STAT 547: Bayesian Workflow

Course Notes

Charles C. Margossian
University of British Columbia
Winter 2026
https://charlesm93.github.io/stat547/

DRAFT

Preface

These course notes constitute my lesson plan for a topics class on *Bayesian Workflow*. The course was primarily designed for graduate students in statistics at the masters and PhD level, although I was fortunate to also have students from other disciplines, including mathematics, computer science, and ecology. The course was 1/2 semester long (6 weeks) with two meetings per week, each 1.5 hours long. The course websites (`https://charlesm93.github.io/stat547/`) provides the reading and homework assignments.

Overview. Bayesian inference refers to the use of conditional probability to learn about unknown parameters given data and a probabilistic model. *Bayesian workflow* is the tangled process through which we iteratively build, fit, and criticize many models for the purpose of model development and data analysis. It is in the context of this comprehensive workflow that we can best understand how useful various statistical methods are to the data analyst. In this course, we will study the methods and algorithms that enable scientists to deploy Bayesian workflow within a probabilistic software, such as Stan. In addition to developing a foundation in Bayesian workflow, we will discuss shortcomings in existing methods and identify open research questions. The course will be grounded in examples from scientific applications.

Contents

1 Why Bayesian Workflow?

1.1 Review of Bayesian modeling and Bayesian inference

What is a Bayesian model ?
$\rightarrow$ A joint distribution over model parameters θ and observations y,

$$p(\theta, y) = \underbrace{p(\theta)}_{\text{prior}} \underbrace{p(y \mid \theta)}_{\text{likelihood}}. \tag{1}$$

The prior encodes knowledge (broadly defined) about the parameters before observing the data and the likelihood tells us how to simulate data given a fixed θ.

Bayesian inference reverse-engineers the data generating process: given observations y, what are plausible values of θ? In a Bayesian setting, all inferential conclusions follow from the *posterior*, obtained via Bayes' rule,

$$p(\theta \mid y) = \frac{p(\theta)p(y \mid \theta)}{p(y)} = \frac{p(\theta, y)}{\int p(\theta, y)\mathrm{d}\theta}. \tag{2}$$

More generally, for any quantity of interest $f(\theta)$, want to learn $p(f(\theta) \mid y)$ and we can also consider quantities which stochastically depend on θ.

Key benefits of Bayesian inference:

(i) The prior encodes expert knowledge and serves as a regularization tool.

(ii) The posterior provides principled ways to quantify uncertainty.

Question: Which properties of the posterior do we care about?

Examples of applications of Bayesian modeling:

- Epidemiology: "Estimation of SARS-CoV-2 mortality during the early stages of an epidemic: a modeling study in Hubei, China and six regions in Europe" [Hauser et al., 2020]
- Pharmacokinetics: "Bayesian aggregation of average data: An application in drug development" [Weber et al., 2018]
- Cosmology: "Listening to the noise: Blind Denoising with Gibbs Diffusion" [Heurtel-Depeiges et al., 2024]
- Deep Learning: "Auto-Encoding Variational Bayes" [Kingma and Welling, 2014]

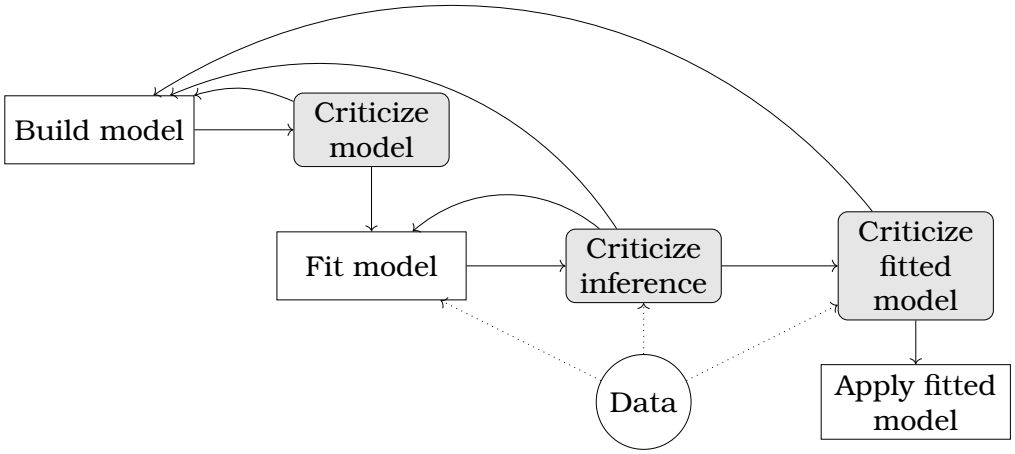

Figure 1: Model development as an iterative process. *Fitting the model is only one step in the broader workflow we use to analyze data. Figure borrowed from Grinsztajn et al. [2021].*

1.2 Going beyond: Bayesian workflow

Key questions:

(i) How do we build a model $p(\theta, y)$?

(ii) Given a model, how do we approximate $p(\theta \mid y)$?

(i) is *modeling* (scientific + statistical) and (ii) is *inference*.

Workflow is the iterative process of building, fitting and criticizing models. Typically an analysis involves a model development phase, understanding the limitations of that model, building an improved model, comparing it to other models... It's a tangled process!

Box's loop suggests an iterative process (Figure 1).

Example (SEIR model). The model we used in Hauser et al. [2020] is the 15^{th} "model iteration". Grinsztajn et al. [2021] discuss earlier models, why they failed and how to improve them. For example:

- Predictions are far-off $\rightarrow$ add a reporting rate parameter.
- Skewed predictions $\rightarrow$ add incubation period.
- ODE solver is too slow $\rightarrow$ adjust parameterization of ODE (then model runs in 2 hours instead of 3 days!)
- ...

What tools do we need to support workflow?

(i) An expressive probabilistic programming language that let's us do model composition.

(ii) Black box inference algorithms that can readily fit bespoke models without requiring extensive tuning effort from users.

(iii) Reliable checks to criticize the inference and the trained model.

1.3 Probabilistic programming languages

Examples: Stan, Pigeons.jl, PyMC, Turing, PyTorch, TensorFlow Probability

Highlighted languages have developers here at UBC!

In this course, we'll focus on Stan, although the concepts we learn apply more broadly.

The two primary components of Stan are:

(i) A flexible language to specify a model, i.e. $p(\theta, y)$.

(ii) Several black box gradient-based inference algorithms which return approximate samples from $p(\theta \mid y)$, e.g. Markov chain Monte Carlo, variational inference.

One thing that makes Stan easy to use is that, in general, the user only needs to specify their model. The gradients are calculated automatically via *automatic differentiation* and Stan provides several "off-the-shelf" inference algorithms. When Stan came out, its two main innovations were an extensive automatic differentiation library for probabilistic modeling and a self-tuning Hamiltonian Monte Carlo algorithm.

While Stan provides its own language for specifying a model, it needs to be interfaced with a scripting language (Figure 2).

1.4 Example: Influenza Outbreak

We're going to analyze data from a 1978 influenza outbreak at a British boarding school (Figure 3). Our goal is to understand the dynamics of the disease and, for example, predict the number of future cases, calculate the recovery time, and calculate R_0.

1.4.1 Scientific model

First we introduce a scientific model of the disease: the Susceptible-Infected-Recovered (SIR) model (Figure 4). This model treats the population as con-

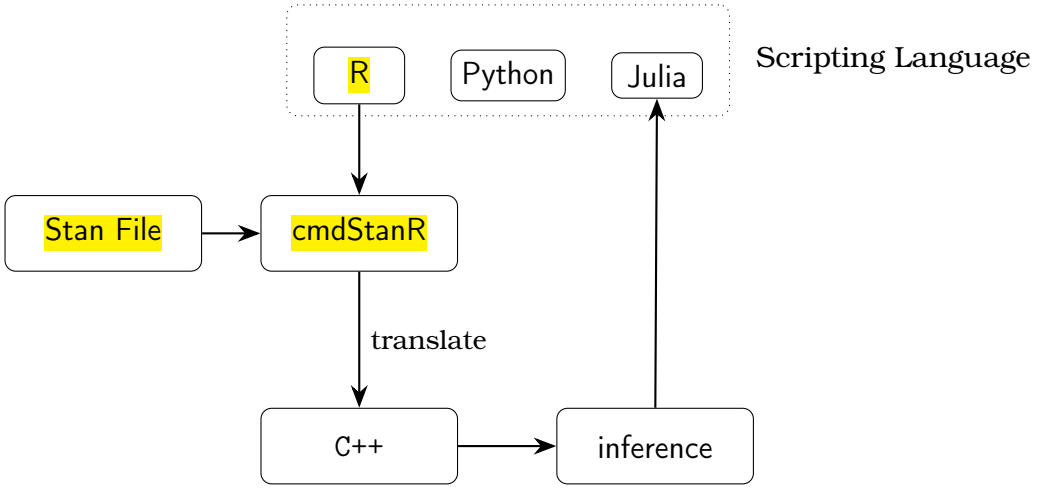

Figure 2: Code workflow when using Stan. *The Bayesian model is specified in a Stan file. Manipulating the data, calling Stan's inference, and analyzing the output of Stan's inference are all done in a scripting language. In this course, we'll use R with the package cmdStanR. Under the hood, the Stan file is translated into a C++ file, which is then compiled and executed when doing inference.*

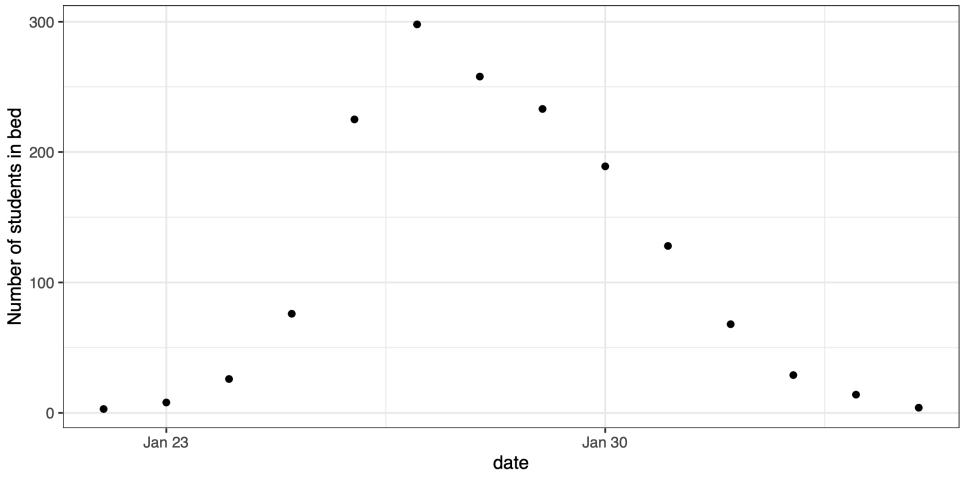

Figure 3: 1978 Influenza outbreak in a British boarding school

Figure 4: *Susceptible-Infected-Recovered model*

tinuous. Each "element" of the population is either susceptible, infected (and infectious) or recovered.

Let N be the total population and $S(t)$, $I(t)$, and $R(t)$ the amount of susceptible, infected and recovered elements at time t. Then the dynamic of the disease are modeled using a system of ordinary differential equations (ODEs):

$$S'(t) = -\beta \frac{S(t)I(t)}{N} \tag{3}$$

$$I'(t) = \beta \frac{S(t)I(t)}{N} - \gamma I(t) \tag{4}$$

$$R'(t) = \gamma I(t), \tag{5}$$

with:

- β: the transmission rate,
- γ: the recovery rate.

In addition, we have an initial condition $S(t_0) = N - 1$, $I(t_0) = 1$, $R(t_0) = 0$ where t_0 is the time at the beginning of the infection. We also have some derived quantities of interest:

- $T = 1/\gamma$: the recovery time
- $R_0 = \beta/\gamma$: how many individual does one person infect.

This scientific model does not tell us how the data is generated. Indeed, none of the quantities in it are observed—and they will need to be infered.

1.4.2 Observational model

Our observations are the number of students in bed $y(t))$ at time t, which is an indirect probe of the number of infected individuals.

Option 1. Poisson likelihood with rate $\lambda(t) = I(t)$. Then $\mathbb{E}y(t) = I(t)$ and $\text{Var}y(t) = I(t)$.

Option 2. Negative binomial with mean $\mu(t) = I(t)$ and overdispersion parameter ϕ. Here $\mathbb{E}y(t) = I(t)$ and $\text{Var}y(t) = I(t) + I^2(t)/\phi$.

Question: what other observational models might we consider?

1.4.3 Prior model

Under option 1, we only have two model parameters: $\beta \in \mathbb{R}^+$ and $\gamma \in \mathbb{R}^+$. We usually have some knowledge of what values these model parameters might take, based on an understanding of the underlying scientific phenomenon and/or other observations on the same disease or similar diseases. This knowledge can be detailed or it might be something like an order of magnitude (for example, we don't expect patients will take 100 years to recover from the disease).

Translating this understanding into a prior model is an active area of research. Let's look at some example:

- $p(\beta) = \text{normal}^+(2, 1) \implies \beta > 0$ and $P(\beta < 4) = 0.975$.
- $p(\gamma) = \text{normal}^+(0.4, 0.5) \implies \gamma > 0$ and $P(\gamma < 1) = 0.9$ (equivalently $P(T > 1) = 0.9$).

Can reason about the prior by seeing what they imply on other quantities of interest such as the recovery time T.

1.4.4 Writing and fitting the joint model in Stan

We now have all the ingredients to write a model in Stan: that is, we've specified a joint distribution over observations and model parameters, which defines a Bayesian model.

Before we start coding, let's pause briefly and ask some fundamental questions:

- Can we trust this model?

  When building our model, we made several choices which can be reasonably contested. For example, it is not correct to treat the number of infected individuals $I(t)$ as a continuous variable. As George Box said, "all models are wrong but some of them are useful." It falls upon us to identify the ways in which the model is useful and the ways in which the model is wrong. This is at the heart of workflow.

- Can we trust the prior?

  Building priors is often challenging and something practitioners struggle with. People often criticize Bayesian modeling on the basis that we can influence our inference by tweaking the prior. This is true; but it is just as true that we can influence our result by tweaking the likelihood.

It's good to be skeptical of the prior, in the sense that it's good to be skeptical of the model. On the other hand, being skeptical of the prior without questioning the likelihood is a double-standard. We'll study ways to check how wrong and how useful different components of the model are. But first, we need to talk about fitting the model.

Coding demo: Write the SIR model with Poisson likelihood.

Remark: Stan does not distinguish between prior and likelihood.

With cmdStanR, we can now:

(i) translate the model from Stan to C++ and compile it using `cmdstan_model()`.

(i) Run Stan's default inference engine with `$sample()`.

The output of Stan is approximate samples from the posterior, which are summarized in table.

variable	**mean**	**median**	**sd**	**mad**	**q5**	**q95**	**rhat**	**ess_bulk**
gamma	0.476	0.476	0.0110	0.0108	0.459	0.495	1.00	2362
beta	1.69	1.69	0.0149	0.0146	1.67	1.71	1.00	2794
R0	3.55	3.55	0.0786	0.0766	3.42	3.68	1.00	2962
T	2.10	2.10	0.0484	0.0474	2.02	2.18	1.00	2362

Let's focus on the first row for the parameter γ. The first six columns are statistical summaries of the marginal posterior $p(\gamma \mid y)$. The next tow columns tell us about the quality of the inference and whether it can be trusted. Our goal in the next section will be to understand how these diagnostics are computed and how to interpret them.

2 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a broad class of algorithms used in many fields, including Bayesian Statistics, Statistical Physics, Molecular Dynamics and more. It is often called the workhorse of Bayesian Inference and it is the default inference engine in Stan and other statistical software.

In this section, we review MCMC with the goal of understanding the algorithm's control parameters and how to check that it produces sufficiently accurate answers.

For excellent references which go into more details, I recommend:

- "General state space Markov chains and MCMC algorithms" [Roberts and Rosenthal, 2004]. Reading this paper requires familiarity with measure theory—but don't let that intimidate you!
- "Probabilistic Inference Using Markov Chain Monte Carlo Methods" [Neal, 1993].

2.1 Monte Carlo

Monte Carlo methods estimate properties of a target distribution π using samples.

Example (estimate of the mean of a function $f(z)$):

$$\hat{f}_N = \frac{1}{N}\sum_{n=1}^{N} f\left(z^{(n)}\right), \qquad z^{(1)}, z^{(2)}, \cdots z^{(N)} \overset{iid}{\sim} \pi. \tag{6}$$

Question: how good is this estimator?

Let's examine the expected squared error:

$$\mathbb{E}\left[(\hat{f}_N - \mathbb{E}f)^2\right] = \underbrace{\left(\mathbb{E}\hat{f}_N - \mathbb{E}f\right)^2}_{\text{squared bias}} + \underbrace{\text{Var}\hat{f}_N}_{\text{variance}}. \tag{7}$$

If we draw exact samples from $\pi(z)$, then $\hat{f}_N$ is unbiased,

$$\mathbb{E}\hat{f}_N = \mathbb{E}f, \tag{8}$$

and if the samples are independent, then the variance is

$$\text{Var}\hat{f}_N = \frac{1}{N}\text{Var}f. \tag{9}$$

In addition, we have:

(i) a strong law of large numbers,

$$P\left(\lim_{N\to\infty} \hat{f}_N = \mathbb{E}f\right) = 1. \tag{10}$$

(ii) a central limit theorem,

$$\frac{\sqrt{N}(\hat{f}_N - \mathbb{E}f)}{\sqrt{\text{Var}f}} \xrightarrow{d} \text{normal}(0,1). \tag{11}$$

Challenge: For many problems, we cannot draw exact samples from π. Instead we use MCMC to draw approximate samples from π. But MCMC produces samples which are neither independent nor identically distributed.

2.2 Asymptotic MCMC

It what follows, I'll consider distributions over $\mathcal{Z} \subseteq \mathbb{R}^d$ and I'll assume that these distributions admit a density in order to simplify a little bit the notation.

MCMC starts from an initial point z_0 and then applies a *transition kernel* from $z^{(i)}$ to $z^{(i+1)}$,

$$\Gamma\left(z^{(i)}, z^{(i+1)}\right) = p\left(z^{(i+1)} \mid z^{(i)}\right). \tag{12}$$

We denote P_n the distribution of the n^{th} element of the Markov chain, obtained by sequentially applying n times the transition kernel starting from z_0. We denote p_n the corresponding density. Similarly, we denote P_π the target distribution and π the corresponding density.

In general, we want to show that for any (measurable) set A and for a sufficiently large n,

$$P_n\left(z \in A\right) \approx P_\pi(z \in A). \tag{13}$$

Question: How should we define "$\approx$"?

We can start by checking convergence in distribution, that is

$$\lim_{n\to\infty} P_n\left(z \in A\right) = P_\pi(z \in A). \tag{14}$$

This can be shown for a broad class of transition kernels. The proof usually involves checking a property called *detailed balance* or *reversibility*.

Definition 1. *(reversibility) We say Γ is reversible with respect to π if*

$$\pi(z)\Gamma(z, z') = \pi(z')\Gamma(z', z). \tag{15}$$

Proposition 2. *If Γ is reversible with respect to π, then π is a stationary distribution of the Markov chain generated by Γ.*

Proof. We need to show that the distribution of a new state z' given a previous state $z \sim \pi$ is also π, that is

$$\int \Gamma(z, z')\pi(z)\mathrm{d}z = \pi(z'). \tag{16}$$

Applying the assumption of reversibility,

$$\int \Gamma(z, z')\pi(z)\mathrm{d}z = \int \Gamma(z', z)\pi(z')\mathrm{d}z = \pi(z') \int \Gamma(z', z)\mathrm{d}z = \pi(z'), \tag{17}$$

where in the last step, we recalled that $\Gamma(z', z) = p(z \mid z')$ is a probability density and must therefore integrate to 1.

□

Interpretation: Once the Markov chain "reaches" π, it stays there.

Remark: Reversibility is a sufficient condition for π to be a stationary distribution but not a necessary one. What is an example of an MCMC algorithm which is not reversible but has the right stationary distribution?

Example (Metropolis-Hastings).

Given z, generate a proposal z^* from a proposal distribution $q(z^* \mid z)$. Then accept this proposal with a probability,

$$\alpha(z, z^*) = \min\left[1, \frac{\pi(z^*)q(z \mid z^*)}{\pi(z)q(z^* \mid z)}\right]. \tag{18}$$

That is, for $u \sim \text{uniform}(0, 1)$,

$$z' = \begin{cases} z^*, \text{ if } u \leq \alpha(z, z^*) \\ z, \text{ otherwise.} \end{cases} \tag{19}$$

Exercise: Show that the stationary distribution of a Markov chain using the Metropolis-Hastings transition has stationary distribution π. (Hint: show that it is reversible with respect to π.)

Once we establish that our Markov chain has the right stationary distribution, we need to make sure that it asymptotically gets there. For this, we need to verify two conditions: (i) ϕ-irreducibility and (ii) aperiodicity. The exact definitions of these terms requires measure theory, which is a bit beyond this course. Instead, we'll provide some intuition:

- ϕ-**irreducibility:** From any z, the Markov chain will eventually "explore" the entire space and reach any measurable set.
- **Aperiodicity:** The chain will not oscillate in a regular pattern between different states.

  Example (Periodic Markov chain with the right stationary distribution). Consider a state space $\mathcal{Z} = \{1, 2, 3\}$ and let π be uniform over $\mathcal{Z}$. Consider a Markov chain with

  $$\Gamma(1, 2) = \Gamma(2, 3) = \Gamma(3, 1) = 1. \tag{20}$$

  Then π is stationary. (Why?)
  Intuitively, the chain is irreducible.
  However, if $z_0 = 1$, then $p\left(z^{(n)}\right) \neq \pi$ for any n.

Equipped with these notions, we now have sufficient conditions to build a Markov chain which asymptotically reaches π. Specifically, we have two results: (i) convergence in distribution, which is a statement about the distribution of $z^{(n)}$ as $n \to \infty$; and (ii) a law of large numbers, which is statement about the Monte Carlo estimator $\hat{f}_N$ itself. For completeness, I'm providing the formal measure-theoretical statement of this result.

Theorem 3. *If a Markov chain on a state space with countably generated σ-algebra is ϕ-irreducible, aperiodic, and has stationary distribution π, then for π-almost-everywhere $z \in \mathcal{Z}$ and for any measurable set A,*

$$\lim_{n\to\infty} P\left(z^{(n)} \in A\right) = P_\pi(z \in A). \tag{21}$$

Furthermore, for $f : \mathcal{Z} \to \mathbb{R}$ with $\mathbb{E}(|f|) < \infty$, we have a strong law of large numbers,

$$P\left(\lim_{N\to\infty} \hat{f}_N = \mathbb{E}f\right) = 1. \tag{22}$$

Question: How might you verify the conditions of Theorem 3 for the Metropolis-Hastings algorithm? What conditions on the proposal distribution q do we need?

2.3 Pre-asymptotic MCMC

Theorem 3 is a remarkable result: under fairly weak conditions, we can construct a Markov chain that has the right stationary distribution and will asymptotically get there. But this does not tell us how our Markov chain behaves over a finite number of iterations. To answer this question, we need to study two properties of the Markov chain:

(i) how quickly does P_n approach P_π? This, as we will see, is a statement about the *bias* of $\hat{f}_N$.

(ii) If the Markov chain is stationary, how quickly does the variance of $\hat{f}_N$ decrease?

2.3.1 How quickly does P_n approach P_π?

To tackle the question of convergence speed, it is common to bound a distance between P_n and P_π by a function of N, the number of iterations. There are many ways to compare distributions and none of them seem to be perfect. Which way to use remains, in my view, a somewhat open question (we'll revisit this topic when talking about variational inference). In the MCMC literature, it is common to bound the *total variation distance*.

Definition 4. *The total variation distance between two probability distributions P_{ν_1} and P_{ν_2} is:*

$$||P_{\nu_1} - P_{\nu_2}|| = sup_A |P_{\nu_1}(z \in A) - P_{\nu_2}(z \in A)|. \tag{23}$$

In words, if we consider all possible measurable sets A and pick the one that "maximizes" (supremizes?) the difference between P_{ν_1} and P_{ν_2}, how big is this difference?

The total variation distance can be written in a way that more clearly relates to our goal of constructing Monte Carlo estimators.

Proposition 5. *The total variation distance between two probability measures P_{ν_1} and P_{ν_2}, respectively with density ν_1 and ν_2, is, for $a < b$,*

$$||P_{\nu_1} - P_{\nu_2}|| = \frac{1}{b-a} sup_{f:\mathcal{Z}\to[a,b]} \left| \int f(z)\nu_1(z)dz - \int f(z)\nu_2(z)dz \right|. \tag{24}$$

Proof. Left as an exercise. □

In words, if we consider all bounded functions f, the total variation distance provides an upper-bound on how much the *expectation value* of f with respect to ν_1 and ν_2 can disagree. Applying this to the setting of MCMC, we obtain,

$$\frac{1}{b-a} \left| \mathbb{E}\hat{f}_N - \mathbb{E}f \right| \leq ||P_n - P_\pi||. \tag{25}$$

This is a bound on the bias of our Monte Carlo estimator but only for bounded functions.

Another useful property of the total variation distance is that it provides bounds on the bias of quantile estimates. For what follows, let's take $\mathcal{Z}$ to be $\mathbb{R}$ since quantiles are defined for univariate quantities. Recall that the α^{th} quantile of measure ν_1, $a_{\nu_1}(\alpha)$, is implicitly defined as

$$P_{\nu_1}(z \leq a_{\nu_1}(\alpha)) = \alpha. \tag{26}$$

Proposition 6. *Suppose $||\nu_1 - \nu_2|| = \varepsilon$ and assume that the cumulative distribution functions (CDFs) of ν_1 and ν_2 are strictly increasing. Then,*

$$a_{\nu_1}(\alpha - \varepsilon) \leq a_{\nu_2}(\alpha) \leq a_{\nu_1}(\alpha + \varepsilon). \tag{27}$$

Proof. Recall the CDF, $F_{\nu_1} : \mathcal{Z} \to [0, 1]$, is,

$$F_{\nu_1}(a) = P_{\nu_1}(z \leq a). \tag{28}$$

Since we assume that F_{ν_1} is strictly increasing, we have that F_{ν_1} is invertible and so for $\alpha = F_{\nu_1}(a)$, we have that $a = F^{-1}_{\nu_1}(\alpha)$.

Denote $A = (-\infty, a]$. Then, from the bound provided by the total variation distance,

$$|P_{\nu_1}(A) - P_{\nu_2}(A)| \leq \varepsilon, \tag{29}$$

or using a different notation,

$$|F_{\nu_1}(a) - F_{\nu_2}(a)| \leq \varepsilon. \tag{30}$$

By assumption, F_{ν_2} is strictly increasing and therefore invertible. Pick $a = a_{\nu_2}(\alpha) = F^{-1}_{\nu_2}(\alpha)$. Then, plugging this in,

$$|F_{\nu_1} \circ F^{-1}_{\nu_2}(\alpha) - F_{\nu_2} \circ F^{-1}_{\nu_2}(\alpha)| \leq \varepsilon \tag{31}$$

$$\iff \quad |F_{\nu_1} \circ F^{-1}_{\nu_2}(\alpha) - \alpha| \leq \varepsilon \tag{32}$$

Then,

$$\alpha - \epsilon \leq F_{\nu_1} \circ F^{-1}_{\nu_2}(\alpha) \leq \alpha + \epsilon. \tag{33}$$

Applying $F^{-1}_{\nu_1}$ on each side (and noting that $F^{-1}_{\nu_1}$ must also be strictly increasing),

$$F^{-1}_{\nu_1}(\alpha - \epsilon) \leq F^{-1}_{\nu_2}(\alpha) \leq F^{-1}_{\nu_1}(\alpha + \epsilon), \tag{34}$$

which is the wanted result.

□

Now that we have some motivation for bounding the total variation distance, we can try to understand how quickly this distance decreases. We'll start with an elementary property that states that applying a transition kernel must decrease the total variation distance.

Proposition 7. *If P_π is a stationary distribution for a Markov chain kernel, then for any $n \in \mathbb{N}$,*

$$||P_{n+1} - P_\pi|| \leq ||P_n - P_\pi||. \tag{35}$$

Proof. For any event A,

$$|P_{n+1}(z \in A) - P_\pi(z \in A)| = \left| \int P_1(z \in A \mid z')p_n(z')\mathrm{d}z' - \int P_1(z \in A \mid z')\pi(z')\mathrm{d}z' \right|, \tag{36}$$

where we used the fact that, by assumption, applying a transition kernel to $z' \sim \pi$ still generates a sample $z \sim \pi$.

Let $f(z') = P_1(z \in A \mid z')$ and notice f is a bounded function, specifically $f : \mathcal{Z} \to [0, 1]$. Then eq. (37) can be rewritten as,

$$|P_{n+1}(z \in A) - P_\pi(z \in A)| = \left| \int f(z')p_n(z')\mathrm{d}z' - \int f(z')\pi(z')\mathrm{d}z' \right| \leq ||P_n - P_\pi||, \tag{37}$$

where the inequality follows from Proposition 5 with $a = 0$ and $b = 1$. Taking the supremum with respect to A on both sides of eq. (37), we obtain the wanted inequality. □

With a more detailed analysis, we can derive a bound on the total variation distance of the form,

$$||P_n - P_\pi|| \leq b(z_0)h(N), \tag{38}$$

where by convention $h(0) = 1$ and $b(z_0)$ is the initial bias $|f(z_0) - \mathbb{E}f|$. Under stronger assumptions, it is possible to characterize $h(N)$. One regime of particular interest arises when h is a geometric function, $h(N) = \lambda^N$ for $\lambda \in (0, 1)$. This regime is called *geometric ergodicity*. In this regime:

(i) the bias of $f\left(z^{(n)}\right)$ decreases exponentially and so does the bias of $\hat{f}_N$ if we discard early samples.

Exercise: how quickly does the bias of $\hat{f}_N$ decrease if we don't discard early samples?

(ii) we have a central limit theorem for $\hat{f}_N$, which is the topic of the next section.

Proposition 8. *When the state space $\mathcal{Z}$ is finite, then any irreducible and aperiodic Markov chain with stationary distribution π is geometrically ergodic.*

Not all Markov chains on discrete spaces are irreducible and aperiodic and furthermore, not all irreducible and aperiodic Markov chains on continuous state spaces are geometrically ergodic. Ensuring geometric ergodicity over continuous spaces requires additional technical requirements.

In general, it is difficult to calculate h and even then, bounds on the total variation distance tend to be conservative, since they need to account for worst case scenarios, rather than focusing on the particular functions we might be interested in. This will motivate us to look at empirical measures of convergence which, while imperfect, can be deployed in practice.

2.3.2 Variance of MCMC at stationarity

A question of interest is how quickly does the variance of $\hat{f}_N$ decrease if we start from the stationary distribution? This idealized case approximates the behavior of Markov chains which have been warmed up for a sufficiently long time and which are *nearly* stationary. In this ideal setting, the Monte Carlo estimator is unbiased but we still need to handle its variance.

Here, a quantity of interest is the *effective sample size* (ESS), defined as follows,

$$\text{ESS} = \frac{N}{1 + 2\sum_{t=1}^{\infty} \rho(t)}, \tag{39}$$

where $\rho(t)$ is the autocorrelation between two samples separated by t steps. Notice that if $\rho(t) = 0$ for all t, ESS $= N$.

The ESS plays a key role in the MCMC central limit theorem.

Theorem 9. *Consider the Monte Carlo estimator $\hat{f}_N$ obtained from a geometrically ergodic MCMC algorithm. Then*

$$\frac{\sqrt{ESS}(\hat{f}_N - \mathbb{E}f)}{\sqrt{Var(f)}} \xrightarrow{d} normal(0, 1). \tag{40}$$

This central limit theorem is almost identical to the one we have for independent samples (eq. (11)), except that the rate of convergence is $\sqrt{\text{ESS}}$ rather than $\sqrt{N}$. Moreover, for a large enough sample, we have

$$\hat{f}_N \overset{\text{approx.}}{\sim} \text{normal}\left(\mathbb{E}f, \frac{\text{Var}f}{\text{ESS}}\right). \tag{41}$$

This suggests another interpretation of the ESS for stationary Markov chains,

$$\text{ESS} \approx \frac{\text{Var}f}{\text{Var}\hat{f}_N} \iff \text{Var}\hat{f}_N \approx \frac{\text{Var}f}{\text{ESS}}, \tag{42}$$

and one can once again check that for i.i.d samples, ESS $= N$. Eq. (42) drives home the point that for stationary Markov chains and for large N, the ESS monitors the variance of $\hat{f}_N$, scaled by the posterior variance.

Based on eq. (42), the ESS can further be interpreted as the number of independent samples we would require to achieve the same expected squared error as our Monte Carlo estimator.

2.4 Practical diagnostics for MCMC

Remember that our goal is to control the expected squared error of the Monte Carlo estimator $\hat{f}_N$. For this we resort to a *warmup* phase and discard early samples to reduce the bias. We then run a *sampling* phase to control the variance. By default, Stan runs 1000 warmup iterations and 1000 sampling iterations.[1]

The bias and variance hint at two sources of error:

- The bias is due to the nonstationary initialization.
- The variance is due to the randomness in the MCMC (and is increased by the fact that the samples tend to be autocorrelated).

A general strategy to check the influence of these two sources of error on $\hat{f}_N$ is to run multiple chains with distinct initializations and seeds, and make sure that they still produce results which are in "good agreement" with one another.

They are multiple ways to measure agreement. First, we can perform visual checks using trace plots and density plots (Figures 5 and 6).

Another way to check for agreement is to compute the sample variance of the Monte Carlo estimator generated by individual chains. Moving forward, we denote with a superscript the chain identity of each Monte Carlo estimator: for example, $\hat{f}_N^{(m)}$ is the Monte Carlo estimator generated by the m^{th} chain, and,

$$\bar{f}_N^{(\cdot)} = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_N^{(m)}, \tag{43}$$

is the Monte Carlo estimator obtained by averaging across all chains. The sample variance of the per chain Monte Carlo estimator is then

$$\widehat{B} := \frac{1}{M-1} \sum_{m=1}^{M} \left(\hat{f}_N^{(m)} - \bar{f}_N^{(\cdot)} \right)^2, \tag{44}$$

[1] We will see that the warmup phase in Stan is not only used to reduce the bias of $\hat{f}_N$ but also to tune the MCMC algorithm so that it performs better during the sampling phase.

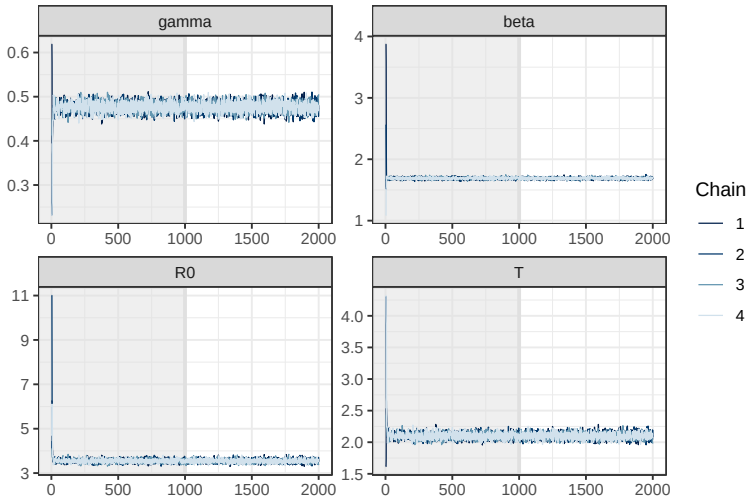

Figure 5: Trace plots for MCMC. *Value for each parameter across iterations. The shaded area corresponds to the warmup phase (first 1000 iterations). Here, we want to check that all the Markov chains have converged to the same "area" despite their distinct initialization and seed. For this model, it only takes a few iterations for all the chains to converge in distribution. The "fuzzy caterpillar" shape indicates that there is little autocorrelation between successive samples.*

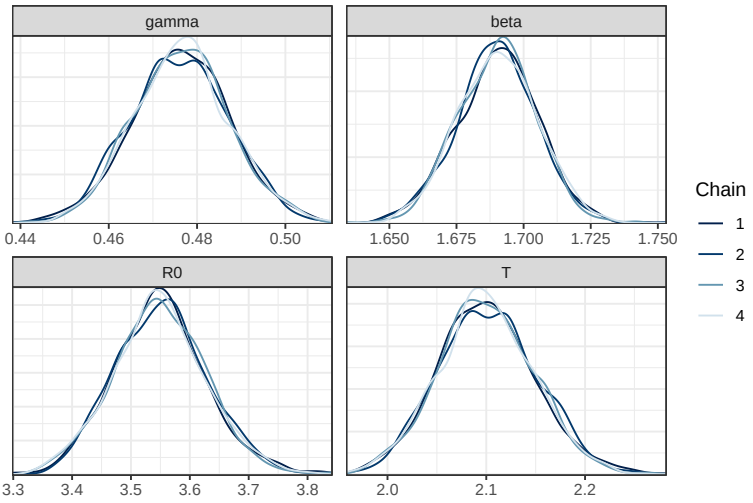

Figure 6: Density plots for MCMC. *Marginal density estimation using samples from the sampling phase. Here too we can check that the marginal densities returned by different chains are in good agreement.*

and we can check that $\widehat{B} \approx 0$ as a measure of how well the Markov chains agree with one another.

You might find this approach at best somewhat convincing. The sample variance lets us measure the variance but can it actually tell us something about the bias?[2] Intuitively, we expect that Markov chains which have not converged to π will generate Monte Carlo estimators with higher variance (why?).

We can formalize this intuition. Suppose we draw the Markov chain's initial point $z_0^{(k)}$ from an initial distribution P_0. Then, applying the law of total variance,

$$\mathrm{Var}\hat{f}_N^{(m)} = \underbrace{\mathrm{Var}\mathbb{E}\left(\hat{f}_N^{(m)} \mid z_0^{(m)}\right)}_{\text{nonstationary}} + \underbrace{\mathbb{E}\mathrm{Var}\left(\hat{f}_N^{(m)} \mid z_0^{(m)}\right)}_{\text{persistent}}. \tag{45}$$

The variance term decomposes into a *nonstationary* and a *persistent* variance:

- The nonstationary variance measures how much the expectation value of $\hat{f}_N^{(k)}$ varies with the initial point $z^{(k)}$ and vanishes as the Markov chain "forgets" its starting point. Hence, it is an indirect measure of the squared bias and how close to convergence the Markov chain is. **Formalizing this connection is a open research problem!**
- The persistent variance eventually dominates the total variance and for stationary Markov chains measures the asymptotic variance.[3]

For many problems, we care about the squared error of $\hat{f}_N$ relative to the posterior variance and so $\widehat{B}$ is scaled by a measure of the posterior variance. For each Markov chain, we compute a sample variance, and then average across all chains,

$$\widehat{W} := \frac{1}{M}\sum_{m=1}^{M}\frac{1}{N-1}\sum_{n=1}^{N}\left(f\left(z^{(nm)}\right) - \hat{f}_N^{(m)}\right)^2. \tag{46}$$

Then we examine the ratio $\widehat{B}/\widehat{W}$ and check that it is close enough to 0 (meaning the variance of the perchain Monte Carlo estimator is small relative to the estimated posterior variance).

For historical reasons, the quantity we typically measure is,

$$\widehat{R} = \sqrt{\frac{N-1}{N} + \frac{\widehat{B}}{\widehat{W}}}, \tag{47}$$

[2] This was a question I started working on during the final year of my PhD!

[3] One can also show that the stationary variance is inflated before the Markov chain convergences due to a "drift" phenomenon: that is, as long as the Markov chain is not stationary, we're likely averaging samples with a different mean and this increases the variance.

a quantity known as either the $\widehat{R}$ statistic, the *potential scale reduction factor* or the *Gelman-Rubin* statistic, and whose initial motivation is a bit different than what I'm presenting here. Nonetheless, $\widehat{R}$ is a 1-to-1 map with $\widehat{B}/\widehat{W}$.

A recent recommendation is to check that $\widehat{R} \leq 1.01$ [Vehtari et al., 2021]. This recommendation is battle-tested and seems to work well, **however finding a principled threshold for $\widehat{R}$ is an open research question.**

In addition to $\widehat{R}$, we also estimate the ESS as in eq. (39), based on estimates of the autocorrelation ρ_t [Geyer, 1992]. Now, from eq. (42), we have that for a stationary Markov chain,

$$\text{ESS} \approx \frac{\text{Var} f}{\text{Var} f_N} \approx \frac{\widehat{W}}{\widehat{B}}. \tag{48}$$

However, the above estimate tends to be less stable than the one obtained using autocorrelation functions, and so it is useful to compute the ESS, once we establish that the Markov chains have approximately converged to the stationary distribution with $\widehat{R}$.

Let's return now to the table of output from fitting the SIR model.

variable	**mean**	**median**	**sd**	**mad**	**q5**	**q95**	**rhat**	**ess_bulk**
gamma	0.476	0.476	0.0110	0.0108	0.459	0.495	1.00	2362
beta	1.69	1.69	0.0149	0.0146	1.67	1.71	1.00	2794
R0	3.55	3.55	0.0786	0.0766	3.42	3.68	1.00	2962
T	2.10	2.10	0.0484	0.0474	2.02	2.18	1.00	2362

For all quantities of interest, we achieve $\widehat{R} \leq 1.01$ which suggests good convergence and we have ESS> 2000 (labeled ess_bulk).

A somewhat **open research question** is to determine what is a useful ESS and more generally an acceptable expected squared error. While this of course depends on the problem at hand, in my experience, field practitioners do not themselves quite know how precise they need their Monte Carlo estimators to be. But this is crucial to determine how long the Markov chains need to be!! See Margossian and Gelman [2024] for further discussion.

For an example where the Markov chains do not converge and disagree, and where $\widehat{R} \gg 1$, see Chapter 29 of *Bayesian Workflow*.

Open discussion: Are Stan's default control parameters (4 Markov chains with a warmup phase of 1000 iterations and a sampling phase of 1000 iterations) optimal for doing Bayesian inference on the SIR model?

3 Prediction and model comparison

Even if we trust the inference, we need to check whether the fitted model "works."

Question: what does it mean for a fitted model to work?

- Accurate predictions
- Accurate recommendations, decisions, ...
- Accurate inference on an unknown quantity of interest.

Remark: In ML, it is common to check a trained model without checking the inference. Is this a good practice?

Most applications of a model imply we are making decent predictions. So let's at least check that! Let $y_{1:N}$ be our data and θ our model parameters. For simplicity, let's assume that, conditional on θ, the observations are independent, that is

$$p(y_{1:N} \mid \theta) = \prod_{n=1}^{N} p(y_n \mid \theta). \tag{49}$$

Let's suppose we want to make a prediction on a new observation, y_{N+1}. There are several ways to make this prediction:

- **Point prediction.** For a fixed $\hat{\theta}$, compute an estimator of $\hat{y}_{N+1} = \mathbb{E}(\hat{y}_{N+1} \mid \hat{\theta})$.
- **Distribution prediction.** Instead of doing a point prediction, we may consider a distribution of predictions, using the likelihood, $p(\hat{y}_{N+1} \mid \hat{\theta})$.
- **Posterior distribution of predictions.** In a Bayesian setting, we want to examine the posterior distribution of the predictions,

$$\begin{aligned} p(y_{N+1} \mid y_{1:N}) &= \int p(y_{N+1}, \theta \mid y_{1:N}) \mathrm{d}\theta \\ &= \int p(y_{N+1} \mid \theta, y_{1:N}) p(\theta \mid y_{1:N}) \mathrm{d}\theta \\ &= \int p(y_{N+1} \mid \theta) p(\theta \mid y_{1:N}) \mathrm{d}\theta, \end{aligned} \tag{50}$$

  where we used conditional independence to get the last line.

A natural way to study this posterior distribution is by drawing samples from $p(y_{N+1} \mid y_{1:N})$. This is easy to do once we have (approximate) samples from $p(\theta \mid y_{1:N})$. In particular, we can use a two step process:

(i) $\theta^{(i)} \sim p(\theta \mid y_{1:N})$

(ii) $y_{N+1}^{(i)} \sim p(y_{N+1} \mid \theta^{(i)})$.

In STAN, we already know how to do (i). It remains to do (ii). This can be done in post-process or using the `generated quantities` block.

3.1 Posterior predictive check

(Sometimes aptly called *posterior retrodictive checks*.[4])

As a first step, we're going to replicate the training data. That is, we're going to sample $\tilde{y}_{1:N}$ and see how closely the model's simulation match the observations we used to train the model.

Coding demo. Posterior predictions for the SIR model with a Poisson likelihood.

Informally, we want to check that the simulated data captures all the meaningful characteristics in the data. One way to do this is to plot the observations along with the simulations. In our disease transmission example, we plot the posterior median and 90 % posterior interval of $p(\tilde{y}_{1:N} \mid \theta)$.

Some questions to ask:

- Does the posterior median capture the central tendency of the observations?
- Is the uncertainty well-calibrated? For example, do 90 % of the observations fall within the 90 % posterior interval? Is the model too confident? Not confident enough?
- Are there some problematic outliers that suggest our model is missing some important mechanisms in the data generating process?

To illustrate how insightful these simple visual checks can be, we're going to fit another model to the same influenza data set. Namely an SIR model with a negative Binomial likelihood.

How do the two posterior predictive checks compare? Does a model better capture the characteristics of the data?

3.2 Out-of-sample predictions

Reconstructing the observations used to train the model provides an important sanity check but it only tells us so much about a model's ability to make predictions.

[4] `https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html`

A common practice is to split the data into a *training* and a *validation* set and see how well the model predicts data it has not "seen".

Question: How should we split the data into a training and a validation set?

3.3 Prediction score

Visual checks are not always convenient, let alone feasible. Usually we also report a quantitative measure of how good the prediction is.

Example 1. (Continuous observation) Suppose we have a normal likelihood, with estimated mean $\hat{\mu}_n$ and standard deviation $\hat{\sigma}_n$, indexed by n. Our best prediction is

$$\hat{y}_n = \hat{\mu}_n. \tag{51}$$

Then, we might report the squared prediction error,

$$\text{Err} = (y_n - \hat{\mu}_n)^2 . \tag{52}$$

This is a valid measure for the quality of our prediction but it doesn't tell us anything about $\hat{\sigma}_n$ or how confident our model is in its prediction.

Instead, we can use the *point-estimate log predictive density*,

$$\begin{aligned} \text{p-lpd} &= \log p(y_n \mid \hat{\mu}_n, \hat{\sigma}_n^2) && (53)\\ &= k - \log \hat{\sigma}_n - \frac{1}{2\hat{\sigma}_n^2} (y_n - \hat{\mu}_n)^2 . && (54) \end{aligned}$$

Example 2. (Binary observation) Suppose we have a Bernoulli likelihood with estimated parameter $\hat{\pi}_n$. Our best prediction is

$$\hat{y}_n = \mathbb{I}(\hat{\pi}_n \geq 0.5). \tag{55}$$

The the prediction error is

$$\text{Err} = \mathbb{I}(\hat{y}_n = y_n). \tag{56}$$

Again, this doesn't really tell us whether the model is too confident or not. Here, the *point-estimate log predictive density* is

$$\begin{aligned} \text{p-lpd} &= \log p(y_n \mid \hat{\pi}_n) && (57)\\ &= y_n \log \hat{\pi}_n + (1 - y_n) \log(1 - \hat{\pi}_n). && (58) \end{aligned}$$

This is the cross-entropy loss function (derived from a probabilistic perspective)!

In a Bayesian setting, we consider the point-estimate log predictive density averaged over the posterior distribution. This leads to the *expected log predictive density*,

$$\text{elpd} = \log p(y_{N+1} \mid y_{1:N}) = \log \int p(y_{N+1} \mid \theta) p(\theta \mid y_{1:N}) \mathrm{d}\theta. \tag{59}$$

Question: Equipped with this notion, how should we compare our two disease transmission models?

We need to agree on which test set to use. A common choice is to do cross-validation, where we average over multiple test-sets. In *leave-one-out* cross-validation, we only exclude one observation and train the model over all the other observations. Then we compute the elpd at each observation and take the average,

$$\text{elp}_{\text{loo}} = \frac{1}{N} \sum_{n=1}^{N} \log p(y_n \mid y_{-n}). \tag{60}$$

But this requires training the model N times, which is expensive...

Idea: We're going to find a fast approximation of $p(\theta \mid y_{-n})$, based on $p(\theta \mid y_{1:N})$.

3.4 Importance sampling estimator

Importance sampling (IS) provides a general framework to do Monte Carlo estimation when we have samples from a proposal distribution $q(z)$, rather than our target distribution $p(z)$.

The main relation we use is,

$$\int f(z) p(z) \mathrm{d}z = \int f(z) \frac{p(z)}{q(z)} q(z) \mathrm{d}z. \tag{61}$$

We denote $w(z) = p(z)/q(z)$ the *importance weights*. With draws from q, we can in principle construct an unbiased Monte Carlo estimator of $\mathbb{E}_p f$,

$$\hat{f}_{IS} = \frac{1}{S} \sum_{s=1}^{S} f\left(z^{(s)}\right) w\left(z^{(s)}\right). \tag{62}$$

The main challenge with importance sampling is that the importance weight may have an infinite variance, if they are regions where $q(z)$ is small and $p(z)$ is large.

When computing the elpd at y_n , $p(\theta \mid y_{1:N})$ is our proposal distribution and $p(\theta \mid y_{-n})$ our target distribution. Then, applying Bayes' rule and using the

conditional independence of the likelihood,

$$\begin{aligned} w(\theta) &= \frac{p(\theta \mid y_{-n})}{p(\theta \mid y_{1:N})} \\ &\propto \frac{p(\theta)p(y_{-n} \mid \theta)}{p(\theta) \mid p(y_{1:N} \mid \theta)} \\ &= \frac{\prod_{j \neq n} p(y_j \mid \theta)}{\prod_{j=1}^{N} p(y_j \mid \theta)} \\ &= \frac{1}{p(y_n \mid \theta)} =: r_n(\theta). \end{aligned} \tag{63}$$

To be explicit, we denote c the missing constant, so that $w(\theta) = cr_n(\theta)$. Then, we have the following importance sampling estimator of $\log p(y_n \mid y_{-n})$,

$$\widehat{\text{elpd}}_n = \frac{1}{S} \sum_{s=1}^{S} cr_n\left(\theta^{(s)}\right) p\left(y_n \mid \theta^{(s)}\right). \tag{64}$$

We need to find a way to deal with the unknown constant c. To do so, we consider the Monte Carlo estimator,

$$\frac{1}{S} \sum_{s=1}^{S} cr\left(\theta^{(s)}\right) = \frac{1}{S} \sum_{s=1}^{S} w\left(\theta^{(s)}\right) \longrightarrow \int \frac{p(\theta \mid y_{-n})}{p(\theta \mid y_{1:N})} p(\theta \mid y_{1:N}) \mathrm{d}\theta = \int p(\theta \mid y_{-n}) \mathrm{d}\theta = 1. \tag{65}$$

This suggests a consistent estimator, sometimes called the *self-normalized IS estimator*,

$$\begin{aligned} \widehat{\text{elpd}}_n &= \frac{1}{S} \frac{\sum_{s=1}^{S} cr_n\left(\theta^{(s)}\right) p\left(y_n \mid \theta^{(s)}\right)}{\frac{1}{S}\sum_{s=1}^{S} cr_n\left(\theta^{(s)}\right)} \\ &= \frac{\sum_{s=1}^{S} r_n\left(\theta^{(s)}\right) p\left(y_n \mid \theta^{(s)}\right)}{\sum_{s=1}^{S} r_n\left(\theta^{(s)}\right)} \\ &= \frac{S}{\sum_{s=1}^{S} \frac{1}{p(\theta^{(s)})}}. \end{aligned} \tag{66}$$

where, bearing an abuse of notation, we denote $\widehat{\text{elpd}}$ the above estimator and where in the final line we used the fact that $r_n = 1/p(y_n \mid \theta^{(s)})$ to simplify the nominator. Crucially, this estimator does not depend on c and can be computed, provided we can evaluate the likelihood $p(y_n \mid \theta)$.

3.5 Trunctated and Pareto-smooth IS estimator

Unfortunately, IS estimators as in eq. (66) can suffer from a large and even unbounded variance. We'll briefly review some strategies to control the variance of IS estimators.

One approach is to bound the ratios via a truncation rule,

$$\tilde{r}_n = \min(r_n, \sqrt{S}\bar{r}). \tag{67}$$

It can be shown that, provided r_n has a finite first moment ($\mathbb{E}(|r_n|) < \infty$), plugging $\tilde{r}_n$ into eq. (66) bounds the variance of $\widehat{\text{elpd}}_n$ but introduces a bias that can be large.

A better strategy is to use Pareto-smoothing. This topic is a bit of a rabbit hole and here we'll only cover it at a high level. The main idea is to fit the tail distribution of r_n,

$$p(r_n \mid r_n > u), \tag{68}$$

for some threshold u. In practice, this distribution is often well approximated by a generalized Pareto distribution,

$$p(y \mid \mu, \sigma, k) = \begin{cases} \frac{1}{\sigma}\left(1 + k\left(\frac{y-\mu}{\sigma}\right)\right)^{-\frac{1}{k}-1} & , k \neq 0, \\ \frac{1}{\sigma}\exp\left(\frac{y-\mu}{\sigma}\right) & , k = 0. \end{cases} \tag{69}$$

We fit the distribution to the tails of the IS weights using estimators $\hat{k}$, $\hat{\mu}$ and $\hat{\sigma}$. A key characteristic of the Pareto distribution is that it has $1/k$ finite moments. That is,

$$\mathbb{E}|x^p| \begin{cases} < \infty & , \text{ if } p \leq 1/k \\ = \infty & , \text{ if } p > 1/k. \end{cases} \tag{70}$$

Hence, the $\hat{k}$ estimator can tell us whether the tail weights have a finite variance or not. Specifically, the variance is finite if $\hat{k} \leq 0.5$. Sometimes, we can get away with $\hat{k} > 0.5$, because even if the tail weights have an unbounded variance, the $\widehat{\text{elpd}}_n$ can still have a finite variance.[5]

One heuristic is to check that $\hat{k} < 0.7$ for all $\widehat{\text{elpd}}_n$. If for a particular n, $\hat{k} > 0.7$ we may need to refit the model with the n^{th} data point excluded or come up with a different estimation strategy.

The Pareto smoothed Importance Sampling (PSIS) estimator can be computed using the R package `loo`.

Coding demo: Use loo-cv to compare the SIR model with a Poisson likelihood and a negative binomial likelihood.

For a more detailed discussion on the topic, see the excellent paper by Vehtari et al. [2017].

[5] For this, we can invoke generalizations of the central limit theorem, with subefficient convergence rates.

4 Hamiltonian Monte Carlo

We've discussed MCMC in a fairly general setting. In this section, we examine a specific subclass of MCMC, called Hamiltonian Monte Carlo (HMC).

HMC has completely modernized MCMC in the 2010's and it arguably remains the most popular "off-the-shelf" inference algorithm for Bayesian analysis—although it certainly does not solve every problem, nor is it the only good candidate you should consider.

Common HMC algorithms tend to yield good results for a broad range of high-dimensional targets with a reasonable geometry, i.e. targets with finite curvature and a single mode (or sometimes multiple modes that are not too disconnected). Certain classes of HMC can handle non-finite curvature and, in general, algorithms designed for more intricate geometries can leverage HMC, for example to do location exploration within a mode.

One motivation for using HMC is that it often scales better in dimension than random-walk Metropolis algorithms, a fact that is often observed in practice. There is also a formal argument for this. Consider a d-dimensional target distribution. Then, under somewhat idealized conditions, the computational cost of HMC scales as $\mathcal{O}(d^{5/4})$, rather than the $\mathcal{O}(d^2)$ cost characteristic of random-walk algorithms.[6]

Interestingly, HMC itself is fairly old: it was introduced in a 1987 paper on quantum chromodynamics and its original name was *hybrid Monte Carlo* [Duane et al., 1987]. Adoption of the technique in the applied statistics community was slow because:

(i) the algorithm requires gradient calculations,

(ii) the algorithm is difficult to tune.

The method had some success in the 1990's and 2000's, thanks to Radford Neal's pioneering work on Bayesian neural networks. Around 2012, the creators of STAN popularized the method by creating a probabilistic language with automated gradient calculation[7] and a self-tuning HMC algorithm, called the

[6] Consider a d-dimensional target distribution p and suppose this target factorizes, with each factor equal, $p(z) = \prod_{i=1}^{d} p(z_i)$. (Naturally, this scenario is simpler than what we encounter in practice, although it can be generalized a bit.) Assume the Markov chain is already stationary. Then in this setting, the computational cost of increasing the ESS by 1 with a random-walk Metropolis algorithm is $\mathcal{O}(d^2)$, but only $\mathcal{O}(d^{5/4})$ for HMC. For further discussion, see Neal [2011] and references therein.

[7] The efficient calculation of gradients is done using *automatic differentiation*, a broad class of techniques to calculate derivatives of functions specified as computer programs. The *reverse-mode* of automatic differentiation is known as backward propagation and underlies much of machine learning. See e.g., Baydin et al. [2018], Margossian [2019] for an introduction on the topic.

No-U-Turn sampler (NUTS, Hoffman and Gelman [2014]).

4.1 Ideal HMC

Suppose we want to construct an MCMC algorithm to sample from a target density $\pi(z)$ defined over $\mathbb{R}^d$. We'll assume that π is differentiable.

A standard[8] HMC transition kernel proceeds as follows:

1. Start at the Markov chain's current position $z_0 = z^{(i)}$.
2. Draw an auxiliary "momentum" variable, from a normal with covariance matrix M,
$$\rho_0 \sim \text{Normal}(0, M). \tag{71}$$
3. Simulate a trajectory over time T by solving Hamilton's equations of motions:
$$\frac{\mathrm{d}}{\mathrm{d}t} z_t = -\nabla_\rho \log \pi(\rho_t) = M^{-1}\rho_t \tag{72}$$
$$\frac{\mathrm{d}}{\mathrm{d}t} \rho_t = \nabla_z \log \pi(z_t), \tag{73}$$
with initial conditions at time $t_0 = 0$ given by (z_0, ρ_0).
4. Update the sate of the Markov chain, $z^{(i+1)} = z_t$.

Showing *why* this algorithm is effective takes a little bit of work.

Intuition. Hamilton's equations can be interpreted as describing the movement of a particle over time, with the position of the particle given by z_t and its momentum by ρ_t. The particle is subject to a potential, determined by $-\log \pi(z)$, and this potential determines how the particle accelerates.

To simplify things, take $M = I$ (the identity matrix). Then, the change in position over time, $\partial z_t / \partial t$—or "velocity"—is given by the momentum ρ_t (eq. (72)). It turn, the change in momentum, $\partial \rho_t / \partial t$—or "acceleration"—is given by $\nabla_z \log \pi(z)$(eq. (73)). That is, the particle accelerates when it moves in a direction with a positive gradient and it decelerates when the gradient is negative.

Tuning. A critical question is how large T should be: that is, for how long do we simulate a Hamiltonian trajectory before we resample the momentum and start the next iteration of MCMC?

- If T is too small, then at each iteration, the Markov chain doesn't travel far in the parameter space and the MCMC samples are strongly autocorrelated.

[8]The algorithm can be specified in a more general way but here we'll focus on the standard implementation.

- But increasing T beyond a certain threshold does not generate less correlated samples (e.g. as the trajectory starts to backtrack) and leads to a large computational cost per iteration.

Demo. Comparison between HMC, Gibbs and Metropolis-Hastings on a high-dimensional and ill-conditioned Gaussian target.

We'll now show that HMC has the correct stationary distribution. To do so, we first review key properties of Hamiltonian trajectories.

We denote H the *Hamiltonian* of the system, defined as the negative log joint density over z and ρ, that is,

$$H(z_t, \rho_t) = -\log \pi(z_t, \rho_t) = -\log \pi(z_t) - \log \pi(\rho_t). \tag{74}$$

With this notion, the equations of motion can be rewritten in their general form,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} z_t &= \frac{\partial H}{\partial \rho} \\ \frac{\mathrm{d}}{\mathrm{d}t} \rho_t &= -\frac{\partial H}{\partial z}. \end{aligned} \tag{75}$$

Next, we denote $\Phi_T : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$, the function that maps (z_0, ρ_0) to (z_t, ρ_t).

Lemma 10. *(Properties of the Hamiltonian trajectory) The Hamiltonian trajectory Φ_T verifies the following properties.*

- *For any T, Φ_T preserves the Hamiltonian,*

$$H(z_t, \rho_t) = H(z_0, \rho_0). \tag{76}$$

  Equivalently, the joint distribution over (z, ρ) is preserved.

- *(Louiville's theorem) The Hamiltonian map Φ_T is volume preserving, that is the determinant of the Jacobian Φ_T is 1.*

- *The Hamiltonian map Φ_T is time-reversible. That is, it admits an inverse Φ_T^{-1} which is obtained by negating the time derivative in eq. (72)–(73). Furthermore, we can obtain the inverse map Φ_T^{-1} by negating ρ, applying Φ_T and negating ρ again.*

Here, I'll only provide a proof of the first item and the proof for the other properties is omitted.

Proof. (Conservation of the Hamiltonian) Taking the derivative of the Hamil-

tonian with respect to time,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}H(z_t,\rho_t) &= \sum_{i=1}^{d}\frac{\partial H}{\partial z_i}\frac{\mathrm{d}z_i}{\mathrm{d}t}+\frac{\partial H}{\partial \rho_i}\frac{\mathrm{d}\rho_i}{\mathrm{d}t}\\ &= \sum_{i=1}^{d}\frac{\partial H}{\partial z_i}\frac{\partial H}{\partial \rho_i}-\frac{\partial H}{\partial \rho_i}\frac{\partial H}{\partial z_i}=0, \end{aligned} \tag{77}$$

where on the second line, we plugged in (75).

□

We now show that HMC has the correct stationary distribution.

Theorem 11. *The transition kernel for HMC admits $\pi(z)$ as its stationary distribution.*

There are multiple ways to prove this result. The first way is to directly show that HMC has the desired stationary distributions. The second way is to show reversibility (reversibility). While the second approach is less direct, it lays the foundation for showing that non-idealized versions of HMC have the right stationary distribution.

Here, we'll go over both proofs.

Proof. (Direct approach)

We will show that the joint distribution $\pi(z,\rho)$ is stationary and from this it will follow that the marginal distribution $\pi(z)$ is also stationary.

First, we start with $(z,\rho)\sim\pi$. The first step of HMC is to refresh the momentum, that is draw $\rho^*\sim\pi$. By construction $\pi(z,\rho)=\pi(z)\pi(\rho)$ and so z and ρ are independent. Hence, it suffices that z and ρ^* are marginally distributed according to π to have (z,ρ^*) be jointly distributed according to π and so the first step of HMC leaves the distribution π invariant.

The next step is to simulate a Hamiltonian trajectory. For ease of notation, we denote $x=(z,\rho^*)$. We need to show that for any measurable set A over $\mathbb{R}^{2d}$, $P(x\in A)=P(\Phi_T(x)\in A)$. Notice that $\{\Phi_T(x)\in A\}\iff\{x\in\Phi_T^{-1}(A)\}$ and so we must show $P(x\in A)=P(x\in\Phi_T^{-1}(A))$.

Now,

$$P(x\in A)=\int_A\pi(x)\mathrm{d}x. \tag{78}$$

Let $y=\Phi_T^{-1}(x)$. Doing a change of variable,

$$\int_A\pi(x)\mathrm{d}x=\int_{\Phi_T^{-1}(A)}\pi(\Phi_T(y))|J_\Phi(y)|\mathrm{d}y, \tag{79}$$

where $|J_\phi(y)|$ is the determinant of the Jacobian of Φ_T.

By Lemma 10, the Hamiltonian is conserved and so $\pi(\Phi_T(y)) = \pi(y)$. Furthermore, $|J_\phi(y)| = 1$.

Therefore,

$$\int_{\Phi_T^{-1}(A)} \pi(\Phi_T(y))|J_\Phi(y)|\mathrm{d}y = \int_{\Phi_T^{-1}(A)} \pi(y)\mathrm{d}y. \tag{80}$$

But the integral on the right-hand-side is exactly $P(x \in \Phi_T^{-1})$, and so $P(x \in A) = P(x \in \Phi_T^{-1}(A))$, as desired.

Thus both steps of HMC preserve the distribution $\pi(x)$ and thence the marginal distribution $\pi(z)$.

□

Next, we consider a proof that shows reversibility.

Proof. (using reversibility)

As in the previous proof, we have that the first step of HMC (momentum refreshment) leaves π invariant.

For the second step, we introduce a modification of the HMC transition kernel: namely, after time T we flip the momentum, so that the final state of the transition is

$$(z', \rho') = (z_T, -\rho_T). \tag{81}$$

This does not actually induce any algorithmic change, since the momentum is refreshed at the beginning of the next iteration and, ultimately, we're only interested in the position variable z. However the momentum flip ensures the algorithm maintains reversibility.

To see this, denote (z_0, ρ_0) the initial state of the trajectory. The Hamiltonian map induces a conditional probability,

$$p(z, \rho \mid z_0, \rho_0) = \delta(z = z_T, \rho = -\rho_T), \tag{82}$$

where δ is the Dirac delta function (meaning all the probability mass concentrates at a single point). Conversely, it follows from the reversibility of the Hamiltonian trajectory that,

$$p(z, \rho \mid z', \rho') = \delta(z = z_0, \rho = -\rho_0). \tag{83}$$

Therefore,

$$\mathbb{P}(z', \rho' \mid z_0, \rho_0) = \mathbb{P}(z_0, \rho_0 \mid z', \rho'). \tag{84}$$

In words, if (z', ρ') and (z, ρ) are each other's "reciprocal" on a Hamiltonian trajectory of length T, meaning $(z' = z_T, \rho' = -\rho_T)$ and $(z = z_0, \rho = \rho_0)$, then the probability on either side is 1, else it is 0.

Next, recall that the Hamiltonian is conserved and so $H(z_T, \rho_T) = H(z_0, \rho_0)$. Furthermore, $H(z, \rho) = -\log \pi(z, \rho) = -\log \pi(z) - \log \pi(\rho)$. Since $\pi(\rho)$ is a symmetric distribution, $\pi(-\rho) = \pi(\rho)$. Therefore $H(z', \rho') = H(z_T, \rho_T) = H(z_0, \rho_0)$ and moreover,

$$\pi(z', \rho') = \pi(z_0, \rho_0). \tag{85}$$

Combining equations (84) and (85), we have

$$\pi(z_0, \rho_0)\mathbb{P}(z', \rho' \mid z_0, \rho_0) = \pi(z', \rho')\mathbb{P}(z_0, \rho_0 \mid z', \rho'), \tag{86}$$

which is reversibility. Thus, the Hamiltonian map leaves π invariant, which completes the proof.

□

4.2 Discretized HMC

In practice, we cannot solve Hamilton's equations of motion exactly and so we resort to a numerical integrator. The most elementary integrator for solving differential equations is *Euler's method*, which uses a tangent approximations.

Specifically, at each iteration of Euler's method, we increment (z, ρ) by a step ϵ in the direction of the tangent $(\mathrm{d}z/\mathrm{d}t, \mathrm{d}\rho/\mathrm{d}t)$. If integrating from 0 to T, we repeat this process for L steps, such that $L\epsilon = T$. That is, we compute a discretized trajectory over $\{0, \epsilon, 2\epsilon, \cdots, L\epsilon\}$.

(Euler's method)

for i in $\{1, \cdots, L\}$ **do**
    $\rho(t + \epsilon) \leftarrow \rho_t + \epsilon \nabla \log \pi(z_t)$
    $z(t + \epsilon) \leftarrow z_t + \epsilon M^{-1} \rho_t$
end for

In practice, Euler's method works poorly and accumulates a large error as L increases.

A simple but surprisingly effective modification leads to the *leapfrog integrator*, which is what is used in practice to run HMC.

Question: How many gradient evaluations per step does the leapfrog integrator require?

Tuning problem. The leapfrog integrator requires choosing a step size ϵ. If ϵ is too large, then we do not simulate accurate Hamiltonian trajectories. On

(Leapfrog integrator)
for i in $\{1, \cdots, L\}$ **do**
$\rho_{t+\epsilon/2} \leftarrow \rho_t + \frac{\epsilon}{2}\nabla \log \pi(z_t)$
$z_{t+\epsilon} \leftarrow z_t + \epsilon M^{-1}\rho_{t+\epsilon/2}$
$\rho_{t+\epsilon} \leftarrow \rho_{t+\epsilon/2} + \frac{\epsilon}{2}\nabla \log \pi(z_{t+\epsilon})$
end for

the other hand, a small ϵ means we require more steps (i.e. a larger L) in order to perform integration over $[0, T]$.

One way to verify the accuracy of the leapfrog integrator is to check that the Hamiltonian $H(z_t, \rho_t)$ is indeed conserved over time. But even for an adequate ϵ, the error remains non-zero, which invalidates our argument that the Hamiltonian map verifies reversibility.

Metropolis adjustment. To ensure that the discretized HMC verifies reversibility, we can perform a Metropolis correction. Starting at a state (z, ρ), we obtain a new state (z_t, ρ_t) by simulating the Hamiltonian for time T. We then generate a proposal,

$$(z^*, \rho^*) = (z_t, -\rho_t). \tag{87}$$

Notice the sign change in ρ! We then accept the proposal with probability,

$$\alpha = \min\left(1, e^{-H(z^*,\rho^*)+H(z,\rho)}\right). \tag{88}$$

Computing the exponentiated difference in the Hamiltonian is equivalent to evaluating a ratio of the densities, $\pi(z^*, \rho^*)/\pi(z, \rho)$, usually seen in the Metropolis acceptance rule.

In practice, we can ignore the sign flip in ρ. Since ρ is distributed according to a normal centered at 0, ρ_t and $-\rho_t$ have the same density. In other words, flipping the momentum does not change the Hamiltonian. Furthermore, the new state ρ^* can be ignored because the first step of HMC is to refresh the momentum. Still, the sign flip is a useful theoretical tool which lets us prove reversibility.

Other strategies. There exist other strategies to correct the numerical error introduce by the leapfrog integrator. For example:

- **Multinomial sampling:** draw a sample from the *entire* discrete trajectory with the probability of drawing any sample weighted by $\exp(-H(z_t, \rho_t))$. This can be effective if the integration error at (z_T, ρ_T) is large, even though it is acceptable along other points on the trajectory.
- **Unadjusted sampling:** In some cases, the error introduced by the leapfrog integrator is sufficiently small that the correction is ineffective and it is better to accept aggressively. This leads to *unadjusted* samplers.

4.3 Adaptive Hamiltonian Monte Carlo

There are three tuning parameters in HMC: the step size ϵ of the leapfrog integrator, the number L of leapfrog steps (with the trajectory length given by $T = L\epsilon$), and the mass matrix M.

4.3.1 Adaptively setting the path length L

We start by assuming ϵ is fixed and to simplify the notation, we take $M = I$.

No-U Turn criterion. Conceptually, large steps in the state space reduce the autocorrelation between samples in the Markov chain and, at stationarity, leads to a larger ESS per iteration, at least when estimating the first moment. Therefore, running a Hamiltonian trajectory for a longer time is useful in so far as it increases the distance from the initial point. Once the trajectory starts backtracking, the benefits of running a longer trajectory decreases and we get less out of the computational work we do to simulate the Hamiltonian trajectory.

This reasoning motivates the *No-U Turn* criterion, whereby we monitor whether increasing the trajectory length T increases the distance from the initial point. This idea is at the heart of the *No-U Turn Sampler* (NUTS) which underlies the MCMC algorithm in Stan, PyMC and other probabilistic languages.

Fortunately, there is a straightforward way to monitor whether a longer trajectory would increase the distance from the starting point. Let z_0 be the initial position and z_t the current position. Then,

$$\frac{\mathrm{d}}{\mathrm{d}t}||z_0 - z_t||^2 = \frac{\mathrm{d}}{\mathrm{d}t}(z_0 - z_t)^T(z_0 - z_t) = 2(z_t - z_0)^T\rho_t. \tag{89}$$

A natural idea then would be to simulate a Hamiltonian trajectory until $(z_t - z_0)^T\rho_t < 0$, at which point we stop in order to not backtrack closer to our starting point.

Reversibility. Unfortunately, the map that simulates a Hamiltonian trajectory with a dynamic trajectory length is not time-reversible (Figure 7) and as a result, we cannot guarantee that the resulting MCMC algorithm is reversible and has the right stationary distribution.

A more sophisticated scheme can address this problem. To make the notation more compact, let $x = (z, \rho)$. The key idea is to stochastically generate a trajectory or *orbit* $\mathcal{I}$ from a starting point x_0, such that

$$\mathbb{P}(\mathcal{I} \mid x_0) = \mathbb{P}(\mathcal{I} \mid x_t), \tag{90}$$

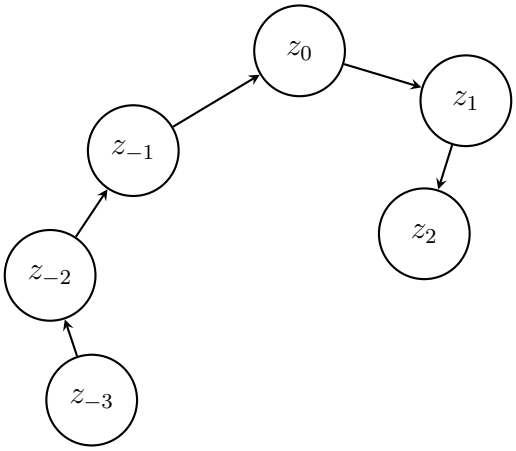

Figure 7: *Example of a (discrete) Hamiltonian trajectory. If starting at point z_0, the algorithm would simulate a trajectory forward in time all the way to z_2, where a U-Turn is detected. On the other hand, starting a trajectory from z_2 (with flipped momentum) may not lead back to z_0 since a U-turn may only occur later, for example at z_{-3}.*

for any $x_t \in \mathcal{I}$. In words, the probability of generating a particular orbit $\mathcal{I}$ is the same no matter which point in the orbit we start from.

Once this orbit is constructed, we set the new point x' in our Markov chain to a random point from the orbit, for example using a multinomial distribution,

$$\mathbb{P}(x' = x_t \mid x_t \in \mathcal{I}) \propto \exp(-H(z_t, \rho_t)) \propto \pi(z_t, \rho_t) = \frac{\pi(x_t)}{\sum_{t' \in \mathcal{I}} \pi(x_{t'})}. \tag{91}$$

We can explicitly check that this scheme verifies reversibility,

$$\begin{aligned} \pi(x)p(x' \mid x) &= \pi(x)\mathbb{P}(\mathcal{I} \mid x)\mathbb{P}(x' \mid \mathcal{I}) \\ &= \pi(x)\mathbb{P}(\mathcal{I} \mid x)\frac{\pi(x')}{\sum_{t \in \mathcal{I}} \pi(x_t)} \\ &= \pi(x')\mathbb{P}(\mathcal{I} \mid x')\frac{\pi(x)}{\sum_{t \in \mathcal{I}} \pi(x_t)} \\ &= \pi(x')\mathbb{P}(\mathcal{I} \mid x')\mathbb{P}(x \mid \mathcal{I}) \\ &= \pi(x')p(x \mid x'). \end{aligned} \tag{92}$$

Of course, this begs the question of *how* do we construct an orbit which verifies eq. (90)? First, it should be clear that any valid orbit must expand the Hamiltonian trajectory from a starting point z_0 both forward and backward in time.

Additive expansion. A natural if naive way to construct an orbit is with an *additive expansion*.

At each iteration of the expansion, we expand the trajectory forward or backward in time by one leapfrog step. The probability of choosing either direction in time is 1/2. We then stop expanding the trajectory, when a termination criterion based on the No-U-Turn condition is met.

Specifically, each time a new point z_t is added, we check whether it induces a U-Turn with any other existing point in the trajectory. Since we can evolve forward and backward in time, the U-Turn condition is checked at both extremities of each subtrajectory,

$$(z_+ - z_-)^T \rho_+ < 0 \quad \text{AND} \quad (z_- - z_+)^T \rho_- < 0. \tag{93}$$

Unfortunately, the additive expansion can be quite expansive for high-dimensional models, because

(i) each extension of the orbit requires us to check the U-Turn condition against all points on the trajectory.

(ii) we need to store every state (z_t, ρ_t), which is memory intensive.

Efficient NUTS. Practical implementation of NUTS rely on a more sophisticated expansion, which I'll only briefly review here. For more details, you may consult Betancourt [2017, Appendix A] and Hoffman and Gelman [2014].

Here's a summary of the strategy used in Stan:

- Starting from an initial point z_0, we randomly pick a direction: backward or forward in time. When then compute <u>one</u> leapfrog step in the chosen direction.
- We repeat this process, but at each step, we double the length of the simulated sub-trajectory. For example, the growing trajectory may look as follows:

  (1) `forward=1`, orbit: $\{x_0, x_1\}$.

  (2) `forward=1`, orbit: $\{x_0, x_1, x_2, x_3\}$

  (3) `forward=0`, orbit: $\{x_{-4}, x_{-3}, x_{-2}, x_{-1}, x_0, x_1, x_2, x_3\}$,

  where at each step the new sub-trajectory is colored in orange. This construction admits a representation as a binary tree.
- At each step, we check for U-turns within the new sub-trajectory and eliminate points which verify the termination criterion in eq. (93). We then sample a candidate (z_t, ρ_t) from the new sub-trajectory, assign a probability weight (which accounts for $H(z_{t'}, \rho_{t'})$ for all points in the sub-trajectory), and only save that particular point.

- Once we've constructed $\mathcal{I}$, we sample from the candidate sample retained from each sub-trajectory using a multinomial distribution. The weight of each sub-trajectory candidate can be chosen to match the distribution in eq. (91). Alternatively, we can favor newer trajectories in order to increase the distance from the starting point z_0. This approach is termed *Biased Progressive Sampling*.

4.3.2 Adaptively setting the step size ϵ

The step size ϵ must be chosen to ensure that the leapfrog integrator is both sufficiently accurate and efficient. Ultimately, this balance ideally results in a Monte Carlo estimator that achieves a target accuracy as quickly as possible.

Step size in Metropolis algorithms. In random-walk Metropolis, we need to tune the size of the random step taken at each iteration and we must navigate the following trade-off: A small step leads to a higher acceptance probability but more correlated samples. Conversely, a large step leads to a lower acceptance probability but less correlated samples. Under certain conditions, it can be shown that a step size with an average acceptance probability of $\alpha = 0.234...$ achieves an optimal ESS/per iteration [Roberts et al., 1997].

Likewise, adaptive HMC targets the acceptance probability α of the Metropolis step in eq. (88). This acceptance probability is driven by fluctuations in the Hamiltonian $H(z_t, \rho_t)$. In the limit $\epsilon \to 0$, the leapfrog integrator simulates exact Hamiltonian trajectories and $\alpha \to 1$.

Step size adaptation as stochastic optimization. The problem of adapting ϵ can be framed as a *stochastic optimization problem*. To see this, we first define the expected acceptance,

$$h(\epsilon) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}(\alpha_n \mid \epsilon), \tag{94}$$

where α_n is the acceptance probability at the n^{th} sampling iteration. Then our goal is to drive,

$$h(\epsilon) \to \delta, \tag{95}$$

for some target acceptance probability δ. For example, Stan's default is $\delta = 0.8$. But the optimal δ can vary largely depending on the problem. The default is a battle-tested heuristic but it may be necessary to adjust δ after a first attempt at running MCMC. In practice, $h(\epsilon)$ cannot be evaluated exactly, rather it is approximated by Monte Carlo using α_n's computed as we run MCMC.

The above task is equivalent to a stochastic optimization problem where our goal is to drive the gradient of an objective function—in this case $h(\epsilon) - \delta$—to

0 and where this gradient can only be evaluated stochastically. This means stochastic optimization algorithms can be used to update ϵ. A common choice for HMC is *dual averaging*.

Strictly speaking, NUTS does not use an accept/reject Metropolis step. Instead, α_n is computed by averaging hypothetical acceptance probabilities over the final sub-trajectory computed when constructing the orbit $\mathcal{I}$.

Freezing adaptation. One final and important caveat is that step size adaptation can alter the stationary distribution. Therefore, it is common practice to only do adaptation during the *warmup* and freeze adaptation (meaning ϵ no longer changes) during the sampling phase.

Certain algorithms, such as delayed-rejection HMC, try to adaptively find an optimal ϵ at each MCMC iteration, including during the sampling phase. But just as with adaptive trajectory lengths, such algorithms required careful constructions to ensure reversibility.

4.3.3 Adapting the mass matrix

The last tuning parameter to consider is the mass matrix M. In practice, we focus on the inverse-mass matrix, M^{-1}, since this is the quantity that appears in the leapfrog integrator. Specifically, the relevant step is,

$$z_{t+\epsilon} \leftarrow z_t + \epsilon M^{-1} \rho_{t+\epsilon/2}. \tag{96}$$

Let's first consider a diagonal mass matrix. Then, from eq. (96), we can interpret M^{-1} as a rescaling of the step size ϵ along each dimension.

Such a rescaling can make sense when the distribution has wildly different scales along each dimension. If we fix the mass matrix to I, then ϵ must typically be small enough to accommodate the dimension with the smallest scale and the integrator can be wildly inefficient along dimensions with a larger scale, where a less granular discretization of the Hamiltonian trajectory may be required.

With this conceptual motivation in mind, Stan uses the inverse sample variance based on samples collected during the warmup phase,

$$M_{ii}^{-1} = \frac{1}{\widehat{\text{var}}(z_i)}. \tag{97}$$

(As with step size, the mass matrix adaptation is frozen after the warmup phase.) That said, finding an optimal adaptation strategy for the mass matrix remains an open question.

Sometimes, the direction along which ϵ needs to be rescaled does not correspond to a direction along one particular dimension. Think for instance of a distribution with strongly correlated variables. In that case, we may use the a non-diagonal mass matrix and set it to the sample covariance matrix.

But this choice can be costly for high-dimensional targets. Indeed, the matrix-vector multiplication in eq. (96) costs $\mathcal{O}(d^2)$ for a dense matrix. By contrast, with a diagonal mass matrix, the cost of this operation reduces to $\mathcal{O}(d)$. Hence, the benefits of using a dense mass matrix must justify the more expensive leapfrog step. Sometimes, a compromised can be struck by using a mass matrix structured as a diagonal matrix + a low-rank matrix.

For some targets, the correct mass matrix depends on where in the target space we are simulating a Hamiltonian trajectory. (The most notorious example of this may well be Neal's funnel, which arises in hierarchical models.) In this case, the mass matrix can be set locally. One choice is to use the *curvature* of the target distribution,

$$\mathcal{C}(z) = -\nabla^2 \log \pi(z). \tag{98}$$

For justification for this choice, see e.g., Girolami and Calderhead [2011]. Unfortunately, this approach tends to be costly, notably through the requirement to compute and store higher-order derivatives of $\log \pi(z)$.

Notice that if the target is Gaussian, then $\mathcal{C}(z) = \Sigma^{-1}$, which resonates with the heuristic of using the covariance matrix to set the mass matrix.

5 Bayesian Modeling and Markov chain Monte Carlo on Modern Hardware

Much of the recent progress in compute power has come from parallel-processing, in particular the development of graphical processing units (GPUs). GPUs were originally designed for graphical rendering and then developed for numerical computation, specifically the matrix operations that underlie neural networks.

In this section, we'll examine the ways in which Bayesian modeling and MCMC can exploit GPUs and more generally parallel accelerators. We'll see that popular MCMC workflows often fail to leverage parallelization and that certain implementations, while remarkably effective on classic hardware such as central processing units (CPUs), are not mindful of the engineering constraints imposed by GPUs.

Perhaps most important takeaway from this lesson is to realize that parallel accelerators do not offer "arbitrary parallelization", i.e. the ability to perform completely distinct operations on different cores. Rather, accelerators are most effective when using Single Instruction Multiple Data (SIMD) instructions, wherein each core performs the same operation but on a different "data" or input.

5.1 Parallelizing the model

Let's examine some examples of Bayesian models, i.e. functions which return (log) joint densities, and see how compatible these functions are with parallelization and SIMD instructions.

Logistic regression. Consider a model with N observations (x_n, y_n), where $x_n \in \mathbb{R}^D$ and $y_n \in \{0, 1\}$:

$$\begin{aligned} \beta_d &\sim \text{normal}(0, 1) \\ y_n &\sim \text{Bernoulli}(\sigma(\beta^T x_n)), \end{aligned} \tag{99}$$

where σ is the logistic function. There are several opportunities for parallelization, when evaluating $\log p(y, \beta; x_n)$. Due to the marginal independence of the β_d's and the conditional independence of the y's,

$$\log p(y, \beta; x_n) = \sum_{d=1}^{D} \log \text{Normal}(\beta_d; 0, 1) + \sum_{n=1}^{N} \log \text{Bernoulli}(y_n; \beta^T x_n). \tag{100}$$

The terms inside each sum can be parallelized and calculated using SIMD instructions. On the other hand, we cannot parallelize *across*

the sums with SIMD instructions, since different cores would need to perform different operations. Hence, each sum is parallelized and performed sequentially.

Fortunately, we do not need to explicitly dictate how the parallelization happens on GPU, provided we use specialized packages such as `JAX` or `PyTorch`.

Pharmacokinetic model. We now consider a more sophisticated example (see your homework assignment for additional details). In this example, we have observations $y_{1:T}$ on a patient, physiological parameters θ, and a parameter σ for the observational model. The generative model is,

$$\begin{aligned} \theta &\sim p(\theta) \\ \sigma &\sim p(\sigma) \\ c_t &= f(\theta, t) \\ y_t &\sim \text{logNormal}(\log c_t, \sigma), \end{aligned} \tag{101}$$

where f is a function that returns the solution to a differential equation parameterized by θ at time t.

Once again, there are opportunities for parallelization. If the prior $p(\theta)$ factorizes into identical factors with different inputs, the calculation of $\log p(\theta)$ can be parallelized with SIMD instructions. Similarly, we can take advantage of the conditional independence of the y_t's when computing $\log p(y_{1:T} \mid c_{1:T}, \sigma)$.

The function f requires solving a differential equation. Depending on how we solve the differential equation, there may be some opportunities for parallelization. A numerical integrator usually requires doing sequential operations and so is not immediately amiable to parallelization (although some operation within the sequence may be).

Population pharmacokinetic model. An extension of the pharmacokinetic model considers observations collected across multiple patients. This leads to a hierarchical or population model, wherein each patient has their own individual parameters θ_n, with a prior driven by a population level parameters ϕ.

$$\begin{aligned} \phi &\sim p(\phi) \\ \theta_n &\sim p(\theta_n \mid \phi) \\ \sigma &\sim p(\sigma) \\ c_{nt} &= f(\theta_n, t) \\ y_{nt} &\sim \text{logNormal}(\log c_{nt}, \sigma). \end{aligned} \tag{102}$$

In addition to the parallelization opportunities we identified previously, it also seems natural to parallelize across patients.

This is straightforward to do if computing f involves the same operations for all patients, as will be the case if we use:

- an analytical expression for f.
- a matrix exponential, if the ODE is linear.
- a numerical integrator with a fixed step size.

On the other hand, if we use a numerical integrator with an adaptive step size, the number of operations to compute f likely differs between patients. This is in fact what we can expect from most standard numerical integrators (like the Runge-Kutta $4^{\text{th}}/5^{\text{th}}$ order in `Stan`).

In practice, population pharmacokinetic models are often fitted on a CPU cluster, with each core handling one patient in a non-SIMD fashion. `Stan` is well equipped to handle this type of parallelization, notably via the `reduce_sum` function.

Finding ways to fit population pharmacokinetic models on GPUs largely remains an open question. Some ideas:

- Use a fixed step size for the ODE solver, with a carefully chosen step size using information during the warmup.
- Use a surrogate model to solve the ODE, for example with a neural network.

Remark 12. *In an automatic differentiation framework, parallelization considerations for evaluating* $\log p(\theta, y)$ *naturally extend to computing the gradient* $\nabla \log p(\theta, y)$.

5.2 Parallelizing across chains

A general strategy to leverage access to multiple cores is to run Markov chains in parallel. Running multiple chains has several benefits:

- It increases the number of samples, which in turn reduces the variance of our Monte Carlo estimators.
- It is necessary to compute certain diagnostics, such as $\widehat{R}$.
- It can lead to adaptation strategies which share information between multiple chains.

In a classical framework, it is common to run 4 to 8 chains on a CPU.

With a GPU, it is straightforward to run hundreds or even thousands of Markov chains in parallel, especially with the help of packages such as `TensorFlow Probability` and `BlackJax`. (My personal record is 16,000.)

5.2.1 The Many-Short-Chains Regime

The prospect of running many chains in parallel suggests a strategy whereby the variance of Monte Carlo estimators is entirely handled by a large number of chains.

Suppose we run M independent Markov chains of length N and assume they are sufficiently warmed up that the bias is negligible. Let $\hat{f}$ be the Monte Carlo estimator obtained by averaging samples across all chains,

$$\hat{f} = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} f\left(z^{(mn)}\right). \tag{103}$$

Then,

$$\mathrm{Var}\hat{f} \leq \frac{\mathrm{Var}f}{M}, \tag{104}$$

and this upper-bound is attained when $N = 1$, that is each Markov chain contributes exactly one sample. Interpreting the ESS as a ratio of variances, we can say that the number of chains lower-bounds the ESS.

For many problems, it suffices to achieve an ESS of $\sim$100 or perhaps a few hundreds. For models of moderate dimension, it is well within a GPU's capacity to run 100 Markov chains in parallel. In that case, retaining one sample per chain is enough to achieve our target variance.

This configuration, wherein the variance is almost entirely handled by the number of chains rather than the length of the chain is the *many-short-chains* regime.

Of course, while it is possible to average out variance, we cannot average out bias. That is, regardless of the number of chains, we still need to warmup each Markov chain. A question then is: how quickly can we reduce the bias of our Monte Carlo estimator?

To build some intuition, let's consider the regime of geometric ergodicity where,

$$\left|\mathbb{E}f\left(z^{(N)} \mid z_0\right) - \mathbb{E}f\right| \leq b(z_0)\lambda^N, \tag{105}$$

for some initial point z_0 and $\lambda \in (0, 1)$. Integrating over the initial distribution

π_0 from which z_0 is drawn and using Jensen's inequality,

$$\begin{aligned} \left|\mathbb{E}f\left(z^{(N)}\right) - \mathbb{E}f\right| &\leq \left|\mathbb{E}\left(\mathbb{E}f\left(z^{(N)} \mid z_0\right)\right) - \mathbb{E}f\right| \\ &\leq \mathbb{E}\left|\left(\mathbb{E}f\left(z^{(N)} \mid z_0\right)\right) - \mathbb{E}f\right| \\ &= \mathbb{E}(b(z_0))\lambda^N. \end{aligned} \tag{106}$$

Hence, the squared bias decays at an exponential rate $\mathcal{O}(\lambda^{2N})$, provided we discard early samples (otherwise, it only decreases at a quadratic rate).

By contrast, the variance decreases at a linear rate $\mathcal{O}(1/N)$, with a constant which is 1 if we draw independent samples but which is otherwise larger. For N sufficiently large, we have that

$$\lambda^{2N} \leq 1/N. \tag{107}$$

In fact, for $\lambda \leq 0.841$, the above inequality holds for any $N \geq 1$ and within 20 iterations, the squared bias is already two orders of magnitude smaller than the variance. Hence, provided λ does not approach 1, decreasing the squared bias is much cheaper than decreasing the variance.

Moreover, the many-short-chains strategy holds the promise of turning an algorithm whose squared error decreases linearly with the length of the Markov chains to one whose error deceases exponentially with the length of the warmup, at least in some regimes.

5.2.2 Diagnostics and performance metrics

The many-short-chains regime has implications on how we measure the performance of MCMC.

Many analysis of MCMC focus on the asymptotic variance. For example, the most commonly reported empirical metric for the performance of an MCMC algorithm is the ESS/operation or ESS/s. This makes sense if we're running one long chain, since the the bias is negligible by the time we control the variance.

The many-short-chains regime flips the script: the variance is negligible (because we have many chains) and so the bias becomes the computational bottleneck. This is an invitation to revise how we analyze MCMC.

> **Example: $\widehat{R}$ convergence diagnostics.** Recall the $\widehat{R}$ statistics (module 2),
>
> $$\widehat{R} = \sqrt{\frac{N-1}{N} + \frac{\widehat{B}}{\widehat{W}}}, \tag{108}$$

where $\widehat{B}$ is the sample variance of the *per chain* Monte Carlo estimator,

$$\widehat{B} = \frac{1}{M-1} \sum_{m=1}^{M} \left(\hat{f}_N^{(m)} - \bar{f}_N^{(\cdot)} \right)^2, \tag{109}$$

and $\widehat{W}$ is the average within-chain sample variance,

$$\widehat{W} = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{N-1} \sum_{n=1}^{N} \left(\hat{f}\left(z^{(nm)}\right) - \hat{f}_N^{(m)} \right)^2. \tag{110}$$

A diagnostic for convergence is to check that $\widehat{R}$ is sufficiently small, for example, $\widehat{R} \leq 1.01$.

Unfortunately, in the many-short-chains regime, the per chain variance is never small and, no matter how long the warmup phase, $\widehat{R}$ remains large. This is true even if we achieved a low bias with an adequately long warmup and a small variance with a large number of chains.

We saw that as the number of chains increases and in particular as $M \to \infty$, $\widehat{B}$ is a consistent estimator of the *per chain* variance, which admits the following decomposition,

$$\mathrm{Var} \hat{f}_N^{(m)} = \underbrace{\mathrm{Var}\mathbb{E}\left(\hat{f}_N^{(m)} \mid z_0^{(m)} \right)}_{\text{nonstationary}} + \underbrace{\mathbb{E}\mathrm{Var}\left(\hat{f}_N^{(m)} \mid z_0^{(m)} \right)}_{\text{persistent}}. \tag{111}$$

The nonstationary variance decreases with a long warmup, as the Markov chains forget their starting point. On the other hand, the persistent variance requires a long sampling phase—a condition which is incompatible with the many-short-chains regime. Ideally, we would want a direct measure of the nonstationary variance to diagnose convergence.

This can be achieved with a **nesting scheme**. Suppose we partition our Markov chains into groups of chains or *superchains*. The chains within a superchain are initialized at the same starting point and then run independently.

Adjusting our notation, we take K to be the number of superchains and M the number of subchain within each superchain, for a total of KM chains. Then the *superchain Monte Carlo estimator* is,

$$\bar{f}_{NM}^{(k)} = \frac{1}{MN} \sum_{m,n} f\left(z^{(kmn)}\right). \tag{112}$$

We then compute the sample variance,

$$B_{\mathfrak{n}} = \frac{1}{K-1} \sum_{k=1}^{K} \left(\bar{f}_{MN}^{(k)} - \bar{f}_{KMN}^{(\cdot)} \right)^2. \tag{113}$$

As $K \to \infty$, $B_{\mathfrak{n}}$ converges to,

$$\begin{aligned} \mathrm{Var}\bar{f}_{MN}^{(k)} &= \mathrm{Var}\mathbb{E}\left(\hat{f}_{MN}^{(k)} \mid z_0^{(k)}\right) + \mathbb{E}\mathrm{Var}\left(\hat{f}_{MN}^{(k)} \mid z_0^{(k)}\right) \\ &= \mathrm{Var}\mathbb{E}\left(\hat{f}_{N}^{(m)} \mid z_0^{(k)}\right) + \frac{1}{M}\mathbb{E}\mathrm{Var}\left(\hat{f}_{N}^{(m)} \mid z_0^{(k)}\right), \end{aligned} \tag{114}$$

where in the second line, I used the conditional independence of the Markov chains within a superchain to show that (i) the super chain has the same expectation value as each individual subchain and (ii) the superchain's variance is reduced by a factor of $1/M$.

Our nesting strategy allows us to reduce the persistent variance, while leaving the nonstationary variance intact. With additional work, the contribution of the persistent variance can be exactly corrected for and a convergence diagnostic can be devised based on how quickly the nonstationary variance decreases.

Remark 13. *The behavior of the nonstationary variance reflects that of the squared bias, which also does not change with the number of chains. Additional arguments can be made to show that, under certain conditions, the nonstationary variance decreases at a "comparable" rate as the squared bias. (This is ongoing research!)*

Remark 14. *With some additional adjustments, it is possible to compute $\widehat{R}_{\mathfrak{n}}$ for chains of length 1. This opens the possibility of "continuously" monitoring the nonstationary variance at each iteration and developing an automatic stopping rule.*

5.2.3 GPU-friendly MCMC

To leverage the parallelization capacities of GPUs, we need SIMD instructions. The first step is to ensure that evaluation of the log density $\log \pi(z)$ and its gradient $\nabla \log \pi(z)$ can be done with SIMD instructions (Section 5.1). Then, at each "step", the GPU returns an array of gradients (one for each Markov chain).

Unfortunately, sophisticated adaptation strategy may not be compatible with SIMD.

NUTS is not GPU-friendly. NUTS and its variants are not GPU-friendly. Indeed, at each iteration, we need to construct orbits until the No-U-Turn termination criterion is met. Depending on the current "location" z_0 of a Markov

chain and the sampled momentum ρ_0, the number of leapfrog steps to compute a valid orbit will vary between chains.

For example, if we consider a pair of Markov chains, one which takes one leapfrog step and another one two, the SIMD execution model runs both chains for two leapfrog steps and discards the second leapfrog from the second chain. Moreover, the number of leapfrog steps is always driven by the largest orbit, yielding wasteful computation for most chains.

ChESS-HMC: a GPU friendly alternative. The SIMD constraint motivates new adaptive algorithms, such as ChEES-HMC [Hoffman et al., 2021], which ensures that each iteration of the Markov chain can be run in lock-step. This is simply done by fixing the trajectory length L.

Which L should we choose then? Rather than achieving the No-U-Turn criterion for each chain at each iteration, we instead try to maximize the *expected squared jump distance* (ESJD),

$$\text{ESJD}_f = \mathbb{E}\left[||f(z_T) - f(z_0)||^2\right]. \tag{115}$$

If we take f to be the identity, we obtain the expected squared distance from the initial point z_0 to the final point z_T. In other words, while we cannot hope to maximize the travelled distance for each chain at each iteration, we can try to maximize that distance on average.

Other choices of f can be considered for the ESJD. For example, ChEES-HMC focuses on the second moment of z, arguing that it is a more difficult quantity to estimate than the first moment.

Remark 15. *When the target distribution has varying scales across dimensions (or somewhat equivalently, if we're not able to construct a good mass matrix M), we can improve the performance of HMC by jittering the trajectory length, i.e. sampling the number of steps size uniformly from $\{1, 2, \cdots, L\}$.*

Opportunities from running many chains. There are a number of MCMC strategies that benefit from running many chains. For example, the mass matrix and step size adaptation can pool information across many chains, meaning the adaptation per iteration is much faster. In some cases, this can lead to faster bias decay.

Certain methods use Markov chains targeting different distributions to sample from a challenging distribution. An example of this is tempering schemes.

6 Variational Inference

We have seen how approximate sampling techniques, such as Markov chain Monte Carlo and importance sampling, can help us construct Monte Carlo estimators with which we can learn summaries of the posterior.

Another paradigm is to find a tractable distribution which approximates the posterior. By "tractable", I mean that the distribution is easy to manipulate: for example, it is straightforward to compute the mean and variance, and to draw samples. Variational inference (VI) is a broad class of methods to find tractable approximations to the posterior.

The task of VI is to find the best distribution q inside a pre-specified family $\mathcal{Q}$ of tractable distributions to approximate the target distribution p,

$$q^* = \text{argmin}_{q \in \mathcal{Q}} D(p||q), \tag{116}$$

where D is a divergence between p and q. Hence VI turns Bayesian inference into an optimization problem.

The two fundamental tuning choices of VI are (i) the family $\mathcal{Q}$ of approximations and (ii) the objective function which we minimize. The trade-offs for $\mathcal{Q}$ are rather intuitive: a rich family of approximations leads to a better approximation but often complicates the optimization.

The choice of objective function is subtler: there exists many ways to compare distributions. By definition, a valid divergence must be such that

$$D(p||q) = 0 \text{ if and only } p = q, \text{ and otherwise } D(p||q) > 0.$$

But this constraint only somewhat restricts our options. There are many valid divergences and, if we have a restricted family of approximations, i.e., $p \notin \mathcal{Q}$, then different divergences produce different solutions. Hence, we must examine how well different solutions approximate summaries of the posterior. At the same time, certain divergences are much more difficult to optimize than others. For example, the *total variation distance*, which we studied for MCMC, is a valid divergence but cannot be minimized through standard optimization techniques.

Before getting into the details of how VI works, let's highlight two classical applications of VI:

- **Topics model [Blei et al., 2003].** VI is used to train a probabilistic model, with which 1.8 million New York times article were sorted into topics based on the words contained in each article ("bag of words" representation of the article). Once trained, the model determines itself the topics, the words in each topic, and ascribes a distribution of topics to each article, e.g., 80% international affairs, 10% finance, $\cdots$.

- **Variational autoencoder [Kingma and Welling, 2014, Rezende et al., 2014].** This algorithm—we will see how it breaks into a probabilistic model and an inference procedure—is used to compress high-dimensional images. The model parameterizes each high-dimensional image by a low-dimensional latent variable and the image can be reconstructed by learning a neural network (the "decoder") which takes in the low-dimensional representation and outputs the reconstructed image.

6.1 Gaussian Variational Inference

A classic off-the-shelf VI algorithm is Gaussian variational inference (G-VI). We will use G-VI as a concrete example to introduce several concepts.

Suppose we have a target $p(z)$ with $z \in \mathbb{R}^d$. In practice, the space of z, denoted $\mathcal{Z}$, may not be $\mathbb{R}^d$ but as long as this space is continuous, we can apply a transformation to map $\mathcal{Z}$ to $\mathbb{R}^d$. For example, if a variable z_i is constrained to be positive, we can apply a log transformation.

In G-VI, $\mathcal{Q}$ is the family of Gaussian distributions. Each member is fully defined by the *variational parameters* $\lambda = (\nu, \Psi)$, where $\nu \in \mathbb{R}^d$ is the mean and Ψ is the covariance matrix. Often times, Ψ is taken to be a diagonal matrix, primarily to reduce the number of variational parameters from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$. This approach is called the *mean-field approximation*, although I also like the more descriptive name, *factorized approximation*. The resulting algorithm is factorized Gaussian variational inference (FG-VI).

Eq. (116) can be now be written as an optimization problem over the space of variational parameters,

$$\lambda^* = \text{argmin}_\lambda D(p||q_\lambda). \tag{117}$$

The most common objective function in VI is the *reverse Kullback-Leibler divergence*,

$$\text{KL}(q||p) = \int (\log q(z) - \log p(z)) q(z) \mathrm{d}z, \tag{118}$$

which compares densities over measurable sets. In your homework assignment, you showed that $\text{KL}(q||p)$ was a valid divergence, by showing that $\text{KL}(q||p) \geq 0$ and $\text{KL}(q||p) = 0$ if and only if $q = p$. Compared to other divergences, the reverse KL is relatively straightforward to analyze theoretically and to minimize.

> **Example: FG-VI applied to a multivariate Gaussian.** Suppose p is a multivariate Gaussian target with mean μ and a non-diagonal covariance matrix Σ. Then it can be shown analytically (as you did in your homework assignment) that the optimal variational parameters are
>
> $$\nu = \mu \qquad \Psi_{ii} = 1/[\Sigma^{-1}]_{ii}. \tag{119}$$

In words, FG-VI recovers the mean and the marginal precisions of p. (Recall that the precision matrix is the inverted covariance matrix.) On the other hand, q misestimates other properties of p: for example, it underestimates the variance (as you showed in the homework) and the entropy.[9]

More generally, VI can recover certain properties of p even if $q \neq p$. The ability of VI to produce accurate estimators has notably been studied in asymptotic limits, where the number of observations $N \to \infty$ and where, under certain regularity conditions, the posterior becomes Gaussian per the *Bernstein-von Mises theorem* [e.g. Katsevich and Rigollet, 2024].

6.2 Variational inference in the presence of symmetry

Recent work has demonstrated VI's ability to recover properties of p when p and $\mathcal{Q}$ exhibit certain symmetries [Margossian and Saul, 2025]. Here, we'll examine how VI can recover the mean in the presence of even-symmetry.

Definition 16. *(Even- and odd-symmetry) We say a function f is even-symmetric about ν if for all $z \in \mathbb{R}^d$,*

$$f(z + \nu) = f(-z + \nu). \tag{120}$$

Similarly, we say a function f is odd-symmetric about ν if for all $z \in \mathbb{R}^d$,

$$f(z + \nu) = -f(-z + \nu). \tag{121}$$

Consider now a location family of distributions with location parameter ν and base distribution q_0, that is

$$q_\nu(z) = q_0(z - \nu), \tag{122}$$

and suppose that q_0 is even-symmetric. Examples of such distributions are the Gaussian, student-t and Laplace distributions.

Theorem 17. *Let $\mathcal{Q}$ be a location-family with even-symmetric based distribution q_0. Suppose p is even-symmetric about μ.*

Furthermore, suppose p and $\mathcal{Q}$ are such that the order of differentiation and integration can be changed when computing $\nabla_\nu KL(q_\nu||p)$.

Then $\nu = \mu$ is a stationary point of $KL(q_\nu||p)$.

[9] This second result deserves to be nuanced: while the entropy is underestimated, FG-VI can still yield reasonable estimates of the entropy in certain limits [Margossian and Saul, 2023].

Proof. We start from the definition of the KL-divergence, eq. (118), and operate a change of variable $\zeta = z - \nu$,

$$\begin{aligned} \mathrm{KL}(q_\nu||p) &= \int (\log q_\nu(z) - \log p(z)) q_\nu(z) \mathrm{d}z \\ &= \int (\log q_0(\zeta) - \log p(\nu + \zeta)) q_0(\zeta) \mathrm{d}\zeta \\ &= -\mathcal{H}(q_0) - \log p(\nu + \zeta)) q_0(\zeta) \mathrm{d}\zeta, \end{aligned} \tag{123}$$

where $\mathcal{H}(q_0)$ is the entropy of q_0 and doesn't depend on the variational parameter ν. We now differentiate with respect ν and use the fact that we can change the order of integration and differentiation,

$$\begin{aligned} \nabla_\nu \mathrm{KL}(q_\nu||p) &= -\int \nabla_\nu \log p(\nu + \zeta) q_0(\zeta) \mathrm{d}\zeta, \\ &= -\int \nabla_\zeta \log p(\nu + \zeta) q_0(\zeta) \mathrm{d}\zeta, \end{aligned} \tag{124}$$

where in the second line we use the symmetry in the argument of ν and ζ to obtain a gradient with respect to ζ.

Now suppose $\nu = \mu$. Then $p(\nu + \zeta)$, taken as a function of ζ, is even-symmetric and its gradient $\nabla_\zeta p(\nu + \zeta)$ is odd-symmetric. Furthermore, we have by assumption that $q_0(\zeta)$ is even-symmetric. Hence, the integrand is the product of an odd-symmetric and an even-symmetric distribution, and is itself odd-symmetric. Therefore, the integral goes to 0. □

Under some additional assumptions (i.e. p is log-concave[10]), we can show that this stationary point is a unique minimizer.

If p has a finite first moment, then the point of symmetry corresponds to the mean, which is then recovered by ν^*. It's worth noting that the theorem allows for several misspecification. In particular:

- q may be factorized, even though p is not.
- q and p may have different tail behaviors.

An ongoing research endeavor is to understand how VI's symmetry-matching properties generalizes to other symmetries.

6.3 Stochastic optimization for variational inference

In a Bayesian context, the target distribution is the posterior $p(z \mid y)$ and is typically only known up to a normalizing constant. Then, the variational

[10] Although I'm working on relaxing this condition...

objective becomes,

$$\begin{aligned} \text{KL}(q_\lambda||p) &= \int (\log q_\lambda(z) - \log p(z,y) + \log p(y)) q_\lambda(z) \mathrm{d}z \\ &= \int (\log q_\lambda(z) - \log p(z,y)) q_\lambda(z) \mathrm{d}z + \log p(y), \end{aligned} \tag{125}$$

where the laster term, $\log p(y)$, can be ignored for the purposes of optimizing λ.

Often times, the VI optimization problem is reformulated as follows,

$$\lambda^* = \text{argmax}_\lambda \int (\log p(z,y) - \log q_\lambda(z)) q_\lambda(z) \mathrm{d}z, \tag{126}$$

with the maximized objective termed the *evidence lower bound* (ELBO). The name comes from the fact that $p(y)$ is sometimes called the *evidence* and

$$\begin{aligned} & 0 \leq \text{KL}(q_\lambda||p) \\ \iff \quad & 0 \leq \log p(y) - \text{ELBO} \\ \iff \quad & \text{ELBO} \leq \log p(y). \end{aligned} \tag{127}$$

When doing full Bayesian inference, $p(y)$ does not typically play a role, however we will see some non-Bayesian applications where $p(y)$ plays a pivotal role.

The first challenge with maximizing the ELBO is that the integral cannot be solved analytically. Instead, the ELBO is approximated via Monte Carlo and using draws from q_λ (which, in theory, is easy to sample from),

$$\text{ELBO} \approx \frac{1}{B} \sum_{b=1}^{B} \log p(z^{(b)}, y) - \log q_\lambda(z^{(b)}) q_\lambda(z^{(b)}) \mathrm{d}z, \quad z^{(b)} \sim q_\lambda. \tag{128}$$

To do gradient-based optimization, we also need to evaluate the gradient of the target. Assuming the order of integration and differentiation can be changed,

$$\nabla_\lambda \text{ELBO} = \int \nabla_\lambda (\log p(z,y) - \log q_\lambda(z)) q_\lambda(z) \mathrm{d}z. \tag{129}$$

But constructing a Monte Carlo estimator of the gradient is less straightforward, because the samples $z \sim q_\lambda$ carry a somewhat hidden dependence on λ, which we need to differentiate through. (In the integral, this dependence is made explicit in the density term $q_\lambda(z)$).

This motivates the *reparameterization trick*, whereby the dependence on λ and the stochastic component in z are disentangled. In particular, we want a two-step sampling procedure of the form,

$$\begin{aligned} \zeta &\sim p(\zeta) \\ z &= f_\lambda(\zeta), \end{aligned} \tag{130}$$

with f an invertible function. For example, suppose $q_\lambda(z)$ is a univariate normal with mean μ and variant ψ, and $\lambda = (\mu, \psi)$. Then, a sample from q_λ can be obtained as,

$$\begin{aligned} \zeta &\sim \text{normal}(0,1) \\ z &= \sqrt{\psi}\zeta + \mu. \end{aligned} \tag{131}$$

Then, applying standard rules for a change of variable with respect to a probability measure, eq. (129) is rewritten as an integral with respect to ζ,

$$\nabla_\lambda \text{ELBO} = \int \nabla_\lambda (\log p(f_\lambda(\zeta), y) - \log q_\lambda(f_\lambda(\zeta)) q(\zeta) \mathrm{d}\zeta. \tag{132}$$

Our Monte Carlo estimator for the gradient is then,

$$\widehat{\nabla_\lambda \text{ELBO}} = \frac{1}{B} \sum_{b=1}^{B} \nabla_\lambda (\log p(f_\lambda(\zeta^{(b)}), y) - \log q_\lambda(f_\lambda(\zeta^{(b)})), \quad \zeta^{(b)} \sim q(\zeta). \tag{133}$$

Optimization is then performed via gradient-descent. That is, starting from an initial guess λ_0, we iteratively update our parameter estimate by following the gradient,

$$\lambda_t = \lambda_{t-1} + \epsilon_t \widehat{\nabla_\lambda \text{ELBO}}, \tag{134}$$

until some convergence criterion is met; for example, we may require that $|\widehat{\nabla_\lambda \text{ELBO}}| \leq \delta$ for some tolerance δ or check that the (estimated) ELBO remains stable after several iterations.

A crucial question is how to choose the step sizes ϵ_t. There is a rich literature on the subject—going all the way back to Newton's method,[11] which sets ϵ_t to the inverted Hessian of the objective function $g(\lambda)$,

$$\epsilon_t = \nabla^2 g(\lambda). \tag{135}$$

In standard settings—i.e., when minimizing a convex objective functions—Newton's method enjoys a quadratic convergence rate. However the cost of evaluating and inverting a second-order derivative has made Newton's method unpopular in settings where λ is high-dimensional.

Another complication in our setting is that the gradient is not evaluated exactly but stochastically. This raises the question of whether gradient-descent can converge to a solution. Proofs of convergence can be obtained by requiring that:

[11] Newton's method is accredited to Newton himself. The wikipedia page also points to a special case of the method dating back to the Babylonian in the 16th-19th century BC!

(i) The gradient estimator is unbiased, as is the case when drawing exact samples from $q(\zeta)$.

Interestingly, this condition is met when $B = 1$, that is we use a single sample. In my experience, running $B \ll 1$ improves the stability and convergence rate of the algorithm, and what is more samples can be drawn in parallel.

(ii) The step sizes ϵ_t decay at a rate that is neither too slow nor too fast. This condition is captured by the (somewhat mystifying) Robbins-Monroe condition,

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \quad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{136}$$

Much of the recent literature on stochastic optimization concerns finding a good learning schedule $\{\epsilon_t\}$.

There exist a few off-the-shelf optimizers and the most popular ones are currently `Adam` and `LBFGS`. These methods enjoy high-performance implementations in libraries such as `JAX` and `PyTorch`, and provide a good starting point.

6.4 Example: Variational inference for the SIR model

`Stan` provides a G-VI algorithm, called *automatic differentiation variational inference* (ADVI) (which unfortunately is not a very descriptive name). The algorithm proceeds as follows:

1. Transform the parameter θ to an unconstrained variable $\tilde{\theta} \in \mathbb{R}^d$, using an invertible transformation f, with $\tilde{\theta} = f(\theta)$.
2. Approximate $p(\tilde{\theta} \mid y)$ by a Gaussian $q_\lambda(\tilde{\theta})$—the Gaussian can either have a diagonal covariance matrix, meaning the Gaussian is factorized ("mean-field") or have a dense covariance matrix ("full-rank"). The approximation is found by minimizing $\mathrm{KL}(q_\lambda(\tilde{\theta}) || p(\tilde{\theta} \mid y))$.
3. Draw $\tilde{\theta} \sim q_\lambda(\tilde{\theta})$ and transform back to the original space, $\theta = f^{-1}(\tilde{\theta})$.

Coding demo. We can run mean-field ADVI for the SIR influenza model. Specifically, we'll use the version of the model with a negative binomial likelihood. Since the joint distribution $p(\theta, y)$ remains unchanged, there is no need to revise the model.

In addition to approximating the posterior for the model parameters $\theta = (\gamma, \beta, \phi^{-1})$, we can also examine the posterior distribution of some derived quantities, specifically the recovery time T and the R_0 number. Table 1 shows the posterior summaries obtained with ADVI and is directly translated from the `summary()` function in `cmdStanR`.

	Mean	Median	SD	MAD	q_5	q_{95}
γ	0.537	0.536	0.0416	0.0420	0.472	0.610
β	1.75	1.75	0.0499	0.0500	1.68	1.84
ϕ^{-1}	0.148	0.129	0.0881	0.0702	0.0544	0.320
T	1.87	1.87	0.145	0.147	1.64	2.12
R_0	3.29	3.29	0.267	0.268	2.86	3.73

Table 1: *ADVI posterior summary statistics*

We compare these results to the output obtained with MCMC (Table 2).

	Mean	Median	SD	MAD	q_5	q_{95}	$\hat{R}$	ESS	ESS (tail)
γ	0.541	0.539	0.0450	0.0415	0.470	0.618	1.00	2771	2492
β	1.74	1.73	0.0537	0.0491	1.65	1.82	1.00	2298	2189
ϕ^{-1}	0.137	0.121	0.0744	0.0600	0.0520	0.276	1.00	2303	2363
T	1.86	1.85	0.155	0.143	1.62	2.13	1.00	2771	2492
R_0	3.23	3.21	0.276	0.250	2.82	3.71	1.00	2906	2324

Table 2: *MCMC posterior summary statistics*

For this particular example, the estimates of posterior summaries returned by MCMC and ADVI are in close agreement. The model is relatively simple—after all, it's only 2-dimensional—but it does involve a likelihood parameterized by an ODE. So it is a bit surprising that ADVI works so well, despite using a simple approximation.

Notice that the MCMC summary contains three additional columns for diagnostics to check that the posterior inference is reliable. Unfortunately, these diagnostics are MCMC specific and do not apply to VI.

`Stan`'s ADVI checks that the stochastic optimization converges. But unlike for MCMC, convergence of VI does not guarantee that we have accurate estimates of the posterior—merely that we found the "best" candidate $q^* \in \mathcal{Q}$ to approximate p. (And even then, convergence diagnostics for optimization are often fooled by local optimas.) If $\mathcal{Q}$ is too restrictive, then q^* may still be a poor approximation of p.

The lack of automated inference checks for VI is a gap in the workflow.

But checks exist. For example, we can use *Pareto-smoothed importance sampling* [Yao et al., 2018], as we did for leave-one-out cross-validation (Section 3). In VI, q^* is the proposed distribution and p is the target distribution. Recall that because p is only known up to a normalizing constant, we must compute self-normalized importance weights.

Even without validating the inference, we can still perform model criticism,

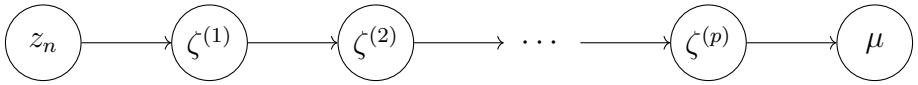

Figure 8: *A neural network is a sequence of transformations ("arrows") between layers ("nodes"). The first layer is the input layer, the final layer the output layer and all the layers in between are hidden layers. Each transformation is characterized by a linear transformation followed by a nonlinear transformation.*

e.g., posterior predictive checks and predictive scores on a validation set.

In many papers which use VI, you'll find that the authors only perform limited checks on the inference but always evaluate the performance of the trained model. We may then reasonably question whether the posterior is accurately estimated, at least not by MCMC standard, e.g., the kind of accuracy we would get with $\widehat{R} \leq 1.01$ and ESS ≥ 100, which corresponds to a negligible bias and standard deviation of $\sqrt{\text{Var} f(z)}/10$. Nonetheless, an imperfectly trained model can still produce useful insights and fulfill its scientific purpose.

Class discussion. Why is it important (or not important) to do inference check in Bayesian Workflow?

6.5 Example: Variational autoencoder

The variational autoencoder (VAE) is class of machine learning models/algorithms, which can be decoupled into a probabilistic model and an inference procedure.

6.5.1 Probabilistic model: latent variables and neural network

A *latent variable model* is a *hierarchical* model where each observation x_n has a corresponding latent variable z_n. The model also admits a "global" *hyperparameter* θ, which is common to all observations x_n. A Bayesian model can be specified by the joint distribution

$$p(x_{1:N}, z_{1:N}, \theta) = p(\theta)p(z_{1:N} \mid \theta) \prod_{n=1}^{N} p(x_{1:N} \mid z_{1:N}, \theta). \tag{137}$$

In a frequentist setting, the hyperparameter θ is treated as fixed and the model is specified as a joint over $(z_{1:N}, x_{1:N})$ parameterized by θ, $p_\theta(x_{1:N}, z_{1:N})$.

In the canonical VAE, x_n is a high-dimensional image, e.g. three color channels for each pixel, and z_n is a low-dimensional representation of the image. θ is the parameter of a *decoder*, which stochastically maps z_n to x_n. One interpretation of the model is that z_n is a compression of x_n and the decoder decompresses z_n back to x_n.

For example, we may model x_n (the colors of each pixel) as normally distributed,[12] whose mean μ and covariance Σ is the output of a function,

$$\mu = f_\mu(z_n; \theta) \quad \Sigma = f_\Sigma(z_n; \theta). \tag{138}$$

In the VAE, the function f is modeled by a *neural network*. A neural network is a composition of transformations starting from an input layer and mapping to an output layer. The layers in-between are called hidden layers (Figure 8). The transformation between layers takes in an input $\zeta^{(i)}$ and outputs $\zeta^{(i+1)}$, typically by applying a linear transformation followed by a nonlinear transformation,

$$\zeta^{(i)} = \sigma\left(W^{(i)}\zeta^{(i)} + b^{(i)}\right), \tag{139}$$

where the matrix $W^{(i)}$ is called the *weights* of the network, $b^{(i)}$ is the "bias", and σ is a non-linear transformation. The weights and biases constitute the parameters of the neural network,

$$\theta = \left(W^{(1)}, W^{(2)}, \cdots, W^{(p)}, b^{(1)}, b^{(2)}, \cdots, b^{(p)}\right). \tag{140}$$

The weight matrix $W^{(i)}$ need not be a square matrix, meaning that the dimension of $\zeta^{(i)}$ and $\zeta^{(i+1)}$ can differ. In practice, the hidden layers are often taken to have a larger dimension than the input and outputs layers.

A common choice for σ is the rectified linear unit (ReLu) function,

$$\sigma(\alpha) = \mathbf{max}(0, \alpha). \tag{141}$$

There are many choices on the architecture of the neural network. For example what should the depth of the network be, i.e. the number of layers? What should be its width, i.e. dimension of intermediate layers? Which nonlinear transformation should we use?

You may ask yourself why use a neural network in the first place? Neural networks have had a tremendous success empirically and there exists *some* theory to explain this success. In particular, *universal approximation theorems* provide ideal conditions under which a neural network can approximate any function f arbitrarily well. A well-known theorem states that a neural network with one infinitely-wide hidden layer can approximate any "well-behaved" function. But despite these results, the theory is often considered to be lagging behind the practice and explaining the remarkable success of neural networks remains an active area of research.

[12] A more exact model would use a categorical distribution, since the color channels are an integer over $[0, 255]$ but for a hight "count", a continuous distribution can work reasonably well.

6.5.2 Inference: Amortized variational inference

Now that we have our model, we can come up with a training procedure.

In a Bayesian setting, the task is to learn the posterior $p(\theta, z_{1:N} \mid x_{1:N})$.

In a frequentist setting, we learn θ by maximizing the marginal likelihood,

$$\theta^* = \text{argmax}_\theta \; p_\theta(x_{1:N}) = \int \prod_{n=1}^{N} p_\theta(x_{1:N} \mid z_{1:N}, \theta) p(z_{1:N}) \mathrm{d}z_{1:N}. \tag{142}$$

Notice the unknown $z_{1:N}$ is still treated as a random variable and so the procedure can be described as a hybrid between a Bayesian and a frequentist approach.

In practice, the integral on the R.H.S of eq. (142) is intractable. To approximate $p_\theta(x_{1:N})$, we will use the ELBO. Applying eq. (127) to our problem, we have that

$$\text{ELBO}(\theta, \lambda) = p_\theta(x_{1:N}) - \text{KL}(q_\lambda(z_{1:N}) || p_\theta(z_{1:N} \mid x_{1:N})), \tag{143}$$

where λ are the variational parameters for a family of approximations $\mathcal{Q}$. To have a concrete example in mind, you may once again take $\mathcal{Q}$ to be the family of Gaussians.

The ELBO provides a lower-bound on $p_\theta(x_{1:N})$ and the gap in this bound is given by $\text{KL}(q_\lambda(z_{1:N}) || p_\theta(z_{1:N} \mid x_{1:N}))$. This suggests a joint optimization problem over θ and the variational parameters λ,

$$\begin{aligned} \theta^*, \lambda^* &= \text{argmax}_{\theta,\lambda} \; \text{ELBO}(\theta, \lambda) \\ &= \text{argmax}_{\theta,\lambda} \; \left[p_\theta(x_{1:N}) - \text{KL}(q_\lambda(z_{1:N}) || p_\theta(z_{1:N} \mid x_{1:N})) \right]. \end{aligned} \tag{144}$$

If $\mathcal{Q}$ is rich enough, then we can drive $\text{KL}(q_\lambda(z_{1:N}) || p_\theta(z_{1:N} \mid x_{1:N}))$ to 0 and optimize the true objective function. However, $\mathcal{Q}$ is often restricted and so the KL can usually not be driven to 0, meaning minimize an approximation of $p_\theta(x_{1:N})$ rather than $p_\theta(x_{1:N})$ itself.

Conditional on θ, each observation x_n only depends on z_n,

$$p(x_{1:N} \mid z_{1:N}) = \prod_{n=1}^{N} p(x_n \mid z_n). \tag{145}$$

We will also assume that the priors on $z_{1:N}$ factorizes, $p(z_{1:N}) = \prod_n p(z_n)$.

If the likelihoods $p(x_n \mid z_n)$ follow the same distribution and similarly for the priors $p(z_n)$, then the posterior $p(z_n \mid x_n)$ can be expressed as a function of x_n and z_n,

$$p(z_n \mid x_n) = g(z_n, x_n). \tag{146}$$

This suggests that, instead of learning a variational parameter λ_n for each factor $q(z_n)$, we can instead *amortize* the procedure and learn a function f such that,

$$\lambda_n = f(x_n). \tag{147}$$

If $\mathcal{Q}$ is the family of Gaussian, then f maps x_n to a posterior mean and covariance matrix. Once again we can approximate f with a neural network. This neural network is often termed the *encoder*, because it maps the original image x_n to a low-dimensional representation z_n.

There are several arguments for using amortized VI:

- The number of variational parameters is determined by the size of the encoder neural network, rather than by the dimension of the observations.
- Information is pooled across observations, empirically leading to faster optimization.
- The learned function, $\hat{f}$ can be used to compress images in a validation set.

On the other hand, amortization can also have drawbacks:

- Amortization restricts the family $\mathcal{Q}$ of approximation—this is most obvious if we use a simple encoder, for example a linear function—and the solution can be suboptimal relative to standard VI. This sub-optimality is known as the *amortization gap*.

Amortized VI can also be used to do full Bayesian inference on the joint posterior $p(\theta, z_{1:N} \mid x_{1:N})$ [e.g Margossian and Blei, 2024].

6.6 Variational inference beyond the Gaussian approximation

Many VI algorithms rely on a family $\mathcal{Q}$ of approximations which is not Gaussian. In many applications, it is common to construct a bespoke family $\mathcal{Q}$ to match the characteristics of the target p [e.g., Blei et al., 2003]. But this approach requires a large algorithmic effort from the user and works poorly in Bayesian workflow, since every revision of the model requires a revision of the variational family $\mathcal{Q}$.

The ideal of *black box VI* is to use a family which works well across a broad range of models and does not require bespoke manipulations from the user. The Gaussian approximation is a reasonable starting point:

- Many Bayesian models have Gaussian components in the likelihood or prior.

- Under certain conditions, the Bernstein-von Mises theorem tells us that as we accumulate data the posterior becomes more Gaussian.
- The resulting variational optimization problem tends to be manageable, especially if the approximation is factorized.

But in many applications, we can improve on the Gaussian approximation. Here I'll very briefly review some strategies:

- **Pathfinder VI [Zhang et al., 2022].** Pathfinder offers a slight twist on traditional VI, but in end effect, it produces a normal approximation.

  The *multi-pathfinder* runs Pathfinder VI I times in parallel and generates I normal approximations. If p is far from Gaussian, different runs of the optimizer produce different solutions. Combining these solutions produces a mixture of normals and the relative weight of each mixture component is determined by an importance sampling scheme.

- **Normalizing flow.** Normalizing flows are defined by an initial draw $\zeta \sim q_0$ which is then transformed via a learned function $z = f(\zeta)$. Usually, we'll pick q_0 to be a simple distributon, for example a Gaussian. This simple distribution is then transformed into a more sophisticated distribution using f.

  The final approximation to p is,

  $$q(z) = q(f(\zeta)) \left| J_{f^{-1}} \right|, \tag{148}$$

  where $J_{f^{-1}}$ is the Jacobian of f^{-1},

  $$J_{f^{-1},ij} = \frac{\partial}{\partial z_j} f^{-1}(\zeta)_i. \tag{149}$$

  We can elect f to be a highly flexible function, once again using a neural network. However, in order to evaluate $\log q(z)$ when computing the ELBO and its gradient, we must be able to evaluate the Jacobian determinant $|J_{f^{-1}}|$. This imposes constraints on the neural network architecture. In particular, the final function f needs to be invertible.

  Naturally, the more "complicated" f is, the more difficult the variational optimization problem—both in terms of computational cost per step and number of steps required to find a minima (in fact, once neural networks are involved, we can expect the objective function to be non-convex and have many local minimas).

References

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

M. Betancourt. A conceptual introduction to hamiltonian monte carlo, 2017.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987. doi: 10.1016/0370-2693(87)91197-X.

C. J. Geyer. Practical markov chain monte carlo. *Statistical Science*, 7(4):473–483, 1992. doi: 10.1214/ss/1177011137.

M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011. doi: 10.1111/j.1467-9868.2010.00765.x.

L. Grinsztajn, E. Semenova, C. C. Margossian, and J. Riou. Bayesian workflow for disease transmission modeling in stan. *Statistics in Medicine*, 40(27):6209–6234, 2021. doi: https://doi.org/10.1002/sim.9164.

A. Hauser, M. J. Counotte, C. C. Margossian, G. Konstantinoudis, N. Low, C. L. Althaus, and J. Riou. Estimation of sars-cov-2 mortality during the early stages of an epidemic: a modeling study in hubei, china and six regions in europe. *PLOS Medicine*, 17(7), 2020.

F. Heurtel-Depeiges, C. C. Margossian, R. Ohana, and B. Régaldo-Saint Blancard. Listening to the noise: Blind denoising with gibbs diffusion. *International Conference on Machine Learning*, PMLR 235:18284–18304, 2024.

M. Hoffman, A. Radul, and P. Sountsov. An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo. In *International Conference on Artificial Intelligence and Statistics*, volume 130, pages 3907–3915, 2021.

M. D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.

A. Katsevich and P. Rigollet. On the approximation accuracy of gaussian variational inference. *Annals of Statistics*, 52(4):1384–1409, 2024.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR) 2014*, 2014.

C. Margossian and L. K. Saul. Variational inference in location-scale families: Exact recovery of the mean and correlation matrix. In *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pages 3466–3474. PMLR, 2025.

C. C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019. doi: 10.1002/widm.1305.

C. C. Margossian and D. M. Blei. Amortized variational inference: When and why? In *Uncertainty in Artificial Intelligence*, volume 244 of *Proceedings of Machine Learning Research*, pages 2434–2449, 2024.

C. C. Margossian and A. Gelman. For how many iterations should we run Markov chain Monte Carlo? 2024.

C. C. Margossian and L. K. Saul. The shrinkage-delinkage trade-off: An analysis of factorized gaussian approximations for variational inference. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, volume 216 of *Proceedings of Machine Learning Research*, pages 1358–1367. PMLR, 2023.

R. M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993. URL `https://www.cs.toronto.edu/~radford/review.abstract.html`.

R. M. Neal. Mcmc using hamiltonian dynamics. In S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman and Hall/CRC, 2011.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*, 2014. arXiv:1401.4082.

G. O. Roberts and J. S. Rosenthal. General state space markov chains and mcmc algorithms. *Probability Surveys*, 1:20–71, 2004. ISSN 1549-5787. doi: 10.1214/154957804100000024.

G. O. Roberts, A. Gelman, and W. R. Gilks. Weak convergence and optimal scaling of random walk metropolis algorithms. *The Annals of Applied Probability*, 7(1):110–120, 1997.

A. Vehtari, A. Gelman, and J. Gabry. Practical bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5):1413–1432, 2017.

A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner. Rank-normalization, folding, and localization: An improved rhat for assessing convergence of mcmc (with discussion). *Bayesian Analysis*, 16(2):667–718, 2021. doi: 10.1214/20-BA1221.

S. Weber, A. Gelman, D. Lee, M. Betancourt, A. Vehtari, and A. Racine-Poon. Bayesian aggregation of average data: An application in drug development. *Annals of Applied Statistics*, 12(3):1583–1604, 2018. doi: 10.1214/17-AOAS1122.

Y. Yao, A. Vehtari, D. Simpson, and A. Gelman. Yes, but did it work?: Evaluating variational inference. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5581–5590, 2018.

L. Zhang, B. Carpenter, A. Gelman, and A. Vehtari. Pathfinder: Parallel quasi-newton variational inference. *Journal of Machine Learning Research*, 23: 1–49, 2022. URL `http://jmlr.org/papers/v23/21-0889.html`.