Probabilistic and Inferential Programming

Belhal Karimi
Mines ParisTech, Université Paris Sciences et Lettres

Juin 2016

Abstract

The following report is exploring all my areas of interests during my visit at the Probabilistic Computing Lab `http://probcomp.csail.mit.edu` at MIT, Brain and Cognitive Science Department.
I would like to thank Vikash Mansighka for his supervision throughout the visit. He allowed me to tackle several issues towards the field and developed several skill sets I needed to pursue a career in Technological fields.
I would like to thank the engineering team for its support on the tools and softwares and the research team for the multiple discussions that led to amazing results.

Contents

1 Introduction

My strong interest in computational social science since my master thesis on behaviors towards private data disclosure on the internet during my last year at Telecom ParisTech and my recent focus on applied mathematics in signal processing and machine learning led me directly to the several papers published by the Probabilistic Computing Project at the Brain and Cognitive Science Departement at the Massachusetts Institute of Technology.
Their willingness to apply Bayesian statistics to deeply understand the underlying causal inferences within for instance global development or healthcare datasets was exactly what I was expecting for my internship in academic research.
On one hand, this lab is building a new probabilistic programming language, like Stan or Julia, in order to give basic tools for non statisticians and non computer scientists to build simple and efficient inference program for their dataset. On the other hand, it has created a tool, called BayesDB, which relies on Bayesian statistics, in order to query and find the distribution of high dimensional dataset.

I started working on the different projects in November 2015 remotely from Paris. It allowed to get familiar with the tools and especially grasp the state of mind and the concept of the lab. When I arrived early January, my schedule became pretty clear in a week. I was going to the lab every day, enrolled to three classes in machine learning and entrepreneurship on Mondays, Wednesdays and Fridays (basically 1h30 per day) and had engineering meetings on Mondays at noon, reading groups on Wednesdays at 1 pm and student clinic (all the students and the supervisor gathering to share the progress on each project) followed by a social hour. My weekends were great to learn concepts I could not understand during the week and also to polish work when deadlines were earlier next week. Most of the work was done in the lab, at the brain and cognitive sciences department, except some conferences where the lab was showcasing some work. As far as task related scheduling, every day was kind of different working sometimes more on research and sometimes more on analysis for upcoming deadlines for the Gates Foundation for instance.

2 The probabilistic Computing Project

The MIT Probabilistic Computing Project aims to build software and hardware systems that augment human and machine intelligence. We are currently focused on probabilistic programming. Probabilistic programming is an emerging field that draws on probability theory, programming languages, and systems programming to provide concise, expressive languages for modeling and general-purpose inference engines that both humans and machines can use.

Our research projects include BayesDB and Picture, domain-specific probabilistic programming platforms aimed at augmenting intelligence in the fields of data science and computer vision, respectively. BayesDB, which is open source and in use by organizations like the Bill Melinda Gates Foundation and JPMorgan, lets users who lack statistics training understand the probable implications of data by writing queries in a simple, SQL-like language. Picture, a probabilistic language being developed in collaboration with Microsoft, lets users solve hard computer vision problems such as inferring 3D models of faces, human bodies and novel generic objects from single images by writing short (¡50 line) computer graphics programs that generate and render random scenes. Unlike bottom-up vision algorithms, Picture programs build on prior knowledge about scene structure and produce complete 3D wireframes that people can manipulate using ordinary graphics software. The core platform for our research is Venture, an interactive platform suitable for teaching and applications in fields ranging from statistics to robotics.

2.1 BayesDB

BayesDB is a Bayesian database that lets users query the probable implications of their data as easily as a SQL database lets them query the data itself. Using the built-in Bayesian Query Language (BQL), users with no statistics training can solve basic data science problems, such as detecting predictive relationships between variables, inferring missing values, simulating probable observations, and identifying statistically similar database entries.
BayesDB is suitable for analyzing complex, heterogeneous data tables with up to tens of thousands of rows and hundreds of variables. No preprocessing or parameter adjustment is required, though experts can override BayesDB's default assumptions when appropriate.
BayesDB's inferences are based in part on CrossCat, a new, nonparametric Bayesian machine learning method, that automatically estimates the full joint distribution behind arbitrary data tables.

2.2 Crosscat

CrossCat is a domain-general, Bayesian method for analyzing high-dimensional data tables. CrossCat estimates the full joint distribution over the variables in the table from the data, via approximate inference in a hierarchical, nonparametric Bayesian model, and provides efficient samplers for every conditional

distribution. CrossCat combines strengths of nonparametric mixture modeling and Bayesian network structure learning: it can model any joint distribution given enough data by positing latent variables, but also discovers independencies between the observable variables.

A range of exploratory analysis and predictive modeling tasks can be addressed via CrossCat[2], including detecting predictive relationships between variables, finding multiple overlapping clusterings, imputing missing values, and simultaneously selecting features and classifying rows. Research on CrossCat has shown that it is suitable for analysis of real-world tables of up to 10 million cells, including hospital cost and quality measures, voting records, handwritten digits, and state-level unemployment time series. Croscat is thus a metamodel creating models and analyzing them. Each "model" in the sense that you initialized above represents one Markov-Chain Monte-Carlo particle. In lay terms, one instance where the generative model can walk around a hypothesis space, one step per iteration. The hypothesis space for crosscat in particular explores the number of Gaussians to mix, their means and standard deviations, and which variables to group as sharing some of those parameters.

On its first step, a particle's parameters are drawn from its prior probability distribution, a naive one that we pre-specified. On each step thereafter, the particle chooses a nearby set of parameters that are both likely according to that prior, and likely to generate the data you observed (your csv). As the particle takes more steps, the data have more influence, and the prior has less.

The full model averages over all the particles. This can be a weighted average, if you initialized different generators different ways (we will talk more about that shortly), but by default we use equal weights.

The probability of dependence between two variables is the prevalence of being in the same group in these random walks. It approximates the probability that the mutual information between those two variables is non-zero. Note that it does not estimate the value of the mutual information. Note that the calculation is independent of the size of the inputs, or their particular values, and dependent instead on the generative process we choose and our exploration of its implications through these chains of steps in each particle's random walk. The essence of probabilistic programming, in this view, is in:

1. specifying separately the generative processes that we use to model data, (not always Gaussians)
2. specifying their prior probability distributions,
3. specifying the way to get to the next step in the chain,
4. specifying the interpretation of all the particles as an aggregate, and
5. measuring separately the performance of each of these factors.

For each iteration crosscat is doing gibbs step for all the rows within the views and all the columns within all the partitions (it can also create new partition)

and roll a die with each side having the log likelihood of the probability of this row given the data of the view it's in and the result will fix the view where the row has to go. Gibbs step decide where next step to go to.

2.3 VentureScript

VentureScript is a higher-order probabilistic programming language that aims to be sufficiently expressive, extensible, and efficient for general-purpose use. Some distinctive features include:

· Sufficient expressiveness to handle problems and data sources from multiple fields, such as machine learning, robotics and statistics.

· An inference programming language that supports combinations of exact and approximate inference techniques, including Metropolis-Hastings, mean field, sequential Monte Carlo, Hamiltonian Monte Carlo and gradient descent.

· A JavaScript-like front-end syntax and a read-write textual representation of the abstract syntax derived from Lisp

· A flexible foreign interface that makes it straightforward to add new primitives, including higher-order probabilistic procedures, exchangeable sequences, and "likelihood free" primitives, and to equip them with custom inference schemes.

· An interactive console that provides tools for inspection, profiling and debugging.

VentureScript is also the main language for programming the Venture platform.

3 US College Scorecard

This project is the first project that led me to using BayesDB, the AI assistant i've described earlier.
This dataset has been carefully chosen by me and Vikash. We needed to build an analysis where the historical ones were misleading the readers, inclined to emotional behaviors and definitely splitting the communiity with different point of views. The Education was ideal. Ironically enough, the College Scorecard dataset was published by the U.S. Department of Education in October 2015[1].

The Economist triggered discussion about this dataset in an article describing a ranking system based only on earnings after graduation[2]. While the vision of BayesDB, and of probabilistic programming languages in general, is to come as an AI-enhanced medium to create consensus on different fields where historically, researchers, scientists, domains experts, barely succeed in agreeing with each other, the analysis on the US Education Scorecard would give a good material to show how non experts of the domains (I am certainly not, besides the only input about the models are only mathematics wise and surely not involving cultural knowledge for instance) could compute logical and complete analysis of a specific field.

3.1 Problem Statement

This huge dataset is going to be a good field for BayesDB to showcase its ability to understand any dataset and extracting useful insights of the fiels.

· The issue is that traditional rankings of American colleges do not focus on many variables and base its core analysis on metrics related to graduates earnings. In the meantime, the challenge here is to be able to compute a transparent value-added for each college in order to quantify the salary boost the students are receiving from attending such schools.

· Also, current rankings prefer translating the opportunities given by a school, via its network, its partnerships... and not on the hard work and intelligence of its graduates.

· In this following document, we will ask BayesDB different questions in terms of relations between variables and distribution of some metrics to compare these results to our intuition and to other iterations of the same models.

As a result, we should expect our tool to take into consideration every relation between variables and show us intuitive dependencies and simulations.

We will organize the study in different parts:

[1] https://collegescorecard.ed.gov/data/
[2] http://www.economist.com/blogs/graphicdetail/2015/10/value-university

1. The first part will consist of analyzing a small subsample, given our initial selection of 50 variables, of 100 observations (colleges). We'll be plotting the dependencies heatmap and be adding more and more observations with fixed and variable number of models and iterations. Both cases would show interesting results.

2. Then, using gpmcc, a new implementation of CrossCat from the the lens of generative population model (GPM), we'll simulate distributions of a few variables and compare them with the existing distributions and our intuition. We'll try the same experiment by inferring missing values only.

3. On the same variables, we'll ask BayesDB to find unlikely data, whether they are outliers or input errors.

4. Finally, we'll simulate colleges by size, selectivity and students earnings after graduation and challenge stereotypes about colleges thanks to these results.

3.2 Dataset

The dataset published by the U.S. Department of Education is composed of 18 years (from 1996 to 2013) of data about more than 7,000 schools. More than 1,700 variables have been measured each year for each university. The number of universities fluctuates according to the creation of new schools and the closing of existing ones.

The overall dataset contains more than 215 million values with 43% of them missing.

Figure 1 highlights the number of variables measured each year. Obviously, we've been able to measure more and more variables through the years, but for some reason many data are missing in 2013 (the last year this study has been conducted). Obama's administration college database is way more transparent and better quality compared to datasets used by US rankings publishers. It has better data on family income, family education levels of entering students and more sophisticated measures of degree completion as well as loan repayment. These numbers have been generated in particular by matching students loans to their actual tax returns. As a result, we are able to compare professionals' earnings to their student characteristics.

Note: The data only includes students who applied for financial aid and thus is missing all the students with well-off parents. Also, as the earnings data only take into consideration 10 years after starting college, one could argue that this scope of time is too small to include future high earners, as they would still be students (e.g. Ph.D., post-doctorate).

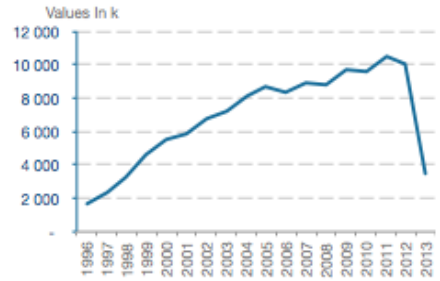


Figure 1: Data available per year

The number of schools from one year to another is also varying. Here is an overview of how many schools we have metrics of. One can tell that the overall trend is showing an increase in number of schools even though from 1996 to 1998 we can observe more schools closing than being created. One interesting result could be to characterize the fall of trends of these schools and be able to predict future outlook of current schools based on their most recent data.

```
bdball = bayeslite.bayesdb_open("dfall.bdb", builtin_metamodels=False)
bdbcontrib.barplot(bdball, '''
SELECT year ,
COUNT(OPEID) AS "Number of schools"
FROM dfall
GROUP BY year
ORDER BY year asc
''');
```

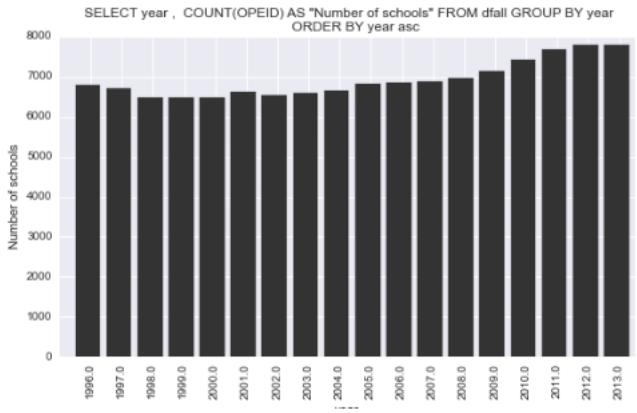


Figure 2: Number of colleges in the US per year

Our 57 variables include some financials and specs about the universities. Here is a full list of the variables selected for the study

var_name	description
Financials	
DEP_DEBT_MDN	The median debt for dependent students
LO_INC_DEATH_YR2_R	Percent of low income (between 0 and 30k in nominal family income) students who died within 2 years
MD_INC_DEATH_YR2_R	Percent of middle-income (between $30k and 75k in nominal family income) students who died within 2 years
HI_INC_DEATH_YR2_R	Percent of high income (more than 75k in nominal family income) students who died within 2 years
PELL_DEATH_YR2_RT	Percent of students who received a Pell Grant at the institution and who died within 2 years at original institution
NOPELL_DEATH_YR2_RT	Percent of students who never received a Pell Grant at the institution and who died within 2 years at original institution
LOAN_DEATH_YR2_RT	Percent of students who received a federal loan at the institution and who died within 2 years at original institution
NOLOAN_DEATH_YR2_RT	Percent of students who never received a federal loan at the institution and who died within 2 years at original institution
NOLOAN_ENRL_ORIG_YR	Percent of students who never received a federal loan at the institution and who were still enrolled at original institution within a year
DEATH_YR2_RT	Percent died within 2 years at original institution
LO_INC_COMP_ORIG_Y	Percent of female students who transferred to a 2-year institution and whose status is unknown within 8 years
MD_INC_COMP_ORIG_Y	Percent of middle-income (between $30k and 75k in nominal family income) students who died within a year
HI_INC_COMP_ORIG_Y	Percent of high-income (over in nominal family income) students who died within a year
PELL_COMP_ORIG_YR2	Percent of students who received a Pell Grant at the institution and who completed in 2 years at original
NOPELL_COMP_ORIG_YR2	Percent of students who did not receive a Pell Grant at the institution and who completed in 2 years at original
LOAN_COMP_ORIG_YR2	Percent of students who received a federal loan at the institution and who completed in 2 years
NOLOAN_COMP_ORIG_YR2	Percent of students who never received a federal loan at the institution and who were still enrolled at original institution within 2 years
MD_INC_RPY_5YR_RT	Five-year repayment rate by family income ($30k-75k)
GRAD_DEBT_MDN	The median debt for students who have completed
WDRAW_DEBT_MDN	The median debt for students who have not completed
LO_INC_DEBT_MDN	The median debt for students with family income between $0 and 30k
MD_INC_DEBT_MDN	The median debt for students with family income between $30k and 75k
HI_INC_DEBT_MDN	The median debt for students with family income between over 75k
IND_DEBT_MDN	The median debt for independent students
PCTPELL	Percentage of Pell Grant
AVGFACSAL	Average faculty salary
TUITIONFEE_PROG	TUITIONFEE_PROG
faminc	Average family income
md_faminc	Median family income
mn_earn_wne_p10	Mean earnings of students working and not enrolled 10 years after entry
md_earn_wne_p10	Median earnings of students working and not enrolled 10 years after entry
Ethnies	
PBI	Flag for predominantly black institution
AANAPII	Flag for Asian American Native American Pacific Islander-serving institution
MENONLY	Flag for men-only college
WOMENONLY	Flag for women-only college
Selectivity	
ADM_RATE_ALL	Admission rate for all campuses rolled up to the 6-digit OPE ID
SATVR25	25th percentile of SAT scores at the institution (critical reading)
SATVRMID	Midpoint of SAT scores at the institution (critical reading)
SATVR75	75th percentile of SAT scores at the institution (critical reading)
SATMT25	25th percentile of SAT scores at the institution (math)
SATMTMID	Midpoint of SAT scores at the institution (math)
SATMT75	75th percentile of SAT scores at the institution (math)
SATWR25	25th percentile of SAT scores at the institution (writing)
SATWRMID	Midpoint of SAT scores at the institution (writing)
SATWR75	75th percentile of SAT scores at the institution (writing)
SAT_AVG_ALL	Average SAT equivalent score of students admitted for all campuses rolled up to the 6-digit OPE ID
ACTCM25	25th percentile of the ACT cumulative score
ACTCMMID	Midpoint of the ACT cumulative score
ACTCM75	75th percentile of the ACT cumulative score
University specs	
st_fips	FIPS code for state
region	Region (IPEDS)
locale2	Degree of urbanization of institution
OPEID	8-digit OPE ID for institution
CCBASIC	Carnegie Classification -- basic
CCUGPROF	Carnegie Classification -- undergraduate profile
UGDS	Enrollment of undergraduate degree-seeking students
PFTFAC	Faculty Rate

Figure 3: Codebook for our variables

For a purpose of space I will attach my whole report on this analysis.

4 Bill&Melinda Gates Foundation

For a couple of years now, the Gates Foundation has been putting tremendous efforts to have an impact in the developing countries. In particular, they are collecting huge amount of data through surveys and on-field experiments in poor countries around the world. Data about education, health, finance, etc. led the Foundation to create a unique project, named HBGDki (for Healthy Birth, Growth and Development knowledge integration) aiming at developing knowledge towards all this data. The lab I joined is a member of the latter. The main duties I was assigned to were:

1. Developing new tools to allow program managers at Gates Foundation to access the full potential of Bayesian inference through our tool BayesDB.

2. Analyzing several datasets and show proof of concepts of the tools

4.1 Visualization tools

I started by writing a python script that would allow any user to generate a data sketch, of any dataset, from the command line. The challenges were multiple. First, we had to figure out what kind of information to include in the two pager sketch so that any reader could get a sense of the content of the dataset and even more, understand a bit the field. Second, the user experience was crucial since most of the targeted users for this sketch generation tool were not computer scientists and did not want to deal with lines of codes.

To address the first issue, several iterations with my supervisor were necessary and led to the following template:

1. The title of the dataset with its description (generally the name of the dataset was opaque)

2. A warning message automatically displayed in case the user did not have the codebook. The codebook is the description of each variable name, that are most of the time opaqe (for instance SIFTMM for Supraillac Skinfold Thickness (in mm))

3. A table summarizing the shape of the entire dataset (columns and rows) and the shape of the random subsample to run the analysis on. Indeed, since the analysis takes time, we decided to do the sketch on a subsample of 1000 rows and 20 columns.

4. The number of models (markov chains), iterations (of each chain) and the time of analysis

5. A dependence probability heatmap showing dependence probabilities from pairwise columns of the sample

6. A pairplot of 2 random variables

7. Three first entries for each of the variable to give a sense of the actual data

8. A table summarizing the type (numerical or categorical) of the random variables in the metamodel and the percentage of missing values.

The second issue, with respect to the user experience and the ease of use of the tool, was a bit tricky. Indeed, many choices have to be taken by whoever uses the script to generate it: codebook file to include, how many columns and rows to subsample, include the description of the dataset for external readers, the number of models one should create, the number of iterations, the type of each variable, etc.

To make it super simple and straight forward we decided to let less freedom to the user since the purpose of this tool is not to do a proper statistical study but to have a glance at a dataset and have some insights on a particular field. As a result we limited the number of models to 50 and iterations to 100. Enough for the markov chains to converge and quite reasonable to run on any personal computer. Also, we use the GUESS function of our metamodel that automatically detect categorical and numerical variable (and assigning them broad prior distributions) avoiding the user to select manually each distribution for each variable.

At the end, we came up with a script that can be run from the command line. You have to whether run:

```
python sketch.py file.csv file_col.csv
```

or

```
python sketch.py
```

to run it on all the datasets in your current folder.

Figure 8 showcases an example of a sketch:

Sample Sketch of BngR_Study: data.csv

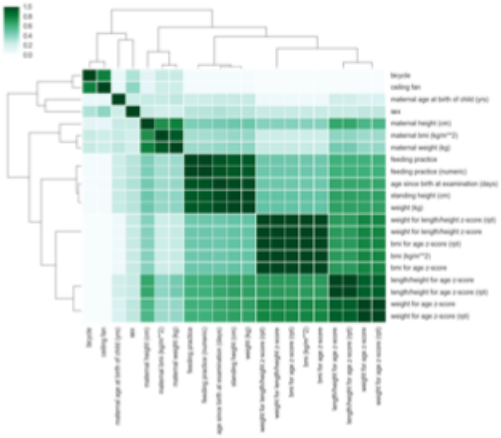
Cross-Sectional Dietary Survey in Children Aged 24-48 months in Bangladesh

Shape:

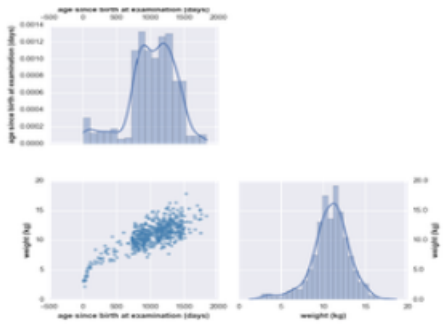
Object	Rows	Columns
Dataset	548	79
Sample	548	21

Initialization of **50** models with **100** iterations for **561** seconds

Sketchmap of dependancies



Pairplot



Data for 10 variables (rows) from 3 records (cols):

Record ID	0	1	2
age since birth at examination (days)	1055.0	1746.0	1245.0
weight (kg)	13.7	13.5	11.7
standing height (cm)	89.1	96.05	88.95
bmi (kg/m**2)	17.256994436	14.633190624	14.787472736
weight for age z-score	-0.25	-2.16	-2.12
length/height for age z-score	-1.67	-2.75	-2.61
weight for length/height z-score	0.96	-0.73	-0.97
bmi for age z-score	1.22	-0.46	-0.56
weight for age z-score (rpt)	-0.25	-2.16	-2.12
length/height for age z-score (rpt)	-1.67	-2.74	-2.61

Figure 4: Sample Sketch of BngR Study

The outcomes were very encouraging and the lab is currently negotiating with a visualization third party to explore new opportunities.

4.2 Growth & Development datasets

I've analyzed dozens of datasets from malnutrition to IQ. Here are an example of some of them:

1. CONTENT Study of Growth, Diarrhea and Socioeconomic Status
2. Randomized, community-based trial of the effect of zinc supplementation, with and without other micronutrients, on the duration of persistent childhood diarrhea in Lima
3. Cross-Sectional Dietary Survey in Children Aged 24-48 months in Bangladesh
4. Systems Dynamics Modeling of Growth in Children
5. Dataset of a 1959 cohort with data on IQ at age 1, 4 and 7. Plus some maternal parity and cigarettes use data

The work was basically to find quick and dirty dependencies that could make sense between variables with the minimum amount of expertise, obviously, and of human processing. Some outstanding results came out of an analysis ran on GUSTO (Growing Up in Singapore toward Healthy Outcomes) where simple analysis and plots enabled us to write a convincing stories about the dependencies between level of Zinc, Iron and Magnesium of the mother and some physical characteristics of the baby such as the head circumference, BMI for age at z-score, etc.

Finally, I finished my visit with a work on the satellites example the lab particularly cherish for all the options it gives to showcase our work. The datasets include hundreds of satellites and 23 metrics such as the name, perigee, apogee, period, lifetime, etc. Program managers from the Foundation were dubious about the ability of our metamodel, Crosscat, to cluster the rows and the columns into meaningful clusters. This is indeed a crucial question since the whole dependency matrix is based on this clustering (the estimate of the mutual information between two variables is the number of times they are in the same cluster).

The idea we had was to draw the final state of the markov chain (of one model only) and compare this to pairplots and simulated data.

I wrote a function to draw the state of any model and highlight the clustering of the variables. Figure 9 shows the raw data (none of the rows and columns are clustered). Red color is nan values and the shade of black are the values (dark is low).

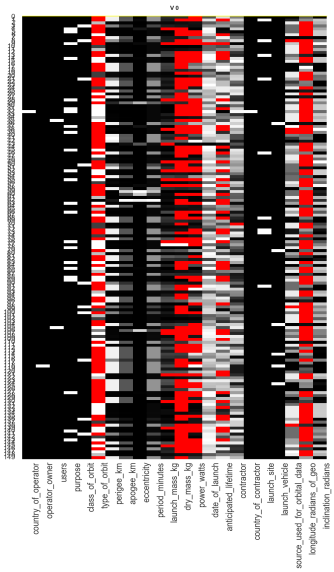


Figure 5: Raw Data

After analysis, the Figure 10 shows the actual clustering Crosscat made. The yellow lines are the row cluster assignments and each variables are organized into views.

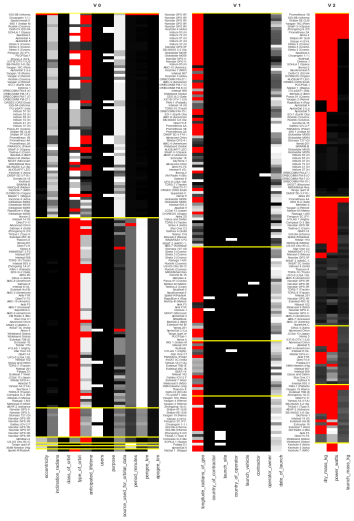
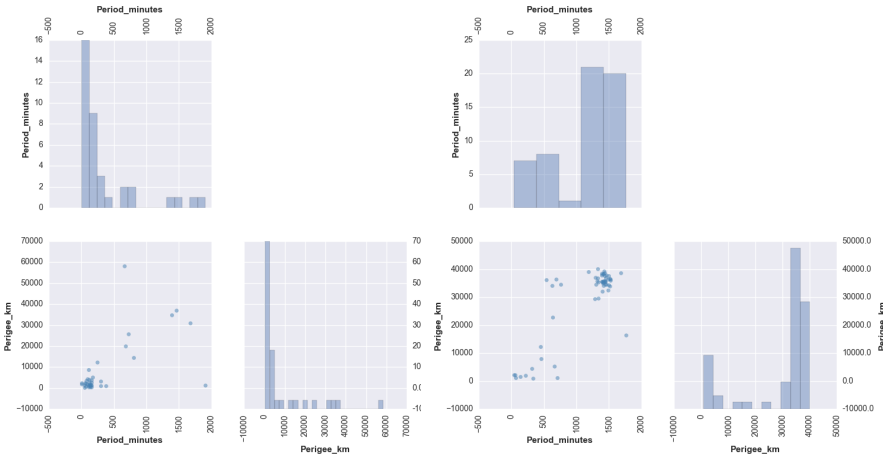


Figure 6: State view: clustering rows and columns in views

We can easily notice two satellites clusters defined by the Anticipated_Lifetime (whether low or high). We then plotted the joint distribution of the perigee and period conditioned on the anticipated lifetime,
$P(period\_minutes, pergigee\_km | Anticipated\_Lifetime = 3)$ and
$P(period\_minutes, pergigee\_km | Anticipated\_Lifetime = 15)$ to assess that the clustering makes sense. The pairplots shown on figure 11 are pretty straight forward.



(a) Cluster conditioned on low lifetime (b) Cluster conditioned on high lifetime

Figure 7: Pairplots showing the conditional clustering

These results were useful to convince the program manager during a call the week after. Some more efforts are being put to extend this to different metamodels and with simulated data.

5 Measuring accuracy of inference-based algorithm

This following section relates to the main research I've been doing during my visit. The next sections will deal with the applications I've pursued whether on high dimensional datasets or even my contribution to the general engineering effort.

Many of the most demanding applications of computing tolerate a spectrum of solutions to a given problem. Examples can be found in fields as diverse as numerical methods, lossy compression, microchip layout, robotics ([5]), computer graphics ([6]), and genetics ([1]). The additional flexibility in these applications makes programming more challenging: There are often several reasonable algorithms, each with its own tunable parameters and tradeoffs between com-

putational resource consumption and solution quality, and the introduction of approximate hardware platforms promises to contribute additional degrees of freedom. Reliable measurements of accuracy are a first prerequisite for guiding programmers in their exploration of the solution space, and for enabling automated accuracy-aware program optimization ([4]).

Reliable measurement of the accuracy of approximate inference is important for guiding inference programmers and automated systems in the exploration of the large space of conceivable solutions to a given inference problem. However, due to difficulty in analysis of simulation-based inference for finite computation, generally applicable techniques and tools for principled measurement of the most popular inference strategies on a common scale have been historically lacking. the following technique measures Markov chain Monte Carlo (MCMC)-based inference programs on a common accuracy scale based on Kullback-Leibler (KL) divergence. The vision is to increase the productivity of inference programmers, provide quantitative evaluation and comparison between inference strategies and allow meta-inference-programming.

First of all, it is important to define what we mean by inference programming.

5.1 Inference programming

In probabilistic programming (PP), the programmer writes a declarative specification of a probabilistic inference problem, which the PP platform solves or assists in solving, using built-in inference primitives. PP systems vary in the amount of flexibility the programmer has in selecting a solution technique. The Venture PP platform ([3]) offers the programmer sufficient flexibility in selecting custom solution techniques to warrant use of the term *inference programming* to describe the process of choosing a solution technique.

Let's first formalize the notion of probabilistic programming as used in this work. We denote probability distributions and random variables by capital letters (e.g. $P(X, Y)$), and we assume that all distributions are associated with associated discrete or continuous densities, which we denote by lower case letters. We denote joint densities $p_{X,Y}(x, y)$, marginal densities $p_X(x)$, and conditional densities $p_X(x|Y = y)$ by the shorthand $p(x, y)$, $p(x)$, and $p(x|y)$, respectively. We often refer to a distribution by its density.

1. A *model program* defines a joint probability distribution P over the latent (unobserved) variables X and observed variables Y, through a procedure for simulating latent values x according to the prior $p(x)$, followed by simulation of observed values y (also known as 'observations') given latents according to $p(y|x)$.

2. Together the model program and observations y imply a *target distribution* over the latent variables X, which we denoted by P_y. The target distribution has density $p(x|y)$ and un-normalized density $\tilde{p}_y(x) = p(x, y)$. The task of generating samples from a distribution over the latent variables that approximates P_y defines a *problem instance*.

3. An *inference program* is an executable procedure for constructing and sampling values for the latent values from its *output distribution* Q, which is intended to approximate the target distribution P_y. We distinguish between the output distribution and the inference program because distinct inference programs implementations can have the same output distribution. Generally, the output distribution Q depends on the observations y of the problem instance, but we omit this dependence in the notation. We denote the type of inference strategy used with superscript label and denote parameters of the inference program in the subscript. For example, Q_K^{mcmc} is an MCMC inference program with parameter K (to be discussed in detail later). We assume that an inference program is *primitive* when it simulates from and evaluates the density under the prior ($q^{prior}(x) = p(x)$).

MCMC inference programs are examples.

5.2 MCMC inference programs

Markov chain Monte Carlo (MCMC) is a class of inference programs based on stochastic simulation. The MCMC inference programs that we consider are defined by *MCMC kernels* K_y. In our formulation, an MCMC kernel K_y is map from the set of 'previous' latent states $X = x$ to distributions over a 'new' latent state X'. We denote the distribution induced by a kernel for a specific previous latent state $X = x$ by $K_y(X'; X = x)$ and its density by $k_y(x'; x)$. We require that the following condition ('detailed balance') holds:

$$k_y(x'; x)p(x'|y) = k_y(x; x')p(x|y) \quad \text{for all } x, x' \tag{1}$$

Note that each kernel is parameterized by the observations y, and that Equation 1 relates the kernel to the target distribution P_y associated with these observations. The Metropolis-Hastings algorithm provides a general and simple way of constructing kernels with this property.

MCMC inference programs sample an initial state from the prior $p(x)$, and 'apply' a kernel K_y by sampling x' according to $k_y(x'; x)$. The marginal distribution over x' is the output distribution $Q_{K_y}^{mcmc}$ of the inference program:

$$q_{K_y}^{mcmc}(x') = \int p(x)k_y(x'; x)dx$$

Typically, a primitive kernel K_y is repeatedly applied a large number of times M, resulting in a Markov chain over latent states $\left(x = x^{(0)}, x^{(1)}, \ldots, x^{(M-1)}, x^{(M)} = x'\right)$. We marginalize out the intermediate latent states and construct a composite kernel, denoted K_y^M, according to:

$$k_y^M(x^{(M)}; x^{(0)}) = \int \prod_{m=1}^{M} k_y(x^{(m)}; x^{(m-1)})dx^{(1)} \cdots dx^{(M-1)} \tag{2}$$

The composite kernel K_y^M also satisfies Equation 1. Since the notion of an MCMC kernel associated with observations y is closed under this composition operation, we allow K_y to refer to either primitive or composite kernels.

The purpose of applying an MCMC kernel K_y is to generate a new latent state whose marginal distribution is closer to the target distribution P_y than the marginal distribution of the previous latent state. In general, the KL divergence cannot be increased by the application of a kernel (a proof that KL divergence must decrease, in other words must improve, has been done before assuming this result) :

$$\mathrm{D}_{\mathrm{KL}}(Q^{mcmc}_{K_y^{M+1}}||P_y) \leq \mathrm{D}_{\mathrm{KL}}(Q^{mcmc}_{K_y^{M}}||P_y) \leq \mathrm{D}_{\mathrm{KL}}(Q^{prior}||P_y) \tag{3}$$

$$\mathrm{D}_{\mathrm{KL}}(Q^{mcmc}_{K_y^{M}}||P_y) > 0 \implies \mathrm{D}_{\mathrm{KL}}(Q^{mcmc}_{K_y^{M+1}}||P_y) < \mathrm{D}_{\mathrm{KL}}(Q^{mcmc}_{K_y^{M}}||P_y) \tag{4}$$

In general, as the number of repetitions of a primitive kernel in an MCMC inference program increases asymptotically, the output distribution converges to the target distribution. However, the rate of this convergence for finite M is difficult to assess. That's where I decided to focus.

5.3 Models and Problem Instances

We consider generative probabilistic models in which there are global latent random variables Θ, local latent random variables $Z_{1:T}$, and observed random variables $Y_{1:T}$, which can be simulated given the latents.

Next, a problem instance, is defined by a joint distribution $P(\Theta, Z_{1:T}, Y_{1:T}) = P(\Theta)P(Z_{1:T}|\Theta)\prod_{t=1}^{T} P(Y_t|\Theta, Z_t)$. A *ProblemInstance* is defined as a *Model* along with realized values for the observations, denoted $y_{1:T}$. A *ProblemInstance* may be created by applying *simulate* to a *Model*, or by manually combining a *Model* with compatible observations $y_{1:T}$. Each *ProblemInstance* encodes a task of approximate conditional sampling from a target distribution, which is the conditional distribution $P(\Theta, Z_{1:T}|Y_{1:T} = y_{1:T})$. The un-normalized density of the target distribution is denoted $\tilde{P}(\Theta, Z_{1:T}|y_{1:T}) = P(\Theta, Z_{1:T}, y_{1:T})$, with normalizing constant $P(y) = \int \tilde{P}(\Theta, Z_{1:T}|y_{1:T}) d\Theta dZ_{1:T}$.

5.4 Algorithms

An *inference algorithm*, indexed by $a \in A$, is a procedure that takes a problem instance as input and produces samples from a distribution Q_a as output, such that Q_a is an approximation to the target distribution $P(X|Y = y)$. We will study inference algorithms that make use of three types of primitive elements:

1. Sequences of MCMC transiton kernels $K = (K_1, \ldots, K_{T-1})$: Each K_t is a reversible MCMC transition operator whose stationary distribution is $P(\theta, z_{1:t}|y_{1:t})$.

2. Sequences of augmentation proposal distributions $Q^{aug} = (Q_1^{aug}, \ldots, Q_T^{aug})$: Each $Q_t^{aug}(z_t|\theta, z_{1:t-1}, y_{1:t})$ samples a value for a new local latent variable z_t conditioned on all previous latents, and all observations up to and including y_t.

3. Sets of variational distributions $\{Q_\phi^{var}(\theta, z_{1:T}) : \phi \in \Phi\}$, where Φ is a set of variational parameters.

Specifically we will condider the following class of inference algorithms:

5.4.1 Single-particle MCMC

A single-particle MCMC algorithm [**Andrieu**] is defined by a sequence of MCMC transition kernels, K_1 through K_T, and a sequence of augmentation proposals Q_1^{aug} through Q_T^{aug}. Each K_t is defined on the space $(\Theta, Z_1, \ldots, Z_t)$, and satisfies detailed balance for the corresponding target distribution:

$$\frac{K_t(\theta', z'_{1:t}|\theta, z_{1:t})}{K_t(\theta, z_{1:t}|\theta', z'_{1:t})} = \frac{P(\theta', z'_{1:t}|y_{1:t})}{P(\theta, z_{1:t}|y_{1:t})} \quad \text{for all } \theta, z_{1:t}, \theta', z'_{1:t}, 1 \leq t \leq T \tag{5}$$

The K_t do not need to be *assessable*, meaning their log-density does not need to be evaluated. The Q_t^{aug} are assessable distributions over z_{t+1}, given $z_{1:t}$ and $y_{1:t+1}$:

$$Q_t^{aug}(z_{t+1}|z_{1:t}, y_{1:t+1}) \quad \text{for all } 1 \leq t \leq T \tag{6}$$

The algorithm samples from a Markov chain over the state space $\mathbf{x} = \{\theta^{(t)}, z_{1:t+1}^{(t)}\}_{t=0}^{T-1}$. The algorithm initializes the Markov chain by sampling $\theta^{(0)}$ from the prior $P(\Theta)$, and sampling $z_1^{(0)}$ from $Q_1^{aug}(z_1|\theta^{(0)}, y_1)$. Then, the algorithm proceeds by alternating between sampling from the kernels K_t and sampling from the augmentation proposals Q_{t+1}^{aug}, for $1 \leq t \leq T-1$, producing the full distribution over the Markov chain:

$$\mathbf{Q}_a(\mathbf{x}) = P(\theta^{(0)})Q_1^{aug}(z_1^{(0)}|y_1, \theta^{(0)}) \prod_{t=1}^{T-1} K_t(\theta^{(t)}, z_{1:t}^{(t)}|\theta^{(t-1)}, z_{1:t}^{(t-1)})Q_{t+1}^{aug}(z_{t+1}^{(t)}|\theta^{(t)}, z_{1:t}^{(t)}) \tag{7}$$

The algorithm defines the output distribution Q_a as the marginal distribution of the final latent sample $\theta^{(T-1)}, z_{1:T}^{(T-1)}$ in the sequence:

$$Q_a(\theta, z_{1:T}) = \int \cdots \int \mathbf{Q}_a(\mathbf{x}) d\theta^{(0)} \cdots d\theta^{(T-2)} dz_{1:T}^{(0)} \cdots dz_{1:T}^{(T-2)} \tag{8}$$

The algorithm is reproduced below:

5.5 The data and the technique

As a working example, we will use a univariate finite Gaussian mixture model with fixed mixture proportions π, fixed data variance σ, fixed prior $\mathcal{N}(\mu_0, \sigma_0)$

Algorithm 1 Single-particle MCMC algorithm

Require: $K_1, \ldots, K_{T-1}$ are MCMC kernels and $Q_1^{aug}, \ldots, Q_T^{aug}$ are augmentation distributions compatible with problem instance defined by $\tilde{P}(\theta, z_{1:T}, y_{1:T})$

$w \leftarrow 1$

$\theta^{(0)} \sim P(\Theta)$ ▷ Sample θ from the prior

$z_1^{(0)} \sim Q_1^{aug}(z_1|\theta^{(0)}, y_1)$

for $t \leftarrow 1$ to $T-1$ **do**

  $(\theta^{(t)}, z_{1:t}^{(t)}) \sim K_t(\theta, z_{1:t}|\theta^{(t-1)}, z_{1:t}^{(t-1)})$ ▷ Run the MCMC kernel K_t

  $z_{t+1}^{(t)} \sim Q_{t+1}^{aug}(z_{t+1}|\theta^{(t)}, z_{1:t}^{(t)}, y_{1:t+1})$ ▷ Augment the state space

  $w \leftarrow w \cdot \frac{P(z_{t+1}^{(t)}, y_{t+1}|\theta^{(t)}, z_{1:t}^{(t)}, y_{1:t})}{Q_{t+1}^{aug}(z_{t+1}^{(t)}|\theta^{(t)}, z_{1:t}^{(t)}, y_{1:t+1})}$

end for

return $(\theta^{(T-1)}, z_{1:T}^{(T-1)}), w$ ▷ Return the sample $(\theta^{(T-1)}, z_{1:T}^{(T-1)}) \sim Q_a$ and the weight w

on the component means, and K mixture components. The generative model is:

$$\begin{aligned} \mu_k &\sim \mathcal{N}(\mu_0, \sigma_0) \quad k = 1, \ldots, K \\ z_t &\sim \text{Categorical}(\pi) \quad t = 1, \ldots, T \\ y_t &\sim \mathcal{N}(\mu_{z_t}, \sigma) \quad t = 1, \ldots, T \end{aligned} \tag{9}$$

The realized values of the global latents are $\theta = \{\mu_k\}_{k=1}^K$, the realized values of the local latents are $\{z_t\}_{t=1}^T$, and the realized values of the observations are $y = \{y_t\}_{t=1}^T$. Given values of y, defining a problem instance, we will apply various inference algorithms a to this problem. Each algorithm a will produce a sampling distribution Q_a over the values $\{\mu_k\}_{k=1}^K$ and $\{z_t\}_{t=1}^T$ that attempts to approximate the target distribution $P(\mu_1, \ldots, \mu_K, z_1, \ldots, z_T|y_1, \ldots, y_T)$. This work describes a technique for objectively measuring on a the same scale the accuracy on a given problem instance of algorithms a that span a wide set of different types of inference strategies. Specifically, the distributions Q_a produced by Markov chain Monte Carlo (MCMC) algorithms can be compared on the basis of the following quantity:

$$\mathcal{L}(Q_a) = \mathrm{E}_{x \sim Q_a}\left[\log \frac{\tilde{P}(x|y)}{Q_a(x)}\right] \tag{10}$$

This quantity is the Evidence Lower BOund (ELBO). $\mathcal{L}(Q_a)$ gains meaning through its relationship with the KL-divergence:

$$\mathrm{D}_{\mathrm{KL}}(Q_a(X)||P(X|y)) + \mathcal{L}(Q_a) = \log P(y) \tag{11}$$

Specifically, even if $\log P(y)$ is not known, the following relationship holds:

$$\mathcal{L}(Q_{a_1}) \geq \mathcal{L}(Q_{a_2}) \implies \mathrm{D}_{\mathrm{KL}}(Q_{a_1}(X)||P(X|y)) \leq \mathrm{D}_{\mathrm{KL}}(Q_{a_2}(X)||P(X|y)) \tag{12}$$

A lower bound on $\mathcal{L}(Q_a)$ for MCMC algorithm can be estimated: the log weight (see appendix for the proof), to produce estimates of upper bounds on the KL-divergence $\mathrm{D}_{\mathrm{KL}}(Q_a(X)||P(X|y))$ (as the equation (7) suggests it) for algorithms a.

5.6 Measures and Validation

According to [**Grosse2015**], single particle learning parameters can be described as follow: In what follows, there are T observations $y_1, \ldots, y_T$, local latents $z_1, \ldots, z_T$, and global latents θ, with joint distribution $P(\theta, z_{1:T}, y_{1:T}) = P(\theta) \prod_{i=1}^{T} p(z_i|\theta)p(y_i|z_i, \theta)$.

5.6.1 Pure MCMC algorithm

l — l

Extended state space $\mathbf{x} = (\theta, z_{1:T})^{(t)}$ for $t = 0 \ldots T-1$

MCMC Kernels $K_t(\theta^{(t)}, z_{1:T}^{(t)}|\theta^{(t-1)}, z_{1:T}^{(t-1)})$ target $P(\theta, z_{1:T}|y_{1:t})$ for $t = 1 \ldots T-1$ (reversible)

Extended sampling distribution $\mathbf{Q}(\mathbf{x}) = P(\theta^{(0)}, z_{1:T}^{(0)}) \prod_{t=1}^{T-1} K_t(\theta^{(t)}, z_{1:T}^{(t)}|\theta^{(t-1)}, z_{1:T}^{(t-1)})$

Extended target distribution $\mathbf{P}(\mathbf{x}) = P(\theta^{(T-1)}, z_{1:T}^{(T-1)}|y_{1:T}) \prod_{t=1}^{T-1} K_t(\theta^{(t-1)}, z_{1:T}^{(t-1)}|\theta^{(t)}, z_{1:T}^{(t)})$

Particle weight $w(\mathbf{x}) = \prod_{t=1}^{T-1} P(y_t|\theta^{(t-1)}, z_{1:T}^{(t-1)})$

Exact MCMC KL-bound $\mathrm{D}_{\mathrm{KL}}(\mathbf{Q}||\mathbf{P}) = \sum_{t=0}^{T-1} \mathrm{D}_{\mathrm{KL}}(P(\theta, z_{1:t+1}|y_{1:t})||P(\theta, z_{1:t+1}|y_{1:t+1}))$

5.6.2 Validation

In order to test the implementations of these algorithms, we compared the asymptotic bounds to the actual asymptotic expected log-weight produced by our implementations for several small problems (Independent Metropolis Hastings Random Walk Metropolis Hastings) in which the asymptotic bounds can be computed exactly. We tested our instance of the pure MCMC algorithm (Algorithm 5.6.1) in which the model has no local latents z_i.
10^5 independent runs for each distinct MCMC kernel and count of MCMC iterations were used to produce estimates $\widehat{\mathcal{L}_{\tilde{\mathbf{P}}}(\mathbf{Q})}$ of the expectation $E_{\mathbf{x}\sim Q}[\log w(\mathbf{x})] = \mathcal{L}_{\tilde{\mathbf{P}}}(\mathbf{Q})$. The true $\log Z$ was analytically computed, and the estimated KL-bounds were was computed as: $\log Z - \widehat{\mathcal{L}_{\tilde{\mathbf{P}}}(\mathbf{Q})}$

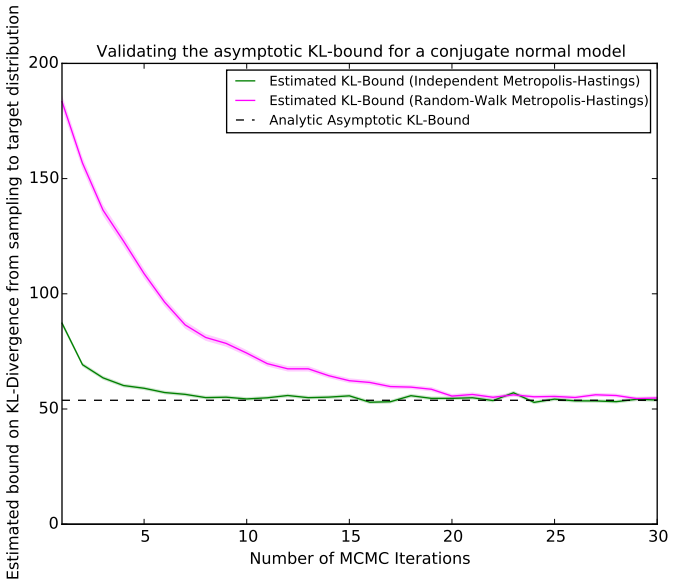


Figure 8: Validation of Algorithm 5.6.1 applied to a conjugate normal model ($\mu_0 = 0, \sigma_0 = 10, \sigma = 1$) using an independent Metropolis-Hastings (IMH) with proposal from the prior, and a random-walk Metropolis Hastings with a $\mathcal{N}(\mu, 2)$ proposal.

As you can see, the estimated bounds converged to the expected asymptotic values, validating both the use of the log weight (its expectation) and the correctness of implementation.

6 Software engineering

Finally, my contribution throughout the visit was also regarding the engineering efforts. Not only the research team, where I was in, had the responsibility to report bugs when they were facing them (even the non blocking ones) but also had to put efforts into suggesting ways of fixing them. And in general we were constantly searching for better features or improvements of existing ones. This side task allowed me to develop my skills with respect to working in a software environment and dealing with engineering teams. Either on communication and organization plan, I think it was very useful for me to contribute.

References

[1] Mark a Beaumont and Bruce Rannala. "The Bayesian revolution in genetics." In: *Nature reviews. Genetics* 5.4 (2004), pp. 251–261. ISSN: 1471-0056. DOI: 10.1038/nrg1318.

[2] Tenenbaum J.B and Mansighka VK. "A probabilistic model of cross-categorization. (English)". In: *Cognition* (2011).

[3] Vikash Mansinghka, Daniel Selsam, and Yura Perov. "Venture: a higher-order probabilistic programming platform with programmable inference". In: (2014), pp. 1–78. arXiv: arXiv:1404.0099v1.

[4] Stelios Sidiroglou et al. "Managing performance vs. accuracy trade-offs with loop perforation". In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (SIGSOFT/FSE)* (2011), pp. 124–134. DOI: 10.1145/2025113.2025133. URL: http://dl.acm.org/citation.cfm?doid=2025113.2025133.

[5] Sebastian Thrun et al. "Robust Monte Carlo localization for mobile robots". In: *Artificial Intelligence* 128.1-2 (2001), pp. 99–141. ISSN: 00043702. DOI: 10.1016/S0004-3702(01)00069-8.

[6] T.Y.a Yeh et al. "Fool me twice: Exploring and exploiting error tolerance in physics-based animation". In: *ACM Transactions on Graphics* 29 (2009). ISSN: 07300301. DOI: http://doi.acm.org/10.1145/1640443.1640448. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-77955884922%5C&partnerID=40%5C&md5=59f3f6a10fc59b1a54105d36c3de3ae1.

7 Appendix

7.1 Proof that log weight is unbiased estimator of lower bound on ELBO for single particle MCMC

First, consider the distribution of $x^{(0:T)} = (x^{(0)}, \ldots, x^{(T)})$ induced by a single run of Algorithm 0. which we will refer to as the *extended sampling distribution*, and denote by $\mathbf{Q}_K^{mcmc}(x^{(0)}, \ldots, x^{(T)})$, with density:

$$\mathbf{q}_K^{mcmc}(x^{(0:T)}) = p(x^{(0)}) \prod_{t=1}^{T} k_{y_{1:t}}(x^{(t)}; x^{(t-1)}) \tag{13}$$

First, note that by Equation 3 the application of the final kernel $K_{y_{1:T}}$ cannot increase the KL divergence:

$$\mathrm{D}_{\mathrm{KL}}(\mathbf{Q}^{mcmc}(X^{(T)})||P_{y_{1:T}}) \leq \mathrm{D}_{\mathrm{KL}}(\mathbf{Q}^{mcmc}(X^{(T-1)})||P_{y_{1:T}}) \tag{14}$$

$$\log p(y) - \mathcal{L}_{P_y}(\mathbf{Q}_K^{mcmc}(X^{(T)})) \leq \log p(y) - \mathcal{L}_{P_y}(\mathbf{Q}_K^{mcmc}(X^{(T-1)})) \tag{15}$$

$$\mathcal{L}_{P_y}(\mathbf{Q}_K^{mcmc}(X^{(T-1)})) \leq \mathcal{L}_{P_y}(\mathbf{Q}_K^{mcmc}(X^{(T)})) \tag{16}$$

Therefore, it suffices to find a lower bound on the ELBO for the marginal $Q^{mcmc}(X^{(T-1)})$. Following the derivation of [**Neal1998**], we construct the following *extended target distribution* on the space $x^{(0)}, \ldots, x^{(T-1)}$, with density:

$$\mathbf{p}_{y,K}^{mcmc}(x^{(0:T-1)}) = p(x^{(T-1)}|y_{1:T}) \prod_{t=1}^{T-1} k_{y_{1:t}}(x^{(t-1)}; x^{(t)}) \tag{17}$$

Conceptually, $\mathbf{P}_{y,K}^{mcmc}$ is an imaginary distribution that first samples $x^{(T-1)}$ from the target distribution P_y and proceeds sequentially, sampling from the kernels $K_{y_{1:t}}$ in reverse order for $t = T-1, \ldots, 1$. We define the un-normalized density of $\mathbf{P}_{y,K}^{mcmc}$ as:

$$\tilde{\mathbf{p}}_{K,y}^{mcmc}(x^{(0:T-1)}) = p(x^{(T-1)}, y_{1:T}) \prod_{t=1}^{T-1} k_{y_{1:t}}(x^{(t-1)}; x^{(t)}) \tag{18}$$

The normalizing constant of $\tilde{\mathbf{p}}_{y,K}^{mcmc}$ is then the same as the normalizing constant of the the un-normalized density of the original target distribution $p(x|y) = \frac{p(x,y)}{\int p(x,y)dx} = \frac{p(x,y)}{p(y)}$:

$$\int \tilde{\mathbf{p}}_{y,K}^{mcmc}(x^{(0:T-1)}) dx^{(0)} \cdots dx^{(T-1)} = \int p(x,y)dx = p(y) \tag{19}$$

The ratio of $\mathbf{p}_{y,K}^{mcmc}(x^{(0:T-1)})$ to $\mathbf{q}_{K}^{mcmc}(x^{(0:T-1)})$ is exactly the weight w returned by the algorithm:

$$\frac{\tilde{\mathbf{p}}_{y,K}^{mcmc}(x^{(0:T-1)})}{\mathbf{q}_{K}^{mcmc}(x^{(0:T-1)})} = \frac{p(x^{(T-1)})p(y_{1:T}|x^{(T-1)})}{p(x^{(0)})} \prod_{t=1}^{T-1} \frac{k_{y_{1:t}}(x^{(t-1)}; x^{(t)})}{k_{y_{1:t}}(x^{(t)}; x^{(t-1)})} \tag{20}$$

$$= \frac{1}{p(x^{(0)})} \left[\prod_{t=1}^{T-1} \frac{p(x^{(t-1)})p(y_{1:t}|x^{(t-1)})}{p(x^{(t)})p(y_{1:t}|x^{(t)})} \right] p(x^{(T-1)})p(y_{1:T}|x^{(T-1)}) \tag{21}$$

$$= \prod_{t=1}^{T-1} p(y_t|x^{(t-1)}) \tag{22}$$

$$= w \tag{23}$$

where we have used the fact that the kernels $K_{y_{1:t}}$ satisfy detailed balance (Equation 1) to simplify:

$$\frac{k_{y_{1:t}}(x^{(t-1)}; x^{(t)})}{k_{y_{1:t}}(x^{(t)}; x^{(t-1)})} = \frac{p(x^{(t-1)}|y_{1:t})}{p(x^{(t)}|y_{1:t})} = \frac{p(x^{(t-1)})p(y_{1:t}|x^{(t-1)})}{p(x^{(t)})p(y_{1:t}|x^{(t)})} \tag{24}$$

Consider the following identity, analogous to Equation **??**, but for the extended sampling and extended target distributions of the mcmc algorithm:

$$\mathrm{D}_{\mathrm{KL}}(\mathbf{Q}_{K}^{mcmc} || \mathbf{P}_{K}^{mcmc}) + \mathcal{L}_{\mathbf{p}_{y,K}^{mcmc}}(\mathbf{Q}_{K}^{mcmc}) = \log p(y) \tag{25}$$

Note that $\mathbf{Q}_{K}^{mcmc}(X^{(T-1)})$ and $P_y(X^{(T-1)})$ are the marginal distributions of $X^{(T-1)}$ for $\mathbf{Q}_{K}^{mcmc}$ and $\mathbf{P}_{y,K}^{mcmc}$, respectively. Therefore:

$$\mathrm{D}_{\mathrm{KL}}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)}) || P_y) \leq \mathrm{D}_{\mathrm{KL}}(\mathbf{Q}_{K}^{mcmc} || \mathbf{P}_{y,K}^{mcmc}) \tag{26}$$

Combining this with Equation 25 gives:

$$\begin{aligned}
\log p(y) - \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)}) &\leq \log p(y) - \mathcal{L}_{\mathbf{p}_{y,K}^{mcmc}}(\mathbf{Q}_{K}^{mcmc}) \\
\mathcal{L}_{\mathbf{p}_{y,K}^{mcmc}}(\mathbf{Q}_{K}^{mcmc}) &\leq \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)})) \\
\mathrm{E}_{x^{(0:T-1)} \sim \mathbf{Q}_{K}^{mcmc}} \left[\log \frac{\tilde{\mathbf{p}}_{y,K}^{mcmc}(x^{(0:T-1)})}{\mathbf{q}_{K}^{mcmc}(x^{(0:T-1)})} \right] &\leq \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)})) \\
\mathrm{E}\left[\log w\right] &\leq \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)}))
\end{aligned}$$

Combining this with Equation 16 gives the desired result:

$$\mathrm{E}\left[\log w\right] \leq \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T-1)})) \leq \mathcal{L}_{P_y}(\mathbf{Q}_{K}^{mcmc}(X^{(T)})) = \mathcal{L}_{P_y}(Q_{K}^{mcmc}) \tag{27}$$