École des Ponts ParisTech
Department of Statistics, University of Oxford

2017
Master's Internship Report

Émile Mathieu
Élève-ingénieur, Third year

Bayesian Nonparametric Inference within Probabilistic Programming Languages

Internship carried out at Department of Statistics, University of Oxford
From the 22nd of May, to the 15th of September 2017.

Company tutor: Teh, Yee Whye
Training supervisor: Obozinski, Guillaume

Acknowledgments

First of all, I would like to express my indebtedness appreciation to my departmental supervisor Prof. Yee Whye Teh. His belief in me and his advices played a decisive role in making the execution of my work and thus this report.

I also express my deepest thanks to Benjamin Bloem-Reddy, who as a postdoctoral, oversaw me during this internship and with whom I have continuously worked.

Moreover, my gratitude goes to Guillaume Obozinski, my school training supervisor, whose guidance has continually shaped my career path since I have been at Ecole des Ponts ParisTech.

Abstract

On one side, Bayesian Nonparametric (BNP) models have gained attraction because of their flexibility. These models, which automatically adapt with the number and complexity of data, avoid having to define *a priori* the number of parameters of the model, such as the number of components for a mixture model. On the other side, Probabilistic Programming Languages (PPLs) allow practitioners to express probabilistic models in a universal way, and bring generic inference algorithms. These systems avoid designing specific inference schemes, which is error-prone and time consuming. Specific representations of BNP models must be used so as to denote such models in PPLs, since BNP models live in infinite dimensional space and machines only have finite memory and computational resources.

In this report, we review well-known BNP models with a focus on mixture models and discrete random probability measures, but we also give background on the design of PPLs and associated inference schemes. We utilize generative construction of BNPs and show that more generic BNP classes than the Dirichlet Process can be represented in PPLs. We prototype our approach by contributing to an existing PPL.

Keywords : Probabilistic Programming, Bayesian Non-parametric, Bayesian Inference, Generative process, Sampling methods.

Contents

Introduction

For data science practitioners, statistical inference is typically just one step in a more elaborate analysis workflow. The first stage of this work involves data acquisition, pre-processing and cleaning. This is often followed by several iterations of exploratory model design and testing of inference algorithms. Once a sufficiently robust statistical model and a corresponding inference algorithm have been identified, analysis results must be post-processed, visualized, and in some cases integrated into a wider production system. Probabilistic programming systems [38, 35, 58, 92, 31] represent generative models as programs written in a specialized language that provides syntax for the definition and conditioning of random variables. The code for such models is generally concise, modular, and easy to modify or extend. Inference can be performed for any probabilistic program using one or more generic inference techniques provided by the system back end. These systems therefore avoid having to design specific inference schemes, which is error-prone and time consuming.

In the previously described data scientist's workflow, model selection is often a particularly difficult step. Most scientists address this problem by first fitting several models, with different numbers of clusters or factors, and then selecting one using model comparison metrics [15]. Model selection metrics usually include two terms. The first term measures how well the model fits the data. The second term, a complexity penalty, favours simpler models (i.e., ones with fewer components or factors). Bayesian Nonparametrics models provide a different approach to this problem. Rather than comparing models that vary in complexity, the BNP approach is to fit a single model that can adapt its complexity to the data. Indeed, they allow the complexity to grow as more data are observed, such as when using a model to perform prediction. This is an attractive property for many settings. Nonetheless, such models cannot be used out-of the box within Probabilistic Programming Languages since they have by definition at least one infinite dimensional component and computers only have finite memory and computational resources. Consequently, specific representations of these models must be constructed so as to denote them in probabilistic programming systems. Our work focuses on generative constructions of such models, and on developing efficient algorithms to perform inference within PPLs.

After giving some background on the Department of Statistics in Chapter 1 we develop on the context and organisation of the project. We then review in Chapter 4, well-known Bayesian Nonparametric models, with a focus on infinite mixture models and discrete random probability measures. This chapter also highlights the tailored representations of BNP models used by Markov chain Monte Carlo samplers in the literature. After that in

Chapter 5, we review the design of Probabilistic Programming Languages and describe the framework in which inference schemes can be generically applied for these programs. We then recall a generative construction of Poisson Kingman Processs in Chapter 6, and stress out how it relates with a specific class of PPLs. We also implement this generative process, along with a mixture model in an already existing PPL, and run experiments to assess the performance of posterior samplers. Finally, in Chapters 7 and 8 we conclude an point out possible directions for future work.

The main contributions of this report are the following:

- Being able to sample from some discrete random probability measures in PPLs, and therefore represent infinite mixture models.
- Efficiently sample from posterior distributions of hidden variables in infinite mixture models.

Presentation of the Deparment of Statistics

2.1 Creation

The Department of Statistics [1] is part of the University of Oxford, along with the other departments and the 38 constituent colleges. The University of Oxford was founded in the 11th century, which makes it the oldest university in the English-speaking world and the world's second-oldest university in continuous operation.

The Department of Statistics was officially created in 1988, even though first moves in the development of Oxford statistics can be dated to the 19th century.

Indeed, In the 1870s, Florence Nightingale – the pioneer of modern nursing – discussed the possibility of endowing a Professorship of Statistics in Oxford, but the proposal eventually foundered. However, Oxford did appoint a statistician to a chair in 1891, although not to a chair in statistics.

The next significant moves in the development of Oxford statistics were by economists, who were increasingly keen to build economic theory on a foundation of sound data analysis. This led to the creation in 1935 of an Institute of Statistic, swhich was then renamed as the Institute of Economics and Statistics in 1962.

The sequence of events which led directly to the establishment of the present Department of Statistics began with the appointment in 1945 of David Finney as the universitys first Lecturer in the Design and Analysis of Scientific Experiment (LIDASE).

Then in the 1980s, after the Department of Biomathematics' head increasingly felt that Oxford was losing out in the face of developments in statistics, a working party appointed by the general board of the university to assess a careful analysis of the organisation of statistics in Oxford. They found fragmentation to be the dominant feature of Oxford statistics and concluded that fragmentation has serious disadvantages ..... The working partys report recommended the creation of a university statistics department, which were to include the former Department of Biomathematics, together with a new Professorship in Statistical Science and the two existing lecturerships in statistics within the Mathematical Institute.

[1] `https://www.stats.ox.ac.uk`

These major recommendations were all accepted by the university and the new Department of Statistics was created in 1988.

2.2 Activities

The Department of Statistics at Oxford is a world leader in research including computational statistics and statistical methodology, applied probability, bioinformatics and mathematical genetics. The main research groups in the Department are Computational statistics and machine learning, Probability, Statistical genetics and bioinformatics, Protein Informatics and Statistical Genetics.

I am part of the Computational Statistics and Machine Learning Group (OxCSML) [2], which have research interests spanning Statistical Machine Learning, Monte Carlo Methods and Computational Statistics, Statistical Methodology and Applied Statistics.

The department offers an undergraduate degree (BA or MMath) in Mathematics and Statistics, jointly with the Mathematical Institute. At postgraduate level there is an MSc course in Applied Statistics (MSc in Statistical Science from 2017), as well as a lively and stimulating environment for postgraduate research (DPhil or MSc by Research). The department also has a consulting activity called *Oxford University Statistical Consulting.*

[2] `http://csml.stats.ox.ac.uk/people/mathieu/`

Mission

3.1 Themes of research

Prof. Yee Whye Teh [1] has worked for a long time on inference sampling schemes for BNP mixture models [27, 26, 53, 54], but also on stick-breaking constructions [82, 25]. He also has recently been interested in PPL and consequently in inference schemes within PPL for BNP models. This theme requires knowledge in several fields – Probabilities, Computational Statistics, Programming Languages – which makes it deeply interesting. He proposed me working with him on this topic as part of a 3-years DPhil program, and to start earlier as an intern.

3.2 Context

In addition to inviting me to work with him, Prof. Yee Whye Teh also opened two postdoctoral positions for working on the same project, which have been filled by Tom Rainforth and Benjamin Bloem-Reddy. Tom Rainforth [2] is finishing is third year of DPhil in the Dept. of Engineering Science in Oxford, supervised by Prof. Frank Wood. His interests include probabilistic programming, Bayesian optimization, probabilistic numerics, sequential Monte Carlo and particle Markov Chain Monte Carlo methods. He will join the group in October, but he has already attended several reading group meetings. On the other hand, Benjamin Bloem-Reddy [3] arrived in May in Oxford and has already started working on the project. He was supervised by Peter Orbanz at Columbia University, and his research were focused on probabilistic and statistical analysis of networks and other discrete data.

3.3 Reading group

Four reading groups are organised with a bi-weekly period: Kernel methods, Deep Learning, Bayesian Nonparametrics and Probabilistic Inference. I have been leading the Prob-

[1] https://www.stats.ox.ac.uk/~teh
[2] http://www.robots.ox.ac.uk/~twgr
[3] http://www.stats.ox.ac.uk/~bloemred/

abilistic Inference reading group [4] since July. Since, Ben and I have presented four papers [77, 85, 78, 17] with an emphasise on probabilist programming.

3.4 Organisation

In this section I develop my current organisation and workflow as a researcher. At first, I had much trouble to organise my workflow, I wrote my papers' review and new ideas on flying sheets, papers were saved in my computer's folders, citations for report was time-consuming, my code was locally saved, etc... Thus, I worked on a better workflow and after trials and errors, I eventually arrived on what I describe below. I aim to modify this process with time, so as to continually enhance my productivity and be able to focus on the interesting part of the job.

3.4.1 Managing papers

My biggest trouble was keeping organised the dozens of new articles I read each week. I was saving them in a tree-like structure of folders, but with the number of articles saved growing, it became more and more difficult to find specific article. Moreover, this structure inherently prohibits cross-categories articles which is annoying for a project situated at the intersections of several fields. Furthermore, I had no fast way to cite an article, neither in plain format (for markdown [5] files for instance) nor in *BibTeX* format.

Then, I heard of papers managing library such as Papers3 [6] or Mendeley [7]. I have eventually opted for Papers3 but Mendeley is also a popular choice in the academic community. These applications features many tools easing the life of a researcher, the main one being from my point of view:

- Synchronisation: between multiple computers or devices.
- Multi-labels: these are used in the search tool.
- Local search: search in titles, authors, labels and even papers' content.
- Online search: can import articles in a fast manner by being connected with online search engines such as *arXiv*.
- Collections: create a reading list, or group of papers which can be cited at once
- Citations: get *BibTeX* reference or *BibTeX* cite command in clipboard

3.4.2 Managing research

Another of my organizational issue was keeping track of ideas. I happened to find that research is a result of a long chain of ideas which were continually iterated upon. I am now maintaining a single *master document* for keeping tracks of this chain of ideas.

[4] `https://github.com/BigBayes/oxsml/wiki/Probabilistic-Inference-meetings`
[5] `https://en.wikipedia.org/wiki/Markdown`
[6] `https://www.readcube.com/papers/mac/`
[7] `https://www.mendeley.com`

It has a bulleted list of all ideas, problems, and topics that Id like to think more carefully about. This list is succinct but subsequent sections go in depth on particular entries. This list is sorted according to what Id like to work on next, but I continually revise my priorities according to whether I think the direction aligns with my broader research vision, and if I think the direction is necessarily impactful for the community at large.

3.4.3 Managing projects

Then, when an idea has matured enough and I have seriously started working on it, I create a Github [8] repository for the project. Each project has its separated repository. It contains a `/readme.md` file maintaining a list of todos, with also questions (and sometimes answers!) both for myself and collaborators. This makes it transparent how to keep moving forward and whats blocking the work. There is also a `/doc/` folder for all the write-ups, usually in LaTeXformat. The `etc/` folder is used for everything not relevant to other directories such as pictures of whiteboards during conversations about the project. Finally, the `/src/` folder is where all code is written. Runnable scripts are written directly in `/src/` , and classes and utilities are written in `/src/codebase/`.

3.4.4 Writing scientific documents

Concerning the writing of scientific documents, LaTeXis the language of choice. Yet the commonly used editor for OSX – TeXShop – is missing some useful features. Hopefully, LaTeXTools [9], a LaTeX plugin for Sublime Text [10] has been developed. It has the following useful features; (i) a build command to compile LaTeX, (ii) forward and inverse search with PDF previewers, (iii) fill everything including references, citations, packages, graphics, figures, etc, (iv) smart command completion for a variety of text and math commands, (v) full support for project files and multi-file documents. I have gained in productivity since I started using Sublime with this plugin instead of TeXShop.

[8] `https://github.com`
[9] `https://github.com/SublimeText/LaTeXTools`
[10] `https://www.sublimetext.com`

Bayesian nonparametrics

4.1 Definition

A Bayesian nonparametric model is a model that (i) constitutes a Bayesian model (a statistical model with parameters being random variables) on an infinite-dimensional parameter space and (ii) can be evaluated on a finite sample in a manner that uses only a finite subset of the available parameters to explain the sample. The parameter space in (i) typically consists of functions or of measures, while (ii) is usually achieved by marginalizing out surplus dimensions over the prior. Random functions and measures, and more generally probability distributions on infinite-dimensional random objects, are called stochastic processes.

4.2 Motivation

Most scientists address the model selection problem by first fitting several models, with different numbers of clusters or factors, and then selecting one using model comparison metrics [15]. Model selection metrics usually include two terms. The first term measures how well the model fits the data. The second term, a complexity penalty, favors simpler models (i.e., ones with fewer components or factors).

BNP models provide a different approach to this problem. Rather than comparing models that vary in complexity, the BNP approach is to fit a single model that can adapt its complexity to the data. BNP models allow the complexity to grow as more data are observed, such as when using a model to perform prediction. This is an attractive property for many settings.

4.2.1 Exchangeability

The underlying assumption of all Bayesian methods is that the parameter specifying the observation model is a random variable. However, there is a very general type of so-called *exchangeable* observations for which the existence of such a random variable is a mathematical consequence of the data's properties. This is an important notion since all models we will be considering in Chapter 6 are assuming exchangeability.

Formally, a sequence of variables $X_1, \dots, X_n$ over the same probability space $(\mathbb{X}, \Omega)$ is *exchangeable* if their joint distribution is invariant to permuting the variables. That is, if for any permutation σ of $\{1, \dots, n\}$, then

$$\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) = \mathbb{P}(X_1 = x_{\sigma(1)}, \dots, X_n = x_{\sigma(n)})$$

An infinite sequence $X_1, X_2, \dots$ is *infinitely exchangeable* if $X_1, \dots, X_n$ is *exchangeable* for every $n \geq 1$. Exchangeability reflects the assumption that the variables do not depend on their indices although they may be dependent among themselves. This is typically a reasonable assumption in machine learning and statistical applications, even if the variables are not themselves iid, since it is a much weaker assumption.

If θ parametrizes the underlying distribution, and one assumes a prior distribution over θ, then the resulting marginal distribution over $X_1, \dots, X_n$ with θ marginalized out will still be exchangeable. A fundamental result credited to de Finetti [16] states that the converse is also true. That is, if $X_1, \dots, X_n$ is (infinitely) exchangeable, then there is a random θ such that:

$$P(X_1, \dots, X_n) = \int P(\theta) \prod_{i=1}^{n} P(X_i|\theta) d\theta \tag{4.1}$$

In other words, the seemingly innocuous assumption of exchangeability automatically implies the existence of a hierarchical Bayesian model with θ being the random latent parameter.

In de Finetti's Theorem it is important to stress that θ can be infinite dimensional (it is typically a random measure), thus the hierarchical Bayesian model 4.1 can be a nonparametric one.

4.3 Examples

Bayesian nonparametric models have recently been applied to a variety of machine learning problems, including regression, classification, clustering, latent variable modeling, sequential modeling, and others.

4.3.1 Dirichlet Process

The Dirichlet Process (DP) is a distribution over distributions. It is parameterized by a concentration parameter $\alpha > 0$ and a base distribution H_0, which is a distribution over a space $\mathbb{X}$. A random variable drawn from a Dirichlet Process (DP) is itself a distribution over $\mathbb{X}$. A random distribution P drawn from a DP is denoted $P \sim \mathrm{DP}(\alpha, H_0)$.

The DP was first developed in [30], who showed its existence by appealing to its finite dimensional distributions. Consider a measurable partition of $\mathbb{X}$, $\{A_1, \dots, A_K\}$ If $P \sim \mathrm{DP}(\alpha, H_0)$ then every measurable partition of $\mathbb{X}$ is Dirichlet-distributed,

$$(P(A_1), \dots, P(A_K)) \sim \mathrm{Dir}(\alpha H_0(A_1), \dots, \alpha H_0(A_K)).$$

This means that if we draw a random distribution from the DP and add up the probability mass in a region $A \in \mathbb{X}$, then there will on average $H_0(A)$ mass in that region. The concentration parameter plays the role of an inverse variance; for higher values of α, the random probability mass $P(A)$ will concentrate more tightly around $H_0(A)$.

[30] proved two properties of the Dirichlet process. The first property is that random distributions drawn from the Dirichlet process are discrete. They place their probability mass on a countably infinite collection of points, called atoms,

$$P = \sum_{k=1}^{\infty} p_k \delta_{X_k^\star}. \tag{4.2}$$

where p_k is the probability assigned to the kth atom and $X_k^\star$ is the location or value of that atom. Further, these atoms are drawn independently from the base distribution H_0.

The second property connects the Dirichlet process to the Chinese restaurant process. Consider a random distribution drawn from a DP followed by repeated draws from that random distribution,

$$P \sim \mathrm{DP}(\alpha, H_0) \tag{4.3}$$
$$X_i \sim P \quad \forall i = \{1, \ldots, n\} \tag{4.4}$$

Ferguson [30] examined the joint distribution of $X_{1:n}$, which is obtained by marginalizing out the random distribution P,

$$p(X_1, \ldots, X_n | \alpha, H_0) = \int \left(\prod_{i=1}^{n} p(X_i|P) \right) d\mathbb{P}(P|\alpha, H_0) \tag{4.5}$$

Ferguson showed that, under this joint distribution, the X_i will exhibit a *clustering property*–they will share repeated values with positive probability. The structure of shared values defines a partition of the integers from 1 to n, and the distribution of this partition is a Chinese restaurant process with parameter α. Finally, he showed that the unique values of X_i shared among the variables are independent draws from H_0.

The stick-breaking construction Ferguson [30] proved that the DP exists via its finite dimensional distributions. Sethuraman [81] provided a constructive definition based on the stick-breaking representation.

Consider a stick with unit length. We divide the stick into an infinite number of segments $\tilde{p}_k$ by the following process. First, choose a beta random variable $Z_1 \sim \mathrm{beta}(1, \alpha)$ and break off V_1 of the stick. For each remaining segment, choose another beta distributed random variable, and break off that proportion of the remainder of the stick. This gives us an infinite collection of weights

$$V_k \sim \mathrm{Beta}(1, \alpha) \tag{4.6}$$
$$\tilde{p}_k = V_k \prod_{j=1}^{k-1} (1 - V_j) \quad k = 1, 2, \ldots \tag{4.7}$$

Finally, we construct a random distribution using Equation 4.2, where we take an infinite number of draws from a base distribution H_0 and draw the weights as in Equation 4.7.

Sethuraman [81] showed that the distribution of this random distribution is a $DP(\alpha, H_0)$. He also showed that the sequence of weights $(\tilde{p}_k)_{k\geq 1}$ is a size-biased permutation of the DP's weights $(p_k)_{k\geq 1}$, i.e. the bigger is a weight p_k, the sooner it will be sampled by the previously described process.

Chinese Restaurant Process The Chinese Restaurant Process (CRP) is a distribution over infinite partitions of the integers [73, 2]. The CRP derives its name from the following metaphor. Imagine a restaurant with an infinite number of tables [1], and imagine a sequence of customers entering the restaurant and sitting down. The first customer enters and sits at the first table. The second customer enters and sits at the first table with probability $\frac{1}{1+\alpha}$, and the second table with probability $\frac{\alpha}{1+\alpha}$, where α is a positive real. When the nth customer enters the restaurant, he sits at each of the occupied tables with probability proportional to the number of previous customers sitting there, and at the next unoccupied table with probability proportional to α. At any point in this process, the assignment of customers to tables defines a random partition.

More formally, let z_n be the table assignment of the nth customer. A draw from this distribution

$$P(z_n = k|\mathbf{z}_{1:n-1}) \propto \begin{cases} \frac{m_k}{n-1+\alpha} & \text{if } k \leq K_+ \text{ (i.e., k is a previously occupied table)} \\ \frac{\alpha}{n-1+\alpha} & \text{otherwise (i.e., k is the next unoccupied table)} \end{cases}$$

where m_k is the number of customers sitting at table k, and K_+ is the number of tables for which $m_k > 0$. The CRP can also be obtained by marginalising out the random measure P.

Dirichlet process mixtures A DP mixture adds a third step to the model above: $Y_i \sim f(\cdot|X_i)$. Marginalizing out P reveals that the DP mixture is equivalent to a CRP mixture. Moreover, the stick-breaking representation is a generative process enabling to lazily sample according to the associated DP, in a *size-biased* order. We will talk more about this size-biased of the DP atoms in Proposition 1 and its discussion.

4.3.2 Nonlinear regression

The aim of regression is to infer a continuous function from a training set consisting of input-output pairs $\{(t_i, x_i)\}_{i=1}^n$. Parametric approaches parametrize the function using a finite number of parameters and attempt to infer these parameters from data. The prototypical Bayesian nonparametric approach to this problem is to define a prior distribution over continuous functions directly by means of a Gaussian Process (GP). As explained in [75], a GP is a distribution on an infinite collection of random variables X_t, such that the joint distribution of each finite subset $X_{t_1}, \dots, X_{t_n}$ is a multivariate Gaussian. A value x_t taken by the variable X_t can be regarded as the value of a continuous function f at t, that is, $f(t) = x_t$. Given the training set, the Gaussian process posterior is again a distribution on functions, conditional on these functions taking values $f(t_1) = x_1, \dots, f(t_n) = x_n$.

[1]The Chinese restaurant metaphor is due to Pitman and Dubins, who were inspired by the seemingly infinite seating capacity of Chinese restaurants in San Francisco.

4.3.3 Latent feature models

Latent feature models represent a set of objects in terms of a set of latent features, each of which represents an independent degree of variation exhibited by the data. Such a representation of data is sometimes referred to as a distributed representation. A Bayesian nonparametric approach to latent feature modeling allows for an unknown number of latent features. The canonical stochastic processes involved here are known as the Indian Buffet Process (IBP) [33, 82] and the Beta Process (BP). Draws from BPs are random discrete measures, where each of an infinite number of atoms has a mass in $(0, 1)$ but the masses of atoms need not sum to 1. Yet, their sum is finite almost surely. Each atom corresponds to a feature, with the mass corresponding to the probability that the feature is present for an object. We can visualize the occurrences of features among objects using a binary matrix, where the (i, k) entry is 1 if object i has feature k and 0 otherwise. The distribution over binary matrices induced by the BP is called the IBP. Thibaux and Jordan [83] showed that a particular subclass of Beta processes is to the IBP as the DP is to the CRP.

4.3.4 Hidden Markov models

Hidden Markov Models (HMMs) are popular models for sequential or temporal data, where each time step is associate with a state, with state transitions dependent on the previous state. An infinite HMM [5] is a Bayesian nonparametric approach to HMMs, where the number of states is unbounded and allowed to grow with the sequence length. It is defined using one DP prior for the transition probabilities going out from each state. To ensure that the set of states reachable from each outgoing state is the same, the base distributions of the DPs are shared and given a DP prior recursively. The construction is called an Infinite Hierarchical Dirichlet Process (HDP).

4.3.5 Density estimation

A nonparametric Bayesian approach to density estimation [23] requires a prior on densities or distributions. However, the DP is not useful in this context, since it generates discrete distributions. A useful density estimator should smooth the empirical density (such as a Parzen window estimator), which requires a prior that can generate smooth distributions. Priors applicable in density estimation problems include DP mixture models and Pólya trees.

4.4 Mixture models

4.4.1 Finite dimensional

Mixture models provide a statistical framework for modeling data where each observation is assumed to have arisen from one of k groups, with k possibly unknown, and each group being suitably modeled by a distribution function from some parametric family. We refer to the monographs by Titterington et al. [84] and McLachlan and Basford [60] for accounts on mixture models with a fixed number of components. The distribution function of each group is referred to as a component of the mixture model and is weighted by the relative frequency of the group in the population. Specifically, assuming k being fixed, a collection of observations $Y_1, \dots, Y_n$ is modeled as independent draws from a mixture distribution function with k components, that is,

$$Y_i \sim \sum_{j=1}^{k} p_j F(\cdot|X_j), \tag{4.8}$$

where $(X_1, \dots, X_k)$ are parameters associated with component j, $F(\cdot|X)$ is a given parametric family of distribution functions indexed by a parameter X and $(p_1, \dots, p_k)$ are the mixture proportions constrained to be nonnegative and sum to unity. Thus, finite mixture models define a density function over data items y of the form $p(y) = \sum_{k=1}^{k} p_j f(y|X_j)$, where f is the density of F. The density can be written in a non-standard manner as an integral: $p(y) = \int F(y|X)P(X)dX$, where $P = \sum_{j=1}^{k} p_j \delta_{X_j}$ is a discrete mixing distribution encapsulating all the parameters of the mixture model and δ_X is a Dirac distribution (atom) centered at X.

A convenient formulation of the mixture model (4.8) can be stated in terms of latent allocation random variables, namely, each observation Y_i is assumed to arise from a specific but unknown component Z_i of the mixture model. Accordingly, an augmented version of (4.8) can be written in terms of a collection of latent random variables $(Z_1, \dots, Z_n)$, independent and identically distributed with probability mass function $\mathbb{P}[Z = j] = p_j$, such that the observations are modeled as

$$Y_i|Z_i \sim F(\cdot|X_{Z_i}). \tag{4.9}$$

Integrating out the random variables $(Z_1, \dots, Z_n)$ then yields (4.8). In a Bayesian setting the formulation of the mixture model (4.9) is completed by specifying suitable prior distributions for the unknown quantities that are objects of the inferential analysis: the parameter $(X_1, \dots, X_k)$ and the vector of proportions $(p_1, \dots, p_k)$.

4.4.2 Infinite dimensional

Bayesian Nonparametric generalizations of finite mixture models provide an approach for estimating both the number of components in a mixture model and the parameters of the individual mixture components simultaneously from data [64]. As opposed to finite mixtures, BNP mixtures use mixing distributions consisting of a *countably infinite* number of atoms:

$$P = \sum_{j=1}^{\infty} p_j \delta_{X_j^\star} \tag{4.10}$$

This gives rise to mixture models with an infinite number of components. When applied to a finite training set, only a finite (but varying) number of components will be used to model the data, since each data item is associated with exactly one component but each component can be associated with multiple data items. Inference in the model then automatically recovers both the number of components to use and the parameters of the components. Being Bayesian, we need a prior over the mixing distribution P. Discrete Random Probability Measures are of the form of 4.10, defined on a suitable measurable space $\mathbb{X}$ with $X_i \in \mathbb{X}$ for $i = 1, \dots$. The most common prior to use is a

DP. Consequently, the mixture components would be drawn accordingly to the base distribution of the DP.

We are therefore interested in hierarchical models of the form

$$\begin{gathered} P \text{ a Discrete Random Probability Measure,} \\ X_i|P \sim P, \\ Y_i|X_i \sim F(\cdot|X_i) \end{gathered}$$

4.5 Random probability measures

In this section, we explore classes of nonparametric priors for our previously defined mixture model. There are two possible random discrete distributions that can be obtained from a specific class of discrete random measures called Completely Random Measures (CRMs). Informally, a Completely Random Measure (CRM) is a discrete random measure with an independence property. One of them is the well known random discrete distributions called Normalised Random Measures with Independent Increments and the other is the class of Poisson-Kingman Random Probability Measures. Both are obtained from CRMs after a suitable normalisation operation. Poisson Kingman Process (PKP) specify a distribution for the total mass T, whereas Normalised Random Measures (NRMs) use the "natural" distribution of T induced by their Lévy measure ρ.

4.5.1 Normalised Random Measure

We start with a description of CRMs. See the monograph by Kingman [46] for a good reference on such a topic.

Definition 1 (Completely Random Measure)**.** *Let $\mathbb{X}$ be a complete and separable metric space endowed with the Borel σ-field $\mathcal{B}(\mathbb{X})$. A Completely Random Measure μ is a random element taking values on the space of boundedly finite measures on $\mathbb{X}$ such that, for any collection of measurable subsets, $A_1, \ldots, A_n$ in $\mathcal{B}(\mathbb{X})$, with $A_i \cap A_j = \emptyset \ \forall i \neq j$, the random variables $\mu(A_1), \ldots, \mu(A_n)$ are mutually independent.*

CRMs were first proposed and studied by Kingman [45], who showed that a CRM μ can always be decomposed into a sum of three independent parts

$$\mu = \mu_0 + \sum_{k \geq 1} J_k \delta_{X_k^\star} + \sum_{l=1}^{N} v_l \delta_{\psi_l}$$

where μ_0 is a (non-random) measure over $\mathbb{X}$, $\{\psi_l\}_{l \in [N]} \subset \mathbb{X}$ is a collection of N, $1 \leq N \leq \infty$, atoms at fixed locations and independent random masses $(v_l)_{l \in [N]}$, and $(J_k, X_k^\star)_{k \geq 1}$ is a collection of atoms with random masses and random locations.

In applications of Bayesian nonparametrics it is usually assumed that $\mu_0 = 0$ and $N = 0$, so that μ consists only of the atoms with random masses and locations. However, the posterior distribution of μ given data would typically contain atoms at fixed locations, hence, the usefulness of the larger class of CRMs.

The distribution of the random atoms $(X_k^\star)_{k\geq 1}$ and their masses $(J_k)_{k\geq 1}$ under a CRM is characterized by the Lévy-Khintchine representation of its Laplace functional transform [45]. Specifically,

$$\mathbb{E}\left[e^{-\int g(y)\mu(dy)}\right] = \exp\left\{-\int_{\mathbb{R}^+\times\mathbb{X}}\left(1-e^{-sg(y)}\right)\rho(ds)H_0(dy)\right\}, \tag{4.11}$$

for any measurable function $g:\mathbb{X}\to\mathbb{R}$ such that $\int_{\mathbb{X}}|g(x)|\mu(dx) < +\infty$ almost-surely. The underlying measure $\nu = \rho\times H_0$ uniquely characterises the random atoms in μ.

The only intensity measures that are considered herein are those that factorise $\nu(ds,dy) = \rho(ds)H_0(dy)$ for some measure ρ on $\mathbb{R}^+$ absolutely continuous with respect to the Lebesgue measure, and some non-atomic probability measure H_0 on $\mathbb{X}$. The corresponding CRM is said to be *homogeneous* and write $\mathrm{CRM}(\rho, H_0)$ for the law of μ. Successively, the measure ρ is referred to as the Lévy measure, while H_0 is the base distribution. Homogeneity implies independence between $(J_k)_{k\geq 1}$ and $(X_k^\star)_{k\geq 1}$, where $(X_k^\star)_{k\geq 1}$ is a sequence of random variables independent and identically distributed according to H_0 while the law of $(J_k)_{k\geq 1}$ is governed by ρ. Intuitively, the point process $(J_k, X_k^\star)_{k\geq 1}$ is described by a Poisson process over $\mathbb{R}^+\times\mathbb{X}$ with intensity measure $\nu = \rho\times H_0$, as illustrated in Figure 4.1.

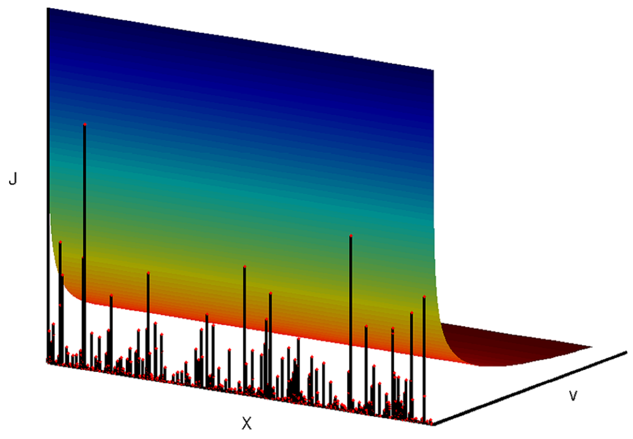


Figure 4.1: *A draw $\sum_{j\geq 1} J_j\delta_{X_j}$ from a* CRM. *Each stick denotes an atom in the* CRM, *with mass given by its height J_j and location given by X_j. Behind the* CRM *is the density of its Lévy intensity measure ν. Source: [27].*

It is required that μ has almost-surely finite total mass, Equation 4.11 with $g(y) = 1$ shows that the Lévy measure is required to satisfy the property

$$\int_{\mathbb{R}^+\times\mathbb{X}}\left(1-e^{-s}\right)\rho(ds)H_0(dy) = \int_{\mathbb{R}^+}\left(1-e^{-s}\right)\rho(ds) < \infty$$

Furthermore, the expected number of random atoms in μ is obtained by Campbells Theorem to be the total mass of the Lévy measure $\rho(\mathbb{R}^+)$. In typical applications of Bayesian nonparametrics this is infinite, so we can work with mixture models with infinite number of components. This also guarantees that the total mass is positive almost-surely.

However, since $T \neq 1$ in general, CRMs cannot directly be used as priors for mixture models. One can normalize a CRM by its finite total mass to construct a BNP prior for mixture models.

Definition 2 (Normalised Random Measure [76]). *Let μ be a homogeneous* CRM*, with Lévy measure ρ and base distribution H_0, with almost-surely positive and finite total mass. A* NRM *is an almost-surely discrete random probability measure P on $\mathbb{X}$ obtained by normalising μ*

$$P = \frac{\mu}{T} = \sum_{k\geq 1} p_k \delta_{X_k^\star}$$

with $T = \sum_{k\geq 1} J_k$ and $p_k = J_k/T$. Since μ is homogeneous, the law of $(p_k)_{k\geq 1}$ is governed by the Lévy measure ρ and the atoms $(X_k^\star)_{k\geq 1}$ are a sequence of random variables independent of $(p_k)_{k\geq 1}$, and independent and identically distributed according to H_0. We denote it by $P \sim NRM(\rho, H_0)$.

Let $\mu \sim \mathrm{CRM}(\rho, H_0)$ with an almost-surely finite and positive total mass T, and $P = \mu/T$. Suppose that T is positive and finite almost-surely, and absolutely continuous with respect to Lebesgue measure with density $f_\rho(t)$. Let $(X_i)_{i\geq 1}$ be a sequence of random variables that, given P, are independent and identically distributed according to P. Since μ is almost-surely discrete, there is a positive probability that $X_i = X_j$ for each pair $i \neq j$, i.e. when both are assigned to the same atom in P. This induces a partition π on $\mathbb{N}$, where i and j are in the same block in Π if and only if $X_i = X_j$.

We can thus define a mixture model with an NRM acting as a BNP prior as following:

$$\begin{aligned}
\mu &\sim \mathrm{CRM}(\rho, H_0),\\
P &= \frac{\mu}{T},\\
X_i|P &\sim P,\\
Y_i|X_i &\sim F(\cdot|X_i),
\end{aligned}$$

An example of NRM is the Normalised Generalised Gamma Process (NGGP) [72, 50]. The Generalised Gamma Process (GGP) Lévy measure is

$$\rho(dy) = \frac{a}{\Gamma(1-\sigma)} y^{-\sigma-1} e^{-\tau y} dy, \tag{4.12}$$

where $\tau \in [0, \infty)$, $a \in (0, \infty)$ and $\sigma \in [0, 1)$. The NGGP is then obtained, as described above, by normalisation of the GGP. Notable special case of NGGP are $\sigma = 0$ where we obtain the DP, and $\sigma = 0.5$, where we obtain the inverse-Gaussian (IG) process.

4.5.2 Poisson-Kingman random probability measure

Poisson-Kingman Random Probability Measure (RPM) were introduced in [72] as a generalisation of homogeneous NRMs.

Definition 3 (Poisson-Kingman Random Probability Measure). *Let $\mu \sim CRM(\rho, H_0)$ and let $T = \mu(\mathbb{X})$ be finite, positive almost-surely, and absolutely continuous with respect to Lebesgue measure. For any $t \in \mathbb{R}^+$, let us consider the conditional distribution of μ/t*

given that the total mass $T \in dt$. This distribution is denoted by PK(ρ, δ_t, H_0), were δ_t denotes the Dirac delta function. Poisson-Kingman RPMs *form a class of* RPMs *whose distributions are obtained by mixing PK(ρ, δ_t, H_0), over t, with respect to some distribution γ on the positive real line. Specifically, a Poisson-Kingman* RPM *has the hierarchical representation*

$$\begin{aligned} T &\sim \gamma, \\ P|T = t &\sim PK(\rho, \delta_t, H_0), \end{aligned} \tag{4.13}$$

The RPM *P is referred to as the Poisson-Kingman* RPM *with Lévy measure ρ, base distribution H_0 and mixing distribution γ. The distribution of P is denoted by PK(ρ, γ, H_0). If the density for the total mass equals the density obtained from its Lévy measure f_ρ, i.e. $\gamma(dt) = f_\rho(t)dt$, then the distribution PK(ρ, f_ρ, H_0), coincides with NRM(ρ, H_0).*

Since μ is homogeneous, the atoms $(X_k^\star)_{k\geq1}$ of P are independent of their masses $(p_k)_{k\geq1}$. They form a sequence of independent random variables identically distributed according to H_0. Finally, the masses of P have distribution governed by the Lévy measure ρ and the distribution γ.

Same as for NRMs, Poisson-Kingman RPMs can be used as BNP prior for mixture models. The associated hierarchical model is the following:

$$\begin{aligned} T &\sim \gamma, \\ P|T = t &\sim \text{PK}(\rho, \delta_t, H_0), \\ X_i|P &\sim P, \\ Y_i|X_i &\sim F(\cdot|X_i), \end{aligned}$$

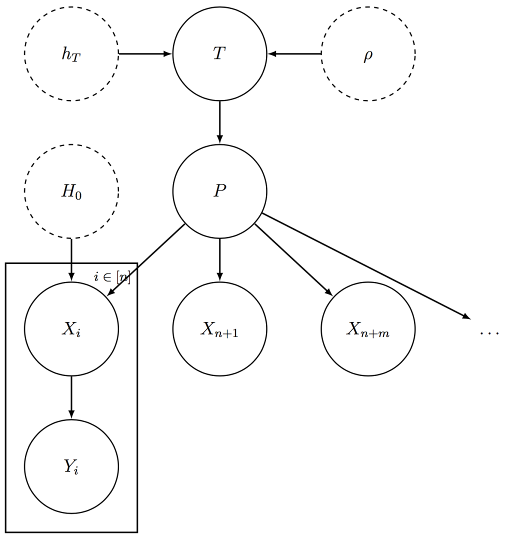


Figure 4.2: PKP *intractable graphical model: the latent variables are countably infinitely many which makes the model computationally intractable. Source: [52].*

Figure 4.2 represents the associated graphical model.

σ-Stable Poisson-Kingman rpms One famous class of Poisson-Kingman RPM is the class σ-Stable Poisson-Kingman RPMs which encompasses most of the popular discrete RPMs used in Bayesian nonparametrics, for instance, the Pitman-Yor process and the normalised generalised Gamma process. For any $\sigma \in (0,1)$ the density function of a positive σ-Stable random variable is $f_\sigma(t) = \frac{1}{\pi}\sum_{j=0}^{\infty}\frac{(-1)^{j+1}}{j!}\sin(\pi\sigma j)\frac{\Gamma(\sigma j+1)}{t^{\sigma j+1}}$. A σ-Stable Poisson-Kingman RPM is a Poisson-Kingman RPM with Lévy measure given by

$$\rho(dy) := \rho_\sigma(dy) = \frac{a}{\Gamma(1-\sigma)}y^{-\sigma-1}dy, \tag{4.14}$$

and base distribution H_0. The mixing distribution for the total mass T takes the following factored form $\gamma(dt) \propto h(t)f_\sigma(t)dt$, for any non-negative measurable function h such that $\int_0^\infty h(t)f_\sigma(t)dt < \infty$. Accordingly, σ-Stable Poisson-Kingman RPMs form a class of discrete RPMs indexed by the parameters (ρ, h). The Dirichlet process can be recovered as a limiting case, if $\sigma \to 0$, for some choices of h. The following examples of σ-Stable Poisson-Kingman RPMs are obtained by specifying the tilting function h. See [52] for more examples and more details on those.

- Normalised σ-Stable process (NS): $h(t) = 1$
- Normalised Generalised Gamma Process (NGGP) : $h(t) = \exp\{\tau - \tau^{1/\sigma}t\}$, for any $\tau > 0$
- Pitman-Yor process (PY): $h(t) = \frac{\Gamma(\theta+1)}{\Gamma(\theta/\sigma+1)}t^{-\theta}$ with $\theta \geq -\sigma$
- Gamma-tilted process (GT): $h(t) = t^{-\theta}\exp\{-\eta t\}$ for any $\eta > 0$ or $\eta = 0$ and $\theta > -\sigma$
- Poisson-Gamma class (PG): $h(t) = \int_{\mathbb{R}^+}\exp\{\tau - \tau^{1/\sigma}t\}F(d\tau)$

Size-Biased Sampling process An important object induced by a Poisson-Kingman RPM is a size-biased permutation of its atoms. Specifically, order the blocks in the partition Π (induced by the RPM) by increasing order of the least element in each block, and for each $l \in \mathbb{N}$ let Z_l be the least element of the l-th block. Z_l is the index among $(X_i)_{i\geq 1}$ of the first appearance of the l-th unique value in the sequence. Let $\tilde{J}_l = \mu(\{X_{Z_l}\})$ be the mass of the corresponding atom in μ. Then $(\tilde{J}_l)_{l\geq 1}$ is a size-biased permutation of the masses of atoms in μ, with larger masses tending to appear earlier in the sequence. It is easy to see that $\sum_{l\geq 1}\tilde{J}_l = T$, and that the sequence can be understood as a stick-breaking construction: start with a stick of length $T_0 = T$; break off the first piece of length $\tilde{J}_1$; the surplus length of stick is $T_1 = T - \tilde{J}_1$; then the second piece with length $\tilde{J}_2$ is broken off, etc. This construction is called the Size-Biased Sampling (SBS) process.

The conditioning operation from the generative process from Equation 4.13 is not well defined a priori but the following proposition from [70] helps to bypass this difficulty. It also states an interesting Markovian property of the surplus masses, which will be of used to describe a Poisson Kingman (PK) generative process.

Proposition 1 (Perman et al. [70])**.** *The sequence of surplus masses $(T_k)_{k\geq 0}$ forms a Markov chain, where $T_k := T - \sum_{l=1}^{k} J_l$, with initial distribution and transition kernels* *given as follows*

$$\begin{aligned}\mathbb{P}_{\rho,H_0}(T_0 \in dt_0) &= f_\rho(t_0)dt_0\\ \mathbb{P}_{\rho,H_0}(T_k \in dt_k | T_0 \in dt_0, \dots, T_{k-1} \in dt_{k-1}) &= \mathbb{P}_{\rho,H_0}(T_{k-1} \in dt_{k-1})\\ &= \frac{(t_k - t_{k-1})\rho(d(t_k - t_{k-1}))}{t_{k-1}} \frac{f_\rho(t_k)}{f_\rho(t_{k-1})}\end{aligned}$$

Proposition 1 [70] states that the sequence of surplus masses $(T_l)_{l\geq 1}$ forms a Markov chain and gives the corresponding initial distribution and transition kernels. The density of T is denoted by $\gamma(t) \propto h(t)f_\rho(t)$. Lomeli et al [53] gave the following PK generative process for the sequence $(X_i)_{i\geq 1}$, where parts of the PK random measure μ are simulated as required.

(i) Draw from the total mass from its distribution $\mathbb{P}_{\rho,H_0}(T \in dt) \propto h(t)f_\rho(t)$.

(ii) The first draw X_1 from μ/T is a size-biased pick from the masses of μ. The actual value of X_1 is simply $X_1^\star \sim H_0$, while the mass of the corresponding atom in μ is $\tilde{J}_1$, with conditional distribution given by

$$\mathbb{P}_{\rho,H_0}(\tilde{J}_1 \in ds_1 | T \in dt) = \frac{s_1}{t}\rho(ds_1)\frac{f_\rho(t - s_1)}{f_\rho(t)}$$

The leftover mass is $T_1 = T - \tilde{J}_1$.

(iii) For subsequent draws $i \geq 2$:

- Let k be the current number of distinct values among $X_1, \dots, X_{i-1}$, and let $X_1^\star, \dots, X_k^\star$ be the unique values, i.e., atoms in μ. The masses of these first k atoms are denoted by $\tilde{J}_1, \dots, \tilde{J}_k$ and the leftover mass is $T_k = T - \sum_{l=1}^k \tilde{J}_l$.
- For each $l \leq k$, with probability $\tilde{J}_l/T$, we set $X_i = X_l^\star$.
- With probability T_k/T, X_i, takes on the value of an atom in μ besides the first k atoms. The actual value $X_{k+1}^\star$ is drawn from H_0, while its mass is drawn from

$$\mathbb{P}_{\rho,H_0}(\tilde{J}_{k+1} \in ds_{k+1} | T \in dt_k) = \frac{s_{k+1}}{t_k}\rho(ds_{k+1})\frac{f_\rho(t_k - s_{k+1})}{f_\rho(t_k)}$$

  The leftover mass is again $T_{k+1} = T_k - \tilde{J}_{k+1}$.

By multiplying the above infinitesimal probabilities one obtains the joint distribution of the random elements T, Π, $(\tilde{J}_i)_{i\geq 1}$ and $(X_i^\star)_{i\geq 1}$.

Proposition 2 (Perman et al. [70])**.** *Let Π_n be the exchangeable random partition of $[n] := \{1, \dots, n\}$ induced by a sample $(X_i)_{i\in[n]}$ from $P \sim PK(\rho, h, H_0)$. Let $(X_l^\star)_{l\in[k]}$ be the k distinct values in $(X_i)_{i\in[n]}$ with masses $(\tilde{J}_l)_{l\in[k]}$. Then*

$$\begin{aligned}&\mathbb{P}_{\rho,H_0}(T \in dt, \Pi_n = (c_l)_{l\in[k]}, X_l^\star \in dx_l^\star, \tilde{J}_l \in ds_l \text{ for } l \in [k])\\ &\quad = h(t)t^{-n}f_\rho(t - \sum_{l=1}^k s_l)dt \prod_{l=1}^k s_l^{|c_l|}\rho(ds_l)H_0(dx_l^\star)\end{aligned} \tag{4.15}$$

where $(c_l)_{l\in[k]}$ denotes a particular partition of $[n]$ with k blocks, $c_1, \dots, c_k$, ordered by increasing least element and $|c_l|$ is the cardinality of block c_l.

4.6 MCMC Inference

Constructing Markov chain Monte Carlo (MCMC) schemes for models with one or more Bayesian nonparametric components is an active research area since dealing with the infinite dimensional component P forbids the direct use of standard simulation-based methods. These methods usually require a finite-dimensional representation. The general idea for designing inference schemes is to find finite dimensional representations to be able to store the model in a computer with finite capacity.

There are two main sampling approaches to facilitate simulation in the case of Bayesian nonparametric models: random truncation and marginalisation. These two schemes are known in the literature as conditional and marginal, but hybrid samplers between these two also exist. Each sampler rely on a tailored, thus different, representation of the underlying process. Yet, these samplers are all instances of Gibbs sampler.

4.6.1 Marginal Samplers

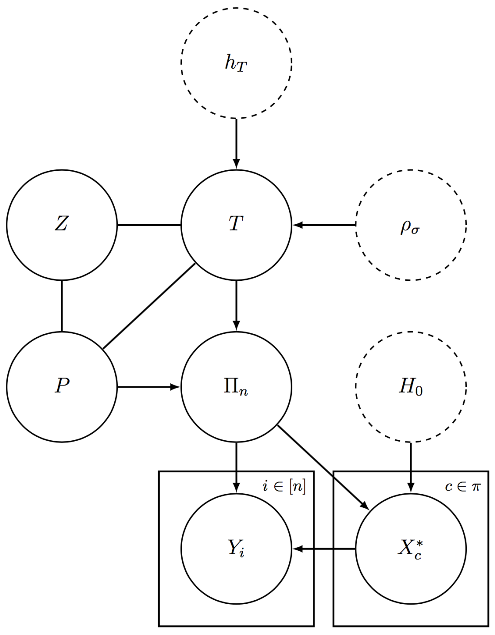


Figure 4.3: *Marginal samplers graphical model. Each node represents a variable used in the augmented representation for the joint distribution. The nodes with dashed lines represent the algorithmic inputs. Source: [52]*

Marginal samplers bypass the need to represent the infinite-dimensional component by marginalising it out. These schemes have lower storage requirements than conditional samplers because they only store the induced partition, but could potentially have worse mixing properties [52].

With a conjugate model (H_0 and F are conjugate), both the RPM P and the cluster parameters $\{X_c^\star : c \in \pi\}$ can be marginalised out efficiently. Yet, for a more generic nonconjugate marginalized sampler, the cluster parameters cannot be easily marginalised out and are thus included into the state of the MCMC algorithm. This yields the difficulty of the introduction of new clusters (along with their parameters) when Gibbs sampling the cluster assignments.

To tackle this issue, [65] conceptualizes the update in terms of an augmented state with additional temporarily existing variables, in a way that the marginal distribution of the permanent variables once the temporary ones are integrated out is still the appropriate posterior distribution. To do so, the state space is augmented as well to include both existing clusters and new empty clusters $[M] = \{1, \ldots, M\}$, which parameters are independent of the existing ones and independent and identically distributed according to H_0. [27] extends this idea originally developed for DP mixtures, for all NRM mixture models.

For instance, if the σ-Stable Poisson-Kingman subclass is picked, then it is possible to obtain a tractable representation, given by Figure 4.2, in terms of some random variables and the partition induced by the RPM. This representation is based on an augmented version of its corresponding exchangeable partition probability function.

4.6.2 Conditional Samplers

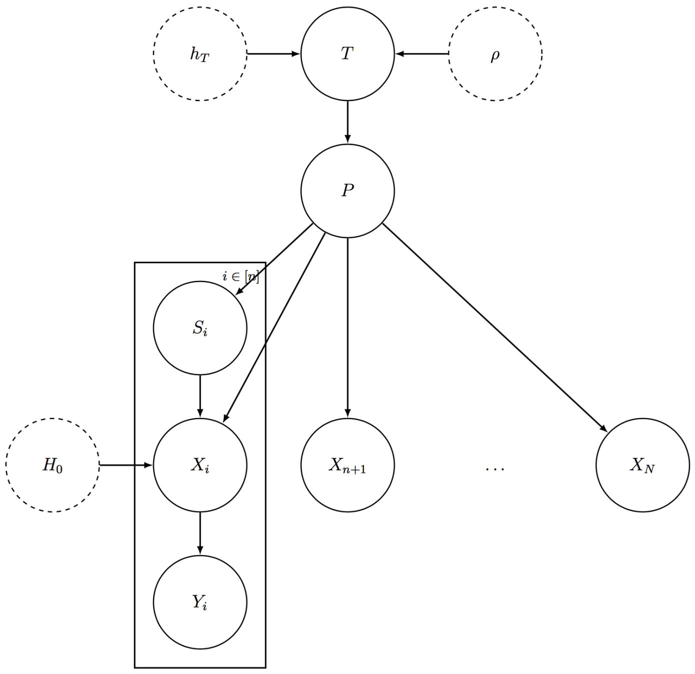


Figure 4.4: *Conditional samplers graphical model. Each node represents a variable used in the augmented representation for the joint distribution. The latent variables represent the number of occupied and unoccupied components. Source: [52].*

Conditional samplers replace the infinite-dimensional prior by a finite-dimensional representation chosen according to a truncation level. Since these samplers do not integrate out the infinite-dimensional component, their output provides a more comprehensive representation of the random probability measure. Figure 4.4 shows such representation or the σ-Stable Poisson-Kingman family for the conditional slice MCMC sampler [28].

The truncation level needs to be randomised to have an exact MCMC scheme. A slice sampling step [66] within the Gibbs sampler is used in [27] so as to get an efficient sampler. It allows to represent a finite but random number of stick-breaking weights. These weights correspond to the ones that fall above the slice variables, sampled at every iteration. Then, the cluster assignment variable for each observation is chosen from a categorical distribution, where the number of categories given by the number of represented sticks. For this reason, both the number of occupied and empty clusters is stored at every iteration. These numbers could be potentially large.

The stick-breaking weights can be sampled either via a stick-breaking representation of the measure, or via thinning [27]. Unfortunately, only few models have a tractable stick-breaking representation, such as the DP, the Pitman-Yor process (PY) or a sub-class of σ-stable Poisson-Kingman models [25].

4.6.3 Hybrid Samplers

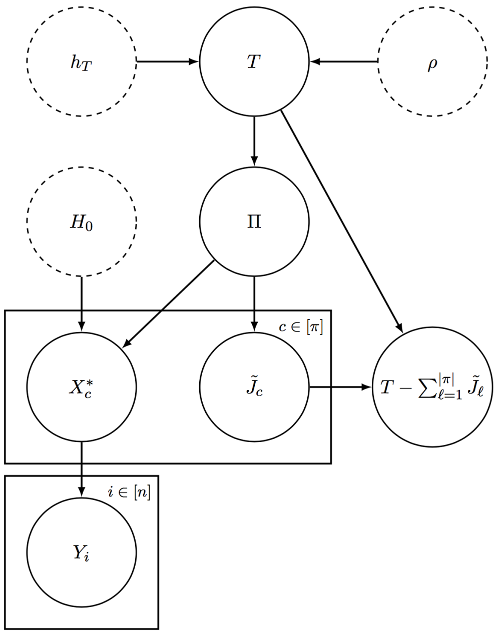


Figure 4.5: *PK hybrid samplers graphical model. Each node represents a variable used in the augmented representation for the joint distribution. The nodes with dashed lines represent the algorithmic inputs. Source: [52].*

An hybrid MCMC scheme has been developed [53], and combines the main strengths of both conditional and marginal MCMC samplers.

Equation's (4.15) joint distribution is written in terms of the first k size-biased weights. In order to obtain a complete representation of the RPM, one would need to size-bias sample from it for a countably infinite number of times. Yet, by exploiting the Size-Biased Sampling process (derived after Proposition 1) when reassigning observations to clusters and reparameterizing the model in term of a surplus mass random variable yields the join distribution used by Lomeli et al. [53].

This hybrid scheme makes use of a representation of the model where only the size-biased weights associated to occupied clusters are needed to be explicitly represented along with a surplus mass term which is associated to the rest of the empty clusters, as Figure 4.5 shows. The cluster reassignment step can be seen as a retrospective sampling scheme which explicitly represent and update the weights associated to occupied clusters and create a new size-biased weight only when a new cluster appears.

This scheme has less memory requirements since it only represents the size-biased weights associated to occupied clusters as opposed to conditional samplers which represent both empty and occupied clusters.

4.6.4 Sequential Monte Carlo

Recently, various Sequential Monte Carlo (SMC) schemes (see Section 5.5.4 for more details on SMC) for Bayesian nonparametric models have been proposed. [29] and [91] propose an SMC scheme for Dirichlet process mixture models and [39] for NRM mixture models. These articles cast the marginal and conditional representations presented previously in a particle algorithm's setting.

4.7 Variational inference

To my knowledge, the first article tackling inference in a BNP setting is [10], where a truncated proposal is introduced to approximate a Dirichlet Process Mixture model.

Truncation-free variational inference methods have also been introduced [11]. These methods adapts model complexity on the fly by lazily representing clusters assignments. Yet, the sticks proportions and mixture components are marginalized out to obtain a closed form distribution for the mixture assignment hidden variables z_i. This marginalization is unfortunately only tractable for few models, such as the Dirichlet Process and the Pitman-Yor process.

Probabilistic programming

5.1 Definition

At a high level, PPLs are Programming Language (PL) techniques to abstract inference algorithms from statistics such that they apply automatically and correctly to the broadest possible set of model-based reasoning applications.

A bit more precisely, Probabilistic programming systems [35, 36, 58, 92] represent generative models as programs written in a specialized language that provides syntax for the definition and conditioning of random variables.

Indeed, "probabilistic programs are usual functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observations." [38]

Probabilistic programs define probability distributions over sequences of values, implicitly by means of program execution.

5.2 Motivation

For data science practitioners, statistical inference is typically just one step in a more elaborate analysis workflow. The first stage of this work involves data acquisition, preprocessing and cleaning. This is often followed by several iterations of exploratory model design and testing of inference algorithms. Once a sufficiently robust statistical model and a corresponding inference algorithm have been identified, analysis results must be post-processed, visualized, and in some cases integrated into a wider production system.

The main goal of PPLs is to increase productivity. The code for models written with these systems is generally concise, modular, and easy to modify or extend. Thus, one of the savings is to be found in the amount of code that needs to written in order to prototype and develop models.

Secondly, PPLs remove the burden of having to develop inference code for each new model which is error-prone and time consuming. This is done by providing a modeling language abstraction layer in which developers can denote their models. Once denoted, generic inference is provided for free.

5.3 Existing languages

The first generation of PPLs had limitation in the range of models that could be represented and in which inference could be performed. BUGS [55] and STAN [13] can only work with graphical models. Similarly, Factorie [59] and Infer.NET [62] only handle factor graphs. These so-called *First-Order* PPLs, can only represent finite dimensional model and have bounded loops.

On the other hand, *High Order* PPLs which arrived a bit before the 2010s, are Turing complete, allow complex control flow, including stochastic recursion, thus can denote infinite dimensional objects. Anglican [92], Venture [58], Church [35] and WebPPL [36] are *High Order* PPLs.

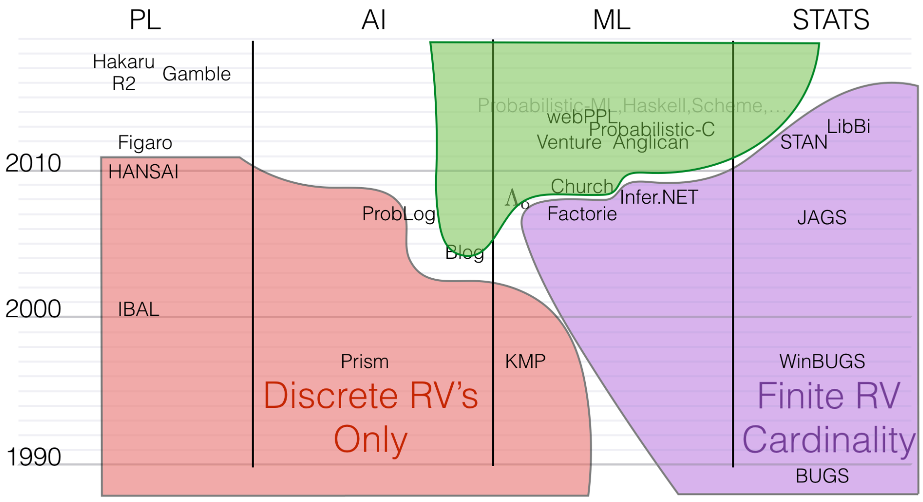


Figure 5.1: *First-order and high-order PPLs. Source: http://www.robots.ox.ac.uk/ fwood.*

Figure 5.1 shows a map of existing PPLs, with first-order ones being in red and purple (bottom left/right), and high-order ones in green (top). On one side, first-order PPLs are quite restrictive since the spectrum of models that can be represented is limited to finite dimensional models, but inference can be performed efficiently. On the other side, high-order PPLs are more flexible but the task of writing efficient inference algorithm is harder. There is therefore a trade-off between flexibility and efficiency, and the restrictions of first-order PPLs are a design choice.

Recently, a new PPL named Edward [85] has been developed. It is different from the classical PPLs since it focuses on variational inference (VI) and Hamiltonian methods.

5.4 Design

Probabilistic programs denote probabilistic generative models as programs that include `sample` and `observe` statements (see Listing 5.1). Both `sample` and `observe` are functions that specify random variables in this generative model using probability distribution objects as an argument, while `observe`, in addition, specifies the conditioning of this random variable upon a particular observed value in a second argument. These observed values induce a conditional probability distribution over the execution traces whose approximations and expected values we aim to characterize by *performing inference*.

```julia
@model model(y) = begin
  x = sample(Bernoulli(.75))
  if x == True
    mu = 2
  else
    mu = 0
  end
  observe(Gaussian(mu, 1), y)
  return x
end

sample(model(0.5), SMC(100))
```

Listing 5.1: *Example of a Turing.jl model*

A good way to understand this is to imagine the following interpreter of probabilistic programs. Starting from a fixed initial state, the interpreter runs the deterministic parts of a program according to the standard semantics, executes the `sample` statement by generating a random sample, and treats the `observe` statement by skip. More importantly, the interpreter keeps a log that records information about all the `sample` and `observe` forms encountered during execution. The information recorded for `sample` is a triple (F, x, α) of (i) a primitive probability distribution F, such as the standard normal, for which we have the probability density f ; (ii) a value x sampled from the distribution F ; and (iii) an address α that uniquely and systematically identifies the random choice made. The information recorded for `observe` is a pair (G, y) where G is a primitive probability distribution with density g and y is an observed value.

An execution trace $\mathbf{x}$ is defined to be a sequence of triples (F, x, α) and $\mathbf{x}$ is said feasible if the trace is precisely the triple part of the log of some execution. The observed values are denoted by $\mathbf{y} := (y_j)_{j=1}^N$.

In most probabilistic programming systems any variable may be declared as being the output of a random procedure. Such variables can take different values in independent interpretations of the program. This leads to a “many-worlds” computational trace tree in which, at interpretation time, there is a branch at every random procedure application.

A (almost-surely terminating) probabilistic program defines a probability distribution over finite feasible traces $\mathbf{x}$ with probability density $\pi(\mathbf{x}) := \gamma(\mathbf{x})/Z$ where

$$\gamma(\mathbf{x}) := p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{|\mathbf{x}|} f_i(x_i \mid x_{1:i-1}) \prod_{j=1}^{|\mathbf{y}|} g_j(y_j \mid x_{1:\tau(j)})$$

where Z is the normalizing constant $Z := \int \gamma(\mathbf{x}) d(\mathbf{x})$ and τ is a mapping from the index j of the `observe` statement to the index of the last `sample` statement encountered before this `observe` statement during the execution of the program. Without any `observe` statement, the probability distribution over traces $\mathbf{x}$ is simply the prior

$$p(\mathbf{x}) := \prod_{i=1}^{|\mathbf{x}|} f_i(x_i \mid x_{1:i-1})$$

5.5 Inference

Inference in probabilistic programming characterizes the conditional distribution of execution traces $\mathbf{x}$ given observed data $\mathbf{y}$ assumed to have been generated by executing the probabilistic program.

Concretely, using the `sample` statements, the PPL's model first defines a so called prior distribution on these execution traces, and then it adjusts this prior distribution based on observations in data using the `observe` statement. Samples from this conditioned distribution (also called posterior distribution) can be obtained by running the model under one of the PPLs inference algorithms.

The inference algorithms may make random choices that do not correspond to any statements in the program, and decide which parts of the program code are executed and how often. Some inference algorithms re-run the program multiple times partially, from a certain point on, while reusing random choices made in the previous runs as much as possible. Upon encountering a sample or observe record, the inference algorithm computes the updated program state and the value to be passed to the continuation. How the state is updated, the number of times the continuation is called, and the value passed to the continuation of sample depend on the inference algorithm executing the program. Implementing an inference algorithm in PPLS amounts to defining checkpoint handlers for `sample` and `observe`.

Typically inference can be performed for any probabilistic program using one or more generic inference techniques provided by the system back end, such as Metropolis-Hastings [89, 58], Hamiltonian Monte Carlo [13], expectation propagation [62], and extensions of Sequential Monte Carlo [86, 69, 93] methods.

5.5.1 Use-case

Even if as highlighted before, inference in PPLS should be able to deal with arbitrary series of targets, for simplicity we will focus on a non-Markovian State-Space Model (SSM).

SSMs are probabilistic models over a set of latent variables $X_t \in \mathcal{X}_t, \forall t = 1 : T$ and observed variables $Y_t \in \mathcal{Y}_t, \forall t = 1 : T$. We can further consider a model to be parameterized by $\theta \in \Theta$. The SSM is then characterized by an initial density $\mu_\theta(x_1)$, a series of transition densities $f_{t,\theta}(x_t|x_{1:t-1})$, and a series of emission densities $g_{t,\theta}(y_t|x_{1:t})$.

$$\begin{aligned}
&X_1 \sim \mu_\theta(\cdot) \\
&X_t|(X_{1:t-1} = x_{1:t-1}) \sim f_{t,\theta}(\cdot|x_{1:t-1}) \\
&Y_t|(X_{1:t} = x_{1:t}) \sim g_{t,\theta}(\cdot|x_{1:t})
\end{aligned}$$

The joint density of the SSM is then as follows

$$\gamma_\theta(\mathbf{x}) := p_\theta(x_{1:T}, y_{1:T}) = \mu_\theta(x_1) \prod_{t=2}^{T} f_{t,\theta}(x_t|x_{1:t-1}) \prod_{t=1}^{T} g_{t,\theta}(y_t|x_{1:t})$$

We are free to choose any density for $\mu_\theta(x_1)$ and each $f_{t,\theta}(x_t|x_{1:t-1})$ and $g_{t,\theta}(y_t|x_{1:t})$. One is usually interested characterizing the posterior

$$p_\theta(x_{1:T}|y_{1:T}) \propto p_\theta(x_{1:T}, y_{1:T})$$

Or expectations of some function ϕ under this posterior

$$I(\phi) = \int \phi(x_{1:T}) p_\theta(x_{1:T}|y_{1:T}) dx_{1:T}$$

5.5.2 Enumeration

The easiest way one could think of to perform inference in a probabilistic program is via *rejection sampling*. An execution trace can be thought of a *path* in a *tree* implicitly defined by a discrete model. This tree could then be explored using depth-first search, breadth-first search, or a probability-based priority queue. Then only the paths matching the observations $\mathbf{x}$ are retained and the others are rejected. These retained execution traces form the targeted posterior distribution.

5.5.3 Markov Chain Monte Carlo

Yet, for many models with large state spaces, enumeration is infeasible. This is particularly clear for models with continuous random variables, where the state space is infinite. In the case of a large number of execution paths, one should avoid exploring all paths individually but only a subset of paths.

A popular way to estimate a difficult distribution is to sample from it by constructing a random walk that will visit each state in proportion to its probability. This class of algorithms are called MCMC. In our case, we are interested in random walk in the space of execution traces of a computation.

For discrete models, an easy way to a build random walk in the space of executions is to execute the probabilistic program by sampling variables at `sample` AND `observe` statements. The proposed execution trace $\mathbf{x}^\star$ shall then be rejected if at least one of the true observation y_k does not match the associated generated observation $y \sim G$. If accepted, that means that this trace is one of the trace that can lead to the observations $\mathbf{y}$, i.e. it belongs to the support of the targeted posterior distribution $p(\mathbf{x}|\mathbf{y})$. This method is called *rejection sampling.* Yet, for highly dimensional space, this method will accept quite rarely a proposed trace. What is more, for continuous models, it will almost-surely never be accepted.

A more complex Markov Chain needs to be built to have a reasonable acceptance rate. Given a current trace $\mathbf{x}$ and score $p(\mathbf{x})$, we proceed by reconsidering one random choice x_k. Each `sample` statement is equipped with a proposal kernel $\mathcal{K}_k(x^\star|x, \psi)$, which is used to generate proposals to x_k. A random walk can the be built by (i) randomly choosing $k \in 1, \dots, |\mathbf{x}|$, (ii) sample a new value $x_k^\star$ with $\mathcal{K}_k(x_k^\star|x_k, \psi)$ and (iii) re-run the program starting from x_k which generate a new trace $\mathbf{x}^\star$. This new execution trace is accepted with probability

$$1 \wedge \frac{\gamma(\mathbf{x}^\star)\mathcal{K}_k(x_k|x_k^\star, \psi)}{\gamma(\mathbf{x})\mathcal{K}_k(x_k^\star|x_k, \psi)}$$

This is better than our previous Metropolis-Hastings (MH) algorithm, but when the chosen k is close to 1, it is almost as sampling from the prior since we only reuse the random choices made before the point of regeneration. So as to avoid having a small acceptance rate, it is generally better to make smaller steps by reusing as many choices as possible. If we knew which sampled value was which, then we could look into the previous trace as the execution runs and reuse its values. That is, imagine that each call to sample was passed a (unique) name: `sample(name, dist)`. Then the `sample` function could try to look up and reuse values. Notice that, in addition to reusing existing sampled choices, we add the name and mark whether this choice has been resampled. We must account for this in the MH acceptance calculation. We now hope to reuse most of the choices from the old trace in making a proposal. This algorithm is called Lightweight MCMC [89], and has been improved in [78].

More generally, for an excellent introduction to MCMC and particle inference in PPLs, see [36].

5.5.4 Importance Sampling

Importance Sampling Importance sampling is an example of a Monte Carlo sampling scheme that provides approximately independent and identically distributed samples from a distribution of interest or target distribution, such as a posterior distribution, by generating a candidate sample from a proposal or importance distribution $q(\mathbf{x}|y_1, \dots, y_T)$. The fact that the weights $w^k = \frac{p(\mathbf{x}, y_1, \dots, y_T)}{q(\mathbf{x}|y_1, \dots, y_T)} \propto \frac{p(\mathbf{x}|y_1, \dots, y_T)}{q(\mathbf{x}|y_1, \dots, y_T)}$, with $k \in 1, \dots, K$ can be computed is exploited, and samples from the target are obtained by sampling from the following weighted empirical distribution

$$\hat{p}(\mathbf{x}|y_1, \dots, y_T) = \sum_{k=1}^{K} \bar{w}^k \delta_{\tilde{\mathbf{x}}^k}(\mathbf{x})$$

where $\bar{w}^k = \frac{w^k}{\sum_{k=1}^K w^k}$ is the normalized weight and δ_z is a Dirac measure centered on z. The expectation $I(\phi)$ can also be approximated using

$$\hat{I}(\phi) = \sum_{k=1}^{K} \bar{w}^k \phi(\tilde{\mathbf{x}})$$

One problem with this method is that it is not easy to choose the proposal distribution q. A good proposal should share most of the support of the target distribution and have the same number of modes, i.e. it should be close to the target. A second problem is that it is a batch estimation method. To tackle this latter issue, in the next section, an extension to a sequential scenario is described.

Sequential Importance Sampling Sequential Importance Sampling (SIS) exploits the structure of a model by breaking down the overall inference problem into a series of target distributions which get incrementally closer to the distribution of interest. These targets are then approximated by propagating a population of samples known as particles. If each intermediary target is kept similar to its predecessor, approximating one target given the last forms a significantly simpler problem than the overall inference. Breaking the overall inference problem into a series of intermediate distributions makes the design of proposals easier since they must now be of the form $q(x_t|x_{1:t-1}, y_{1:t})$ instead of $q(x_{1:T}|y_{1:T})$.

More formally, SIS performs approximate inference on a sequence of target distributions $(\pi_t(x_{1:t}))_{t=1}^T$ of increasing spaces $(\mathcal{X}_1 \times \cdots \times \mathcal{X}_t)_{t=1}^T$. In the context of SSMs, the target distributions are taken to be $(p_\theta(x_{1:t}|y_{1:t}))_{t=1}^T$. At each time step t, we have a set of K particles $\left(\tilde{x}_{1:t}^k\right)_{t=1}^T$, corresponding to samples of the latents, and respective particle weights $\left(w_t^k\right)_{t=1}^T$. Similarly to Importance Sampling (IS), using these weighted particles, one can approximate each posterior $p_\theta(x_{1:t}|y_{1:t})$. In particular, the posterior for the complete model $p_\theta(x_{1:T}|y_{1:T})$ and the expectation $I(\phi)$ can be approximated using the following estimators

$$\hat{p}(x_{1:T}|y_{1:T}) = \sum_{k=1}^{K} \bar{w}_T^k \delta_{\tilde{x}_{1:T}^k}(x_{1:T})$$

$$\hat{I}(\phi) = \sum_{k=1}^{K} \bar{w}_T^k \phi(\tilde{x}_{1:T})$$

where $\bar{w}_T^k = \frac{w_T^k}{\sum_{k=1}^K w_T^k}$ is the normalized weight.

Let's now focus on computing the particles' weights. The joint posterior can be written in the following factorised form

$$p(x_{1:T}|y_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t|x_{1:t-1}, y_{1:t})$$

The corresponding importance weights are thus

$$
\begin{aligned}
w_t =& \frac{p(x_{1:T}|y_{1:T})}{q(x_{1:T}|y_{1:T})} \\
w_t =& \frac{\mu(x_1)\prod_{t=2}^{T} p(x_t|x_{1:t-1}, y_{1:t})}{q(x_1)\prod_{t=2}^{T} q(x_t|x_{1:t-1}, y_{1:t})}
\end{aligned} \tag{5.1}
$$

The target distribution is the posterior distribution up to time t, which changes sequentially as we observe more data. This posterior distribution can be estimated recursively due to

$$
p(x_{1:t+1}|y_{1:t+1}) = p(x_{1:t}|y_{1:t}) \times \frac{p(y_{t+1}|x_{1:t+1})p(x_{t+1}|x_{1:t})}{f(y_{t+1}|y_{1:t})} \tag{5.2}
$$

Substituting the numerator of Equation 5.2 in 5.1 we obtain a recursive equation for the importance weight at time $t+1$

$$
w_{t+1} = w_t \times \frac{p(y_{t+1}|x_{1:t+1})p(x_{t+1}|x_{1:t})}{q(x_{t+1}|x_{1:t}, y_{1:t+1})}
$$

Even if we are not dealing with a SSM, SIS can be used as a general-purpose algorithm. The data are assumed to be observed sequentially so the observations index is the time index.

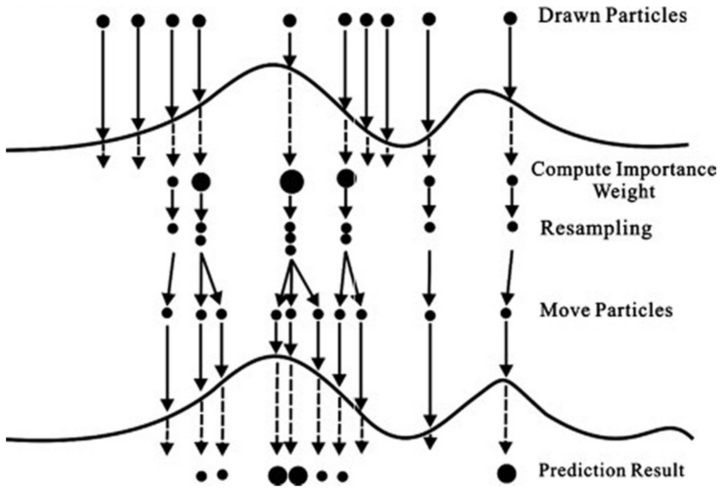


Figure 5.2: *Sequential Monte Carlo's algorithm main steps. Source: https://umbertopicchini.wordpress.com/2016/10/19/sequential-monte-carlo-bootstrap-filter.*

Sequential Monte Carlo The main problem with SIS is that the weights become more skewed as the number of data points increases [20], after a few steps only one particle will have a significant weight. To remedy this, a resampling step can be introduced which allows to eliminate particles with small weights and replicate particles with high weights, as illustrated in Figure 5.2. The resampling step is achieved by, at each time step $t = 2, \dots, T$, selecting the ancestor index a_{t-1}^k for the kth particle from a discrete distribution $\mathcal{F}(\cdot|\bar{w}_{t-1}^1, \dots, \bar{w}_{t-1}^K)$ over parent indices $1, \dots, K$ with probabilities equal to the normalized weights at the previous time step $(\bar{w}_{t-1}^1, \dots, \bar{w}_{t-1}^K)$. The first resampling scheme introduced was the multinomial resampling for which $\mathbb{P}(a_{t-1}^k = i) \propto \bar{w}_{t-1}^i$. [19] provides a comparison of numerous different schemes for sampling from $\mathcal{F}(\cdot|\bar{w}_{t-1}^1, \dots, \bar{w}_{t-1}^K)$ that reduce the variance of the SMC estimates compared with naïve multinomial resampling.

This selection step can be introduced only occasionally or at every step of the algorithm. If the selection step is to be performed occasionally, a possible criterion is when the Effective Sample Size (ESS) is below a given threshold, which is a function of the number of particles, a popular choice is $0.5T$. The ESS for the unnormalised weights is given by

$$ESS_t = \frac{\left(\sum_{k=1}^K w_t^k\right)^2}{\sum_{k=1}^K {w_t^k}^2}$$

At step t, after the ancestors indices $\{a_{t-1}^k\}_{k=1}^K$ have been sampled, new $\{x_t^k\}_{k=1}^K$ are proposed according to $q(\cdot|x_{1:t-1}, y_{1:t})$. Then, the particles' weights can be computed given

$$\begin{aligned} w_t^k :=& \frac{p(x_{1:t}^k, y_t)}{p(x_t^k|x_{1:t-1})q(x_t^k|x_{1:t-1}^k, y_{1:t})} \\ =& \frac{p(y_t|x_{1:t}^k)p(x_t^k|x_{1:t-1}^k)}{q(x_t^k|x_{1:t-1}^k, y_{1:t})} \end{aligned}$$

ppls implementation of smc One a practical point of view, so as to handle particle algorithms, PPLS must have the ability to stop the execution of the program at each time t (more generally at each `observe` statement). Indeed, these *chechpoints* are required by the SMC algorithm for the resampling step. Handling of checkpoints can be implemented through coroutines/co-operative multitasking (as in Turing [31]), and parallel execution/preemptive multitasking, as well as through explicit maintenance of program continuations (as in *Anglican* [92] and *WebPPL* [36]).

Moreover, the proposals $q(\cdot|x_{1:t-1}, y_{1:t})$ are generally implemented as being the prior $p(\cdot|x_{1:t-1})$ in PPLS, i.e. the program is simply ran until an `observe` statement is encountered. Yet, by overriding the `sampler` statements, different proposals can be implemented. In the former case, the weights simplify to the likelihood; $w_t^k = p(y_t|x_{1:t}^k)$, which can be computed at the `observe` statements.

5.5.5 Particle Markov Chain Monte Carlo

In a Bayesian setting, it is usual to consider the parameter θ in $p_\theta(x_{1:T}|y_{1:T})$ as a random variable by specifying a prior $p(\theta)$. In this subsection, we are therefore interested on algorithms targeting $p(\theta, x_{1:T}|y_{1:T})$. Let's think about MCMC algorithms targeting the distribution $p(\theta, x_{1:T}|y_{1:T})$ which rely on sampling exactly from $p_\theta(x_{1:T}|y_{1:T})$, called idealized algorithms. Such algorithms are purely conceptual but a natural idea consists of approximating these idealized algorithms by using the output of an SMC algorithm targeting $p_\theta(x_{1:T}|y_{1:T})$ using $K \geq 1$ particles as a proposal distribution for an MH update. Intuitively this could allow us to approximate with arbitrary precision such idealized algorithms while only requiring the design of low dimensional proposals for the SMC algorithm.

A direct implementation of this idea is impossible as the marginal density of a particle that is generated by an SMC algorithm is not available analytically but would be required for the calculation of the MH acceptance ratio. Yet the SMC algorithm yields an unbiased estimate of the marginal likelihood

$$\hat{p}(y_{1:T}) = \prod_{t=1}^{T} \frac{1}{K} \sum_{k=1}^{K} w_t^k$$

which can be used for the calculation of the MH acceptance ratio. These Particle Markov chain Monte Carlo (PMCMC) updates have been introduced in [3]. The key feature of PMCMC algorithms is that they are in fact exact approximations to idealized MCMC algorithms targeting either $p(\theta, x_{1:T}|y_{1:T})$ in the sense that for any fixed number $N \geq 1$ of particles their transition kernels leave the target density of interest invariant.

Particle Marginal Metropolis-Hastings Particle Marginal Metropolis-Hastings (PMMH) makes use of the standard decomposition $p(\theta, x_{1:T}|y_{1:T}) = p(\theta|y_{1:T})p_\theta(x_{1:T}|y_{1:T})$. It is natural to suggest the following form of proposal density for an MH update:

$$q\left(\theta^\star, x_{1:T}^\star|\theta, x_{1:T}\right) = q(\theta^\star|\theta)p_{\theta^\star}(x_{1:T}^\star|y_{1:T})$$

for which the proposed $x_{1:T}^\star$ is given by a SMC algorithm targeting $p_{\theta^\star}(x_{1:T}|y_{1:T})$. Thus, the only degree of freedom of the algorithm (which will affect its performance) is $q(\theta^\star|\theta)$. The resulting MH acceptance ratio is given by

$$1 \wedge \frac{p_{\theta^\star}(y_{1:T})p(\theta^\star)q(\theta|\theta^\star)}{p_\theta(y_{1:T})p(\theta)q(\theta^\star|\theta)}$$

PMMH uses $\hat{p}(y_{1:T})$ to compute the acceptance ratio. It has been proven [3] that the PMMH update leaves $p(\theta, x_{1:T}|y_{1:T})$ invariant and that under weak assumptions the PMMH sampler is ergodic.

Particle Gibbs An alternative to the previous algorithm to sample from $p(\theta, x_{1:T}|y_{1:T})$ consists of using the Gibbs sampler which samples iteratively from $p(\theta|x_{1:T}, y_{1:T})$ and $p_\theta(x_{1:T}|y_{1:T})$. It is often possible to sample from $p(\theta|x_{1:T}, y_{1:T})$ and thus the potentially tedious design of a proposal density for θ that is necessary in the PMMH update can be bypassed.

It has been shown [3] that the naïve particle approximation to the Gibbs sampler where sampling from $p_\theta(x_{1:T}|y_{1:T})$ is replaced by sampling from an SMC approximation $\hat{p}_\theta(x_{1:T}|y_{1:T})$ does not admit $p(\theta, x_{1:T}|y_{1:T})$ as invariant density.

A valid particle approximation to the Gibbs sampler requires the use of a special type of PMCMC update called the *conditional* SMC update. This update is similar to a standard SMC algorithm but is such that a prespecified path $x^{\star}_{1:T}$ is ensured to survive all the resampling steps, whereas the remaining $N-1$ particles are generated as usual.

Particle Gibbs with Ancestor Sampling A drawback of Particle Gibbs (PG) is that it can be particularly adversely affected by path degeneracy in the Conditional Sequential Monte Carlo (CSMC) step. Conditioning on an existing trajectory means that whenever resampling of the trajectories results in a common ancestor, this ancestor must correspond to this trajectory. Consequently, the mixing of the Markov chain for the early steps in the state sequence can become very slow when the particle set typically coalesces to a single ancestor during the CSMC sweep.

[51] introduces Particle Gibbs with Ancestor Sampling (PGAS), which alleviates the problem with path degeneracy by modifying the original PG kernel with a so-called Ancestor Sampling (AS) step. The idea is to sample a new value for the index variable a_t^N in an ancestor sampling step. While this is a small modification of the algorithm, the improvement in mixing can be quite considerable. The task is to artificially assign a history to the partial path $(x^{\star}_{t:T})$ of the reference path. This is done by connecting $(x^{\star}_{t:T})$ to one of the particles $(x^k_{1:t-1})$. The ancestor index of the reference path $(x^{\star}_{t:T})$ at time t $a_t^N \in \{1, \ldots, K\}$ encodes the ancestry of this particle.

To assign a history to this partial path, first the following weights are computed

$$\tilde{w}^k_{t-1|T} := w^k_{t-1} \frac{p((x^k_{1:t-1}, x^{\star}_{t:T}), y_{1:T})}{p(x^k_{1:t-1}, y_{1:t-1})}$$

for $k = 1, \ldots, K$. Then, a_t^K is sampled via $\mathbb{P}\left(a_t^K = k\right) \propto \tilde{w}^k_{t-1|T}$.

Interacting Particle Markov Chain Monte Carlo Interacting Particle Markov Chain Monte Carlo (IPMCMC) [74] is another way of tackling the path degeneracy issue. In IPMCMC, a pool of CSMC and unconditional SMC algorithms are ran as parallel processes (referred as nodes. After each run of this pool, successive Gibbs updates are applied to the indexes of the CSMC nodes, such that the indices of the CSMC nodes changes. Hence, the nodes from which retained particles are sampled can change from one MCMC iteration to the next. This lets us trade off exploration (SMC) and exploitation (CSMC) to achieve improved mixing of the Markov chains.

5.5.6 Hamiltonian samplers

Hamiltonian Monte Carlo In Hamiltonian Monte Carlo (HMC) [21, 67], a deterministic proposal process based on Hamiltonian dynamics is employed along with additional stochastic proposals that together provide an ergodic Markov chain capable of making large transitions that are accepted with high probability. The Hamiltonian makes use of the gradient of the joint distribution $p(\mathbf{x}, \mathbf{y})$ so as to construct better proposal.

No-U-Turn Sampler In [41], the authors address the issue of choosing the two hyperparameters of HMC: a step size ϵ and a desired number of steps L, since HMC's performance is highly sensitive on those.

First, No-U-Turn Sampler (NUTS) uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution, stopping automatically when it starts to double back and retrace its steps. The number of recursion calls is chosen to be the number of leapfrogs steps. Moreover, if α_t is the Metropolis acceptance probability for iteration t and δ is the desired average acceptance probability, we would like the statistic $H_t = \delta - \alpha_t$ to decreases towards 0. NUTS achieves that goal by adaptively computing the step size using the dual averaging scheme of Nesterov [68], an algorithm for nonsmooth and stochastic convex optimization.

Stochastic Gradient Markov Chain Monte Carlo A recent focus has been on devising scalable variants of MCMC algorithms that subsample the data and use stochastic gradients in place of full-data gradients in the dynamic simulations. Such algorithms are called Stochastic Gradient Markov chain Monte Carlo (SGMCMC) and a recipe has now been given [56] for deriving these schemes. SGMCMC algorithms are well suited for online setting and can scale to big datasets. Stochastic Gradient Langevin Dynamics (SGLD) [88] and Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [14] are examples of these schemes.

5.5.7 Variational Inference

MCMC methods can be slow to converge and their convergence can be difficult to diagnose. One class of alternative methods is provided by variational inference methods [44, 32]. The basic idea of variational inference is to formulate the computation of a marginal or conditional probability in terms of an optimization problem. This (generally intractable) problem is then "relaxed", yielding a simplified optimization problem that depends on a number of free parameters, known as variational parameters. Solving for the variational parameters gives an approximation to the marginal or conditional probabilities of interest.

To my knowledge, Edward [85] is the only PPL handling variational inference.

5.6 Contributions

During this internship I have taken the time to actually implement several inference algorithms, and by so, I contributed to two existing PPLs. Some only for the sake of learning more about sampling schemes and PPLs, but others as specifically part of the project.

First, I implemented [1] both the SGLD [2] and SGHMC inference algorithms in Turing.jl [31], a PPL based on Julia [7] and developed at the University of Cambridge. Then, I

[1]See `https://github.com/yebai/Turing.jl/tree/master/src/samplers`

[2]See for instance, SGLD applied to a Bayesian logistic regression at `https://github.com/yebai/Turing.jl/blob/master/example-models/sgld-paper/lr_sgld.jl`

implemented [3] the Dual Averaging extension [41] of HMC for Edward [85], a PPL built on top of Tensorflow [1] by Blei's group [4] at Columbia University.

I have also implemented other inference algorithms for Turing, that I described in Section 6.4. Thanks to my contributions, I am now officially a *Collaborator* of Turing's repository.

[3]See `https://github.com/blei-lab/edward/pull/728`
[4]`http://www.cs.columbia.edu/~blei/`

BNP inference within PPLs

In this chapter we focus on the task of performing inference within a PPL for models with an infinite dimensional component, also called BNP models. For now we have restricted ourselves to infinite mixture models, but we hope that our framework will enable efficient sampling for other BNP models such as the infinite Hidden Markov Model [5]. We have also focused our study on MCMC sampling methods, but variational inference is considered in Section 7.3. After introducing the ideas allowing to sample from Poisson Kingman Processs prior, this section describes the methodology used for the experimentations, along with our contributions.

6.1 Prior sampling for the Dirichlet Process

In PPLs, to transform a program variable `x` in a random variable, one only needed to write `x = sample(Dist(parameters))`. Then the posterior distribution (given some observations) of `x` will automatically be performed during the inference scheme. The Bayesian setting thus naturally fits the framework of PPLs. Yet, we believe that there is more than just a connection between Bayesian inference and PPLs, but that there is also a connection between Bayesian Nonparametric and High-order PPL.

Working with BNP models can be impossible because of the constraints of computers. A NRM P can be written [45] as

$$P = \sum_{k \geq 1} p_k \delta_{X_k^\star} \quad \text{and} \quad \sum_{k \geq 1} p_k = 1$$

where $(p_k, X_k^\star)_{k \geq 1}$ is an infinite collection of weights and atoms. Discrete probability measures with countable support such as Normalised Random Measures cannot be represented in a computer in a naïve manner since a machine has finite memory. Other representations of such objects must be used, if one hope to denote them in a program.

One key notion in programming languages which will be crucial is the concept of *lazy evaluation* (or *laziness*). It is an evaluation strategy which delays the evaluation of an expression until its value is needed and which also avoids repeated evaluations. Lazy evaluation is often combined with *memoization*, as described in [6]. After a function's value is computed for that parameter or set of parameters, the result is stored in a lookup table that is indexed by the values of those parameters; the next time the function is called, the table is consulted to determine whether the result for that combination of parameter values is already available. If so, the stored result is simply returned. If not, the function is evaluated and another entry is added to the lookup table for reuse.

Instead of hoping to construct the entire infinite sequence $(p_k, X_k^\star)_{k\geq 1}$, we will only “lazily” sample $(p_k, X_k^\star)$, i.e. when we need them. Since we work with homogeneous CRMs, the $\{X_k^\star\}_{k\geq 1}$ are independent and identically distributed according to the base distribution. Sampling the $(X_k^\star)_{k\geq 1}$ as we need them is therefore trivial. Yet, how could we lazily sample the $(p_k)_{k\geq 1}$? In Subsection 4.3.1, we presented the stick-breaking process, a generative process for constructing of a size-biased permutation of $(p_k)_{k\geq 1}$, denoted $(\tilde{p}_k)_{k\geq 1}$. In Section 6.2 we explore other generative processes, for more generic BNP classes. For now, Let us recall the construction given by [81] for the stick-breaking process of the DP:

$$V_k \overset{iid}{\sim} \text{Beta}(1, \alpha) \;\; k \geq 1$$

$$\tilde{p}_k = V_k \prod_{j=1}^{k-1} (1 - V_j) \;\; k \geq 1$$

Therefore, $\tilde{p}_k$ can be seen as the probability of the following sequence of outcomes: $k-1$ Bernoulli draws with respective probabilities V_j , $j = 1, \ldots, k-1$, all yield failure, and one last Bernoulli draw with probability V_k yields success. How can this be interpreted as a generative process on an infinite set of discrete outcomes ? Imagine “walking” down the natural numbers in order, flipping a coin with probability V_k (called stick or stick length) for each one; if the coin comes up `false`, we continue to the next natural number; if the coin comes up `true`, we stop and return the current natural number. The probability of getting the natural number k is given by $\tilde{p}_k$ defined above. This is formalised by the procedure `pickStick` in the code sample 6.1 below.

```
function pickStick(sticks, k)
  if rand(Bernoulli(sticks(k))) # flip a coin with weight p̃_k
    return k
  else
    return pickStick(sticks, k+1) # recursive call if coin is tail
  end
end

function DP(α)
  sticks = @memoize (index) -> rand(Beta(1, α))
  return pickStick(sticks, 1)
end
```

Listing 6.1: *Dirichlet Process stick-breaking representation written in Julia.*

The individual V_k are drawn lazily, only generating new ones when we have walked out further than the furthest previous time. This is the role of the procedure `DP` which uses memoization to enforce this property. Even though we started by imagining an infinite set of sticks we only ever construct a finite subset of them. The above generative construction of the DP defines a distribution over the infinite set of natural numbers, which is equivalent to the CRP defined in Section 4.3.1.

Note that the function `pickStick` is written in a special fashion: it is built via a *recursion*. A *recursive* function has one or more base cases for which the function produces a result trivially (without recurring), and one or more recursive cases for which the program recurs (calls itself). For programming languages, allowing recursion means allowing a function to call itself within the program text. Moreover, before calling `pickStick`, the program does not know the integer which will be returned, i.e. the number of recursion calls, since it is a random variable by construction.

Recall that high-order PPLs allow complex control flow, including stochastic recursion (stated in Section 5.3). *Stochastic* or *unbounded* recursion simply means that the number of recursive calls (also called *depth* of the recursion) is random or unbounded. Therefore, the DP stick-breaking process can be written and executed with such high-order PPLs.

PPLs are some sort of *simulators*, models need to be written as generative processes. Executing such a model – as is – via a PPL's interpreter, is sampling from its prior distribution. Once one can write a model as a generative process, (i.e can sample from the model's prior), the PPL can perform inference on the latent variables. Consequently, for BNP models based on Random Probability Measures, a representation similar to the stick-breaking process for the DP, seems to be necessary.

6.1.1 From sampling in $\mathbb{N}$ to sampling in Ω

The stick-breaking construction previously described defines a distribution over the infinite set of natural numbers via the sequence of weights $(\tilde{p}_k)_{k\geq 1}$, as the CRP does. Yet, ultimately we would like a distribution not over the natural numbers themselves, but over an infinite set of samples from some other distribution H_0 (called the base distribution) which will represent our mixture components. Using the homogeneous assumption (of the underlying CRM), we can easily generalise the DP code that we wrote to this setting by using `@memoize` to associate to each natural number, a draw from the base distribution. This is what the code sample (6.2) below formalises.

```
function DPmem(α,H₀)
  augmentedDP = @memoize (stickIndex) -> rand(H₀)
  return () -> augmentedDP(DP(α)) # returns a lambda function
end
```

Listing 6.2: *Dirichlet Process with base distribution H_0 written in Julia.*

6.1.2 Stochastic memoization

Discrete Random Probability Measures can be seen through the interesting perspective of *stochastic memoization* [1], which is an in-between (deterministic) memoization and no memoization. Higher-order procedures such as `DPmem` (defined above) transform a procedure into one that sometimes reuses its returned values, and sometimes generate new values. These procedures are called *stochastic memoizers* by Goodman et al. [35]. For instance with the DP, setting $\alpha = 0$ recovers a deterministic memoization whereas setting $\alpha \to \infty$ yields no memoization at all.

[1]`https://probmods.org/chapters/12-non-parametric-models.html`

6.1.3 Prior sampling algorithm

We have implemented in Listing 6.1 the stick-breaking construction with a recursive function. Yet, this is not the most efficient implementation of such a generative process. Consider that the concentration α is quite big, then the sticks $V_k \sim \text{Beta}(1, \alpha)$ will tend to be small. Consequently, calling `DP` (thus `pickStick(sticks, 1)`) will tend to flip many coins before one yields head (way more than the number of atoms already sampled). Having to go through lots of these sticks, for each point that is sampled from the DP sample can be computationally intensive.

Since $\sum_{k=1}^{\infty} p_k = 1$, the DP can be seen as an infinite categorical distribution. Yet, it can be lazily cast as a finite categorical distribution with $k + 1$ categories, after having sampled k atoms; the k existing atoms and one other category corresponding to a non-sampled yet atom. The weights of such a finite categorical distribution are $(\tilde{p}_1, \dots, \tilde{p}_k, 1 - \sum_{k=1}^{\infty} \tilde{p}_k)$. Such weights can easily be computed since $V_k \sim \text{Beta}(1, \alpha)$ and $\tilde{p}_k = V_k(1 - \sum_{j=1}^{k-1} \tilde{p}_j)$.

We have empirically assessed the computation needs of these two implementation by sampling 10000 different DPs, and then sampling 1000 points from each of them. Results have been summed up in Figure 6.1. By using a recursive sampling, it appears that the computation time grows linearly with α, whereas it is constant by using the previously described algorithm.

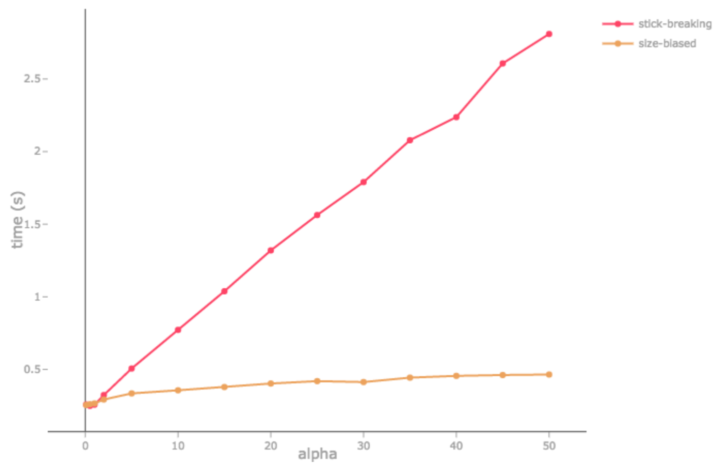


Figure 6.1: *Computational time to sample* 1000 *points from a* DP*, averaged on* 10000 *runs.*

Even if the stochastic recursion perspective is an elegant one, the categorical sampling algorithm is a better way to sample from a DP from a computational point of view.

6.1.4 Existing work

Most high-order PPLs indeed already handle some BNP models [35, 92, 37]. See [2] for Anglican's examples of Hierarchical Dirichlet Process or Probabilistic Deterministic In-

[2]`http://www.robots.ox.ac.uk/~fwood/anglican/examples/index.html`

finite Automata. See also [3] for WebPPL's example of Infinite Hidden Markov Models or Infinite Relational Model. Yet, their experimentations are mostly limited to Dirichlet Process, Pitman-Yor process or hierarchical versions of these processes, and often focused on sampling from priors. When performing inference, the used scheme is usually a particle algorithm such as Sequential Monte Carlo or Conditional Sequential Monte Carlo.

6.2 Generative constructions of random probability measures

Previously, we recalled that to be able to denote a model in a PPL, one should know how to sample from its prior, i.e. know a generative construction.

6.2.1 Stick-breaking processes

The first comprehensive treatment of stick-breaking priors dates back to the paper by Ishwaran and James [42]. There, they introduced a class of stick-breaking priors including as special cases the Dirichlet Process by Ferguson [30] and the two parameter Poisson-Dirichlet process by Perman et al. [70]. Specifically, let H_0 be a nonatomic probability measure on a complete and separable metric space $\mathbb{X}$. Also, let $(V_i)_{i\geq 1}$ be a sequence of independent random variables such that $\sum_{i\geq 1}\mathbb{E}[\log(1-V_i)] = -\infty$. Based on such V_i's, they define a sequence of random probabilities $(\tilde{p}_k)_{k\geq 1}$ as $\tilde{p}_1 = V_1$ and

$$\tilde{p}_k = V_k \prod_{j=1}^{k-1}(1-V_j) \tag{6.1}$$

for each $k > 1$, and let $(X_k^\star)_{k\geq 1}$ be a sequence of random variables, independent of $(\tilde{p}_k)_{k\geq 1}$, and independent and identically distributed according to H_0. Then,

$$P = \sum_{k\geq 1} \tilde{p}_k \delta_{X_k^\star}$$

is a stick-breaking prior in the class of Ishwaran and James [42].

For any $\sigma \in [0,1)$ and $\theta > -\sigma$, the stick-breaking representation of the two parameter Poisson-Dirichlet process is recovered by assuming the V_k's to be distributed according to the Beta distribution with parameter $(1-\sigma, \theta + k\sigma)$ for each $k \geq 1$. The stick-breaking representation of the DP, which was first derived by Sethuraman [81], is also recovered as special case, by setting $\sigma = 0$.

Apart from the two parameter Poisson-Dirichlet process, most of the discrete random probability measures do not admit a stick-breaking representation in terms of a collection of independent V_i's.

As an example, Favaro et al. [24] derived the stick-breaking representation of the normalized inverse Gaussian process introduced in Lijoi et al. [49]. Specifically, let $b > 0$ and let $(V_k)_{k\geq 1}$ be a sequence of dependent random variables such that, for each $k \geq 1$, the conditional distribution of V_k given $(V_1, \ldots, V_{k-1})$ coincides with the distribution of the random variable

$$\frac{X_k}{X_k + Y_k} \tag{6.2}$$

[3] https://probmods.org/chapters/12-non-parametric-models.html

where X_k is distributed according to the generalized inverse Gaussian distribution and Y_k is distributed according to the positive $\frac{1}{2}$-stable distribution.

According to Favaro et al [25], the normalized inverse Gaussian process [24] is the first example of a prior admitting a stick-breaking representation in terms of dependent V_k's, and such that for any $k \geq 1$ the conditional distribution of V_k given $(V_1, \dots, V_{k-1})$ is characterized by means of a straightforward transformation of random variables as in 6.2. Favaro et al [25] construct a similar transformation to build a stick-breaking process for a subclass of σ-stable Poisson-Kingman processes. Similarly, James [43] builds a stick-breaking process for the class of $\text{PG}(\alpha, \zeta)$-Generalized Gamma Processes.

Since the size-biased sequence of weights $(\tilde{p}_k)_{k\geq 1}$ is defined through Equation 6.1 for stick-breaking processes, it can be generated in the same way as described in the previous Section 6.1: drawing Bernoulli's with parameters V_k's until success, then return integer k.

6.2.2 Size-biased generative processes

The PKP generative process (Size-Biased Sampling) described in Section 4.5.2, is reminiscent of the well known stick breaking construction from Ishwaran & James [42], where a stick of length one is successively broken as in Figure 6.3. This Size-Biased Sampling (SBS) process is illustrated in Figure 6.2 and schematically recalled below:

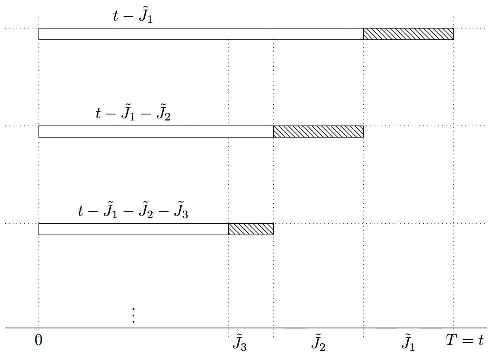
$$\begin{aligned}
T &\sim \gamma \\
\tilde{J}_1 | T &\sim \text{SBS}(T) \\
\tilde{J}_2 | T, \tilde{J}_1 &\sim \text{SBS}(T - \tilde{J}_1) \\
&\vdots \\
\tilde{J}_l | T, \tilde{J}_1, \dots, \tilde{J}_{l-1} &\sim \text{SBS}(T - \sum_{i<l} \tilde{J}_i) \\
&\vdots
\end{aligned}$$

However, by starting with Equation (4.15), one can recover the usual stick-breaking construction due to two useful identities (in distribution):

$$\tilde{p}_k := \frac{\tilde{J}_k}{T} \text{ for } k = 1, \dots \tag{6.3}$$

$$V_k := \frac{\tilde{p}_k}{1 - \sum_{j=1}^{k-1} \tilde{p}_j} \text{ for } k = 1, \dots \tag{6.4}$$

Indeed, equivalently to Equation (6.4), we have $\tilde{p}_k = V_k \prod_{j=1}^{k-1} (1 - V_j)$ for all $k \geq 1$, which is of the same form as Equation (6.1) for the usual stick-breaking process. In general, the stick random variables $(V_k)_{k\geq 1}$ from Equation (6.3), form a sequence of dependent random variables, except for some simple processes as the DP and PY, see Pitman [71] for details. Similarly to stick-breaking constructions, Poisson Kingman Processs can therefore be constructed in a generative manner and implemented via (stochastic) recursion.



$t - \tilde{J}_1$
$t - \tilde{J}_1 - \tilde{J}_2$
$t - \tilde{J}_1 - \tilde{J}_2 - \tilde{J}_3$
0
$\tilde{J}_3$
$\tilde{J}_2$
$\tilde{J}_1$
$T = t$

Figure 6.2: *Generative process of Poisson-Kingman Process. Source: [52].*

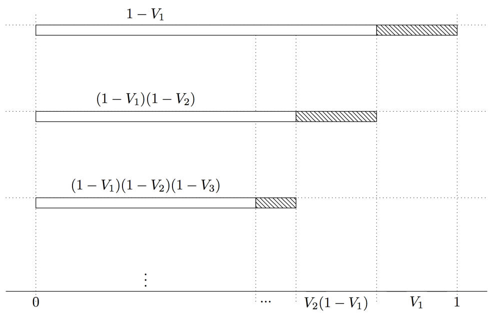


Figure 6.3: *Stick-breaking construction. Source: [52].*

Practical aspects of SBS A useful identity for computing sticks length $(V_k)_{k\geq 1}$ is given by

$$V_k = \frac{\tilde{J}_k}{T_{k-1}} \quad \text{for } k = 1, \dots \tag{6.5}$$

Consequently, to compute V_k, one only needed to sample $\tilde{J}_k | T, \tilde{J}_1, \dots, \tilde{J}_{k-1}$, and to update the surplus mass given by $T_1 = T$ and $T_k = T - \tilde{J}_k$ for $k \geq 2$. The normalised weights $(\tilde{p}_k)_{k\geq 1}$ are therefore never computed.

In the Size-Biased Sampling process (described in Section 4.5.2), when a random variable X_i takes the value of an atom which has not yet been sampled, a new atom $X^{\star}_{k+1}$ must be sampled, along with its associated weight $\tilde{J}_{k+1}$. The new atom $X^{\star}_{k+1}$ is drawn from H_0, while its associated normalised weight is drawn from

$$\mathbb{P}_{\rho,H_0}(\tilde{J}_{k+1} \in ds_{k+1} | T \in dt_k) = \frac{s_{k+1}}{t_k} \rho(ds_{k+1}) \frac{f_\rho(t_k - s_{k+1})}{f_\rho(t_k)} \tag{6.6}$$

Yet, even if we know the density of the distribution of $\tilde{J}_k | T, \tilde{J}_1, \dots, \tilde{J}_{k-1}$, we do not necessarily know how to sample from it. We are currently working on a generic way to sample from such as distribution. We could restrict to Lévy measures with tractable intensity – which can be evaluated pointwise – thus excluding σ-Stable PK. A simple rejection sampling scheme with proposal $\mathcal{U}(0, T_k)$ might be sufficient. Nonetheless, there are particular cases for which we already know how to sample from these distributions. Lomeli et al [53] proposes a simple rejection sampling scheme with uniform proposal to tackle the $-\log$Beta PK process. Favaro et al [25] gave an identity from which iid samples can be drawn for a subclass of σ-Stable PKP.

One of our contribution is the implementation of building blocks to sample from Normalised Random Measures and Poisson Kingman Processs. The code can be found online [4]. The only function which needs to be implemented to be able to sample from a specific class of Random Probability Measure is `sampleWeight` which samples $\tilde{J}_{k+1}$ given $\{T, \tilde{J}_1, \dots, \tilde{J}_k\}$. We have specifically implemented this for the DP and the $-\log$Beta PK process. See for instance samples from a $-\log$Beta PK process in Figure 6.4.

[4] `https://github.com/emilemathieu/Turing.jl/tree/feature-328/src/models`

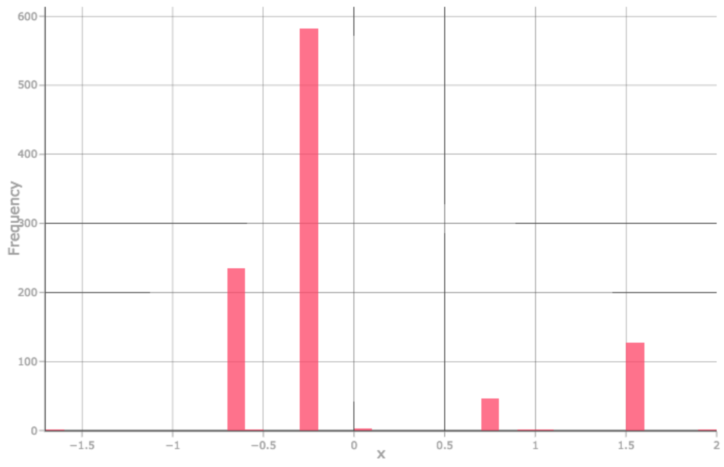


Figure 6.4: *Histogram of* 1000 *samples from a* $-\log Beta(1,2)$ *Poisson Kingman random measure.*

6.3 Models of interest

We focus on infinite mixture models described in Section 4.4. We use a Dirichlet Process as a prior over mixture components so as to build an infinite Gaussian mixture model. Yet, the DP could be replaced by another Normalised Random Measure or Poisson Kingman.

More specifically we consider for our experimentations the following hierarchical model

$$\begin{aligned} \theta &\sim \text{InverseGamma}(\alpha_0, \beta_0) \\ P &\sim \text{DP}(\alpha, \mathcal{N}(m_0, \sqrt{v_0})), \\ X_i|P &\sim P, \\ Y_i|X_i &\sim \mathcal{N}(X_i, \sqrt{\theta}) \end{aligned}$$

The equivalent model written in Turing can be found in the code sample 6.3 below. The real code of the size-biased processes and the infinite mixture model which has been used for our experimentations can be found on Github [5].

```
@model InfiniteMixutre(y) = begin
  N = length(y)
  α = 10.0
  θ ~ InvGamma(precisionShape, precisionScale) # Sample statement
  P ~ DP(α, Normal(meanMean, meanStd)) # Sample statement

  x = zeros(N)
  for i in 1:N
    x[i] ~ P # Sample statement
```

[5] https://github.com/emilemathieu/Turing.jl/tree/feature-328/src/models

```
10      y[i] ~ Normal(x[i], sqrt(θ)) # Observe statement
11    end
12  end
```

Listing 6.3: *Nonconjugate infinite mixture model written in Turing.jl.*

This model is a Gaussian mixture model with a shared variance of all clusters according to line `10`. This variance is distributed according an inverse gamma distribution as specified in line `4`. The base measure of the DP is a normal distribution as shown in line `5`.

Turing uses an unique symbol $\sim$ to denote sampling and observation. This model (function) is then compiled to replaced this symbol by the appropriate function: `sample` or `observe`. If the left-hand side term of $\sim$ is unknown, it is a random variable and `sample` is consequently chosen. Otherwise, if the term is a deterministic value (such as `y[i]`s), `observe` is chosen. This is written in the comments of the above code sample 6.3 after symbols `#`.

6.4 Posterior samplers

Now that we have described and implemented a way to sample from the (prior) distribution of an infinite mixture model, we focus on the issue of sampling from the posterior distribution $p(\theta, x_{1:T}|y_{1:T}) \propto p(\theta, x_{1:T}, y_{1:T})$. In Chapter 5 on Probabilistic Programming Languages, we presented many different inference schemes. These schemes are generic, meaning that for any model, they will produce samples that are distributed according to the targeted posterior distribution. Yet, this assertion does not give any guaranty on the quality of the produced samples.

Indeed, PPLs makes a trade-off between flexibility and efficiency. In an high-order PPL where a huge class of models can be represented, it is hard to develop inference schemes which are efficient for all possible models. Whereas for a first-order PPL as Stan which can only represent finite dimensional graphical models, developing an efficient inference scheme is easier since the denotable class of models is relatively small. By *efficient*, we mean for a sampler to be able to produce samples with "high" *quality* for some given computational resources. The quality is usually defined in term of Effective Sample Size (ESS) [6] which is a measure of correlation between produced samples. The spirit of PPLs from our belief is then to have multiple inference schemes, which could be combined, such that these schemes are efficient on different subclasses of representable models.

The goal of this internship has therefore been to find or develop an efficient algorithm to produce samples distributed according to the posterior distribution of the previously described infinite mixture model.

6.4.1 Sequential Monte Carlo

By writing the infinite mixture model as in Listing 6.3 above – using the `for-loop` form – the model is expressed as a State-Space Model. Such a formulation allows to make use of particle algorithms (see Subsections 5.5.4 and 5.5.5) thanks to the sequence of `observe` statements.

[6] `https://en.wikipedia.org/wiki/Effective_sample_size`

As described in Section 5.5.4, the Sequential Monte Carlo (SMC) algorithm breaks down the overall inference problem into a series of target distributions which get incrementally closer to the distribution of interest. These intermediate target distributions live in smaller spaces than the overall posterior distribution. This class of methods is consequently well suited as a generic inference method for the high-dimensional models that we are considering. It is straightforward to prove that by using an SMC scheme within a PPL with our model, we are indeed rightly targeting the posterior distribution.

6.4.2 Particle Gibbs

By using SMC, the random global parameter θ (precision of the Gaussian mixtures) would also be included in the latent space and sampled at the same time as the mixture components and assignments. More precisely, it would be re-sampled with all hidden (non-observed) variables appearing before the first `observe statement`; therefore with x_1 (`x[1]` at line `9`). If the mixing of the posterior samples of x_1 (and consequently of θ) is poor, it would negatively affect the mixing of the $(x_2, \dots, x_T)$ samples.

Particle Markov chain Monte Carlo methods – introduced by Andrieu et al [3] – alleviate this issue by explicitly targeting $p(\theta, x_{1:T}|y_{1:T}) \propto p(\theta, x_{1:T}, y_{1:T})$, thus decoupling the global parameters θ from the (states) $x_{1:T}$. These algorithms – such as Particle Gibbs and Particle Marginal Metropolis-Hastings – use SMC algorithms to design efficient high dimensional proposal distributions for MCMC algorithms.

As already described in Section 5.5.5, the PG algorithm is a Gibbs sampler which iteratively samples from $p(\theta|x_{1:T}, y_{1:T})$ and $p_\theta(x_{1:T}|y_{1:T})$, with $x_{1:T} \sim p_\theta(\cdot|y_{1:T})$ being generated by a Conditional SMC update. This update is similar to a standard SMC algorithm but is such that a prespecified path $x^\star_{1:T}$ (the previously generated value $x_{1:T}$ in the Gibbs step) is ensured to survive all the resampling steps, whereas the remaining $N-1$ particles are generated as usual.

We therefore experimented using Particle Gibbs targeting the cluster components and assignments $x_{1:T}$ with the conditional SMC and the global parameter θ with an HMC. The PG algorithm was already implemented in Turing, so there is no contribution from us for the PG algorithm. All our experimentations are using the galaxy data set [7] as observations, which consists of the velocities at which 82 galaxies are receding away from our own. We are using Turing for our experimentations. Figure 6.5 shows 100 samples from the posterior distributions of mixture components X_1 and X_{82}. These samples have been generated from a PG with 50 particles. The ESS is computed on 50 samples for each t, and these ESSs are then averaged on 8 runs. One can note from Figure 6.5 that samples from X_1 are very “sticky”, compared to samples from X_{82}. Figure 6.6 highlights this phenomenon, since we can see that the first X_t's drastically have lower ESS (meaning poorer mixing) than the X_t's at the end of the sequence (t close to 82).

Indeed, a drawback of PG is that it can be particularly adversely affected by *path degeneracy* in the CSMC step. Conditioning on an existing trajectory means that whenever

[7]Consists of the following values: 9172,9350,9483,9558,9775,10227,10406,16084,16170,18419,18552,18600, 18927,19052,19070,19330,19343,19349,19440,19473,19529,19541,19547,19663,19846,19856,19863,19914,19918, 19973,19989,20166,20175,20179,20196,20215,20221,20415,20629,20795,20821,20846,20875,20986,21137,21492, 21701,21814,21921,21960,22185,22209,22242,22249,22314,22374,22495,22746,22747,22888,22914,23206,23241, 23263,23484,23538,23542,23666,23706,23711,24129,24285,24289,24366,24717,24990,25633,26960,26995,32065, 32789,34279.

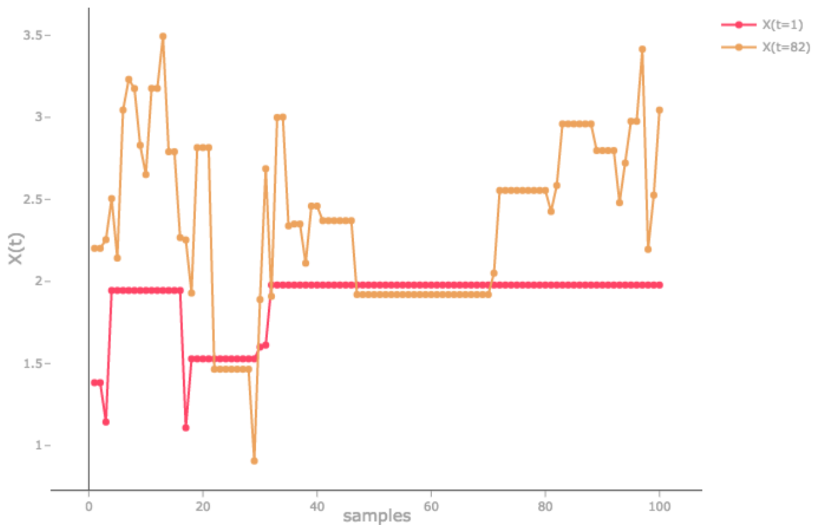


Figure 6.5: *Samples from the Particle Gibbs sampler (with 50 particles) for the first ($t = 1$) and the last ($t = 82$) mixture components.*

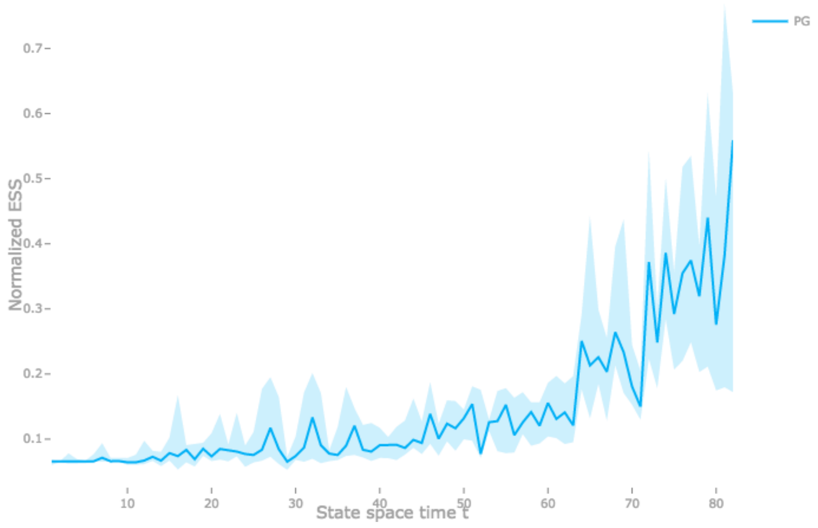


Figure 6.6: *Normalized Effective Sample Size (ESS) for 50 samples generated via Particle Gibbs with 50 particles. This ESS is averaged on 8 runs.*

resampling of the trajectories results in a common ancestor, this ancestor must correspond to this trajectory. Consequently, the mixing of the Markov chain for the early steps in the state sequence can become very slow when the particle set typically coalesces to a single ancestor during the CSMC sweep. The higher is the number of particles used in the Particle Gibbs algorithm, the better is the mixing of the generated samples. Yet having a fairly good ESS for all indices t is computationally out of reach since computation time grows linearly with the number of particles.

6.4.3 Particle Marginal Metropolis-Hastings

Another well-known PMCMC algorithm is PMMH, introduced by Andrieu et al [3]. PMMH is an Metropolis-Hastings algorithm with the following form of proposal density:

$$q\left(\theta^\star, x^\star_{1:T}|\theta, x_{1:T}\right) = q(\theta^\star|\theta)p_{\theta^\star}(x^\star_{1:T}|y_{1:T})$$

for which the proposed sample $x^\star_{1:T}$ is given by a SMC algorithm targeting $p_{\theta^\star}(x_{1:T}|y_{1:T})$.

One of our contribution is the implementation of this PMMH algorithm in the Turing [8] Probabilistic Programming Language. The code has indeed been merged in the official repository and can be found online [9]. Our implementation alternates between sampling $\theta^\star \sim q(\cdot|\theta)$, with q being by default θ's prior. In that case $\theta^\star$ is sampled by running the model (i.e. the program) and returning the value of θ sampled. Otherwise, the user could specify a proposal, such as a Gaussian kernel (see Listing 6.4).

```
q_θ =(θ) ->Normal(θ, 0.1*std(InvGamma(precisionShape, precisionScale)))
N_samples = 50
N_particles = 50
sampler = PMMH(N_samples, SMC(N_particles, :x), (:θ, q_θ))
```

Listing 6.4: *PMMH sampler constructor written in Turing.jl.*

An issue which often arises when working with non prior-proposal is that the proposed value may not lies in the support of its distribution. This can frequently happen for instance with a Gaussian kernel with high variance proposing values for a Gamma random variable having positive support; a negative might be proposed. The easiest way to solve this issue is simply to reject the proposed value and to go on the next iteration of the MH algorithm. One might think about sampling from the proposal until the proposed value lies on its distribution's support, which is equivalent to sampling from a truncated proposal. Yet, such a scheme is invalid if the acceptance ratio is not modified. A nice blog post [10] explains how to correct the acceptance ratio for Gaussian proposal so that the MH update be correct.

We ran similar experimentations than for the PG sampler so as to assess the quality of samples produced by our PMMH algorithm. For each t we compute the ESS of the 50 points sampled by the algorithm, and averaged these ESSs on 8 runs. We used 50 particles and a Gaussian proposal $q(\cdot|\theta) = \mathcal{N}(\cdot|\theta, 4.5)$ for the global parameter update. Figure 6.7 shows the results of these experimentations. Since there is no retained particle to condition on (as for PG), we do not observe the path degeneracy problem anymore. Yet, posterior samples have really low ESS, they are very sticky. In our experimentations the mean acceptance rate was around 0.1, which is way lower than the optimal acceptance rate. Adaptively updating the variance of the Gaussian kernel so as to target to optimal acceptance rate could improve a bit this sampler.

6.4.4 Particle Gibbs with Ancestor Sampling

To our knowledge, the first article to tackle the path degeneracy problem is the PGAS paper [51] (described in Subsection 5.5.5). Yet, Van de Meent et al [86] who proposed an adapted

[8]https://github.com/yebai/Turing.jl
[9]https://github.com/yebai/Turing.jl/blob/master/src/samplers/pmmh.jl
[10]http://fsaad.scripts.mit.edu/randomseed/metropolis-hastings-sampling-with-gaussian-drift-proposal-on-bounded-s

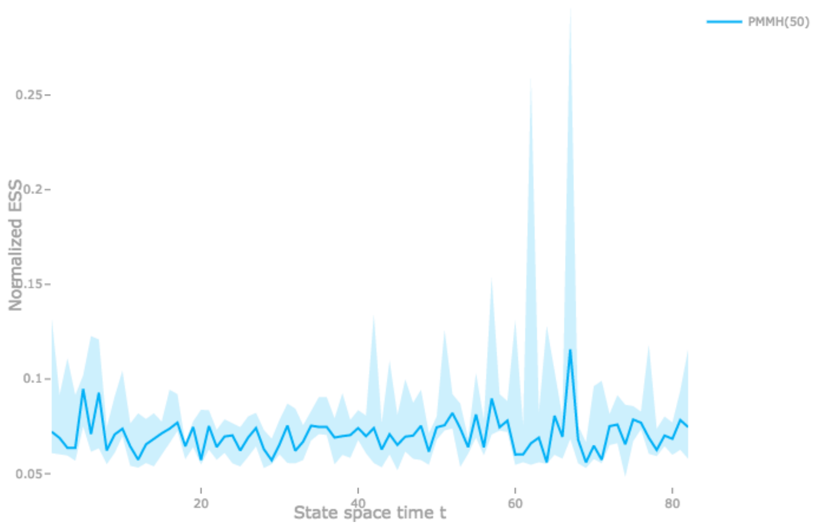


Figure 6.7: *Normalized Effective Sample Size (ESS) for 50 samples generated via the PMMH algorithm with 50 particles. The ESSs are averaged on 8 runs.*

version for PPLs, wrote "Implementing PGAS in the context of probabilistic programs poses technical challenges when programs can make use of recursion and memoization". PGAS is therefore non-trivial to implement so that it can handle the type of models we are interested in, which make use of memoization.

6.4.5 Interacting Particle Markov Chain Monte Carlo

Hopefully, Rainforth et al [74] developed another algorithm to alleviate the path degeneracy issue. In IPMCMC, a pool of CSMC and unconditional SMC algorithms are ran as parallel processes, referred as nodes. After each run of this pool, successive Gibbs updates to the indexes of the CSMC nodes are applied, such that the indices of the CSMC nodes changes. Hence, the nodes from which retained particles are sampled can change from one MCMC iteration to the next. This lets trading off exploration (SMC) and exploitation (CSMC) to achieve improved mixing of the Markov chains. Crucially, the pool provides numerous candidate indices at each Gibbs update, giving a significantly higher probability that an entirely new retained particle will be switched in than in non-interacting alternatives.

Another of our contribution is the implementation of this IPMCMC algorithm in Turing. The implementation does not yet make use of parallelization, but sequentially samples from CSMC and SMC nodes, by making use of existing implementations of these two algorithms. The code can be found online [11], it is currently reviews as a pull request so as to be merged in the official repository [12]. In the Gibbs sampling step of the new indices of the CSMC nodes, we have to sample from a categorical distribution but we only have the log-weights for numerical stability reasons. We therefore make use of a smart trick [13]

[11] see `https://github.com/emilemathieu/Turing.jl/blob/feature-334/src/samplers/ipmcmc.jl`
[12] `https://github.com/yebai/Turing.jl/pull/351`
[13] `https://stats.stackexchange.com/questions/64081/how-do-i-sample-from-a-discrete-categorical-distribution-in-`

using Gumbels distributions to sample from this categorical distribution and staying in the log-space.

We ran a similar experiment than for the previous samplers. We chose a IPMCMC sampler with 15 particles, 2 CSMC nodes and 2 SMC nodes. We reduced the number of particles compared to our experiment with PG and PMMH, so that the computational time be approximately similar. Figure 6.8 shows the average (also 1st and 3rd) ESS of the posterior samples for each x_t. Even if the results are a bit noisy, we can experientially see the improvement in sample quality compared to PG and PMMH. The ESS is relatively stable across all t and vary around 0.42. Moreover, Figure 6.9 shows posterior samples for x_1 and x_{82} generated by IPMCMC, similarly to Figure 6.5 for PG. We can see that neither samples of x_1 and x_{82} are sticky, especially compared with samples from PG. Figure 6.10 shows the corresponding histograms. Furthermore, Figure 6.11 shows posterior samples of $\frac{1}{\theta}$, along with a gamma distribution whose parameters have been fitted by maximising likelihood. We can conclude that the IPMCMC algorithm is the most efficient sampler we have considered for our model. Moreover it could be improved by running the CSMC and SMC nodes in a parallel way.

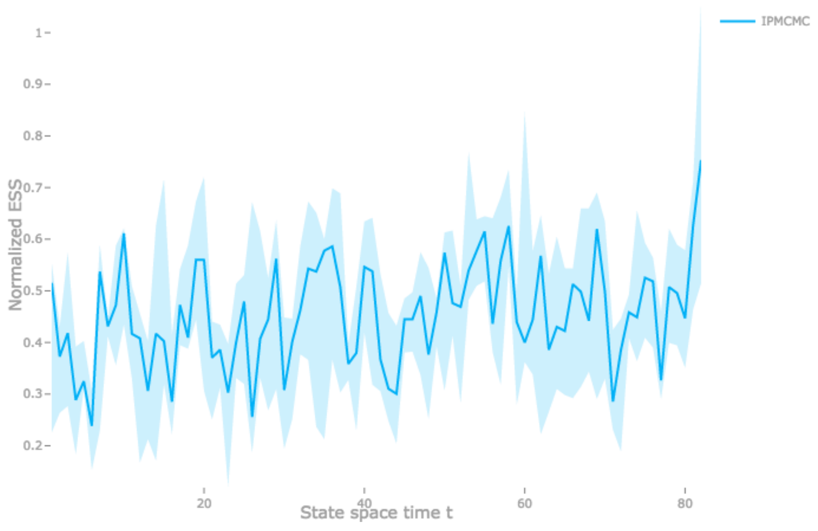


Figure 6.8: *Normalized Effective Sample Size (ESS) for 50 samples generated via the IPMCMC algorithm with 15 particles, 2 CSMS nodes and 2 SMC nodes. The ESSs are averaged on 8 runs.*

The current IPMCMC algorithm [74] does not allow the explicit sampling of the global parameters; similarly to the Conditional Sequential Monte Carlo algorithm. It only targets $p(x_{1:T}|y_{1:T}) \propto p(x_{1:T}, y_{1:T})$, with the global parameters θ being implicitly included in the states variables $x_{1:T}$ (specifically with x_1 in our case). I have therefore designed an extended version of this algorithm – and am currently working on the proof – which explicitly targets $p(\theta, x_{1:T}|y_{1:T})$. The implementation I have already implemented this new algorithm in Turing so that I can easily compare it to other existing algorithms such as PG and IPMCMC, and in the hope to be merged in the official repository of Turing. This work is likely to be submitted as a workshop paper. The main idea is to alternate sampling $x^j_{1:T} \sim p^j_\theta(\cdot|x^j_{1:T}, y_{1:T})$ for the CSMC nodes and $x^j_{1:T} \sim p^{\text{SMC}}_\theta(\cdot|y_{1:T})$ for SMC

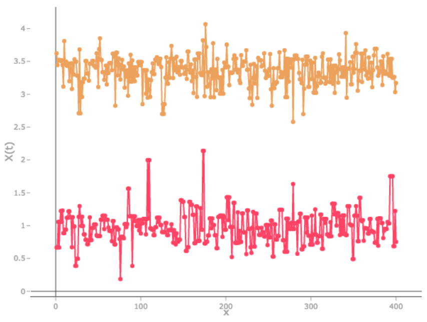


Figure 6.9: *Samples from the* IPMCMC *sampler of the first x_1 and the last x_{82} mixture components.*

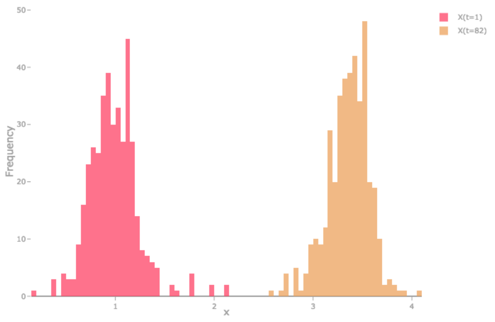


Figure 6.10: *Histograms of posterior samples of x_1 and x_{82} generated by an* IPMCMC *sampler.*

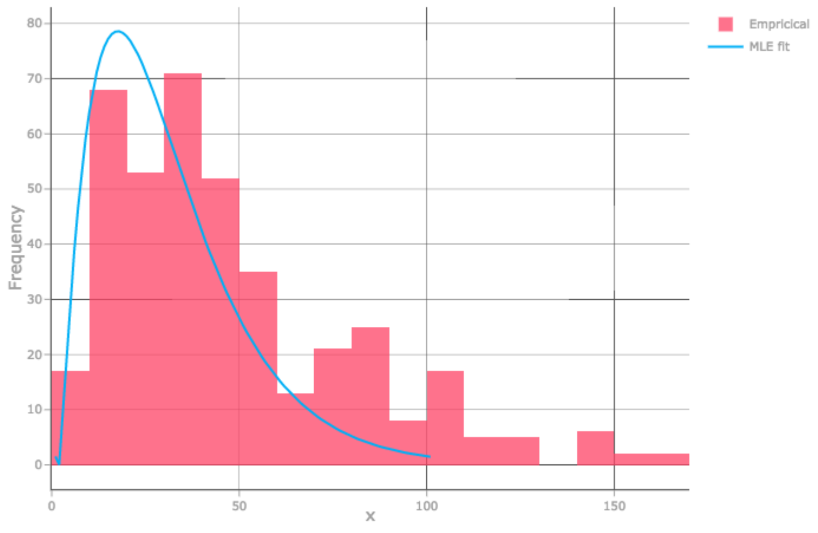


Figure 6.11: *Histogram of posterior samples of $\frac{1}{\theta}$, along with a Gamma distribution fitted by maximisation of the likelihood.*

nodes, with global parameter Gibbs steps $\theta^j \sim p(\cdot|x^j_{1:T}, y_{1:T})$ for CSMC nodes, but a unique step $\theta^{\text{SMC}} \sim p(\cdot|x^{\text{SMC}}_{1:T}, y_{1:T})$ (with $x^{\text{SMC}}_{1:T}$ being randomly chosen between CSMC's $\{x^j_{1:T}\}_{j=1,\dots,P}$). Listing 6.5 below shows how the constructor of this IPMCMC's extension is called with our implementation.

```
N_CSMC = 2
N_nodes = 2 * N_CSMC
N_samples = 50 / N_CSMC
N_particles = 50
sampler = IPMCMC(N_particles, N_samples, N_nodes, N_CSMC, HMC(1,0.2,3,:θ), :x)
```

Listing 6.5: *Extension of IPMCMC sampler constructor written in Turing.jl.*

We also ran experimentations on the extended version of IPMCMC, by using an HMC update for the global parameter θ (with 3 leapfrog steps of size 0.2). The mixing results are shown in Figure 6.12, unfortunately it is unclear that the mixing of posterior samples is improved compared to the usual IPMCMC. Further experimentations are needed, especially with more samples and more runs for averaging, since the noise of the experimentations is unfortunately quite high. It would also be interesting to compare algorithms by minimum ESS (across all t), divided by computation time. Indeed, the user is interested to have high quality sample for all t, in a minimum time. It would also be interesting to compare these metrics with samples generated by model-specific samplers, such as the ones described in Section 4.6.

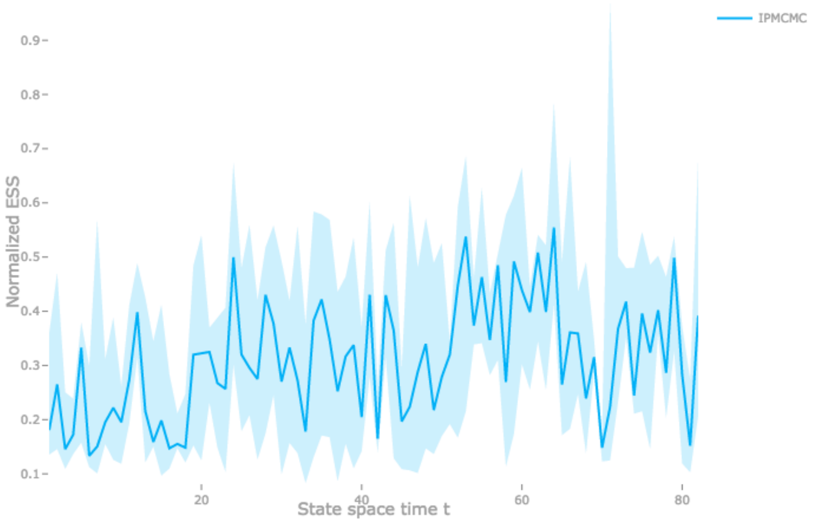


Figure 6.12: *Normalized Effective Sample Size (ESS) for 50 samples generated via the extended version of IPMCMC algorithm with 15 particles, 2 CSMS nodes and 2 SMC nodes. The ESSs are averaged on 8 runs.*

6.5 Open questions

In this section we introduce related open questions on which we are currently working on.

Models For now we focused on infinite mixture models but our approach may be easily extended to other BNP models, such as the infinite Hidden Markov Model [5] or latent feature models [33] since a stick-breaking process for the Indian Buffet Process has been developed [82].

Posterior sampler For now we worked on existing samplers (SMC, PG, PMMH, PGAS and IPMCMC), but we are thinking about developing (or extending) a sampler tailored for a specific model. A generic proposal, different (and hopefully better) than the prior might be developed. Moreover, an interesting idea from Del Moral & Murray [17] extending SMC could be used. They introduce a schedule of intermediate weighting and resampling times between observation times, which guide particles towards the final state.

Future Work

In this chapter we present several ideas which could be further developed, and if matured enough, these ideas may become projects on themselves.

7.1 Learning parameters in PPL

7.1.1 Motivation

Since variational methods have arisen, ideas of mixing sampling with VI have emerged, including in the PPL literature. In [90], the authors introduce the idea of automatically learning the parameters of proposals for SMC within a PPL. A lower bound on the *KL* divergence between the proposal and the true posterior distribution is optimize via gradient descent. In [77, 47], this idea is further developed using neural networks (such as LSTMs [40]) to parametrize these proposal distributions. These networks are fed with the previous latent and observed variables. In AESMC [48], FIVO [57] and VSMC [63], both the model and the SMC's proposal are learned by maximizing the marginal likelihood estimator given by the SMC. The interest in learning parameters (for proposals and for the model) and performing inference on some random variables at once is thus great. PPLs allow to easily write probabilistic models and perform inference on latent variables. One may be interested in building a PPL with the capability of automatically optimizing some parameters given a loss/estimator.

Automatic Differentiation (AD) methods [4] enable the computation of gradients of some variables with respect to some parameters. The reverse differentiation is particularly popular in the machine learning community, where the history of each variable (how it has been constructed) is saved as a computational graph, and gradients can then be computed via the *chain rule*. Some libraries such as TensorFlow [1] require the users to define static computation graphs within the syntactic and semantic constraints of a domain-specific mini language with limited support for control flow whereas the lineage of projects leading from autograd [1] to PyTorch [2] provide truly general-purpose reverse mode AD capability. The latter mode is to be preferred for fully and easy support of control flow such as stochastic recursion which is needed for stick-breaking processes.

[1]`https://github.com/HIPS/autograd`
[2]`http://pytorch.org/`

7.1.2 Choice of language/library

We this idea in mind, one can now think how to pragmatically build such a AD PPL.

Python: One of the most famous language for scientific computing is Python [79]. As *Edward* [85] is built on top of *Tensorflow*, one could build a PPL layer on top of PyTorch. *Edward* implements each MCMC step (specific for each algorithms) as a computational graph in *Tensorflow* which is thereafter run with the updated input so as to sample a new value. Similarly for VI, *Edward* implements a loss function as a computational graph, for which its gradient can be computed via auto-differentiation.

However, *Edward* focuses on VI and HMC-like schemes and does not handle particle algorithms. Indeed, so as to handle such algorithms, a PPL must have access to *breakpoints* at `assume` statement. This can be implemented via Continuation-Passing Style (CPS) [3] or coroutine [4] copying. Unfortunately implementing CPS is something non-trivial.

Julia: One could also think of using Julia [7], which has been specifically built for scientific usage. Julia has the advantage of natively handling coroutine copying, which is used in Turing [31] to implement particle methods. Reverse mode AD libraries exist in Julia, *ReverseDiff.jl* [5] and *Knet.jl* [6] which respectively build a static and dynamic graph. I am particularly interested in the perspective of adapting a AD library for Turing [31].

7.1.3 New models

With such as PPL in mind, one can think of new model or algorithms to be developed. The idea of AESMC [48] might be extended to PG and PMMH so as to learn proposals' parameters for their SMC and for $p(\theta^\star|\theta)$ parameters (specific of PMMH).

7.1.4 Difficulties

Yet this is not a trivial task, one have to put proper care when computing unbiased gradient of a loss function defined by an expectation over a collection of random variables. Hopefully, a stochastic computation graph [80] can be converted into a deterministic computation graph, to which the backpropagation algorithm can then be applied on a surrogate loss function which results in an unbiased gradient estimator of the loss.

7.2 Piecewise Deterministic Markov Processes

A novel class of non-reversible Markov chain Monte Carlo schemes relying on continuous-time piecewise deterministic Markov Processes has recently emerged [87]. In these algorithms, the state of the Markov process evolves according to a deterministic dynamics

[3]`http://matt.might.net/articles/by-example-continuation-passing-style/`
[4]`https://en.wikipedia.org/wiki/Coroutine`
[5]`https://github.com/JuliaDiff/ReverseDiff.jl`
[6]`https://github.com/denizyuret/Knet.jl`

which is modified using a Markov transition kernel at random event times. A general framework is presented in [8], and includes among others the Zig-Zag Process [9], the Bouncy Particle Sampler [12] and the Generalized Bouncy Particle Sampler [94].

It has been claimed [8] that the non-reversibility property of these algorithms enhances the mixing rate of the chain. I am consequently interested in understanding to what extent this class of MCMC schemes could fit the PPL's setting.

7.3 Variational Inference for bnp in ppl

As explained in Section 7.3, Truncation-free variational inference methods rely on a lazy representing of the clusters assignments. Yet, the marginalisation used seems to be only available for few models. However, we may be able to use a similar approach for more flexible BNP models, by extending the latent space with the sticks proportions and mixture components (since they cannot be marginalized out). Moreover, there might be a deeper link between *Truncation-free* VI and stick-breaking processes.

7.4 Adversarial Inference for bnp in ppl

Adversarial inference methods [22, 18, 61] inspired by GANs [34] jointly learns a generation network and an inference network using an adversarial process. The decoder/generator network $x' \sim p(x|z)$ maps samples from stochastic latent variables to the data space while the encoder/inference network $z' \sim q(z|x)$ maps training examples in data space to the space of latent variables. An adversarial game is cast between these two networks and a discriminative network is trained to distinguish between joint latent/data-space samples (x', z) from the generative network and joint samples (x, z') from the inference network.

Adversarial inference seems to be closely related to VI. Yet, in adversarial inference the model can also be learned as opposed to VI where only the proposal is learned. Moreover, in VI the marginal likelihood is optimised via a lower bound (ELBO) whereas in adversarial inference, a classification loss is optimised. This approach could be interesting in the BNP setting, if a tractable and tight lower bound on the marginal likelihood cannot be found.

Conclusion and personal review

8.1 Conclusion

After giving some background on the mission, we reviewed well-known Bayesian Nonparametric models, with a focus on infinite mixture models and discrete random probability measures. After we reviewed the design of Probabilistic Programming Languages and described the framework in which inference schemes can be generically applied for these programs. Then we recalled a generative construction of Poisson Kingman Processs called Size-Biased Sampling and stressed out that it relates with a high order PPLs, via stochastic recursion. We implemented such as a generative process for some BNP classes, along with a mixture model in a PPL named Turing, and ran experiments to assess the performance of posterior samplers.

Further work is still needed to design and implement an effective posterior sampler. Furthermore, this project could lead to other interesting ideas. Such as performing inference whithin a PPL with a Piecewise Deterministic MCMC scheme, with variational inference or maybe via adversarial inference. Also, enabling a PPL to automatically compute gradients via auto-differentation could lead to new models.

8.2 Personal review

I am much satisfied by this internship which brought me much experience and knowledge. I worked during four months on a subject involving Bayesian nonparametrics, Computational Statistics and Programing Languages, while my knowledge of these fields was limited. I therefore had to learn a lot about these, obviously by reading articles and books, but also by actually implementing some inference schemes. Talking with other students, post-docs and professors is also a stimulating way to get to know more about other related subjects. I was for instance invited by Frank Wood [1] in the Department of Engineering [2] to have a talk with him and some of his students. Reading groups are good places to frequently meet with the other students and keep up with the state of the art. I have been leading the probabilistic inference reading group [3] and will also participate to

[1]He works on Probabilistic Programming Languages, http://www.robots.ox.ac.uk/~fwood/
[2]http://www.eng.ox.ac.uk
[3]https://github.com/oxmlcs/ML_bazaar/wiki/Probabilistic-Inference

the Reinforcement Learning reading group [4] organised by Chris Maddison.

I also have much appreciated the great opportunities brought by this internship/PhD position, such as OxCSML [5] weekly talks [6] or other department's talks. I particularly enjoyed Judith Rousseau's [7] talk on posterior consistency and Jim Pitman's [8] talk on transformations of Brownian processes. I have also been invited for a week to the Microsoft Research AI Summer School 2017 [9] in Cambridge, where I met many other PhD students working in machine learning related areas.

Moreover, I made lots of progress in my research workflow. Before, coming here, even if I had already worked on several so called *research projects*, I had never actually done any research. I am still at the beginning of my research "career", yet I believe to better understand the spirit of research; iterating back and forth between questions and answers, by experimenting, reading, writing and discussing with others. I have learned to enjoy and take the most of the freedom and autonomy I possess by being a PhD student, which was quite destabilizing at first.

[4] http://www.stats.ox.ac.uk/~cmaddis/

[5] Oxford Computational Statistics & Machine Learning

[6] http://csml.stats.ox.ac.uk/events/

[7] https://www.ceremade.dauphine.fr/~rousseau/

[8] https://www.stat.berkeley.edu/~pitman/

[9] https://www.microsoft.com/en-us/research/event/ai-summer-school-2017/

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[2] D. Aldous, *Exchangeability and related topics*, in École d'Été St Flour 1983, Springer-Verlag, 1985, pp. 1–198. Lecture Notes in Math. 1117.

[3] C. Andrieu, A. Doucet, and R. Holenstein, *Particle Markov chain Monte Carlo methods*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72 (2010), pp. 269–342.

[4] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: a survey*, arXiv.org, (2015).

[5] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, *The infinite hidden markov model*, in Machine Learning, MIT Press, 2002, pp. 29–245.

[6] J. L. Bentley, *Writing Efficient Programs*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[7] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A Fresh Approach to Numerical Computing*, SIAM Review, 59 (2017), pp. 65–98.

[8] J. Bierkens, A. Bouchard-Côté, A. Doucet, A. B. Duncan, P. Fearnhead, G. Roberts, and S. J. Vollmer, *Piecewise Deterministic Markov Processes for Scalable Monte Carlo on Restricted Domains*, arXiv.org, (2017).

[9] J. Bierkens, P. Fearnhead, and G. Roberts, *The Zig-Zag Process and Super-Efficient Sampling for Bayesian Analysis of Big Data*, arXiv.org, (2016).

[10] D. M. Blei and M. I. Jordan, *Variational inference for Dirichlet process mixtures*, Bayesian Analysis, 1 (2006), pp. 121–143.

[11] D. M. Blei and C. Wang, *Truncation-free Stochastic Variational Inference for Bayesian Nonparametric Models*, Advances in Neural Information Processing Systems th Annual Conference on Neural Information Processing Systems, (2012), pp. 422–430.

[12] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet, *The Bouncy Particle Sampler: A Non-Reversible Rejection-Free Markov Chain Monte Carlo Method*, Journal of the American Statistical Association, 4 (2017), pp. 0–0.

[13] B. Carpenter, D. Lee, M. A. Brubaker, A. Riddell, A. Gelman, B. Goodrich, J. Guo, M. Hoffman, M. Betancourt, and P. Li, *Stan: A probabilistic programming language.*

[14] T. D. Chen and C. Fox, Emily B.and Guestrin, *Stochastic gradient hamiltonian monte carlo*, in International Conference on Machine Learning, 2014.

[15] G. Claeskens and N. L. Hjort, *Model selection and model averaging*, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge Univ. Press, Leiden, 2008.

[16] B. de Finetti, *Funzione caratteristica di un fenomeno aleatorio*, Atti della R. Accademia Nazionale dei Lincei, Ser. 6. Memorie, Classe di Scienze Fisiche, Matematiche e Naturali 4, (1931), pp. 251–299.

[17] P. Del Moral and L. M. Murray, *Sequential Monte Carlo with Highly Informative Observations*, SIAM/ASA Journal on Uncertainty Quantification, 3 (2015), pp. 969–997.

[18] J. Donahue, P. Krähenbühl, and T. Darrell, *Adversarial Feature Learning*, arXiv.org, (2016).

[19] R. Douc, O. Cappé, and E. Moulines, *Comparison of Resampling Schemes for Particle Filtering*, arXiv.org, (2005).

[20] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*, Springer New York, New York, NY, 2001.

[21] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, *Hybrid monte carlo*, Physics Letters B, 195 (1987), pp. 216 – 222.

[22] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, *Adversarially Learned Inference*, arXiv.org, (2016).

[23] M. D. Escobar and M. West, *Bayesian density estimation and inference using mixtures*, Journal of the American Statistical Association, 90 (1994), pp. 577–588.

[24] S. Favaro, A. Lijoi, and I. Prunster, *On the stick-breaking representation of normalized inverse Gaussian priors*, Biometrika, 99 (2012), pp. 663–674.

[25] S. Favaro, M. Lomeli, B. Nipoti, and Y. W. Teh, *On the stick-breaking representation of σ-stable Poisson-Kingman models*, Electronic Journal of Statistics, 8 (2014), pp. 1063–1085.

[26] S. Favaro, M. Lomeli, and Y. W. Teh, *On a class of σ-stable Poisson–Kingman models and an effective marginalized sampler*, Statistics and Computing, 25 (2014), pp. 67–78.

[27] S. Favaro and Y. W. Teh, *MCMC for Normalized Random Measure Mixture Models*, Statistical Science, 28 (2013), pp. 335–359.

[28] S. Favaro and S. G. Walker, *Slice sampling -stable poisson-kingman mixture models*, Journal of Computational and Graphical Statistics, 22 (2013), pp. 830–847.

[29] P. Fearnhead, *Particle filters for mixture models with an unknown number of components*, Statistics and Computing, 14 (2004), pp. 11–21.

[30] T. S. Ferguson, *A Bayesian analysis of some nonparametric problems*, The Annals of Statistics, 1 (1973), pp. 209–230.

[31] H. Ge, A. Ścibior, K. Xu, and Z. Ghahramani, *Turing: A fast imperative probabilistic programming language.*, (2016).

[32] Z. Ghahramani and M. J. Beal, *Propagation algorithms for variational bayesian learning*, in Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS'00, Cambridge, MA, USA, 2000, MIT Press, pp. 486–492.

[33] Z. Ghahramani and T. L. Griffiths, *Infinite latent feature models and the Indian buffet process*, Advances in neural information, (2006).

[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative Adversarial Networks*, arXiv.org, (2014).

[35] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, *Church: a language for generative models*, arXiv.org, (2012).

[36] N. D. Goodman and A. Stuhlmüller, *The Design and Implementation of Probabilistic Programming Languages.* `http://dippl.org`, 2014. Accessed: 2017-8-29.

[37] N. D. Goodman and J. B. Tenenbaum, *Probabilistic Models of Cognition.* `http://probmods.org/v2`, 2016. Accessed: 2017-9-5.

[38] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, *Probabilistic programming*, in Proceedings of the on Future of Software Engineering, FOSE 2014, New York, NY, USA, 2014, ACM, pp. 167–181.

[39] J. E. Griffin, *Sequential Monte Carlo methods for mixtures with normalized random measures with independent increments priors*, Statistics and Computing, 27 (2017), pp. 131–145.

[40] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[41] M. D. Homan and A. Gelman, *The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo*, J. Mach. Learn. Res., 15 (2014), pp. 1593–1623.

[42] H. Ishwaran and L. F. James, *Gibbs Sampling Methods for Stick-Breaking Priors*, Journal of the American Statistical Association, 96 (2001), pp. 161–173.

[43] L. F. James, *Stick-breaking PG(α,ζ)-Generalized Gamma Processes*, arXiv.org, (2013).

[44] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, *An Introduction to Variational Methods for Graphical Models*, in Learning in Graphical Models, Springer Netherlands, Dordrecht, 1998, pp. 105–161.

[45] J. Kingman, *Completely random measures*, Pacific Journal of Mathematics, 21 (1967), pp. 59–78.

[46] J. F. C. Kingman, *Poisson processes*, vol. 3 of Oxford Studies in Probability, The Clarendon Press Oxford University Press, New York, 1993. Oxford Science Publications.

[47] T. A. Le, A. G. Baydin, and F. Wood, *Inference Compilation and Universal Probabilistic Programming*, arXiv.org, (2016).

[48] T. A. Le, M. Igl, T. Jin, T. Rainforth, and F. Wood, *Auto-Encoding Sequential Monte Carlo*, arXiv.org, (2017).

[49] A. Lijoi, R. H. Mena, and I. Prünster, *Hierarchical Mixture Modeling With Normalized Inverse-Gaussian Priors*, Journal of the American Statistical Association, 100 (2005), pp. 1278–1291.

[50] A. Lijoi, R. H. Mena, and I. Prünster, *Controlling the reinforcement in bayesian non-parametric mixture models*, Journal of the Royal Statistical Society Series B, 69 (2007), pp. 715–740.

[51] F. Lindsten, M. I. Jordan, and T. B. Schön, *Particle Gibbs with Ancestor Sampling*, arXiv.org, (2014).

[52] M. Lomeli, *General bayesian inference schemes in infinite mixture models*, arXiv.org, (2017).

[53] M. Lomeli, S. Favaro, and Y. W. Teh, *A hybrid sampler for Poisson-Kingman mixture models*, arXiv.org, (2015).

[54] ——, *A Marginal Sampler for σ-Stable Poisson–Kingman Mixture Models*, Journal of Computational and Graphical Statistics, 26 (2017), pp. 44–53.

[55] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter, *Winbugs - a bayesian modelling framework: Concepts, structure, and extensibility*, Statistics and Computing, 10 (2000), pp. 325–337.

[56] Y.-A. Ma, T. Chen, and E. B. Fox, *A complete recipe for stochastic gradient mcmc*, in Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15, Cambridge, MA, USA, 2015, MIT Press, pp. 2917–2925.

[57] C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh, *Filtering Variational Objectives*, arXiv.org, (2017).

[58] V. Mansinghka, D. Selsam, and Y. Perov, *Venture: a higher-order probabilistic programming platform with programmable inference*, arXiv.org, (2014).

[59] A. McCallum, K. Schultz, and S. Singh, *FACTORIE: Probabilistic programming via imperatively defined factor graphs*, in Neural Information Processing Systems (NIPS), 2009.

[60] G. McLachlan and K. Basford, *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker, New York, 1988.

[61] L. M. Mescheder, S. Nowozin, and A. Geiger, *Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks*, in Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, 2017, pp. 2391–2400.

[62] T. Minka, J. Winn, J. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill, *Infer.NET 2.6*, 2014. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[63] C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei, *Variational Sequential Monte Carlo*, arXiv.org, (2017).

[64] C. Navarrete, F. A. Quintana, and P. Müller, *Some issues in nonparametric bayesian modeling using species sampling models*, Statistical Modelling, 8 (2008), pp. 3–21.

[65] R. M. Neal, *Markov Chain Sampling Methods for Dirichlet Process Mixture Models*, Journal of Computational and Graphical Statistics, 9 (2000), p. 249.

[66] R. M. Neal, *Slice sampling*, Ann. Statist., 31 (2003), pp. 705–767.

[67] R. M. Neal, *MCMC using Hamiltonian dynamics*, arXiv.org, (2012).

[68] Y. Nesterov, *Primal-dual subgradient methods for convex problems*, Math. Program., 120 (2009), pp. 221–259.

[69] B. Paige, F. Wood, A. Doucet, and Y. W. Teh, *Asynchronous Anytime Sequential Monte Carlo*, arXiv.org, (2014).

[70] M. Perman, J. Pitman, and M. Yor, *Size-biased sampling of Poisson point processes and excursions*, Probability Theory and Related Fields, 92 (1992), pp. 21–39.

[71] J. Pitman, *Random discrete distributions invariant under size-biased permutation*, Adv. in Appl. Probab., 28 (1996), pp. 525–539.

[72] J. Pitman, *Poisson-kingman partitions*, Lecture Notes-Monograph Series, (2003), pp. 1–34.

[73] ——, *Combinatorial stochastic processes*, vol. 1875 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 2006.

[74] T. Rainforth, C. A. Naesseth, F. Lindsten, B. Paige, J.-W. van de Meent, A. Doucet, and F. Wood, *Interacting Particle Markov Chain Monte Carlo*, arXiv.org, (2016).

[75] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press, 2005.

[76] E. Regazzini, A. Lijoi, and I. Prünster, *Distributional results for means of normalized random measures with independent increments*, Ann. Statist., 31 (2003), pp. 560–585.

[77] D. Ritchie, P. Horsfall, and N. D. Goodman, *Deep Amortized Inference for Probabilistic Programs*, arXiv.org, (2016).

[78] D. Ritchie, A. Stuhlmüller, and N. D. Goodman, *C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching*, arXiv.org, (2015).

[79] G. Rossum, *Python reference manual*, tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

[80] J. Schulman, N. Heess, T. Weber, and P. Abbeel, *Gradient Estimation Using Stochastic Computation Graphs*, arXiv.org, (2015).

[81] J. Sethuraman, *A constructive definition of Dirichlet priors*, Statistica Sinica, 4 (1994), pp. 639–650.

[82] Y. Teh, D. Görür, and Z. Ghahramani, *Stick-breaking construction for the indian buffet process*, in JMLR Workshop and Conference Proceedings Volume 2: AISTATS 2007, Cambridge, MA, USA, Mar. 2007, Max-Planck-Gesellschaft, MIT Press, pp. 556–563.

[83] R. Thibaux and M. I. Jordan, *Hierarchical beta processes and the indian buffet process.*, in AISTATS, M. Meila and X. Shen, eds., vol. 2 of JMLR Proceedings, JMLR.org, 2007, pp. 564–571.

[84] D. Titterington, A. Smith, and U. Makov, *Statistical Analysis of Finite Mixture Distributions*, Wiley, New York, 1985.

[85] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, *Deep probabilistic programming*, in International Conference on Learning Representations, 2017.

[86] J.-W. van de Meent, H. Yang, V. Mansinghka, and F. Wood, *Particle Gibbs with Ancestor Sampling for Probabilistic Programs*, arXiv.org, (2015).

[87] P. Vanetti, A. Bouchard-Côté, G. Deligiannidis, and A. Doucet, *Piecewise Deterministic Markov Chain Monte Carlo*, arXiv.org, (2017).

[88] M. Welling and Y. W. Teh, *Bayesian learning via stochastic gradient langevin dynamics.*, in ICML, L. Getoor and T. Scheffer, eds., Omnipress, 2011, pp. 681–688.

[89] D. Wingate, N. D. Goodman, and A. Stuhlmüller, *Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, USA, Apr. 2011, pp. 770–778.

[90] D. Wingate and T. Weber, *Automated Variational Inference in Probabilistic Programming*, arXiv.org, (2013).

[91] F. Wood and M. J. Black, *A nonparametric Bayesian alternative to spike sorting*, Journal of Neuroscience Methods, 173 (2008), pp. 1–12.

[92] F. Wood, J. W. van de Meent, and V. Mansinghka, *A new approach to probabilistic programming inference*, in Proceedings of the 17th International conference on Artificial Intelligence and Statistics, 2014, pp. 1024–1032.

[93] F. Wood, J.-W. van de Meent, and V. Mansinghka, *A New Approach to Probabilistic Programming Inference*, arXiv.org, (2015).

[94] C. Wu and C. P. Robert, *Generalized Bouncy Particle Sampler*, arXiv.org, (2017).