Bayesian Inference with Stan for Hybrid Species Distribution Modeling

Isaac William Caruso
Advisor: Prof. Matteo Riondato

April 26, 2021

Submitted to the Department of Computer Science of Amherst College
in partial fulfillment of the requirements for the degree of
Bachelor of Arts with Distinction

Contents

Abstract

Bayesian inference is a statistical technique for estimating a quantity of interest from data. Increasingly, Bayesian inference is being applied to problems in a variety of fields of study as it allows for the explicit incorporation of *a priori*, *domain-specific* knowledge into the learning process. While generally appealing for numerous reasons, it has until recently been an impractical tool for learning the parameters of complex, real-world systems. In this work, we first provide the theoretical concepts necessary for appreciating Bayesian inference. We then describe algorithms for performing Bayesian inference with computing machines in a computationally feasible manner, namely Metropolis-Hastings Markov Chain Monte Carlo (MCMC). This is then followed by a discussion of Stan, a domain-specific programming language often used for MCMC and Bayesian inference. Finally, we conclude this work with a case study from a recent publication in the field of biology, which applies Bayesian inference to the task of species distribution modeling. Not only is this task extremely pertinent as it relates directly to climate change, but also the resulting model was sufficiently complex to test the full capabilities of the Stan language.

Acknowledgments

I would like to thank all of the Amherst College Computer Science Department—especially my thesis advisor Professor Matteo Riondato—for facilitating my academic growth over the past four years and in the endeavor of completing this thesis. Moreover, I thank Professor Lauren Buckley from the University of Washington Department of Biology for graciously providing her expertise and guidance in Biology and Functional Ecology. I also am excited to acknowledge my parents for their continued support over the entirety of my educational career and my friends, who provided an extensive network of support during my time at Amherst.

Introduction

As data collection becomes increasingly commonplace, extracting meaning from large datasets is essential for drawing insights and predictions. Understanding how a large number of factors influence a dataset is a topic of certain relevance for almost every scientific domain. In the field of biology, as we expand upon in the fourth chapter, understanding where animal species occur in the wild and how these ranges are shifting over time due to environmental factors is of critical importance for conservation efforts to have the greatest possible effect. Conservation biologists and ecologists have been studying these species' tolerances to the environment for quite some time, so we understand how different properties of the environment influence the occurrence of species we are interested in. Additionally, we are not only interested in where these species have been observed, but also where they have *not* been observed. The fields of statistics and machine learning provides the computational tools necessary for addressing this inquiry and many others. Given existing data, we learn a *model* which incorporates information we already have and the data we have available to us to predict what a situation for which we do not yet have the data will look like. As we elaborate upon in the first chapter of this work, Bayesian approaches to learning allow us to incorporate *a priori* domain-specific knowledge in a manner that is both explicit and intuitive. The second chapter discusses *Markov Chain Monte Carlo* (MCMC) and related algorithmic developments that make Bayesian inference feasible and scalable. Chapter three presents a discussion of Stan, a software project aimed at making Bayesian inference fast and accessible. Since at the outset Bayesian inference may appear quie complex, we aim to provide a gradual introduction first to probability and the concepts necessary to comprehend Bayesian statistics and secondly to how performing Bayesian inference is possible using modern computational techniques.

Chapter 1

Probability primer

This chapter provides an introduction to the probability concepts necessary to understand Bayesian inference. Simply put, Bayesian inference is a statistical technique for estimating a quantity of interest upon the observation of data, while explicitly incorporating *prior knowledge or belief* about that quantity of interest. Before embarking on an exposition of Bayesian statistics, we must first gain a basic understanding of a few key elements of probability theory—random variables, cumulative distribution functions, probability functions/ distributions, expected value, and conditional probability. This chapter concludes by introducing Bayes' theorem and Bayesian inference.

1.1 Random Variables

Imagine for a moment you are tossing a fair coin. There are many experiments you could perform by tossing a coin, but let us consider our quantity of interest to be the fraction of times our coin lands on a tail. It is clear that the number of tails, the outcome of our experiment, is dependent on the eventual realization of some random process. A *random variable* X is a variable whose value is dependent on the outcome(s) of a stochastic phenomenon. The realized value of X is denoted as x.

In the example of tossing a coin, where the data is a sequence of coin tosses, e.g., $[H, T, T, ...]$, we define the random variable X to be the number of tails. If we toss the coin twice, X has three possible realized states, x, depending on the outcome of this stochastic experiment: $x = 0$, $x = 1$, or $x = 2$. Table 1.1 shows the probability that our random variable X takes value x, some actual number of tails.

Table 1.1: P(X = x) for two tosses

x	P(X = x)
0	0.25
1	0.50
2	0.25

The r.v. X is is an example of a *discrete random variable.* Discrete random variables can only assume discrete values. On the contrary, continuous random variables are useful for describing continuous sample spaces. For example, a continuous random variable may be used to represent the outcome of an experiment measuring blossoming heights of flowers, where the data is a sequence of observations of heights at which different flowers blossomed. In this case, the outcome of our blossoming experiment can be any of an infinite number of real values, thus is properly modeled by a continuous random variable.

1.2 Cumulative Distribution Functions

In the previous example, we represented the probability of various outcomes of a coin toss experiment in a tabular format. Another way to represent this set of probabilities is as a *cumulative distribution function* (Wasserman, 2004, Sec 2.2).

Definition 1.1 (Cumulative distribution function 'CDF')**.** *The cumulative distribution function is defined as the function*

$$F_X(x) \doteq P(X \leq x)$$

The cumulative distribution function $F_X(x)$ simply represents the probability that a random variable X takes a value less than or equal to x for each possible input value of x. Figure 1.1 depicts a graphical representation of the CDF for our coin-tossing experiment.

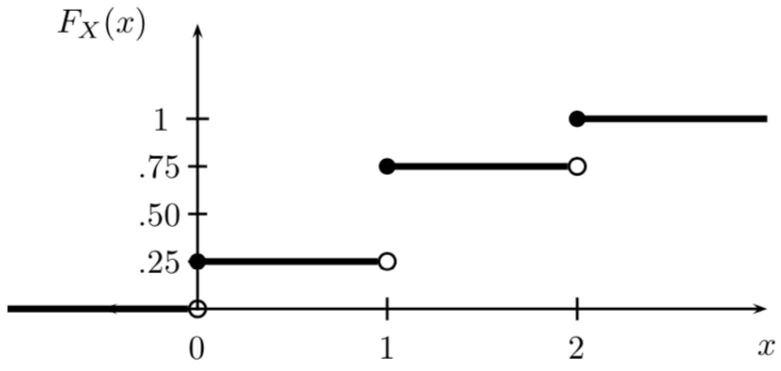

Figure 1.1: CDF for tossing a coin twice (Wasserman, 2004, Sec 2.2)

Here, for every value of x—the number of tails in two coin tosses—the probability that X is equal to or less than this value is represented. In this discrete example, we see that our CDF is represented by several non-decreasing discrete lines defined for all x. In the example of a continuous random variable, this function is a non-decreasing distribution also defined for all x.

1.3 Probability Mass and Density Functions

The *probability mass function* (PMF) and *probability density function* (PDF) allow us to express probabilities of events over a sample space and find their use with discrete and continuous random variables respectively.

In the discrete setting, the probability mass function for a random variable X yields the probability that X takes a value x for every possible value that X can take (Wasserman, 2004, Sec 2.2).

Definition 1.2 (Probability mass function 'PMF')**.** *The probability function for a discrete* *random variable X—known as the* probability mass function* for X—is defined as the function*

$$f_X(x) \doteq P(X = x)$$

Here, the PMF has a few key attributes. Namely $P(X = x) > 0$ for every x in the sample space S_X of X, and $\sum_{x \in S_X} f_X(x) = 1$. With these features in mind, the probability mass function of X follows logically from the cumulative distribution function of X insofar as the CDF is the sum of the PMF for all $x_i \leq x$ (Wasserman, 2004, Sec 2.2), i.e.,

$$F_X(x) \doteq P(X \leq x) = \sum_{x_i \leq x} f_X(x_i).$$

In the case where the random variable X is continuous, its probability density function is defined as follows (Wasserman, 2004, Sec 2.2).

Definition 1.3 (Probability density function 'PDF')**.** *The probability function for a continuous random variable X—known as the* probability density function* for X—is defined as a function $f(x)$ where a and b are two real numbers such that $a \leq b$, so*

$$P(a < X < b) \doteq \int_a^b f_X(x)dx.$$

In other words, the probability that the realized value x of our continuous random variable X is between two numbers a and b is equal to the integral of the probability density function of x from $x = a$ to $x = b$. This formalization of the PDF $f_X(x)$ allows a natural comparison with the CDF $F_X(x)$ of X,

$$F_X(x) \doteq \int_{-\infty}^{x} f_X(x)dx.$$

Specifically, this implies that $F'_X(x) = f_X(x)$ for all differentiable points of F_X. In plain English, this signifies that the derivative of the CDF is the PDF.

1.4 Expectation

One of the final core probability concepts necessary to approach Bayesian statistics on sure footing is the idea of expectation or expected value (Wasserman, 2004, Sec 3.1).

Definition 1.4 (Expectation)**.** *The expectation or expected value of a random variable X is*

$$E(X) \doteq \begin{cases} \sum_x x f_X(x) & \text{if } X \text{ is discrete} \\ \int x f_X(x) dx & \text{if } X \text{ is continuous} \end{cases}$$

To return to a familiar example, consider a random variable X to represent the number of tails in 6 coin tosses. Figure 1.2 depicts the PMF for X using the *binomial distribution* $B(6, 0.5)$ which represents the probability of observing a specific number of successes in a success-failure experiment, where the probability of success is 0.5.

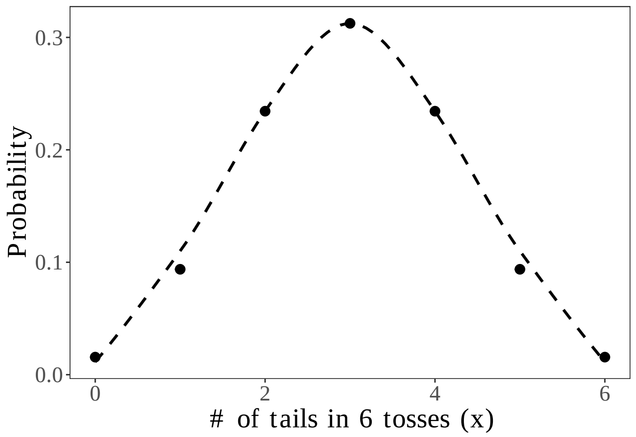

Figure 1.2: The binomial probability mass function for the 6 trial coin toss experiment

In this case, the x-axis represents each possible outcome x and the y-axis is the probability of that outcome. Table 1.2 presents the value of the binomial PMF for every $x \in X$.

Computing $E(X)$ given the values in this table is demonstrated using the discrete case of Definition 1.4 in Example 1.1.

Table 1.2: P(X = x) for six tosses

x	y
0	0.016
1	0.094
2	0.234
3	0.312
4	0.234
5	0.094
6	0.016

Example 1.1. The expected value of $X \sim B(6, 0.5)$,

$$\begin{aligned} E(X) &= \sum_x x f(x) \\ &= (0 \times 0.16) + (1 \times 0.094) + (2 \times 0.234) + (3 \times 0.312) \\ &\quad + (4 \times 0.234) + (5 \times 0.094) + (6 \times 0.016) \\ &= 3 \end{aligned}$$

Importantly while in this case, $E(X)$ corresponds well to the "peak" in the PMF, this should not be assumed to be the case unilaterally, as the same expectation would result from any distribution symmetrical about $x = 3$.

1.5 Conditional Probability

Conditional probability provides a way to model the probability that an event occurs, given that another event is known to have occurred. Conditional probability, as presented in Definition 1.5, requires only that the event assumed to have occurred, i.e., the event we are *conditioning on* has a nonzero probability of occurring (Wasserman, 2004, Sec 3.5).

Definition 1.5 (Conditional Probability)**.** *Assuming* $P(B) > 0$*,*

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Conditional probability asserts that the probability of event A occurring given that event B occurs is equivalent to the probability of both A and B occurring (denoted as $A \cap B$) divided by the probability that B occurs. $P(A|B)$ is not generally equal to $P(B|A)$. For example, the probability that I am swimming given that I am in the water is clearly not the same as the probability that I am in the water given that I am swimming. Example 1.2 explains how to use conditional probability to calculate the probability of rolling a 2 on a 6-sided dice, given I know the outcome of the roll is less than 4.

Example 1.2. Conditional probability can be used to determine the probability of rolling a 2 on a 6 sided dice, given that I know the outcome will be less than 4. This scenario can be represented as:

$$A = \text{rolls } 2,\ P(A) = 1/6$$

$$B = \text{rolls} < 4,\ P(B) = 3/6$$

$$P(A \cap B) = P(A) = 1/6$$

$$\begin{aligned} P(2| < 4) = P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{\frac{1}{6}}{\frac{3}{6}} \\ &= \frac{1}{3} \end{aligned}$$

1.6 Bayes' Theorem

Consider a student, Alice, who was exposed to someone with COVID-19. Being a responsible citizen, Alice decides she should get tested. She receives a test and the accompanying information sheet states the test is 85% accurate, meaning that 85% of the time it gives positive results to recipients who are actually positive. The sheet also says that the test yields a false positive 30% of the time, meaning that if Alice is actually negative she will still receive a positive test 30% of the time. The following day Alice receives a positive test. As a student of probability, Alice recognizes that the 85% accuracy statistic only means the conditional probability that she receives a positive test given she is COVID positive ($P(+test|covid+)$) is 85%. However, Alice is really interested in the conditional probability that she is positive given she just tested positive, $P(covid+|test+)$. As stated in the previous section's discussion of conditional probability, $P(A|B) \neq P(B|A)$. *Bayes' theorem*, which follows intuitively from the theorem of conditional probability, provides this answer for Alice. We can rewrite Definition 1.5 as

$$P(A \cap B) = P(A|B)P(B).$$

Furthermore, it can also be stated that

$$P(B \cap A) = P(B|A)P(A)$$

Clearly, $P(A \cap B) = P(B \cap A)$ as both notations can be used interchangeably to represent the intersection of two sets A and B. This identity means that we can rewrite the previous equations as

$$P(B|A)P(A) = P(A|B)P(B)$$

Moreover, dividing both sides of this equivalency by $P(A)$ yields Bayes' theorem, as formalized in Definition 1.6 (Junker, 2003).

Definition 1.6 (Bayes' Theorem). *Assuming $P(A) > 0$,*

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

The *law of total probability*, Defin-ition 1.7, can be used to compute $P(A)$ for a discrete sample space S_B (Wasserman, 2004, Sec 1.7).

Definition 1.7 (Law of Total Probability)**.** *$B_0, ..., B_k$ is a partition of a discrete sample space S_B, and*

$$P(A) = \sum_{i=1}^{k} P(A|B_i)P(B_i).$$

Returning to our example, the partition of this sample space is $[B_0 = \mathit{covid}+,\ B_1 = \mathit{covid}-]$, as Alice is either COVID positive or she is not. The final piece of information needed is the *prior* probability $P(B)$, which can be thought of as the likelihood of contracting covid from any given exposure. Alice did some research and concluded this likelihood is 20%. Given the information from the factsheet and Alice's prior knowledge about the probability of contracting covid, Example 1.3 shows how Alice can use Bayes' theorem to answer her question and find the probability that she is actually positive given she has tested positive.

Example 1.3. Given the following information:

$P(\mathit{test}+|\mathit{covid}+) = 0.85$, $P(\mathit{test}+|\mathit{covid}-) = 0.30$, $P(\mathit{covid}+) = 0.20$, $P(\mathit{covid}-) = 0.80$

We can represent the conditional probability that Alice is COVID positive given that she tested positive $P(\mathit{covid}+|\mathit{test}+)$ as

$$\begin{aligned} P(\mathit{covid}+|\mathit{test}+) &= \frac{P(\mathit{test}+|\mathit{covid}+)P(\mathit{covid}+)}{P(\mathit{test}+|\mathit{covid}-)P(\mathit{covid}-) + P(\mathit{test}+|\mathit{covid}+)P(\mathit{covid}+)} \\ &= \frac{(0.85)(0.20)}{(0.30)(0.80) + (0.85)(0.20)} \\ &= \frac{0.17}{0.24 + 0.17} \\ &= 0.41 \end{aligned}$$

1.7 Bayesian Inference

In the previous example, we used Bayes' theorem to evaluate the probability that Alice is COVID positive given she received a positive test. Bayes' theorem provided a way for Alice to apply her prior beliefs about the likelihood of actually being positive to the data she observed—a positive test. *Learning*, or *statistical inference*, asks a generalized version of Alice's question. Given some data $X_0, ..., X_i \sim D$, how can we infer the distribution D that generated the data $X_0, ..., X_i$ (Wasserman, 2004, Sec 6.1)? Additionally, as distributions are functions of parameters $\theta_0, ..., \theta_j$, estimating these parameters given the data is one avenue for inferring D. *Bayesian inference* applies Bayes' theorem to the task of statistical inference in an intuitive manner, which by definition maintains the inherent uncertainty in inference. Contrary to the frequentist perspective, a Bayesian approach to inference expresses degrees of belief by producing *posterior probability distributions* for the parameters $\theta_0, ..., \theta_j$ of the distribution (Wasserman, 2004, Sec 11.1). Point estimates and confidence/posterior intervals can then be computed with a post hoc analysis of the posterior distribution. Furthermore, a Bayesian approach to inference explicitly includes previous domain-specific knowledge (or lack thereof) in the learning process. Performing Bayesian inference to compute a posterior distribution $P(\theta|X)$ for a parameter θ first requires choosing a *prior probability distribution* $P(\theta)$ with a PDF $f(\theta)$ which expresses our prior beliefs about θ before observing any data. In the previous example, this was Alice's previous belief about the probability of contracting COVID from any given exposure. Expressing $P(\theta)$ as a distribution allows for the inclusion of the inherent uncertainty of Alice's preconceptions in the modeling schema and thus the computation of the posterior. Secondly, the likelihood function $l(X|\theta)$ must be specified to express our understanding of the probability of the data given a particular value of θ. In the case of Alice's inquiry, a binomial likelihood function would be appropriate as it represents the likelihood of observing a given number of successes (+tests) in $0 \vee 1$ $(+ \vee -)$ trials. The final step in performing Bayesian inference is repeatedly applying Bayes' theorem to update the probability distribution of θ for each observation X_n in the data $n \in (0, i)$, resulting in a posterior distribution $f(\theta | X_0, ..., X_i)$. This is done by iteratively evaluating Bayes' theorem, where the posterior for the $n - 1^{th}$ iteration is the prior for the n^{th} iteration. Bayesian inference relies on Bayes' theorem for continuous variables, which utilizes density functions as presented formally in Definition 1.8 (Junker, 2003).

Definition 1.8 (Bayes' Theorem for Continuous Variables)**.** *Assuming* $f(X) > 0$,

$$f(\theta | X) = \frac{l(X|\theta) f(\theta)}{f(X)},$$

where $f(\theta | X)$ is the posterior PDF, $f(\theta)$ is the prior PDF of θ, $l(X|\theta)$ is the likelihood function or the probability of the data given θ, and $f(X)$ is the marginal likelihood of the data X. In this case, the marginal likelihood $f(X)$ functions only to normalize the posterior distribution and is evaluated for a continuous sample space as:

$$f(X) = \int l(X|\theta) f(\theta) d\theta,$$

meaning that intuitively, the posterior is proportional to the likelihood times the prior.

While Bayesian inference is an appealing schema, learning the parameters of real-world models using this technique requires the use of efficient algorithms to be computationally feasible. As previously mentioned, depending on the system we may be interested in evaluating an arbitrary number of parameters $\theta_0, ..., \theta_j$ given the data, and each parameter may potentially take on an infinite number of values. Even in the case where we only consider a small number of possible values a for every $\theta_i \in (0, j)$, the evaluation of the posterior to perform a single update rapidly becomes a computationally intractable task. This is evidenced by the runtime of the *grid approximation* or *direct discrete approximation*, which is proportional to $O(a^j)$, where a controls the granularity of the approximation (Gelman et al., 2013, Sec 10.1). To perform Bayesian inference on complex systems, fast algorithms are necessary to circumvent this curse of dimensionality.

Chapter 2

Algorithms for Bayesian inference

When performing Bayesian inference, fast algorithms are necessary for computing the posterior probability distributions of parameters of interest in a computationally feasible manner. In modeling scenarios with multiple parameters forming a multidimensional parameter space, evaluating parameters at even a relatively small number of possible values rapidly becomes intractable, as a product of the curse of dimensionality. This problem is further exaggerated in the evaluation of hierarchical models, which are of particular interest in many application domains such as biology, economics, chemistry, and physics. Here we discuss Markov Chain Monte Carlo (MCMC) based sampling algorithms, which allow for efficiently sampling from approximations of probability distributions. MCMC provides a methodology for sampling from the posterior distribution to perform a Bayesian update. Simply put, if we make some sequence of draws $\theta_1, ..., \theta_j$ from the posterior $p(\theta|X)$, then plotting $\theta_1, ..., \theta_j$ as a histogram will provide an approximation of $p(\theta|X)$ (Wasserman, 2004, Sec 11.4). Due to the law of large numbers, this approximation can be used in place of the posterior in post hoc analysis to derive point estimates and confidence intervals. We first provide a theoretical background into Monte Carlo approximation and Markov chains, followed by a discussion of *Metropolis-Hastings*, an algorithm for performing MCMC.

2.1 Monte Carlo Approximation

Monte Carlo approximation provides a convenient method for estimating quantities of interest. This schema will allow for drawing samples from arbitrarily complex probability distributions. The basic example is *Monte Carlo integration* (Wasserman, 2004, Sec 24.2). If we want to evaluate an integral for some function $f(x)$, where

$$I = \int_a^b f(x)dx \ ,$$

we can approximate I using Monte Carlo approximation. $f(x)$ can be alternatively expressed as two functions, $h(x)$ and $w(x)$, where $h(x) = \frac{1}{b-a}$ and $w(x) = f(x)(b-a)$ as follows.

$$I = \int_a^b f(x)dx = \int_a^b w(x)h(x)dx$$

Conveniently, h is the pdf of a uniform r.v. X over $[a, b]$, which means that I can be rewritten in terms of expectation as

$$I = E_f(w(X)) \ .$$

In conjunction with the law of large numbers, this means that if we generate a sequence of random variables from a uniform distribution $X_0, ..., X_N \sim unif(a, b)$ then the standard Monte Carlo integration method asserts

$$\hat{I} = \frac{1}{N}\sum_{i=0}^{N} w(X_i) \rightarrow E(w(X)) = I \ .$$

In other words, $\hat{I}$ approaches I as N grows sufficiently large. As we will expand on in the following sections, this algorithmic framework is particularly useful because it will work for models with non-normal posteriors, including hierarchical models.

Example 2.1. Suppose we have two binomially distributed r.vs—$X \sim Binom(n, p_1)$,

$Y \sim Binom(m, p_2)$—and want to evaluate $\delta = p_2 - p_1$. This problem can be analyzed using Bayesian inference on a beta-binomial model but would require evaluating complex integrals over the joint posterior of X and Y to obtain the density. Monte Carlo Integration allows for estimating an integral using repeated sampling from the two independent posterior densities—P_1 and P_2— and plotting samples using a histogram provides a qualitative estimate for the parameter of interest, δ (Wasserman, 2004, Sec 24.2). In simplest terms, the prior distribution could be a flat prior on the parameters p_1 and p_2 (Wasserman, 2004, Sec 24.2, Bloom and Orloff (2014)). $f(p_1, p_2|X, Y) = f(p_1|X)f(p_2|Y)$, because the two posteriors $f(p_1|X)$ and $f(p_2|Y)$ are conditionally independent given the data. Thus the resultant posteriors are beta distributions, as follows (Bloom and Orloff, 2014).

$$D_1 = Beta(X + 1, n - X + 1)$$

$$D_2 = Beta(Y + 1, m - Y + 1)$$

Monte Carlo integration simulates the quantity δ by drawing $(P_1^{(1)}, P_2^{(2)}), ..., (P_1^{(N)}, P_2^{(N)})$ from the independent beta posteriors D_1 and D_2 for $i = 1, ..., N$, and every $\delta^{(i)} = P_2^{(i)} - P_1^{(i)}$. Assuming N is sufficiently large, δ is approximated using the previously established equivalency,

$$\delta \approx \frac{1}{N}\sum_{i}^{N}\delta^{(i)} = \frac{1}{N}\sum_{i}^{N}(P_2^{(i)} - P_1^{(i)})$$

(Wasserman, 2004, Sec 24.2). To provide a concrete example, if the number of trials is 20, $n = m = 20$, $X = 11$, and $Y = 4$, sampling 10000 times from the posterior in the previously described manner yields the histogram presented in Figure 2.1. We can then use the histogram to give a point estimate of, e.g., -0.32 to δ.

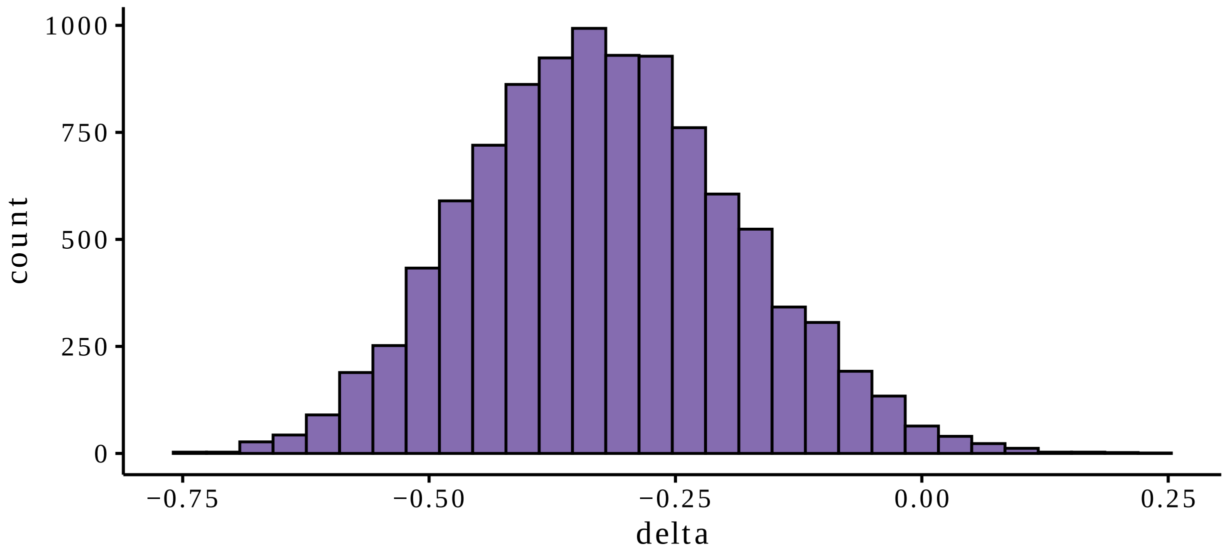

Figure 2.1: Histogram displaying posterior of delta from Monte Carlo integration

2.2 Markov chains

A Markov chain is a stochastic model expressing a sequence of possible states in which the probability of each state, X_i, depends only on the value attained in the previous state, X_{i-1}, for all $i \geq 0$. A Markov chain has several properties which are essential for its application in Bayesian statistics. Definition 2.1 formalizes the requirements for a Markov chain (Wasserman, 2004, Sec 23.2).

Definition 2.1 (Markov chain)**.** *A discrete sequence of random variables $X_0, X_1, ..., X_i$ is a Markov chain iff it satisfies the Markov property; that is, for all i and $x \in X$:*

$$P(X_i = x|X_0, ...X_{i-1}) = P(X_i = x|X_{i-1})$$

A Markov chain is often represented as a directed, weighted graph where possible states are represented as vertices, and edges represent possible transitions between states. The edge weights are the conditional probabilities of the next state of the chain being the one corresponding to the edge target vertex, given that the correct state is the one corresponding to the edge source vertex. These probabilities are also known as transition probabilities.

Markov chains also feature a probability distribution on the states at each step. The distribution on these states changes as the number of steps increases as some states become more or less likely to occur. After sufficiently many steps, the probability of being in any given state converges to the *stationary distribution*, where it remains thereafter. While not all Markov chains feature a stationary distribution, they are guaranteed to exist assuming the chain has certain properties, such as being *ergodic*. See Wasserman, 2004, Sec 23.2 for more details regarding convergence.

2.3 Metropolis-Hastings MCMC

Metropolis-Hastings MCMC is a standard algorithm for sampling from the posterior distribution to perform a Bayesian update. While it is not always the fastest approach in practice and requires manual tuning to work effectively, Metropolis-Hastings provides a foundation for more complex algorithms such as *Hamiltonian Monte Carlo* (Brooks et al., 2011) with *No-U-Turn sampling* (Homan and Gelman, 2014). To reiterate, the purpose of sampling in Bayesian inference is to draw from the posterior density to estimate $P(\theta|data)$ for parameters θ. The Bayesian posterior $P(\theta|data)$, for the purposes of MCMC, is also referred to as the *target distribution*. The basic idea of Metropolis-Hastings, to return to the previously mentioned example of approximating $I = \int h(x)f(x)dx$, is to construct a Markov chain $X_0, ...,$ with a stationary distribution of f. Assuming the sequence has sufficiently many observations, the law of large numbers applies and the following convergence holds.

$$\frac{1}{N}\sum_{i=0}^{N} h(X_i) \xrightarrow{P} E(h(X)) = I$$

The Metropolis-Hastings algorithm provides a method for sampling from the stationary distribution of a Markov chain designed in such a way that this distribution approximates the posterior distribution. Initially, the probability of realizing a state as part of the sampled sequence is skewed, but importantly, over time the sequence will converge to the stationary distribution, i.e., to the posterior. Critically, the law of large numbers guarantees that with sufficiently many iterations of Metropolis-Hastings, the generated sequence of states will converge on this stationary distribution. Essentially, given some sequence of states $X_0, X_1, ..., X_i$ from a Markov-Chain, where X_0 is chosen arbitrarily, an iteration of Metropolis-Hastings produces the next state to include in the sequence. As presented formally in Definition 2.2, this is achieved by generating a *proposal* for X_{i+1} from the *proposal distribution* $q(y|X_i)$ and then accepting the proposal with some probability dependent on the relative, target probabilities of the current state X_i and the proposal (Wasserman, 2004, Sec 24.4).

Definition 2.2 (Metropolis-Hastings MCMC)**.** *X_{i+1} is generated given $X_0, X_1, ..., X_i$ in the following manner:*

1. Sample a proposal Y from the proposal distribution $q(y|X_i)$.

2. Evaluate the acceptance probability

$$r(x, y) = min\{\frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}, 1\} \ .$$

3. Compute the next state X_{i+1}, where

$$X_{i+1} = \begin{cases} Y & \text{with probability } r \\ X_i & \text{with probability } 1 - r \end{cases} \ .$$

While Metropolis-Hastings may seem appealing because it is both memoryless and able to approximate very complex distributions, its downfall lies in the manual tuning of the *step size* parameter necessary to achieve convergence of the stationary distribution to the target distribution in a reasonable number of iterations. Recall that Metropolis-Hastings generates proposals by sampling from some distribution. In the special case of *random walk*

Metropolis-Hastings, this is a normal distribution centered about X_i. Here, the standard deviation of the normal distribution, also known as the *step size*, dictates the relative distance in the sample space between X_i and the generated proposal. If the step size is too low, the algorithm will make very small steps and may miss key features of the target distribution, causing the stationary distribution to require a much larger number of iterations to converge on the target distribution. On the other hand, if the step size is too large, proposals will be overwhelmingly generated from the low-probability tail regions of the distribution, again resulting in a lack of convergence and poorly representative samples from the stationary distribution. While some methods have been proposed for automatically tuning the step size parameter (Graves, 2011), iterating on complex, Bayesian models to tune a parameter is not particularly efficient nor computationally feasible in many cases. For this reason, other approximation algorithms are used in practical implementations of Bayesian inference. One such algorithm, *Hamiltonian Monte Carlo* (HMC), relies on theoretical physics to compute a latent momentum variable that is applied to a hamiltonian, effectively simulating a ball rolling around the multi-dimensional sample space (Brooks et al., 2011). Despite the increased computational cost, this method is appealing because of its ability to generate proposals from distant regions of the stationary distribution with high acceptance probabilities. This means that in practice, the HMC algorithm's sequence converges to the stationary distribution with far fewer iterations than traditional Metropolis-Hastings implementations. Though HMC still requires the user to specify a step size as well as a number of steps to move the hamiltonian before considering a proposal, in practice it is much more efficient and requires fewer iterations on a model. Additionally, a proposed extension to HMC called the *No-U-Turn Sampler* (NUTS) automatically determines the number of steps and was empirically demonstrated to perform at least equally as well as standard HMC (Homan and Gelman, 2014). The efficient algorithms briefly covered in this chapter underpin modern applications for Bayesian inference including Stan, which are discussed in the following chapter.

Chapter 3

Bayesian inference with Stan

Created at Columbia University by Bob Carpenter and Andrew Gelman, Stan is a domain-specific programming language that provides tools for specifying, executing, and evaluating statistical models. Currently comprised of more than 45 core developers from a wide array of educational institutions, private entities, and research organizations, over the past nine years the Stan project has grown to become one of the leading options for performing Bayesian inference computationally (Team, 2021). Among the many capabilities of the Stan language, is the ability to define priors, specify dependencies between variables, and impose known limits on parameters. After all model components have been specified, a Stan model compiles first to C++ and then to binary, which can then be executed to produce an estimation of the posterior densities of all parameters, including intermediate ones, obtained through sampling, with accompanying metadata for evaluating the sampling process.

3.1 The Stan Program

While Stan is not strictly a tool for Bayesian statistical methods, it is most often used for its efficient MCMC sampler for Bayesian inference. In essence, Stan has a sampler that takes a model and creates a separate program that explores the sample space to learn the parameters of interest in the modeling task using what is essentially the previously mentioned NUTS HMC sampling algorithm to generate draws from the posterior distribution. As it is written in C++ and thus produces binary code, a Stan program can take advantage of the full capabilities of the machine on which it is executed. Theoretically, Stan allows computer scientists and statisticians to construct, evaluate, and iterate on extremely complex, hierarchical models in one consolidated environment.

A typical Stan program is comprised of several, optional *program blocks*, which provide a general organizational structure to a Stan program and generally allow for the declaration of variables and accompanying statements about those variables. Block types include functions, data, transformed data, parameters, transformed parameters, model, and generated quantities. Each of these block types serves a different purpose in a general modeling schema; however, no block is required for the Stan compiler to execute, and an empty string is considered a valid Stan program (though it may not yield a very useful model). After the program samples the user-specified number of iterations and completes execution, a fit object and accompanying summary statistics are produced, which allow for performing post hoc evaluation such as trace and pairs plots.

Example 3.1. Bob, a marine biologist at Florida Fish and Wildlife, studies shifting patterns in encounters between humans and large marine wildlife. After seeing a local news headline about a recent string of shark attacks in his town, Bob became interested in trying to determine if the rate of shark attacks from year to year in Florida is changing. To do this, Bob decides to use Bayesian inference to compute a *Poisson regression* for a dataset of shark attack counts each year in Florida (Collier, 2018). Bayesian Poisson regression models are useful for learning parameters that take the form of a count per unit of time or space. Specifically, in a Poisson regression scenario, the data $X_0, ..., X_n$ are Poisson(λ), and the prior distribution (the conjugate prior of the Poisson) is a gamma distribution with shape α and rate β parameters (Hitchcock, 2014). The dataset includes 54 points presented below as columns, each with 4 attributes—year, population, attacks, and fatalities—depicted as rows (Simonoff, 2003).

```r
glimpse(sharks, width=79)
```

```
Rows: 54
Columns: 4
$ year       <int> 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1...
$ population <int> 2473000, 2539000, 2578000, 2668000, 2771305, 2980000, 3...
$ attacks    <int> 0, 1, 0, 0, 1, 0, 3, 1, 0, 0, 1, 5, 2, 4, 2, 4, 2, 3, 7...
$ fatalities <int> 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0...
```

Bob writes the model in Stan with three blocks—data, parameters, and model.

```stan
data {
    int<lower=0> N;               # Number of observations in dataset
    int<lower=0> attacks[N];      # An array of attack counts
    real         year[N];         # A corresponding array of years
    int<lower=0> population[N];   # Array of population for each year
}
parameters {
    real alpha;                   # Shape parameter
    real beta;                    # Rate parameter
}
model {
    beta ~ normal(0.015, 0.015); # Placing a normal prior on beta
    for (n in 1:N) # Every count modeled by a log-link with a linear combination
        attacks[n] ~ poisson_log(log(population[n]) + alpha + beta * year[n]);
}
```

Here, the data block is used to specify the type and quantity of data to be processed by the model. While N could be replaced with an actual integer, keeping fields of this type allows for models to be more flexible and require fewer changes. Additionally, the parameters block is used to declare the model's parameters, which correspond directly to the variables Stan will sample at run time (Team, 2016, Sec 8.1). Finally, the model parameters block is used to define the model, which includes the specification of any custom priors and the overall model formulae. In this case, Bob put a $normal(0.015, 0.015)$ prior on beta to include into the model his domain-specific assumption that shark attacks are occurring at increasing rates each year—specifically, that this rate is increasing at about 1.5% each year. After specifying the model, Bob uses the Stan compiler to compile his code and then executes it to sample his model, in this case the stan function from the rstan interface is used to combine these two steps.

```
sharks_data = list(
    'N' = length(sharks$population),
    'population'= sharks$population,
    'year'= sharks$year,
    'attacks' = sharks$attacks
)
shark_fit = rstan::stan(model_code = sharks_stancode,
                  cores=4, data=sharks_data,
                  chains=4, iter=2000)
```

After his model has finished sampling, Bob first takes a look at the summary statistics.

```
print(summary(shark_fit, pars=c('alpha', 'beta'))$summary, digits=2)
```

```
        mean se_mean     sd    2.5%     25%    50%     75%   97.5% n_eff Rhat
```

```
alpha -73.54 0.35745 8.2343 -91.320 -78.804 -73.17 -67.829 -58.864   531    1
beta    0.03 0.00018 0.0041   0.023   0.027   0.03   0.033   0.039   531    1
```

Bob notes that beta, the rate parameter he is interested in understanding, seems to be increasing by about 3% each year, although the 97.5% confidence interval of ~4% causes Bob to wonder how certain he should be in his conclusion about beta. To get a better understanding of the relative frequencies of different samples of the model's parameter values, Bob plots the sample values for each iteration on a histogram, as presented in Figure 3.1.

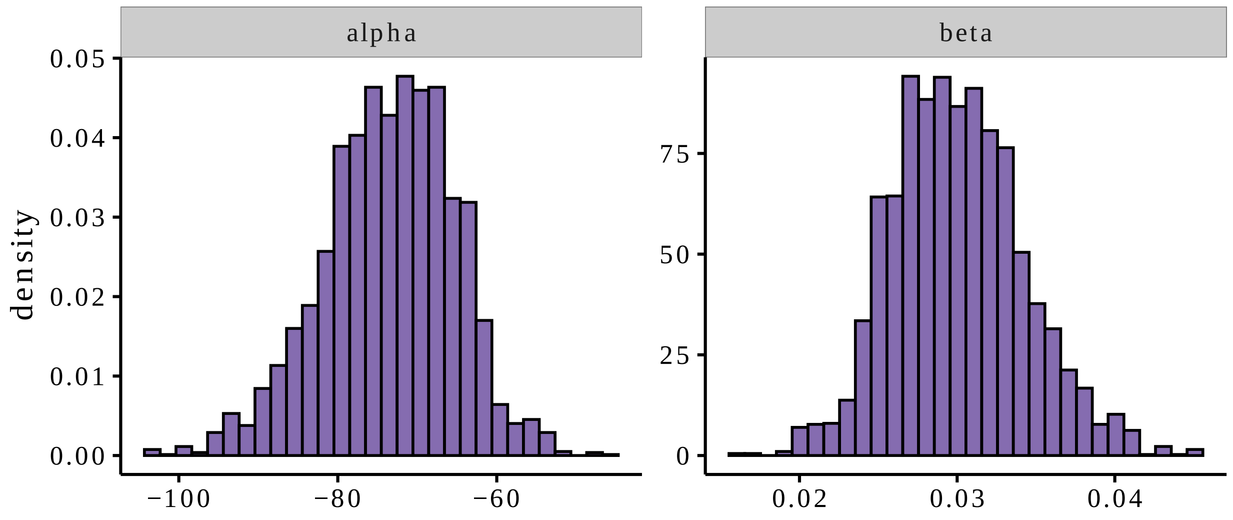

Figure 3.1: Histograms displaying frequencies of varying sampled values for parameters alpha and beta

The final analysis step for Bob's Bayesian model is ensuring the chains mixed properly, which Bob determines qualitatively using a trace plot as shown in Figure 3.2.

As it is clear the chains mixed well—each chain overlaps substantially and frequently with the other chains—Bob is now relatively confident in the results his model is producing. While he does trust the samples evaluated by his model and his analysis of Figure 3.1—that the rate of shark attacks is indeed increasing in Florida by some amount—the general lack of concentrated density in both histograms leads Bob to determine that he should collect more data and continue sampling his fit in an attempt to gain a better understanding of precisely how much shark attacks are increasing each year. Stan, and more generally Bayesian

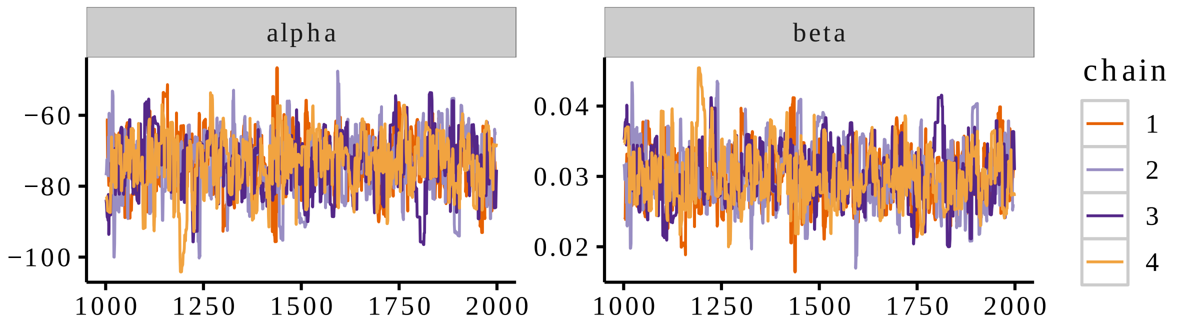

Figure 3.2: Each chain represents a sequence of states sampled from the posterior for parameters alpha and beta. Four chains were sampled for this model, and this trace plot is a qualitative method for determining if the sample space for each parameter was sufficiently explored. Here, the sampled value of each parameter (y-axis) is plotted for every iteration (x-axis) Stan evaluated. In this case, Bob specified 2000 total iterations and 1000 burn in iterations, so the first 1000 samples were discarded and not included in the posterior or the trace.

inference, is a good option for a task like this because Bob can simply collect more data and continue training this model with his new data, without the need to start the fitting process over again.

3.2 Why Stan?

According to Andrew Gelman, a well-known Bayesian statistician at Columbia University, and associates, the motivation behind developing STAN was to address several shortcomings in other technologies for performing Bayesian inference, which made these alternatives impractical for learning large, complex systems (Gelman et al., 2015). The four main shortcomings of previous software Stan's developers aimed to address were flexibility (being able to fit any given model), ease of use/ user programming time, run time, and scalability to larger datasets and more complex models (Gelman et al., 2015). Compared to two of its most direct predecessors, Jags and Bugs, STAN is generally more flexible although notably it cannot handle discrete parameters while both Jags and Bugs can. Stan was demonstrated to be both faster and scale better for complex models than Bugs and Jags, which the developers contribute to general efficiency in implementation, use of memory management, and most importantly the advanced MCMC algorithm STAN employs (Gelman et al., 2015). Stan's use of HMC (Brooks et al., 2011) and a slightly modified No-U-Turn sampler (Homan and Gelman, 2014), means that it often provides large efficiency boosts compared to conventional solutions using Metropolis-Hastings or Gibbs sampling (Bugs and Jags).

In one example, the authors demonstrated that Stan can sample a hierarchical logistic regression model of a large dataset approximately twice as fast as Jags and produce an effective sample size of nearly four times its predecessor (Gelman et al., 2015). While it is highly performant compared to previous options, Stan's overall performance is strongly dependent on the joint posterior of all parameters, and as we will elaborate upon in the following chapter can struggle to sample big datasets with numerous *Gaussian Processes* (Gelman et al., 2015).

Chapter 4

Hybrid species distribution modeling through Bayesian inference

In the field of biology and ecology, predicting the presence or absence of animal species in varying locations is a topic of much interest and critical importance for a variety of reasons. As climate change accelerates the shifting of species' ranges (Parmesan et al., 2003, Steinbauer et al. (2018)), understanding the dynamics of these shifts and predicting future ranges will aid researchers in understanding how best to preserve global biodiversity. Currently, there are two main approaches for constructing *species distribution models* (SDMs)—mechanistic models and correlative models (Buckley et al., 2010, Kotta et al. (2019)). Mechanistic models utilize explicit relationships between environmental variables and species' behaviors/ tolerances as a means for estimating species prevalence in different locations (Buckley et al., 2010, Kotta et al. (2019)). Correlative models, on the other hand, draw estimations by relating species' occurrences to environmental conditions in a particular area at the same time (Buckley et al., 2010, Kotta et al. (2019)). In their review, Buckley et al. (2010) compared predictions from several correlative and mechanistic models of the current and future distributions of two species—*Atalopedes campestris*, a skipper butterfly, and *Sceloporus undulatus*, a fence lizard. Both model types performed similarly in predicting current distributions in previously unobserved locations, though the mechanistic models generally forecasted a larger future range shift in response to climate change. While many researchers believe that correlative models have much potential for forecasting species distributions, it is becoming increasingly clear that a somewhat mechanistic approach is necessary for understanding the limits and intricacies of these ranges (Kearney, 2006). As a result, a variety of hybrid modeling schemas have been proposed for incorporating domain-specific knowledge about the degree of species' tolerances to different environmental variables into the mechanistic modeling schema (Buckley et al., 2010, Kearney (2006)).

Recently, novel methods for building hybrid SDMs using Bayesian inference with *Gaussian Processes* (GPs) have been proposed as a means for combining *a priori* biological knowledge regarding the physical tolerance limits of species to environmental conditions with distributional data aimed at elucidating the natural variability in niche-specific environmental conditions (Kotta et al., 2019). As previously mentioned, Bayesian inference allows for incorporating prior knowledge into the learning schema in an intuitive manner using informative priors. Furthermore, placing priors on Gaussian processes takes advantage of the underlying flexibility of GPs, a distribution of all possible functions over a sample space with arbitrarily many dimensions.

4.1 Experimental design

The authors used two species native to the Baltic Sea region in a case study evaluating the efficacy of Bayesian inference as a tool for distribution modeling—*Fucus vesiculosus*, a microalgae, and *Idotea balthica*, a herbivore. The authors performed this learning task in MATLAB using the GPStuff package (Vanhatalo et al., 2015), which was developed by one of the authors, Jarno Vanhatalo. We attempted to replicate the authors' findings using the Stan programming language for two main reasons—to validate their findings and to evaluate the feasibility of constructing a complex, hierarchical model, which makes heavy use of GPs, in Stan. The authors modeled the occurrence of each species with three separate methodologies, which were subsequently compared in both interpolation and extrapolation scenarios for accuracy. First, the occurrence (*biomass*) of each species was modeled using experimental data collected in a laboratory experiment wherein *F. vesiculosus* and *I. balthica* were reared in varying environmental conditions, and their survival/biomass was recorded for later use. Additionally, the biomass of each species was modeled using distributional data combined from several sources sampled from June to August from 2005 to 2015. Altogether, the distributional dataset features Biomass records from quantitative samples collected using a sieve from 6407 stations around the Baltic Sea. The corresponding data for environmental variables (temperature and salinity) was sourced from the Swedish Meteorological and Hydrological Institute (SMHI) using the echam5/RCAO model and covers the entirety of the Baltic Sea at a grid resolution of 2 NM (Meier et al., 2012). Geospatial data (depth) was ascertained from the Baltic Sea Bathymetry Database (Bat, 2021). Finally, a combined model was constructed to model both experimental and distributional data together. In each case, a *hurdle model* was used to cover both presence/absence and biomass observations. A hurdle model is a class of models where the probability of a random variable Y attaining 0 is modeled separately from the probability of attaining a non-zero value (Team, 2016, Sec 5.6). All data used in our replication of Kotta et al. (2019) was ascertained directly from Jonne Kotta and Jarno Vanhatalo and can be made available upon request.

4.2 Implementation with Stan

Though we were able to sample several models with varying levels of complexity using Stan, post-sampling analysis of all resulting models revealed each to be faulty. Here, we introduce one of the three modeling tasks Kotta et al. (2019) conducted and our implementation in BRMs, an R-based Stan interface (Bürkner, 2017). In our case study, the biomass of *F. vesiculosus* was modeled for distribution data using environmental and geospatial covariates with a hurdle model in a hierarchical schema—the model was written in multiple levels which combine to form the complete model. The data used in this modeling task has the following form:

```
Rows: 6,407
Columns: 6
$ Depth                   <dbl> 0.2, 1.0, 3.1, 5.0, 8.0, 2.0, 5.0, 5.0, 1.3,...
$ Salinity_current        <dbl> 7.372433, 7.371860, 7.370509, 7.370041, 7.36...
$ Temperature_current     <dbl> 15.29270, 15.21756, 15.17266, 15.13744, 15.1...
$ Fucus_vesiculosus_bio <dbl> 760.75, 26.21, 0.00, 0.00, 0.00, 11.49, 0.00...
$ POINT_X                 <dbl> -113627.93, -113451.65, -113221.18, -113087....
$ POINT_Y                 <dbl> 6190269, 6190859, 6190732, 6190645, 6190494,...
```

Given the data, the hurdle model for *F. vesiculosus* was formulated as

$$P(y_i|\pi_i, \mu_i, \tau_i^2) = \begin{cases} (1-\pi_i) & \text{if } y_i = 0 \\ \pi_i \times N(y_i|\mu_i, \tau_i^2) & \text{if } y_i \in R \end{cases}.$$

In this formulation, y_i is the biomass of *F. vesiculosus* for any given observation i, π_i is the probability of occurrence of *F. vesiculosus*, and μ_i and τ_i are the mean and variance of the Gaussian probability density function respectively. A *half Student-t* prior with 0 mean and scale 1 was put on the variance parameter, τ_i. Both π_i and μ_i were modeled using additive models. π_i was represented with a logit link function—logarithm of the odds ratio of occurrence probability—as follows.

$$logit^{-1}(\pi_i) = log(\frac{\pi_i}{1-\pi_i}) = \alpha + f_{ST}(x_{S,i}, x_{T,i}) + f_D(x_{D,i}) + \phi(s_i)$$

Finally, μ_i, the mean of the Gaussian PDF (biomass model), was represented similarly.

$$\mu_i = \alpha + f_{ST}(x_{S,i}, x_{T,i}) + f_D(x_{D,i}) + \phi(s_i)$$

In both cases, f_{ST} is the GP response function along temperature and salinity, and f_D is the GP response function along depth. GP priors were put on f_{ST} and f_D with 0 mean and Gaussian covariance functions. Additionally, α is the intercept, which has $N(0, 10)$ prior, and ϕ is the spatial random effect—a spatial GP with 0 mean and exponential covariance.

To formulate the model in Stan, a custom family was needed for the hurdle model. This family was implemented in BRMs using the `custom_family` function as reproduced here.

```r
hurdle_normal <- brms::custom_family(
  name = "hurdle_normal",
  dpars = c("mu", "hsigma", "hlogitpi"),
  links = c("log", "log", "identity"),
  lb = c(NA, 0, NA),
  ub = c(NA, NA, NA),
  type = "real")
stan_funs <- "real hurdle_normal_lpdf(
                real y, real mu, real hsigma, real hlogitpi) {
    real hpi = inv_logit(hlogitpi);
    if (y == 0) {
      return bernoulli_lpmf(0 | hpi);
    } else {
      return bernoulli_lpmf(1 | hpi) +
        normal_lpdf(y | mu, hsigma);}}"
stanvars <- stanvar(scode = stan_funs, block = "functions")
```

After specifying the custom `hurdle_normal` family, the model formula was provided to Stan in the following manner.

```r
formulae <-
  bf(Fucus_vesiculosus_bio ~ gp(Temperature_current, Salinity_current) +
    gp(Depth) + gp(POINT_X, POINT_Y, gr = TRUE)) +
  bf(hlogitpi ~ gp(Temperature_current, Salinity_current) + gp(Depth) +
    gp(POINT_X, POINT_Y, gr = TRUE)) +
  hurdle_normal
```

The `gp` function is used to specify a Gaussian process term, which gets evaluated with the rest of the model at runtime. Similarly, the `bf` function is used to set up a model formula for input into the model through the brms interface. Finally, the `hurdle_normal` reference is used to specify that the custom `hurdle_normal` family should be used in this model. Next, the priors for each parameter were specified. Priors mentioned thus far were taken directly from Kotta et al. (2019). The remaining priors were either inferred from the authors' code (available upon request), or they were set to the default prior for a given parameter type.

```r
priors <- c(
  prior(student_t(4, 0, 10), class = "hsigma"),
  prior(normal(0, 10), class="Intercept"),
  prior(student_t(4, 0, 1), class="sdgp",
        coef="gpTemperature_currentSalinity_current"),
  prior(lognormal(2, 0.2), class="lscale",
        coef="gpTemperature_currentSalinity_current"),
  # above values inferred from distributionData_fucus_v302.m:233-239
  prior(student_t(4, 0, 1), class="sdgp", coef="gpDepth"),
  prior(normal(5,1), class="lscale", coef="gpDepth"),
  prior(student_t(4,0,1), class="sdgp", coef="gpPOINT_XPOINT_Y"),
  prior(normal(0, 10), class = "Intercept", dpar="hlogitpi"),
  prior(student_t(4, 0, 1), class="sdgp",
        coef="gpTemperature_currentSalinity_current", dpar="hlogitpi"),
  prior(lognormal(2, 0.2), class="lscale",
        coef="gpTemperature_currentSalinity_current", dpar="hlogitpi"),
  # above values inferred from distributionData_fucus_v302.m:233-239
  prior(student_t(4, 0, 1), class="sdgp", coef="gpDepth", dpar="hlogitpi"),
  prior(normal(5,1), class="lscale", coef="gpDepth", dpar="hlogitpi"),
  prior(student_t(4,0,1), class="sdgp", coef="gpPOINT_XPOINT_Y",
        dpar="hlogitpi"))
```

Finally, the model was compiled and sampled using the `brm` function.

```
fit <- brms::brm(formula = formulae,
                 data = Final_Distribution_Dataset,
                 prior = priors,
                 family = hurdle_normal,
                 stanvars = stanvars)
```

The problems we encountered while using Stan for this task were twofold—sampling for this model took on the order of 7-14 days, and post hoc model analysis either revealed a sizable quantity of divergent transitions or that the chains had not converged. The duration of each sampling job made testing and iterating on a model design nearly impossible given the time constraints of this project. For models with divergent transitions, the proportion of samples with divergent transitions was so large (~90%) that the model was evidently not to be trusted. As the Stan manual notes, “The primary cause of divergent transitions in Euclidean HMC (other than bugs in the code) is highly varying posterior curvature, for which small step sizes are too inefficient in some regions and diverge in other regions" (Team, 2016). Though we cannot fully exclude the possibility of bugs in our model, we reviewed our code exhaustively. Essentially, divergent transitions occur because the sampler needs to find the ideal step size where the maximum tree depth is not causing the sampler to halt before making a U-turn too often but is also not so large as to cause sizable divergences in proposals, which occurs in the case of our model. The issue of chains properly mixing was discussed in Example 3.1, and an example of insufficiently mixed chains is presented in Figure 4.1.

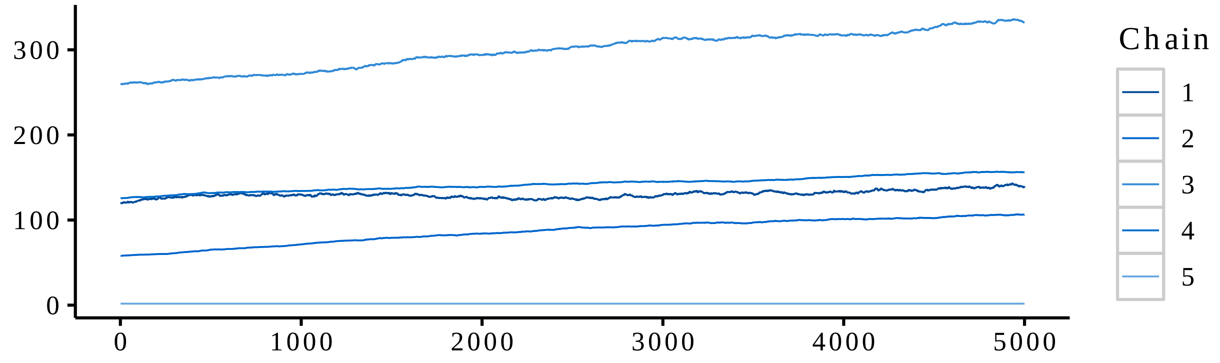

Figure 4.1: Each chain represents a sequence of states sampled from the posterior for the gp(Temperature, Salinity) parameter. The resultant density plot is also presented. The general lack of overlap/mixing in the chains indicates that this density plot, thus the model, should not be trusted.

All told, these limitations in our implementation of Kotta et al.'s (2019) model made our original goal of replicating this study in Stan a very difficult task. Validation of our modeling setup was also made more challenging due to the homegrown nature of the code the authors provided as their implementation. The several thousand lines of code written by the author of the GPStuff package were difficult for us to extract meaning from. Despite these limitations, the incredibly long sampling time required to fit each model validated the underemphasized aside in Gelman et al. (2015)—Stan struggles immensely with any sort of complex GPs. As Gelman put it, "Gaussian processes are currently more challenging. When the number of elements of the process is large... full Bayesian inference for Gaussian processes using Stan can become not just slow but effectively impossible". While Stan is currently one of the most compelling options for Bayesian inference with computing machines, the wide variation in sampling times for varying modeling schemas means it is not quite the plug-and-play solution many in the field hope it to be.

Conclusions and future work

Though we were largely unsuccessful in our numerous attempts to replicate and validate the seminal work presented in Kotta et al. (2019), the aforementioned limitations suggest a closer collaboration may be needed with the original authors and perhaps a different software for implementation. One of the most promising alternatives to Stam is Pyro, a deep universal probabilistic programming language written in Python and supported by PyTorch in the backend (Bingham et al., 2018, Phan et al. (2019)). Pyro is currently used by many large companies including Siemens, IBM, and Uber and research groups at many universities such as Harvard, MIT, Stanford, Oxford, Cambridge, and The Broad Institute. Given additional time and computing resources, using Pyro to implement a similar hierarchical model to the one presented in Kotta et al. (2019) would be a valuable extension to the work in this paper. While the potential applications of Bayesian inference in varying subject fields are numerous, the sheer quantity of statistical expertise needed to design, implement, evaluate, and iterate on models of this sort makes Bayesian inference an impractical tool for many researchers at the moment. Though domain-specific programming languages including Stan claim to lower the barrier to entry for this sort of modeling, the reality is that a sizable quantity of theoretical expertise in Bayesian statistics and practical experience with Bayesian inference is still needed to embark upon such an endeavor. While challenging to implement without a graduate degree in statistics, Bayesian inference is an extremely promising learning methodology for problems where domain-specific knowledge can provide a partial picture of the sample space. In the increasingly automated world we inhabit, tools such as Bayesian inference are necessary for providing learning systems with intuition that has been meticulously gathered by subject experts.

Bibliography

(2021). Baltic sea hydrographic commission, baltic sea bathymetry database.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2018). Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*.

Bloom, J. and Orloff, J. (2014). Introduction to probability and statistics.

Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC.

Buckley, L. B., Urban, M. C., Angilletta, M. J., Crozier, L. G., Rissler, L. J., and Sears, M. W. (2010). Can mechanism inform species' distribution models?

Bürkner, P. C. (2017). brms: An r package for bayesian multilevel models using stan. *Journal of Statistical Software*, 80:1–28.

Collier, A. (2018). Breizh data day, bayesian analysis in python: A starter kit.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. Chapman and Hall/CRC, 3 edition.

Gelman, A., Lee, D., and Guo, J. (2015). Stan: A probabilistic programming language for bayesian inference and optimization.

Graves, T. (2011). Automatic step size selection in random walk metropolis algorithms. *Statistical Sciences*.

Hitchcock, D. (2014). The gamma/poisson bayesian model.

Homan, M. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *The Journal of Machine Learning Research*, 15.

Junker, B. (2003). *Basics of Bayesian Statistics*.

Kearney, M. (2006). Habitat, environment and niche: What are we modelling?

Kotta, J., Vanhatalo, J., Jänes, H., Orav-Kotta, H., Rugiu, L., Jormalainen, V., Bobsien, I., Viitasalo, M., Virtanen, E., Sandman, A. N., Isaeus, M., Leidenberger, S., Jonsson, P. R., and Johannesson, K. (2019). Integrating experimental and distribution data to predict future species patterns. *Scientific Reports*, 9:1–14.

Meier, H. E. M., Ller-Karulis, B. M., Andersson, H. C., Dieterich, C., Eilola, K., Gustafsson, B. G., Höglund, A., Hordoir, R., Kuznetsov, I., Neumann, T., Ranjbar, Z., Savchuk, O. P., and Schimanke, S. (2012). Impact of climate change on ecological quality indicators and biogeochemical fluxes in the baltic sea: A multi-model ensemble study. *Ambio: A Journal of Environment and Society, Royal Swedish Academy of Sciences*, 41:558–573.

Parmesan, C., Yohe, G., and Andrus, J. E. (2003). A globally coherent fingerprint of climate change impacts across natural systems.

Phan, D., Pradhan, N., and Jankowiak, M. (2019). Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*.

Simonoff, J. S. (2003). *Analyzing Categorical Data*. Springer New York.

Steinbauer, M. J., Grytnes, J.-A., Jurasinski, G., Kulonen, A., Lenoir, J., Pauli, H., Rixen, C., Winkler, M., Bardy-Durchhalter, M., Barni, E., Bjorkman, A. D., Matteodo, M., Cella, U. M. D., Normand, S., Odland, A., Olsen, S. L., Palacio, S., Petey, M., Piscová, V., Sedlakova, B., Steinbauer, K., Stöckli, V., Svenning, J.-C., Teppa, G., Theurillat, J.-P., Vittoz, P., Woodin, S. J., Zimmermann, N. E., and Wipf, S. (2018). Accelerated increase in plant species richness on mountain summits is linked to warming. *Nature*, 556:231–234.

Team, S. D. (2016). Stan modeling language user's guide and reference manual.

Team, S. D. (2021). Stan development team.

Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., and Vehtari, A. (2015). Bayesian modeling with gaussian processes using the gpstuff toolbox.

Wasserman, L. (2004). *All of Statistics*. Springer.