1),  $-147.73 \pm 0.26$  (lazyHMC2),  $-147.84 \pm 0.10$  (lazyNUTS),  $-147.62 \pm 0.14$  (lazyLMH),  $-147.75 \pm 0.09$  (NP-HMC). The total times in minutes across the chains are: 72 (lazyHMC1, lazyHMC2), 129 (lazyNUTS), 41 (lazyLMH), 27 (NP-HMC).



(a) Histogram of the degree in the polynomial model, averaged over 10 runs. Sample sizes per run: 3000 for lazyHMC1 ( $\epsilon = 0.005$ ,  $L = 64$ ), lazyNUTS ( $\epsilon = 0.005$ ,  $M = 2^6$ );  $10^6$  for lazyLMH ( $b = 0.5$ ). For lazyLMH we thin the chain to obtain  $\sim 3000$  samples. In all cases, only the second half of the samples is used (the first half is discarded as burn-in).



(b) Posterior polynomial fits based on the same post-burn-in samples as in Fig. 5a. For this figure, we further thin all chains by keeping every second sample.

Fig. 5. Histogram of sampled polynomial degrees and posterior fits for the polynomial regression model.

*Experiment: Polynomial regression model with unbounded degree.* In this example, we generate a synthetic regression dataset from a degree 5 polynomial. We sample 40 training inputs and 25 test



Table 1. For the polynomial regression experiment, we report test dataset LPPD and runtime in minutes summed over all chains. LPPD is computed using the same post-burn-in samples as in Fig. 5a. The true test dataset LPPD is 3.82. Reported runtimes correspond to the full chain execution, including burn-in.

|                     | lazyNUTS         | lazyHMC1         | lazyLMH           | NP-HMC           |
|---------------------|------------------|------------------|-------------------|------------------|
| LPPD mean $\pm$ std | $-5.94 \pm 2.53$ | $-4.69 \pm 2.16$ | $-13.11 \pm 9.72$ | $-7.35 \pm 6.16$ |
| Time (minutes)      | 10.9             | 23.2             | 17.2              | 233.7            |

inputs independently as  $x_i \sim \text{Uniform}(-2, 2)$ . The outputs are generated from a fixed ground-truth polynomial  $P^*$  of degree 5, corrupted by Gaussian noise:  $y_i \sim \mathcal{N}(P^*(x_i), 0.25^2)$ . The polynomial  $P^*$  as well as the training and test points can be seen in Fig. 5b. The polynomial prior for our model is:

$$D \sim \text{Poisson}(4) + 1, \quad (a_k)_{k=0}^D \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, 0.5^2), \quad Q(x) = \sum_{k=0}^D a_k x^k.$$

Since we do not want to constrain the degree  $D$  of the polynomial we sample it from the Poisson distribution. The coefficients of the polynomial  $Q$  are then sampled from the normal distribution. For each train point  $(x_i, y_i)$ , we score the likelihood of it being generated using the polynomial  $Q$  together with the Gaussian noise:

$$y_i \sim \mathcal{N}(Q(x_i), 0.25^2).$$

In Fig. 5a we show the distribution of the degree of the inferred polynomial. While the samples from lazyHMC1 and lazyNUTS concentrate on polynomials of degree 5, those from lazyLMH are stuck mostly with polynomials of degree 7. The performance difference is also visible in Fig. 5b, where the lazyNUTS samples fit the dataset better than the lazyLMH samples. For visual clarity, Fig. 5b shows only lazyNUTS and lazyLMH, since plotting all methods together causes substantial overlap. NP-HMC also has a smaller posterior mass on degree 5 polynomials. Despite similar or longer runtime, lazyLMH and NP-HMC achieve worse LPPD than our methods (Table 1).

**Conclusion.** We have given a Hamiltonian Monte Carlo simulation that works over probabilistic programs involving infinite dimensions, by evaluating at the dimensions lazily. The key idea is that the relevant gradient is only non-zero at finitely many dimensions (because the likelihood functions are PACAP, §3) and the acceptance ratio is designed so that the factors corresponding to unused dimensions cancel (§4). To improve on the issues with leapfrog and step size, we also provide a lazyNUTS procedure.

Our initial experimentation shows that the performance is good even where the changing dimensions play a big role in the model (§6). However, in the broader context of real-world systems and large non-parametric models, to be clear, we don't expect a purely declarative approach to be sufficient: the modeller will likely need to guide or interact with the inference engine (e.g. Gen [9]). In our setting this interaction is mostly the familiar one of choosing HMC hyperparameters (step size, trajectory length  $L$ , or the bound  $M$  in lazyNUTS) and tuning them to the model at hand; and a natural direction for future work is to incorporate automatic step-size adaptation (such as in [12]), reducing the need for manual tuning.

## Acknowledgments

We would like to thank Alex Lew, Matthijs Vákár and Fabian Zaiser for valuable discussions and advice. This work was partially supported by the National Research Foundation, Singapore, under its

RSS Scheme (NRF-RSS2022-009), ERC Grant BLAST, AFOSR under award number FA9550-21-1-0038, and grants from ARIA Safeguarded AI.

## References

- [1] Christophe Andrieu, Anthony Lee, and Samuel Livingstone. 2020. A general perspective on the Metropolis–Hastings kernel. *arXiv preprint arXiv:2012.14881* (2020). <https://arxiv.org/abs/2012.14881>
- [2] Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. 2020. On the Versatility of Open Logical Relations - Continuity, Automatic Differentiation, and a Containment Theorem. In *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12075)*, Peter Müller (Ed.). Springer, 56–83. doi:10.1007/978-3-030-44914-8\_3
- [3] Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. 2020. *Foundations of Probabilistic Programming*. Cambridge University Press. doi:10.1017/9781108770750
- [4] Alexandros Beskos, Frank J Pinski, Jesús Maria Sanz-Serna, and Andrew M Stuart. 2011. Hybrid Monte Carlo on Hilbert spaces. *Stochastic Processes and their Applications* 121, 10 (2011), 2201–2230. doi:10.1016/j.spa.2011.06.003
- [5] Michael Betancourt. 2017. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434* (2017).
- [6] Nawaf Bou-Rabee, Bob Carpenter, and Milo Marsden. 2024. GIST: Gibbs self-tuning for locally adaptive Hamiltonian Monte Carlo. *arXiv preprint arXiv:2404.15253* (2024).
- [7] Maddy Bowers, Alexander K. Lew, Joshua B. Tenenbaum, Armando Solar-Lezama, and Vikash K. Mansinghka. 2025. Stochastic Lazy Knowledge Compilation for Inference in Discrete Probabilistic Programs. *Proc. ACM Program. Lang.* 9, PLDI (2025), 1863–1887. doi:10.1145/3729325
- [8] Marco Cusumano-Towner, Alexander K. Lew, and Vikash K. Mansinghka. 2020. Automating Involutive MCMC using Probabilistic and Differentiable Programming. *arXiv preprint arXiv:2007.09871* (2020). <https://arxiv.org/abs/2007.09871>
- [9] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22–26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 221–236. doi:10.1145/3314221.3314642
- [10] Swaraj Dash, Younesse Kaddar, Hugo Paquet, and Sam Staton. 2023. Affine Monads and Lazy Structures for Bayesian Programming. *Proc. ACM Program. Lang.* 7, POPL (2023), 1338–1368. doi:10.1145/3571239
- [11] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20–23, 2017*. IEEE Computer Society, 1–12. doi:10.1109/LICS.2017.8005137
- [12] Matthew D. Hoffman and Andrew Gelman. 2014. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* 15, 1 (2014), 1593–1623. <http://dl.acm.org/citation.cfm?id=2638586>
- [13] Mathieu Huot, Alexander K. Lew, Vikash K. Mansinghka, and Sam Staton. 2023.  $\omega$ PAP Spaces: Reasoning Denotationally About Higher-Order, Recursive Probabilistic and Differentiable Programs. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26–29, 2023*. IEEE, 1–14. doi:10.1109/LICS56636.2023.10175739
- [14] Mathieu Huot, Sam Staton, and Matthijs Vákár. 2020. Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12077)*, Jean Goubault-Larrecq and Barbara König (Eds.). Springer, 319–338. doi:10.1007/978-3-030-45231-5\_17
- [15] Wonyeol Lee, Hangyeol Yu, Xavier Rival, and Hongseok Yang. 2020. On Correctness of Automatic Differentiation for Non-Differentiable Functions. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual*, Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). <https://proceedings.neurips.cc/paper/2020/hash/4aaa76178f8567e05c8e8295c96171d8-Abstract.html>
- [16] Benedict Leimkuhler and Sebastian Reich. 2004. *Simulating Hamiltonian dynamics*. Number 14. Cambridge University Press. doi:10.1017/CBO9780511614118
- [17] Samuel Livingstone, Michael Betancourt, Simon Byrne, and Mark Girolami. 2019. On the geometric ergodicity of Hamiltonian Monte Carlo. *Bernoulli* 25, 4A (2019), 3109–3138. doi:10.3150/18-BEJ1083
- [18] John Longley and Dag Normann. 2015. *Higher-Order Computability*. Springer, Cham. doi:10.1007/978-3-319-21500-6
- [19] Carol Mak, Fabian Zaiser, and Luke Ong. 2021. Nonparametric Hamiltonian Monte Carlo. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18–24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 7336–7347. <http://proceedings.mlr.press/>

- v139/mak21a.html
- [20] Carol Mak, Fabian Zaiser, and Luke Ong. 2022. Nonparametric Involutive Markov Chain Monte Carlo. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 14802–14859. <https://proceedings.mlr.press/v162/mak22a.html>
  - [21] Radford M. Neal. 2012. MCMC using Hamiltonian dynamics. (6 2012). doi:10.1201/b10905
  - [22] Kirill Neklyudov, Max Welling, Evgenii Egorov, and Dmitry P. Vetrov. 2020. Involutive MCMC: a Unifying Framework. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 7273–7282. <http://proceedings.mlr.press/v119/neklyudov20a.html>
  - [23] Peter Orbanz and Yee Whye Teh. 2010. Bayesian nonparametric models. In *Encyclopedia of Machine Learning*. Springer, 81–89. doi:10.1007/978-0-387-30164-8\_66
  - [24] John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *Information Processing 83 (IFIP Congress Series, Vol. 9)*, R.E.A. Mason (Ed.). Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 513–523.
  - [25] Daniel M. Roy, Vikash K. Mansinghka, Noah D. Goodman, and Joshua B. Tenenbaum. 2008. A stochastic programming perspective on nonparametric Bayes. In *Workshop on Nonparametric Bayes (ICML 2008)*, Helsinki, Finland. Filename: RoyManGooTen-ICMLNPB-2008.pdf; MIT. <https://danroy.org/papers/RoyManGooTen-ICMLNPB-2008.pdf>
  - [26] Amir Shaikhha, Andrew W. Fitzgibbon, Dimitrios Vytiniotis, and Simon Peyton Jones. 2019. Efficient differentiable programming in a functional array-processing language. *Proc. ACM Program. Lang.* 3, ICFP (2019), 97:1–97:30. doi:10.1145/3341701
  - [27] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 <http://arxiv.org/abs/1809.10756>
  - [28] Birthe van den Berg, Tom Schrijvers, James McKinna, and Alexander Vandenbroucke. 2024. Forward- or Reverse-Mode Automatic Differentiation: What’s the Difference? *Science of Computer Programming* 231 (2024), 103010. doi:10.1016/j.scico.2023.103010
  - [29] Aki Vehtari, A Gelman, D Dunson, D Rubin, H Stern, and JB Carlin. 2014. Bayesian data analysis.
  - [30] David Wingate, Andreas Stuhlmüller, and Noah D. Goodman. 2011. Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011 (JMLR Proceedings, Vol. 15)*, Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík (Eds.). JMLR.org, 770–778. <http://proceedings.mlr.press/v15/wingate11a/wingate11a.pdf>
  - [31] Glynn Winskel. 1993. *The formal semantics of programming languages: an introduction*. MIT Press, Cambridge, MA, USA.
  - [32] Frank D. Wood, Jan-Willem van de Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014 (JMLR Workshop and Conference Proceedings, Vol. 33)*. JMLR.org, 1024–1032. <http://proceedings.mlr.press/v33/wood14.html>
  - [33] Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. 2020. 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, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 11534–11545. <http://proceedings.mlr.press/v119/zhou20e.html>

## A Proof of the Fundamental Lemma (2), from Thm. 2

The proof is by induction on typing derivations, and requires preliminary lemmas on support enlargement (Lem. 1), closure under finite intersection (Lem. 2), restriction (Lem. 3), gluing (Lem. 4), and composition (Lem. 5).

We note that  $\mathbb{X} = \mathbb{R}^{\mathcal{A}}$  is itself an analytic cylinder (with empty support), so c-analytic; and  $\emptyset$  is c-analytic (the empty union), with  $\mathcal{P}_\tau^\emptyset$  a singleton for every  $\tau$ .

**LEMMA 1 (SUPPORT ENLARGEMENT).** *If  $U$  is an analytic cylinder with support  $B$  and  $B' \supseteq B$  is finite, then  $U$  is also an analytic cylinder with support  $B'$ .*

**PROOF.** Let  $V \subseteq \mathbb{R}^B$  and  $g_k : V \rightarrow \mathbb{R}$  be the open set and constraints for  $U$ . Let  $\pi : \mathbb{R}^{B'} \rightarrow \mathbb{R}^B$  be the coordinate projection, which is continuous and linear (hence analytic). Take  $V' = \pi^{-1}(V)$  (open since  $\pi$  is continuous) and constraints  $g_k \circ \pi$ .  $\square$

**LEMMA 2 (CLOSURE UNDER FINITE INTERSECTION).** *The intersection of two analytic cylinders is an analytic cylinder. Consequently, c-analytic sets are closed under finite intersection.*

**PROOF.** Given analytic cylinders  $U_1, U_2$  with supports  $B_1, B_2$ , enlarge both to the common support  $B = B_1 \cup B_2$  (Lem. 1), then intersect the open sets and concatenate the constraint lists (cf. [13], Cor. B.10 for the finite-dimensional case). For c-analytic closure: if  $W = \biguplus_i A_i$  and  $W' = \biguplus_j C_j$  are c-analytic, then  $W \cap W' = \biguplus_{i,j} (A_i \cap C_j)$ , which is disjoint since the  $A_i$  and  $C_j$  are separately disjoint, and is a countable union of analytic cylinders.  $\square$

**LEMMA 3 (RESTRICTION).** *If  $f \in \mathcal{P}_\tau^U$  and  $U' \subseteq U$  is c-analytic, then  $f|_{U'} \in \mathcal{P}_\tau^{U'}$ .*

**PROOF.** By induction on  $\tau$ . For  $\tau = \text{real}$ : if  $f$  is PACAP on  $U$  with partition  $\{A_k\}_k$  into analytic cylinders, and  $U' = \biguplus_m C_m$  is c-analytic, then  $U' = \biguplus_{k,m} (A_k \cap C_m)$ . Each  $A_k \cap C_m$  is an analytic cylinder (Lem. 2) with support  $B_k \cup D_m$ . On this piece,  $f(x) = f_k(x|_{B_k})$ , where  $f_k$  is analytic on some open  $V_k \subseteq \mathbb{R}^{B_k}$ . By support enlargement (Lem. 1) from  $B_k$  to  $B_k \cup D_m$ , the function  $f_k$  lifts to  $f_k \circ \pi$  on the enlarged open set, witnessing PACAP for each piece. Hence  $f|_{U'}$  is PACAP. The unit and product cases are immediate. For  $\tau = \sum_{i \in I} \tau_i$ : if  $f \in \mathcal{P}_\tau^U$  via partition  $U = \biguplus_i U_i$ , then  $U' = \biguplus_i (U_i \cap U')$  where each  $U_i \cap U'$  is c-analytic (Lem. 2), and  $f_i|_{U_i \cap U'} \in \mathcal{P}_{\tau_i}^{U_i \cap U'}$  by the induction hypothesis. For  $\tau = \tau_1 \rightarrow \tau_2$ : any c-analytic  $U'' \subseteq U'$  is also a c-analytic subset of  $U$ , so the defining clause for  $\mathcal{P}_{\tau_1 \rightarrow \tau_2}^U$  directly gives the result.  $\square$

**LEMMA 4 (GLUING).** *If  $U = \biguplus_{i \in I} U_i$  is a countable partition into c-analytic sets and  $f_i \in \mathcal{P}_\tau^{U_i}$  for each  $i$ , then the function  $f$  defined by  $f(u) = f_i(u)$  for  $u \in U_i$  satisfies  $f \in \mathcal{P}_\tau^U$ .*

**PROOF.** By induction on  $\tau$ . For  $\tau = \text{real}$ : each  $f_i$  is PACAP on  $U_i$ ; concatenating their analytic-cylinder partitions gives a PACAP partition of  $U$ . The unit and product cases are immediate. For  $\tau = \sum_{l \in I} \tau_l$ : each  $f_k \in \mathcal{P}_{\sum_{l \in I} \tau_l}^{U_k}$  gives a partition  $U_k = \biguplus_l U_{kl}$  with  $g_{kl} \in \mathcal{P}_{\tau_l}^{U_{kl}}$ . Let  $W_l = \biguplus_k U_{kl}$  for each  $l \in I$ ; each  $W_l$  is c-analytic (a countable disjoint union of c-analytic sets). Since the  $U_k$  are pairwise disjoint, so are the  $U_{kl}$  across different  $k$ , giving  $\biguplus_l W_l = \biguplus_k U_k = U$ . By the induction hypothesis at  $\tau_l$  (gluing the  $g_{kl}$  over  $W_l = \biguplus_k U_{kl}$ ), the resulting function on  $W_l$  is in  $\mathcal{P}_{\tau_l}^{W_l}$ . For  $\tau = \tau_1 \rightarrow \tau_2$ : given c-analytic  $U' \subseteq U$  and  $g \in \mathcal{P}_{\tau_1}^{U'}$ , let  $U'_i = U' \cap U_i$ , which is c-analytic by Lem. 2. By restriction (Lem. 3),  $g|_{U'_i} \in \mathcal{P}_{\tau_1}^{U'_i}$ . Since  $U'_i \subseteq U_i$  is c-analytic and  $f_i \in \mathcal{P}_{\tau_1 \rightarrow \tau_2}^{U_i}$ , we get  $\lambda u. f_i(u)(g(u)) \in \mathcal{P}_{\tau_2}^{U'_i}$ . By the induction hypothesis at  $\tau_2$  (gluing over  $U' = \biguplus_i U'_i$ ),  $\lambda u. f(u)(g(u)) \in \mathcal{P}_{\tau_2}^{U'}$ .  $\square$

**LEMMA 5 (COMPOSITION).** *If  $c : \mathbb{R}^n \rightarrow \mathbb{R}$  is PAP and  $g_1, \dots, g_n$  are PACAP on a c-analytic set  $U$ , then  $c \circ (g_1, \dots, g_n)$  is PACAP on  $U$ .*

PROOF. Since  $c$  is PAP, there is a finite partition  $\mathbb{R}^n = \bigsqcup_j P_j$  where each  $P_j$  is an analytic set with open set  $V_j$  and constraints  $h_{j,i} : V_j \rightarrow \mathbb{R}$ , together with analytic  $c_j : V_j \rightarrow \mathbb{R}$  agreeing with  $c$  on  $P_j$ . Let  $\{A_k\}_k$  be a common refinement of the PACAP partitions of  $g_1, \dots, g_n$ , obtained by iterated intersection (Lem. 2); by support enlargement (Lem. 1), on each  $A_k$  the tuple  $(g_1, \dots, g_n)$  is analytic on a common open set  $W_k \subseteq \mathbb{R}^{B_k}$ . Refine further:  $A_k \cap (g_1, \dots, g_n)^{-1}(P_j)$  is an analytic cylinder with support  $B_k$ , open set  $W_k \cap (g_1, \dots, g_n)|_{B_k}^{-1}(V_j)$  (open since  $(g_1, \dots, g_n)|_{B_k}$  is continuous), and constraints consisting of those of  $A_k$  together with  $h_{j,i} \circ (g_1, \dots, g_n)|_{B_k}$  (analytic, as a composition of analytic functions on open sets). On each such piece,  $c_j \circ (g_1, \dots, g_n)|_{B_k}$  is analytic on the same open set.  $\square$

**Fundamental lemma (2), from Thm. 2.** *If  $x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau$  and given  $c$ -analytic  $U$  and  $f_1 \in \mathcal{P}_{\tau_1}^U, \dots, f_n \in \mathcal{P}_{\tau_n}^U$ , then  $\lambda u. \llbracket e \rrbracket(f_1(u), \dots, f_n(u)) \in \mathcal{P}_{\tau}^U$ .*

PROOF. By induction on the typing derivation of  $x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau$ . Throughout, write  $\gamma(u) = (f_1(u), \dots, f_n(u))$  for the substituted valuation.

The real literal ( $\lambda u. r$  is constant, hence PACAP), variable ( $\lambda u. \llbracket x_j \rrbracket(\gamma(u)) = f_j$ ), and unit cases are immediate.

*Lambda.* If  $e = \lambda x : \tau_1. e'$  with  $\Gamma, x : \tau_1 \vdash e' : \tau_2$ . Take any  $c$ -analytic  $U' \subseteq U$  and  $g \in \mathcal{P}_{\tau_1}^{U'}$ . By restriction, each  $f_j|_{U'} \in \mathcal{P}_{\tau_j}^{U'}$ . By the induction hypothesis for  $e'$  at  $U'$ :  $\lambda u. \llbracket e' \rrbracket(f_1(u), \dots, f_n(u), g(u)) \in \mathcal{P}_{\tau_2}^{U'}$ . Since  $\llbracket \lambda x : \tau_1. e' \rrbracket(\gamma(u))(g(u)) = \llbracket e' \rrbracket(\gamma(u), g(u))$ , this shows  $\lambda u. \llbracket e \rrbracket(\gamma(u)) \in \mathcal{P}_{\tau_1 \rightarrow \tau_2}^U$ .

*Application.* If  $e = e_1 e_2$  with  $\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2$  and  $\Gamma \vdash e_2 : \tau_1$ . By the induction hypothesis,  $h_1 := \lambda u. \llbracket e_1 \rrbracket(\gamma(u)) \in \mathcal{P}_{\tau_1 \rightarrow \tau_2}^U$  and  $h_2 := \lambda u. \llbracket e_2 \rrbracket(\gamma(u)) \in \mathcal{P}_{\tau_1}^U$ . Since  $U$  is itself  $c$ -analytic, instantiating with  $U' = U$  and  $g = h_2$ :  $\lambda u. h_1(u)(h_2(u)) \in \mathcal{P}_{\tau_2}^U$ .

The pair and projection cases follow directly from the product-type clause.

*Injection.* If  $e = \text{inj}_i e'$ : by the induction hypothesis  $g := \lambda u. \llbracket e' \rrbracket(\gamma(u)) \in \mathcal{P}_{\tau_i}^U$ . The partition with  $U_i = U$  and  $U_j = \emptyset$  for  $j \neq i$  witnesses  $\lambda u. \llbracket e \rrbracket(\gamma(u)) \in \mathcal{P}_{\sum_{i \in I} \tau_i}^U$ , since  $\emptyset$  is  $c$ -analytic and  $\mathcal{P}_{\tau_j}^\emptyset$  is inhabited.

*Case.* If  $e = \text{case } e' \text{ of } (\text{inj}_i x \Rightarrow e_i)_{i \in I}$  with  $\Gamma \vdash e' : \sum_{i \in I} \tau_i$  and  $\Gamma, x : \tau_i \vdash e_i : \tau$  for each  $i \in I$ . By the induction hypothesis on  $e'$ :  $\lambda u. \llbracket e' \rrbracket(\gamma(u)) \in \mathcal{P}_{\sum_{i \in I} \tau_i}^U$ , giving  $U = \bigsqcup_{i \in I} U_i$  with each  $U_i$   $c$ -analytic and  $h_i \in \mathcal{P}_{\tau_i}^{U_i}$ . For each  $i$ : by restriction,  $f_j|_{U_i} \in \mathcal{P}_{\tau_j}^{U_i}$  for all  $j$ , and  $h_i \in \mathcal{P}_{\tau_i}^{U_i}$ . By the induction hypothesis for  $e_i$  at  $U_i$  with the extended valuation:  $\lambda u. \llbracket e_i \rrbracket(f_1(u), \dots, f_n(u), h_i(u)) \in \mathcal{P}_{\tau}^{U_i}$ . Since  $\llbracket e \rrbracket(\gamma(u)) = \llbracket e_i \rrbracket(\gamma(u), h_i(u))$  for  $u \in U_i$ , the result follows from gluing (Lem. 4).

*Constants.* A constant  $c : \text{real} \times \dots \rightarrow \text{real}$  denotes a total PAP function  $\mathbb{R}^n \rightarrow \mathbb{R}$  (Example 1). Take  $c$ -analytic  $U' \subseteq U$  and  $(g_1, \dots, g_n) \in \mathcal{P}_{\text{real} \times \dots}^{U'}$ , so each  $g_i$  is PACAP on  $U'$ . By the composition lemma (Lem. 5),  $\lambda u. c(g_1(u), \dots, g_n(u)) \in \mathcal{P}_{\text{real}}^{U'}$ .  $\square$

## B Encoding the Examples and Verifying PACAP

The core calculus in Sec. 3.1.2 has countable sum types and case analysis but no explicit fixpoint operator. We show here that this suffices to encode primitive recursion and stream corecursion, as used in Sec. 2.1. For the geometric distribution, which involves unbounded search, we verify the PACAP property of the resulting likelihood directly. These encodings are standard in type theory and programming language semantics, but we include them for completeness, in case the relationship between the Haskell examples and the core calculus may not be immediately apparent.

The key observation is that countable case expressions make the syntax *infinitary*: a single well-typed term may be an infinite (but countably branching) syntax tree. Haskell’s general recursion is used in practice to describe these infinite terms finitely, but the denotational semantics is defined on the infinite terms themselves.

*Primitive recursion over  $\mathbb{N}$ .* Since  $\mathbb{N} = \sum_{n \in \mathbb{N}} 1$  is a type in the calculus, countable case analysis directly provides primitive recursion. Given a base case  $z : \tau$  and a step function  $s : \mathbb{N} \rightarrow \tau \rightarrow \tau$  (both terms in the calculus), the primitive recursor  $\text{rec}(z, s) : \mathbb{N} \rightarrow \tau$  is:

$$\text{rec}(z, s) = \lambda n. \text{case } n \text{ of } (\text{inj}_{i-} \Rightarrow e_i)_{i \in \mathbb{N}}$$

where each branch  $e_i$  is the  $i$ -fold syntactic unrolling of  $s$ :  $e_0 = z$ ,  $e_1 = s \text{ inj}_0 () z$ ,  $e_2 = s \text{ inj}_1 () (s \text{ inj}_0 () z)$ , and so on. Each  $e_i$  is a finite well-typed term of type  $\tau$ , built by literally inlining  $s$  a total of  $i$  times. No fixpoint combinator is needed; the countable case expression directly enumerates all branches.

*Stream corecursion.* A stream of type  $\mathbb{N} \rightarrow \text{real}$  is a function from  $\mathbb{N}$  to  $\text{real}$ . Given a seed  $a : \sigma$ , a head function  $h : \sigma \rightarrow \text{real}$ , and a tail function  $t : \sigma \rightarrow \sigma$ , the corecursively defined stream  $\text{unfold}(a, h, t) : \mathbb{N} \rightarrow \text{real}$  is:

$$\text{unfold}(a, h, t) = \lambda n. \text{case } n \text{ of } (\text{inj}_{i-} \Rightarrow h(\underbrace{t \circ \dots \circ t}_{i \text{ times}}(a)))_{i \in \mathbb{N}}$$

That is, the  $i$ -th branch is the finite term  $h(t^i(a))$ , where  $t^i$  denotes  $i$ -fold syntactic composition of  $t$ .

*Models covered by the core calculus.* Using primitive recursion and stream corecursion, the following models from Sec. 2 and 6 can be encoded in the core calculus, and are therefore PACAP by Thm. 2:

- **Gaussian mixture clustering** (Sec. 2.3). The number of clusters is drawn from a Poisson distribution (a natural number, hence a countable sum type). The function  $\text{mapM}$  then recurses over the finite list  $[1..k]$ , which is primitive recursion over  $\mathbb{N}$ .
- **Step regression** (Listing 3). The Poisson point process  $\text{poissonPP}$  produces an infinite stream of changepoints by stream corecursion ( $\text{unfold}$  with seed = current position). The function  $\text{splice}$  constructs a piecewise function by consuming this stream, also corecursively.
- **Polynomial regression** (Sec. 6). The degree is drawn from a Poisson distribution, and the coefficients are sampled by primitive recursion over the resulting finite list.

*Models requiring direct verification.* Recall from Listing 1 that the geometric distribution samples an IID Bernoulli sequence and returns the index of the first success. For simplicity, we take  $\mathcal{A} = \mathbb{N}$  (the monadic seed-splitting into  $\mathbb{N}^*$  described in Sec. 3.2 does not affect the argument).

The stream  $\text{iid}(\text{bernoulli } p)$  is encoded by a  $\lambda$ -term that reads off coordinates:  $\lambda \omega. \lambda n. (\omega n < p)$ , of type  $(\mathbb{N} \rightarrow \text{real}) \rightarrow (\mathbb{N} \rightarrow \{\text{tt}, \text{ff}\})$ , using the PAP comparison constant. (Recall that booleans  $\{\text{tt}, \text{ff}\} = \sum_{b \in \{0,1\}} 1$  are a special case of countable sums.)

The functions  $\text{findIndex}$  and  $\text{find}$  perform unbounded searches and so cannot be expressed in the core calculus, which for simplicity has no general fixpoint operator. Consequently, Thm. 2 does not directly apply to programs built from these functions, including the geometric distribution (Listing 1) and the random walk model (Listing 2).

Nonetheless, the PACAP property of these likelihoods can be verified directly; we illustrate with the geometric distribution. The geometric distribution partitions the seed space  $\mathbb{R}^{\mathbb{N}}$  (up to a measure-zero set) into countably many analytic cylinders: for each  $i \in \mathbb{N}$ , the set  $\{\omega \mid \omega_0 \geq p, \dots, \omega_{i-1} \geq p, \omega_i < p\}$  is an analytic cylinder with support  $\{0, \dots, i\}$ , since the non-strict



constraints  $p - \omega_j \leq 0$  are analytic and the strict constraint  $\omega_i < p$  is absorbed into the open set  $V$  (Def. 1). The uncovered set  $\{\omega \mid \forall n. \omega_n \geq p\}$  has measure zero under any product measure with  $p \in (0, 1)$ . On each cylinder, the geometric distribution returns a fixed natural number, so any likelihood obtained by composing with a scoring function  $\mathbb{N} \rightarrow \text{real}$  is PACAP by definition. The same cylinder reasoning applies to any program whose likelihood factors through countably many deterministic branches.

More generally, programs involving unbounded search go beyond our core calculus. Formal verification of almost-sure termination is an active research area (e.g. [3]). Combining such formal guarantees with PACAP-ness would require an ‘omega-PACAP’ notion, analogous to the  $\omega$ -PAP spaces of [13], with recursion via chain-complete partial orders that are compatible with the PACAP structure. For the models considered in this paper, the geometric distribution and random walk involve unbounded search and require the direct cylinder partition verification illustrated above, while the clustering, step regression, and polynomial regression models use only bounded recursion or corecursion and are covered directly by Thm. 2.

## C Additional Proofs and iMCMC Kernel

### C.1 iMCMC Kernel

One iteration of the iMCMC algorithm (Sec. 4.2) yields the kernel:

$$\kappa(\mathbf{q}, A) = \int_{\mathbb{Y}} \left( [I(\mathbf{q}, \mathbf{p}) \in (A \times \mathbb{Y}) \cap \mathbb{S}] \tilde{\alpha}(\mathbf{q}, \mathbf{p}) + [( \mathbf{q}, \mathbf{p}) \in (A \times \mathbb{Y}) \cap \mathbb{S}] (1 - \tilde{\alpha}(\mathbf{q}, \mathbf{p})) \right) p_{\mathbf{q}}(\mathbf{p}) \mu_{\mathbb{Y}}(d\mathbf{p}). \quad (4)$$

Informally,  $\kappa(\mathbf{q}, A)$  tells us the probability of starting with a position  $\mathbf{q}$  and returning a position in  $A$ . The first term of the sum corresponds to accepting the proposed state  $I(\mathbf{q}, \mathbf{p})$ , and the second term corresponds to rejecting it and returning the same position  $\mathbf{q}$ .

### C.2 Proof of Prop. 3

PROOF. By Prop. 1, it is enough to let

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{\zeta(\mathbf{q}', \mathbf{p}')}{\zeta(\mathbf{q}, \mathbf{p})} \cdot \left( \frac{d(\text{Leb}_{2n} \circ (\Psi^{(L)})^{-1})}{d\text{Leb}_{2n}} \right) (\mathbf{q}, \mathbf{p}).$$

Using that the leapfrog integrator  $\Psi$  is volume preserving (Prop. 2), i.e.  $\frac{d(\text{Leb}_{2n} \circ (\Psi^{(L)})^{-1})}{d\text{Leb}_{2n}} (\mathbf{q}, \mathbf{p}) = 1$  we can simplify  $\alpha$ :

$$\alpha(\mathbf{q}, \mathbf{p}) = \frac{\zeta(\mathbf{q}', \mathbf{p}')}{\zeta(\mathbf{q}, \mathbf{p})} = \frac{\exp(-H(\mathbf{q}', \mathbf{p}'))}{\exp(-H(\mathbf{q}, \mathbf{p}))} = \frac{l(\mathbf{q}') \varphi_n(\mathbf{p}')}{l(\mathbf{q}) \varphi_n(\mathbf{p})}.$$

□

### C.3 Reduction of $\psi$ to $\tau$ on Sites with 0 Gradient

PROPOSITION 8. For  $\epsilon_q = \sin \theta$  and  $\epsilon_p = \tan(\theta/2)$ , the map  $\psi$  restricted to any site  $a$  with  $G(\mathbf{q})_a = 0$  equals the rotation  $\tau$ .

PROOF. At such a site  $\phi_t^P$  reduces to  $(q, p) \mapsto (q, p - tq)$ , so composing  $\psi = \phi_{\epsilon_p}^P \circ \phi_{\epsilon_q}^Q \circ \phi_{\epsilon_p}^P$  on  $(q_a, p_a)$  gives

$$\begin{pmatrix} q' \\ p' \end{pmatrix} = \begin{pmatrix} 1 - \epsilon_p \epsilon_q & \epsilon_q \\ -\epsilon_p(2 - \epsilon_p \epsilon_q) & 1 - \epsilon_p \epsilon_q \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix}.$$

Since  $\epsilon_p \epsilon_q = \tan(\theta/2) \sin \theta = 2 \sin^2(\theta/2) = 1 - \cos \theta$ , the diagonal entries equal  $\cos \theta$  and the top-right entry is  $\sin \theta$ . Moreover  $\epsilon_p(2 - \epsilon_p \epsilon_q) = \tan(\theta/2)(1 + \cos \theta) = \sin \theta$ , so the bottom-left entry is  $-\sin \theta$ . The matrix is therefore exactly  $\tau$ . □

#### C.4 Proof of Prop. 5

PROOF.

- Condition A3: Let  $I' = F' \circ \tau^L$ , where  $F' : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  is the momentum flip:  $F'(q, p) = (q, -p)$ . We get that  $I'(q, p) = (\cos(L\theta)q + \sin(L\theta)p, \sin(L\theta)q - \cos(L\theta)p)$ .  $I'$  is a measure preserving involution (with respect to  $\text{Leb}_2$ ) on  $\mathbb{R}^2$ . Moreover,  $(\varphi_X \times \varphi_Y)(q, p) = (\varphi_X \times \varphi_Y)(I'(q, p))$ .
- Condition A1: Let  $T$  be a finite subset of  $\mathcal{A}$  and let  $(\mathbf{q}, \mathbf{p}) \in \mathbb{S}$  with  $v(\mathbf{q}, \mathbf{p}) = T$ . This means that  $\bigcup_{i=0}^L v_I(\mathbf{q}^{(i)}) = T$ , where  $(\mathbf{q}^{(i)}, \mathbf{p}^{(i)}) = \psi^i(\mathbf{q}, \mathbf{p})$ . This implies that in order to compute  $G(\mathbf{q}^{(i)})_T$  one only needs to know  $\mathbf{q}_T^{(i)}$ . By induction we can prove that is enough to know  $\mathbf{q}_T, \mathbf{p}_T$  to compute  $\mathbf{q}_T^{(i)}$  and  $\mathbf{p}_T^{(i)}$ . So changing the values at any sites  $\mathcal{A} \setminus T$  in  $\mathbf{q}, \mathbf{p}$  would not change  $(\mathbf{q}_T^{(i)}, \mathbf{p}_T^{(i)})$ . Therefore if  $v(\mathbf{q}, \mathbf{p}) = T$  then  $\{(\mathbf{x}, \mathbf{y}) \in \mathbb{X} \times \mathbb{Y} \mid (\mathbf{x}_T, \mathbf{y}_T) = (\mathbf{q}_T, \mathbf{p}_T)\} \subseteq v^{-1}(T)$ . We can deduce that there exists a measurable set  $C_T \in \Sigma_{\mathcal{X}|T} \otimes \Sigma_{\mathcal{Y}|T}$  such that  $v^{-1}(T) = \{(\mathbf{q}, \mathbf{p}) \mid (\mathbf{q}_T, \mathbf{p}_T) \in C_T\}$ .
- Condition A2: Consider the following maps with domain and codomain included in  $\mathbb{R}^T$ :

$$\begin{aligned} \psi_T &= \phi_T^P \circ \phi_T^Q \circ \phi_T^P, & F_T(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, -\mathbf{p}) \\ \phi_T^P(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, \mathbf{p} - \epsilon_p \mathbf{q} - \epsilon_p G_T(\mathbf{q})), & \phi_T^Q(\mathbf{q}, \mathbf{p}) &= (\mathbf{q} + \epsilon_q \mathbf{p}, \mathbf{p}) \end{aligned}$$

Since  $v(\mathbf{q}, \mathbf{p}) = T$  then it is enough to know the value of  $\mathbf{q}^{(L)}$  at sites  $T$  to check if  $(F \circ \psi^L)(\mathbf{q}, \mathbf{p})$  is in  $\mathbb{S}$  or not. Hence, the map  $I_T : C_T \rightarrow C_T$  is well defined:

$$I_T(\mathbf{q}_T, \mathbf{p}_T) := \begin{cases} (F_T \circ \psi_T^L)(\mathbf{q}_T, \mathbf{p}_T) & \text{if } (F \circ \psi^L)(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \\ (\mathbf{q}_T, \mathbf{p}_T) & \text{otherwise.} \end{cases}$$

It is easy to check that  $I_T$  is a measure preserving involution on  $C_T$ .

Since  $G(\mathbf{q}^{(i)})_j = 0$  for any  $j \in \mathcal{A} \setminus T$  we get  $(\mathbf{q}_j^{(L)}, \mathbf{p}_j^{(L)}) = I'(\mathbf{q}_j, \mathbf{p}_j)$ . Let  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ , then by definition of  $I$ ,  $(\mathbf{q}'_j, \mathbf{p}'_j) = I'(\mathbf{q}_j, \mathbf{p}_j)$  for any  $j \in \mathcal{A} \setminus T$ .

□

#### C.5 Proof of Prop. 6

PROOF.

- Condition A3: On a site  $j$  with  $G(\mathbf{q}^{(i)})_j = 0$  for all  $i$ , the kick  $\phi_j^P(\mathbf{q}, \mathbf{p}) = (\mathbf{q}, \mathbf{p} - tG(\mathbf{q}))$  is the identity on the  $j$ -th coordinate, so  $\psi = \phi_{\theta/2}^P \circ \phi_{\theta}^{Q,P} \circ \phi_{\theta/2}^P$  acts there as its middle factor  $\phi_{\theta}^{Q,P}$ , i.e. the clockwise rotation  $\tau$  by  $\theta$ . Let  $I' = F' \circ \tau^L$ , where  $F'(q, p) = (q, -p)$  is the momentum flip. Then  $I'(q, p) = (\cos(L\theta)q + \sin(L\theta)p, \sin(L\theta)q - \cos(L\theta)p)$  is a measure preserving involution (with respect to  $\text{Leb}_2$ ) on  $\mathbb{R}^2$ . Since  $\tau$  is a rotation we get  $(\varphi_X \times \varphi_Y)(q, p) = (\varphi_X \times \varphi_Y)(I'(q, p))$ .
- Condition A1: As in App. C.4, using that the integrator acts site-wise. Let  $T$  be a finite subset of  $\mathcal{A}$  and let  $(\mathbf{q}, \mathbf{p}) \in \mathbb{S}$  with  $v(\mathbf{q}, \mathbf{p}) = T$ . Both  $\phi^P$  and  $\phi^{Q,P}$  act site-wise, coupled across sites only through the gradient  $G$ , which is supported in  $T$ . By induction it is enough to know  $\mathbf{q}_T, \mathbf{p}_T$  to compute  $\mathbf{q}_T^{(i)}$  and  $\mathbf{p}_T^{(i)}$ , so changing the values at sites  $\mathcal{A} \setminus T$  does not change  $(\mathbf{q}_T^{(i)}, \mathbf{p}_T^{(i)})$ . Therefore there exists a measurable set  $C_T \in \Sigma_{\mathcal{X}|T} \otimes \Sigma_{\mathcal{Y}|T}$  such that  $v^{-1}(T) = \{(\mathbf{q}, \mathbf{p}) \mid (\mathbf{q}_T, \mathbf{p}_T) \in C_T\}$ .
- Condition A2: Let  $\psi_T, F_T, \phi_T^P, \phi_T^{Q,P} : \mathbb{R}^{2|T|} \rightarrow \mathbb{R}^{2|T|}$  be the maps

$$\begin{aligned} \psi_T &= \phi_T^P \circ \phi_T^{Q,P} \circ \phi_T^P, & F_T(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, -\mathbf{p}) \\ \phi_T^P(\mathbf{q}, \mathbf{p}) &= (\mathbf{q}, \mathbf{p} - \frac{\theta}{2} G_T(\mathbf{q})), & \phi_T^{Q,P}(\mathbf{q}, \mathbf{p}) &= (\cos \theta \mathbf{q} + \sin \theta \mathbf{p}, -\sin \theta \mathbf{q} + \cos \theta \mathbf{p}). \end{aligned}$$

Both  $\phi_T^P$  and  $\phi_T^{Q,P}$  are volume preserving so  $\psi_T$  is also volume preserving. Since  $v(\mathbf{q}, \mathbf{p}) = T$ , it is enough to know  $\mathbf{q}^{(L)}$  at sites  $T$  to check whether  $(F \circ \psi^L)(\mathbf{q}, \mathbf{p}) \in \mathbb{S}$ , so the map  $I_T : C_T \rightarrow C_T$  is well defined:

$$I_T(\mathbf{q}_T, \mathbf{p}_T) := \begin{cases} (F_T \circ \psi_T^L)(\mathbf{q}_T, \mathbf{p}_T) & \text{if } (F \circ \psi^L)(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \\ (\mathbf{q}_T, \mathbf{p}_T) & \text{otherwise,} \end{cases}$$

and it is a measure preserving involution on  $C_T$ . Since  $G(\mathbf{q}^{(i)})_j = 0$  for any  $j \in \mathcal{A} \setminus T$ , we get  $(\mathbf{q}_j^{(L)}, \mathbf{p}_j^{(L)}) = I'(\mathbf{q}_j, \mathbf{p}_j)$ . Letting  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ , by definition of  $I$  we have  $(\mathbf{q}'_j, \mathbf{p}'_j) = I'(\mathbf{q}_j, \mathbf{p}_j)$  for any  $j \in \mathcal{A} \setminus T$ .  $\square$

### C.6 Proof of Prop. 7

PROOF. Condition A3 holds by construction:  $I'$  is chosen as a  $\text{Leb}_2$ -preserving involution with  $(\varphi_X \times \varphi_Y) \circ I' = \varphi_X \times \varphi_Y$ . Condition A1 is proved by the same induction on the trajectory index as in Prop. 5: the gradient at each step is supported in  $T = v(\mathbf{q}, \mathbf{p})$ , so  $(\mathbf{q}_T^{(i)}, \mathbf{p}_T^{(i)})$  depends only on  $(\mathbf{q}_T, \mathbf{p}_T)$  and  $v^{-1}(T)$  is a cylinder set. Condition A2 is built into the definition of  $I$ : on the visited coordinates  $I$  restricts to the finite-dimensional leapfrog involution  $I_T = F_T \circ \psi_T^L$  (measure-preserving by Prop. 2), and on each unvisited coordinate  $I$  acts independently as  $I'$ .  $\square$

## D Framework B: Generalizing Framework A from Lazy HMC towards Lazy NUTS

In this section we present a slightly more general framework than the one presented in Sec. 4.4. This framework can still be seen as an instance of the iMCMC framework (Sec. 4.2). The idea is to have multiple involutions and to sample an extra variable that tells us which involution to use. One could see this as the GIST framework [6] which uses Gibbs sampling to tune the HMC hyperparameters to get locally adaptive HMC. NUTS [12] is also part of GIST. We also show in App. D.1 how we can construct lazyNUTS on rose trees as part of Framework B.

*Tuning parameters random variable.* We augment the state space  $\mathbb{S}$  with the random variable  $\beta$  which selects the involution to be applied on the state  $(\mathbf{q}, \mathbf{p})$ . Let  $(\mathbb{B}, \Sigma_{\mathbb{B}}, \mu_{\mathbb{B}})$  be a  $\sigma$ -finite measure space and for any  $\mathbf{s} \in \mathbb{S}$  let  $p(\cdot | \mathbf{s})$  be the probability density with respect to  $\mu_{\mathbb{B}}$  from which  $\beta$  is sampled given  $\mathbf{s}$ . Let  $W(\mathbf{s}, \beta) = w(\mathbf{s})p(\beta | \mathbf{s})$  and  $\mathbb{Z} = \{(\mathbf{s}, \beta) \in \mathbb{S} \times \mathbb{B} \mid w(\mathbf{s})p(\beta | \mathbf{s}) > 0\}$  equipped with  $\sigma$ -algebra  $\Sigma_{\mathbb{Z}} = \{A \cap \mathbb{Z} \mid A \in \Sigma_{\mathbb{S}} \otimes \Sigma_{\mathbb{B}}\}$  and measure  $\mu_{\mathbb{Z}} = \mu_{\mathbb{S}} \times \mu_{\mathbb{B}}$ . We will usually denote with  $\mathbf{s}$  a position-momentum pair  $(\mathbf{q}, \mathbf{p})$ .

*Involutions.* Our main involution  $I$  is now defined on  $\mathbb{Z}$ . We assume  $I : \mathbb{Z} \rightarrow \mathbb{Z}$  is measurable and we can write it as  $I(\mathbf{s}, \beta) = (g(\mathbf{s}, \beta), h(\mathbf{s}, \beta))$  with measurable  $h : \mathbb{Z} \rightarrow \mathbb{B}$  and measurable bijection  $g : \mathbb{Z} \rightarrow \mathbb{S}$  with measurable inverse. Then, under certain assumptions on measurability and absolute continuity (see Prop. 13 in the Appendix), we have :

$$\left( \frac{d\mu_{\mathbb{Z}} \circ I^{-1}}{d\mu_{\mathbb{Z}}} \right) (\mathbf{s}, \beta) = \left( \frac{d(\mu_{\mathbb{S}} \circ g_{t_s^{-1}(\beta)}^{-1})}{d\mu_{\mathbb{S}}} \right) (\mathbf{s}) \left( \frac{d(\mu_{\mathbb{B}} \circ t_s^{-1})}{d\mu_{\mathbb{B}}} \right) (\beta), \quad (5)$$

where  $g_{\beta}(\mathbf{s}) = g(\mathbf{s}, \beta)$  and  $t_s(\beta) = h(g_{\beta}(\mathbf{s})^{-1}, \beta)$ .

One iteration of a method from Framework B given position  $\mathbf{q}$ ,  $p(\cdot | \mathbf{q}, \mathbf{p})$ ,  $\mu_X, \mu_Y, l, I, \alpha$  does:

- (1) sample  $\mathbf{p} \sim \mu_Y$  lazily, sampling each component on-demand
- (2) sample  $\beta \sim p(\cdot | \mathbf{q}, \mathbf{p})$
- (3) let  $(\mathbf{q}', \mathbf{p}', \beta') = I(\mathbf{q}, \mathbf{p}, \beta)$

(4) with probability  $\min\{1, \alpha(\mathbf{q}, \mathbf{p}, \beta)\}$  accept and return  $\mathbf{q}'$ , otherwise reject and return  $\mathbf{q}$ .

Let  $\alpha(\mathbf{s}, \beta) = \frac{W(I(\mathbf{s}, \beta))}{W(\mathbf{s}, \beta)} \left( \frac{d\mu_{\mathbb{Z}} \circ I^{-1}}{d\mu_{\mathbb{Z}}} \right) (\mathbf{s}, \beta)$ , then by Prop. 1 we get the following.

**PROPOSITION 9.** *The kernel resulting from Framework B is stationary with respect to the target  $\nu$ .*

We will now focus on the particular case in which  $h$  depends only on  $\beta$  for any  $(\mathbf{s}, \beta) \in \mathbb{Z}$ , so  $h : \mathbb{B} \rightarrow \mathbb{B}$  and  $I(\mathbf{s}, \beta) = (g(\mathbf{s}, \beta), h(\beta))$ .

Moreover, assume that for each  $\beta$ , there exists  $v_\beta : \mathbb{S} \rightarrow \mathcal{P}_{fin}(\mathcal{A})$  such that  $g(\cdot, \beta)$  is a measure preserving involution that satisfies the conditions stated in Conditions A1 to A3 from Sec. 4.4.

**PROPOSITION 10.** *Under the extra assumptions for  $h$  and  $g$  and using the following acceptance ratio in Framework B:*

$$\alpha(\mathbf{s}, \beta) = \frac{W(I(\mathbf{s}, \beta))}{W(\mathbf{s}, \beta)} \frac{(\varphi_X \times \varphi_Y)(g(\mathbf{s}, h^{-1}(\beta))_{v_{h^{-1}(\beta)}(\mathbf{s})})}{(\varphi_X \times \varphi_Y)(\mathbf{s}_{v_{h^{-1}(\beta)}(\mathbf{s})})} \left( \frac{d(\mu_{\mathbb{B}} \circ h^{-1})}{d\mu_{\mathbb{B}}} \right) (\beta)$$

*we get a stationary kernel with respect to the target measure  $\nu$ .*

**PROOF.** From Eq. (5) and the fact that  $I(\mathbf{s}, \beta) = (g(\mathbf{s}, \beta), h(\beta))$  we get:

$$\left( \frac{d\mu_{\mathbb{Z}} \circ I^{-1}}{d\mu_{\mathbb{Z}}} \right) (\mathbf{s}, \beta) = \left( \frac{d(\mu_{\mathbb{S}} \circ g_{h^{-1}(\beta)}^{-1})}{d\mu_{\mathbb{S}}} \right) (\mathbf{s}) \left( \frac{d(\mu_{\mathbb{B}} \circ h^{-1})}{d\mu_{\mathbb{B}}} \right) (\beta).$$

Since for each  $\beta$ , there exists  $v_\beta : \mathbb{S} \rightarrow \mathcal{P}_{fin}(\mathcal{A})$  such that  $g(\cdot, \beta)$  is a measure preserving involution that satisfies the conditions stated in Conditions A1 to A3 from Sec. 4.4, we can use Prop. 4:

$$\left( \frac{d\mu_{\mathbb{Z}} \circ I^{-1}}{d\mu_{\mathbb{Z}}} \right) (\mathbf{s}, \beta) = \frac{(\varphi_X \times \varphi_Y)(g(\mathbf{s}, h^{-1}(\beta))_{v_{h^{-1}(\beta)}(\mathbf{s})})}{(\varphi_X \times \varphi_Y)(\mathbf{s}_{v_{h^{-1}(\beta)}(\mathbf{s})})} \left( \frac{d(\mu_{\mathbb{B}} \circ h^{-1})}{d\mu_{\mathbb{B}}} \right) (\beta). \quad (6)$$

Then by Prop. 1 and Eq. (6) we get the following:

$$\alpha(\mathbf{s}, \beta) = \frac{W(I(\mathbf{s}, \beta))}{W(\mathbf{s}, \beta)} \frac{(\varphi_X \times \varphi_Y)(g(\mathbf{s}, h^{-1}(\beta))_{v_{h^{-1}(\beta)}(\mathbf{s})})}{(\varphi_X \times \varphi_Y)(\mathbf{s}_{v_{h^{-1}(\beta)}(\mathbf{s})})} \left( \frac{d(\mu_{\mathbb{B}} \circ h^{-1})}{d\mu_{\mathbb{B}}} \right) (\beta).$$

□

## D.1 LazyNUTS: A No-U-Turn Sampler over Infinite Dimensional State Spaces

Tuning the hyperparameter  $L$  of number of steps in the HMC algorithm is not trivial. If  $L$  is too small, then we might get stuck in a local minimum and not explore the whole state space. However, if  $L$  is too large, we might explore the same area multiple times in one iteration, which is not efficient. NUTS [12] aims to solve this by adaptively setting the path length  $L$  using the doubling procedure with the no-U-turn criterion as the stopping condition.

*The doubling procedure.* Given the initial position-momentum state  $\mathbf{s}$ , the idea is to construct a set of proposed states  $C_s$  from which we sample the next state according to the weights of the states in  $C_s$ . In order to preserve detailed balance,  $C_s$  is constructed by recursively doubling the trajectory either forwards in time (with probability 1/2) or backwards in time (with probability 1/2) until the stopping conditions are met.

*Property of  $C_s$ .* The stopping condition must be chosen such that for any  $(\mathbf{s}') \in C_s$ , the sets  $C_s$  and  $C_{\mathbf{s}'}$  are the same. That means the proposed states set resulting from the doubling procedure would be the same if we were to start with any other state in the set and do the doubling procedure.

*The no-U-turn criterion.* NUTS stops the doubling procedure if the trajectory constructed so far makes a U-turn. Let  $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{2n}$  and  $(\mathbf{x}', \mathbf{y}') \in \mathbb{R}^{2n}$  be two states in the trajectory so that the direction is from  $(\mathbf{x}, \mathbf{y})$  to  $(\mathbf{x}', \mathbf{y}')$ , i.e.  $(\mathbf{x}', \mathbf{y}')$  can be obtained by performing some  $i$  leapfrog steps from  $(\mathbf{x}, \mathbf{y})$ . The two states satisfy the U-turn criterion if:

$$(\mathbf{x}' - \mathbf{x}) \cdot \mathbf{y} < 0 \text{ or } (\mathbf{x}' - \mathbf{x}) \cdot \mathbf{y}' < 0.$$

This tells that if we were to move a very tiny step forwards from  $\mathbf{x}'$  with momentum  $\mathbf{y}'$  or a very tiny step backwards from  $\mathbf{x}$  with momentum  $\mathbf{y}$ , we would end up reducing the distance between the positions  $\mathbf{x}$  and  $\mathbf{x}'$ . Therefore, a U-turn is made, so the doubling procedure should be stopped. This is checked for certain pairs of states, in such a way that detailed balance holds. For rigorous details and proof of correctness see [12].

*LazyNUTS setup.* Let  $(\mathbb{B}, \Sigma_{\mathbb{B}}, \mu_{\mathbb{B}})$  be the measure space of the integers together with the counting measure. The hyperparameter will be a number of leapfrog steps, so in this case  $\beta$  will be  $L$ . Writing NUTS as part of Framework B,  $L$  represents how many leapfrog steps and in which direction we should go from the initial state in order to get to the proposed state. In practice this corresponds to performing the doubling procedure and choosing the next state according to the correct weights. Hence,  $L$  would be sampled at the end of the doubling procedure from the categorical distribution defined by the weights of the states that are part of the set of proposed states.

*Constructing the involution.* Let  $I^{(L)} = F \circ \psi^L$  be an involution with  $\psi$  as defined in Sec. 4.5 together with the corresponding visited sites function  $v_L(\mathbf{q}, \mathbf{p}) = \cup_{i=0}^L v_i(\mathbf{q}^{(i)})$ , where  $(\mathbf{q}^{(i)}, \mathbf{p}^{(i)}) = \psi^i(\mathbf{q}, \mathbf{p})$ . The conditions in Conditions A1 to A3 hold for  $I^{(L)}$ . Notice that the definition of  $I^{(L)}$  also makes sense for  $L < 0$ :  $I^{(L)} = F \circ \psi^L = \psi^{-L} \circ F$ . Therefore, the involution  $I^{(L)}$  represents  $L$  leapfrog steps followed by negation of the momentum in the direction of the momentum if  $L > 0$ , otherwise, we first negate the momentum and move  $L$  leapfrog steps. Since  $(I^{(L)})^{-1} = I^{(-L)}$  we get the involution  $I : \mathbb{S} \times \mathbb{B} \rightarrow \mathbb{S} \times \mathbb{B}$  with  $I(\mathbf{q}, \mathbf{p}, L) = (I^{(L)}(\mathbf{q}, \mathbf{p}), -L)$ .

*Distribution of  $L$ .* Suppose we start with an initial state  $(\mathbf{q}, \mathbf{p})$  and we perform the doubling procedure. Consider the set  $C_{(\mathbf{q}, \mathbf{p})}$  resulting from the doubling procedure with a suitable stopping condition. Let  $V(C_{(\mathbf{q}, \mathbf{p})}) = \cup_{(\mathbf{q}', \mathbf{p}') \in C_{(\mathbf{q}, \mathbf{p})}} v_L(\mathbf{q}')$  be the set of visited sites in at least one position in  $C_{(\mathbf{q}, \mathbf{p})}$ . We can define the distribution of  $L$  as:

$$p(L \mid \mathbf{q}, \mathbf{p}) = \frac{l(\mathbf{q}^{(L)}) \prod_{j \in V(C_{(\mathbf{q}, \mathbf{p})})} \varphi_X(\mathbf{q}_j^{(L)}) \varphi_Y(\mathbf{p}_j^{(L)})}{\sum_{(\mathbf{q}', \mathbf{p}') \in C_{(\mathbf{q}, \mathbf{p})}} l(\mathbf{q}') \prod_{j \in V(C_{(\mathbf{q}, \mathbf{p})})} \varphi_X(\mathbf{q}'_j) \varphi_Y(\mathbf{p}'_j)}$$

if  $(\mathbf{q}^{(L)}, \mathbf{p}^{(L)}) \in C_{(\mathbf{q}, \mathbf{p})}$  and  $p(L \mid \mathbf{q}, \mathbf{p}) = 0$  otherwise, where  $(\mathbf{q}^{(L)}, \mathbf{p}^{(L)}) = I^{(L)}(\mathbf{q}, \mathbf{p})$ .

**PROPOSITION 11.** *Suppose the stopping condition is start-independent, i.e.  $C_{s'} = C_s$  (hence  $V(C_{s'}) = V(C_s)$ ) for every  $s' \in C_s$ . Then the acceptance ratio from Prop. 10 is 1.*

**PROOF.** Let  $h(L) = -L$ , then  $h$  is measure preserving with respect to  $\mu_{\mathbb{B}}$ . Let  $L$  be such that  $I^{(L)}((\mathbf{q}, \mathbf{p})) \in C_{(\mathbf{q}, \mathbf{p})}$ . The acceptance ratio from Prop. 10 can be written as:

$$\begin{aligned} \alpha(s, L) &= \frac{W(I(s, L)) (\varphi_X \times \varphi_Y)(I^{(L)}(s, h^{-1}(L))_{v_{h^{-1}(L)}(s)})}{W(s, L) (\varphi_X \times \varphi_Y)(s_{v_{h^{-1}(L)}(s)})} \\ &= \frac{l(\mathbf{q}^{(L)}) l(\mathbf{q}) \prod_{j \in V(C_{(\mathbf{q}^{(L)}, \mathbf{p}^{(L)})})} \varphi_X(\mathbf{q}_j) \varphi_Y(\mathbf{p}_j) \prod_{j \in v_L(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_j^{(L)}) \varphi_Y(\mathbf{p}_j^{(L)})}{l(\mathbf{q}) l(\mathbf{q}^{(L)}) \prod_{j \in V(C_{(\mathbf{q}, \mathbf{p})})} \varphi_X(\mathbf{q}_j^{(L)}) \varphi_Y(\mathbf{p}_j^{(L)}) \prod_{j \in v_L(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_j) \varphi_Y(\mathbf{p}_j)} \\ &= 1 \end{aligned}$$

The last step followed from the fact that  $v_L(\mathbf{q}, \mathbf{p}) \subset V(C_{(\mathbf{q}, \mathbf{p})})$  and for any  $j \in V(C_{(\mathbf{q}, \mathbf{p})}) \setminus v_L(\mathbf{q}, \mathbf{p})$  we have  $\varphi_X(\mathbf{q}_j^{(L)})\varphi_Y(\mathbf{p}_j^{(L)}) = \varphi_X(\mathbf{q}_j)\varphi_Y(\mathbf{p}_j)$ . Here we also use  $V(C_{(\mathbf{q}^{(L)}, \mathbf{p}^{(L)})}) = V(C_{(\mathbf{q}, \mathbf{p})})$ , which is the start-independence assumption.  $\square$

This corresponds to the fact that there is no more accept/reject state after the doubling procedure (since we sample one of the states in the constructed set and then return it). One iteration of lazyNUTS does the following:

- sample  $\mathbf{p} \sim \mu_Y$  lazily, sampling each component on-demand
- construct  $C_{(\mathbf{q}, \mathbf{p})}$  using the doubling procedure with suitable stopping conditions.
- sample the next state  $(\tilde{\mathbf{q}}, \tilde{\mathbf{p}})$  from the distribution that assigns for each state  $(\mathbf{q}', \mathbf{p}') \in C_{(\mathbf{q}, \mathbf{p})}$  the (unnormalized) weight  $l(\mathbf{q}') \prod_{j \in V(C_{(\mathbf{q}, \mathbf{p})})} \varphi_X(\mathbf{q}'_j)\varphi_Y(\mathbf{p}'_j)$
- return  $\tilde{\mathbf{q}}$ .

*No-U-turn condition in LazyNUTS.* To get a stopping condition of the doubling procedure similar to the no-U-turn condition we must take into consideration only finitely many dimensions on which to check if a U-turn was made. In the case in which we know the visited sites for a state, like in Sec. 4.7, the no-U-turn condition takes into consideration all the dimensions corresponding to the visited sites. However, if we do not have access to all the visited sites, we could still check if a U-turn was made using the sites for which the two positions have non-zero gradient. This is the stopping condition that we used in our implementation together with an upper bound  $M$  on the number of proposed states, usually  $M = 2^6$ .

Prop. 11 assumes a start-independent set of proposed states. Our implementation checks the no-U-turn criterion only on sites with non-zero gradient and additionally caps the number of proposed states at  $M$ . The gradient-restricted U-turn check is intended to preserve this invariance; the cap  $M$ , however, is a practical termination (analogous to the maximum tree depth in standard NUTS implementations) and, like the maximum tree depth, is not guaranteed to preserve it. When  $M$  does not bind, the construction coincides with the uncapped doubling for which the invariance holds.

## E Technical Propositions

PROPOSITION 12.

$$\left( \frac{d(\mu_{\mathbb{S}} \circ I^{-1})}{d\mu_{\mathbb{S}}} \right) (\mathbf{q}, \mathbf{p}) = \frac{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_i)\varphi_Y(\mathbf{p}'_i)}{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_i)\varphi_Y(\mathbf{p}_i)},$$

( $\mu_{\mathbb{S}}$  - almost everywhere), where  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ .

PROOF. Let  $r(\mathbf{q}, \mathbf{p}) = \frac{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}'_i)\varphi_Y(\mathbf{p}'_i)}{\prod_{i \in v(\mathbf{q}, \mathbf{p})} \varphi_X(\mathbf{q}_i)\varphi_Y(\mathbf{p}_i)}$ , where  $(\mathbf{q}', \mathbf{p}') = I(\mathbf{q}, \mathbf{p})$ .

Let  $\mathcal{I}$  be a finite subset of  $\mathcal{A}$ ,  $B \in \Sigma_{\mathbb{R}^{2|\mathcal{I}|}}$  and let  $A = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \mid (\mathbf{q}_{\mathcal{I}}, \mathbf{p}_{\mathcal{I}}) \in B\}$ , where  $\mathbf{q}_{\mathcal{I}}$  are the values of the tree  $\mathbf{q}$  at nodes  $\mathcal{I}$ . Our aim is to prove that for all such finite cylinder sets  $A$  we have:

$$(\mu_{\mathbb{S}} \circ I^{-1})(A) = \int_A r(\mathbf{s})\mu_{\mathbb{S}}(d\mathbf{s}). \quad (7)$$

We can partition the state space  $\mathbb{S} = \cup_{T \in \mathcal{P}_{fin}(\mathcal{A})} \mathbb{S}_T$ , where  $\mathbb{S}_T = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \mid v(\mathbf{q}, \mathbf{p}) = T\}$ .

We will first fix  $T \in \mathcal{P}_{fin}(\mathcal{A})$  and show that Eq. (7) holds for  $A \cap \mathbb{S}_T$ . Using property A1 we can write  $\mathbb{S}_T = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \mid (\mathbf{q}_T, \mathbf{p}_T) \in C\}$  for some set  $C \in \Sigma_{\mathbb{R}^{2|T|}}$ .

Let  $\mathcal{J} = \mathcal{I} \cup T$  and  $n = |\mathcal{J}|$ . There exists  $D \in \Sigma_{\mathbb{R}^{2n}}$  such that  $A \cap \mathbb{S}_T = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \mid (\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D\}$ . From property A2 we have that  $I^{-1}(A \cap \mathbb{S}_T) = \{(\mathbf{q}, \mathbf{p}) \in \mathbb{S} \mid (\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in f^{-1}(D)\}$ , where  $f : \mathbb{R}^{2n} \rightarrow \mathbb{R}^{2n}$  is given by  $f(\mathbf{q}, \mathbf{p})_T = I_T(\mathbf{q}, \mathbf{p})$  and  $f(\mathbf{q}, \mathbf{p})_j = I'(\mathbf{q}_j, \mathbf{p}_j)$  for any  $j \in \mathcal{J} \setminus T$ . Therefore:

$$(\mu_{\mathbb{S}} \circ I^{-1})(A \cap \mathbb{S}_T) = (\mu_{X^n} \times \mu_{Y^n})(f^{-1}(D))$$



Notice that for  $\mathbf{q}, \mathbf{p} \in A \cap \mathbb{S}_T$ ,  $r(\mathbf{q}, \mathbf{p})$  only depends on the values  $(\mathbf{q}_T, \mathbf{p}_T)$ . Using Fubini/Tonelli we get:

$$\int_{A \cap \mathbb{S}_T} r(\mathbf{s}) \mu_{\mathbb{S}}(d\mathbf{s}) = \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D] \frac{\prod_{i \in T} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in T} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)} \mu_{X^n}(d\mathbf{q}_{\mathcal{J}}) \mu_{Y^n}(d\mathbf{p}_{\mathcal{J}})$$

From properties A2, A3 if  $v(\mathbf{q}, \mathbf{p}) = T$  we have that  $\varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i) = \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)$  for any  $i \in \mathcal{A} \setminus T$ . Hence,

$$\begin{aligned} \int_{A \cap \mathbb{S}_T} r(\mathbf{s}) \mu_{\mathbb{S}}(d\mathbf{s}) &= \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D] \frac{\prod_{i \in T} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in T} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)} \mu_{X^n}(d\mathbf{q}_{\mathcal{J}}) \mu_{Y^n}(d\mathbf{p}_{\mathcal{J}}) \\ &= \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D] \frac{\prod_{i \in \mathcal{J}} \varphi_X(\mathbf{q}'_i) \varphi_Y(\mathbf{p}'_i)}{\prod_{i \in \mathcal{J}} \varphi_X(\mathbf{q}_i) \varphi_Y(\mathbf{p}_i)} \mu_{X^n}(d\mathbf{q}_{\mathcal{J}}) \mu_{Y^n}(d\mathbf{p}_{\mathcal{J}}) \\ &= \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D] \frac{(\varphi_{X^n} \times \varphi_{Y^n})(f(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}))}{\varphi_{X^n}(\mathbf{q}_{\mathcal{J}}) \varphi_{Y^n}(\mathbf{p}_{\mathcal{J}})} \mu_{X^n}(d\mathbf{q}_{\mathcal{J}}) \mu_{Y^n}(d\mathbf{p}_{\mathcal{J}}) \\ &= \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in D] (\varphi_{X^n} \times \varphi_{Y^n})(f(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}})) \text{Leb}_{2n}(d\mathbf{q}_{\mathcal{J}}, d\mathbf{p}_{\mathcal{J}}) \\ &= \int_{\mathbb{R}^{2n}} [(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \in f^{-1}(D)] (\varphi_{X^n} \times \varphi_{Y^n})(\mathbf{q}_{\mathcal{J}}, \mathbf{p}_{\mathcal{J}}) \text{Leb}_{2n}(d\mathbf{q}_{\mathcal{J}}, d\mathbf{p}_{\mathcal{J}}) \\ &= (\mu_{X^n} \times \mu_{Y^n})(f^{-1}(D)) \end{aligned}$$

The second to last step is a change of variable using the measure preserving bijection  $f^{-1}$  (as both  $I_T$  and  $I'$  are measure preserving involutions) with respect to the Lebesgue measure on  $\mathbb{R}^{2n}$ .

Finally,

$$\begin{aligned} (\mu_{\mathbb{S}} \circ I^{-1})(A) &= (\mu_{\mathbb{S}} \circ I^{-1})(\cup_{T \in \mathcal{P}_{fin}(\mathcal{A})} A \cap \mathbb{S}_T) \\ &= \mu_{\mathbb{S}}(\cup_{T \in \mathcal{P}_{fin}(\mathcal{A})} I^{-1}(A \cap \mathbb{S}_T)) \\ &= \sum_{T \in \mathcal{P}_{fin}(\mathcal{A})} \mu_{\mathbb{S}}(I^{-1}(A \cap \mathbb{S}_T)) \\ &= \sum_{T \in \mathcal{P}_{fin}(\mathcal{A})} \int_{A \cap \mathbb{S}_T} r(\mathbf{s}) \mu_{\mathbb{S}}(d\mathbf{s}) \\ &= \int_A r(\mathbf{s}) \mu_{\mathbb{S}}(d\mathbf{s}). \end{aligned}$$

The sets for which this equality holds is a  $\lambda$ -system which contains the finite cylinder sets; the finite cylinder sets form a  $\pi$ -system generating  $\Sigma_{\mathbb{S}}$ , so by the  $\pi$ - $\lambda$  theorem the equality holds for all  $A \in \Sigma_{\mathbb{S}}$ . □

**PROPOSITION 13.** *Let  $(X, \Sigma_X, \mu_X)$  and  $(Y, \Sigma_Y, \mu_Y)$  be two  $\sigma$ -finite measure spaces and let  $(Z, \Sigma_Z, \mu_Z)$  be the product measure space with  $Z = X \times Y$ . Let  $g : X \times Y \rightarrow X$  and  $h : X \times Y \rightarrow Y$  and involution  $f : X \times Y \rightarrow X \times Y$  with  $f(x, y) = (g(x, y), h(x, y))$  be measurable maps. Let  $g_y : X \rightarrow X$  with  $g_y(x) = g(x, y)$  be a measurable bijection with measurable inverse for each  $y \in Y$  and assume that the Radon-Nikodym derivative  $\frac{d(\mu_X \circ g_y^{-1})}{d\mu_X}(x)$  exists (for all  $y \in Y$ ) and is measurable (as a map from  $Z$  to  $\mathbb{R}_{\geq 0}$ ). Then we have the following:*

- (1) *The function  $t_x(y) = h(g_y^{-1}(x), y)$  is a bijection on  $Y$  for each  $x \in X$*

- (2) Assume the maps  $(x, y) \mapsto t_x(y)$  and  $(x, y) \mapsto t_x^{-1}(y)$  are  $\Sigma_Z$ -measurable and the Radon-Nikodym derivative  $\left(\frac{d(\mu_Y \circ t_x^{-1})}{d\mu_Y}\right)(y)$  exists (for all  $x \in X$ ) and is measurable (as a map from  $Z$  to  $\mathbb{R}_{\geq 0}$ ). Then the following Radon-Nikodym derivative exists:

$$\left(\frac{d((\mu_X \times \mu_Y) \circ f^{-1})}{d\mu_X \times \mu_Y}\right)(x, y) = \left(\frac{d(\mu_X \circ g_{t_x^{-1}(y)}^{-1})}{d\mu_X}\right)(x) \left(\frac{d(\mu_Y \circ t_x^{-1})}{d\mu_Y}\right)(y).$$

- (3) If  $h(x, y) = h(y)$  for all  $(x, y) \in Z$  then the Radon-Nikodym derivative simplifies to:

$$\left(\frac{d((\mu_X \times \mu_Y) \circ f^{-1})}{d\mu_X \times \mu_Y}\right)(x, y) = \left(\frac{d(\mu_X \circ g_{h^{-1}(y)}^{-1})}{d\mu_X}\right)(x) \left(\frac{d(\mu_Y \circ h^{-1})}{d\mu_Y}\right)(y).$$

PROOF. (1) Notice that since  $f$  is an involution it follows that it is bijective. Fix  $x \in X$ .

To prove that  $t_x$  is injective let  $t_x(y_1) = t_x(y_2)$ . Since  $g(g_{y_1}^{-1}(x), y_1) = x$  then  $f(g_{y_1}^{-1}(x), y_1) = (x, h(g_{y_1}^{-1}(x), y_1)) = (x, t_x(y_1))$ . Similarly,  $f(g_{y_2}^{-1}(x), y_2) = (x, t_x(y_2))$ . Since  $(x, t_x(y_1)) = (x, t_x(y_2))$  we get  $f(g_{y_1}^{-1}(x), y_1) = f(g_{y_2}^{-1}(x), y_2)$ . By the injectivity of  $f$  we get that  $y_1 = y_2$ , so  $t_x$  is injective.

Let  $y \in Y$ . Since  $f$  is surjective, there exists  $(u, z) \in X \times Y$  such that  $f(u, z) = (x, y)$ . This implies  $g(u, z) = x$ . Then  $t_x(z) = h(g_z^{-1}(x), z) = h(u, z) = y$ . Therefore,  $t_x$  is surjective.

- (2) Let  $A \in \Sigma_X$  and  $B \in \Sigma_Y$ . It is enough to show that for any such  $\mu_Z$ -measurable set  $A \times B$  we have:

$$((\mu_X \times \mu_Y) \circ f^{-1})(A \times B) = \int_{A \times B} \left(\frac{d(\mu_X \circ g_{t_x^{-1}(y)}^{-1})}{d\mu_X}\right)(x) \left(\frac{d(\mu_Y \circ t_x^{-1})}{d\mu_Y}\right)(y) (\mu_X \times \mu_Y)(dx, dy)$$

for all such sets  $A \times B \in \Sigma_Z$ .

$$\begin{aligned} ((\mu_X \times \mu_Y) \circ f^{-1})(A \times B) &= \int_Y \int_X [g(x, y) \in A] [h(x, y) \in B] \mu_X(dx) \mu_Y(dy) \\ &= \int_Y \left( \int_X [g_y(x) \in A] [h(x, y) \in B] \mu_X(dx) \right) \mu_Y(dy) \end{aligned}$$

Performing a change of variable  $u = g_y(x)$  and using the existence of  $\left(\frac{d(\mu_X \circ g_y^{-1})}{d\mu_X}\right)$ :

$$\begin{aligned} ((\mu_X \times \mu_Y) \circ f^{-1})(A \times B) &= \int_Y \left( \int_X [u \in A] [h(g_y^{-1}(u), y) \in B] (\mu_X \circ g_y^{-1})(du) \right) \mu_Y(dy) \\ &= \int_Y \left( \int_X [u \in A] [h(g_y^{-1}(u), y) \in B] \left(\frac{d(\mu_X \circ g_y^{-1})}{d\mu_X}\right)(u) \mu_X(du) \right) \mu_Y(dy) \\ &= \int_Y \left( \int_A [t_u(y) \in B] \left(\frac{d(\mu_X \circ g_y^{-1})}{d\mu_X}\right)(u) \mu_X(du) \right) \mu_Y(dy) \end{aligned}$$

Swapping the integrals (using Tonelli), then performing a change of variables  $z = t_u(y)$  and using the existence of  $\left(\frac{d(\mu_Y \circ t_u^{-1})}{d\mu_Y}\right)$  we get:

$$\begin{aligned}
((\mu_X \times \mu_Y) \circ f^{-1})(A \times B) &= \int_A \left( \int_Y [z \in B] \left( \frac{d(\mu_X \circ g_{t_u^{-1}(z)}^{-1})}{d\mu_X} \right) (u) (\mu_Y \circ t_u^{-1})(dz) \right) \mu_X(du) \\
&= \int_A \left( \int_Y [z \in B] \left( \frac{d(\mu_X \circ g_{t_u^{-1}(z)}^{-1})}{d\mu_X} \right) (u) \left( \frac{d(\mu_Y \circ t_u^{-1})}{d\mu_Y} \right) (z) \mu_Y(dz) \right) \mu_X(du) \\
&= \int_A \int_B \left( \frac{d(\mu_X \circ g_{t_u^{-1}(z)}^{-1})}{d\mu_X} \right) (u) \left( \frac{d(\mu_Y \circ t_u^{-1})}{d\mu_Y} \right) (z) \mu_Y(dz) \mu_X(du) \\
&= \int_A \int_B \left( \frac{d(\mu_X \circ g_{t_x^{-1}(y)}^{-1})}{d\mu_X} \right) (x) \left( \frac{d(\mu_Y \circ t_x^{-1})}{d\mu_Y} \right) (y) \mu_Y(dy) \mu_X(dx)
\end{aligned}$$

(3) If  $h(x, y) = h(y)$  then  $t_x = h$  so the result follows directly from the previous part.

□

Received 2026-02-19; accepted 2026-05-13