PHYS **574**, 1–11 (2019) Preprint 6 February 2021 Compiled using MNRAS LATEX style file v3.0

Markov Chain Monte Carlo Methods

Mario Ivanov

Date: 6 February 2021

ABSTRACT

Bayesian inference is a statistical method that allows us to update our probability distributions based on what information is available to us. When applied computationally, we use Markov Chain Monte Carlo methods. Five are discussed in this paper, the first of which is Metropolis-Hastings – the most commonly used. Using representative data on the controversial prostate cancer test in medicine, we see why the random walk Metropolis-Hastings algorithm is so widely used. Four other algorithms are also discussed with their advantages, disadvantages, and when they are applicable: (i) the Gibbs algorithm, which focuses on sampling conditional probabilities rather than the proposed distribution, (ii) the Metropolis-adjusted Langevin algorithm, useful in situations where convergence to the target is slow so the algorithm needs a push in the right direction, (iii) the Hamiltonian Monte Carlo algorithm, which recreates a physical analog of the problem and solves it using energy conservation equations, and (iv) the Slice Sampling algorithm, which samples a proposed target distribution directly with the help of an auxiliary variable.

Key words: Markov Chain Monte Carlo – numerical simulations

1 INTRODUCTION

Markov Chain Monte Carlo (MCMC) methods are a class of computer algorithms that are incredibly versatile and their application knows no boundary – from finding the probability of having a certain number of people show up at a specific restaurant on a Wednesday to finding correlations and patterns between variables in equations used to describe the most chaotic places in the universe. MCMC is rooted in Bayesian inference, a branch of statistics focused on estimating parameters based on probability distribution functions. This type of statistical method allows us to update our probabilities as we gain more information about the subject matter we are studying.

In this paper, I first begin with explaining the basics of Bayesian inference and how sometimes conditional probabilities give us results that we would not intuitively expect. Then, I focus on explaining the most popular and widely used algorithm, Metropolis-Hastings, using some data that reflects current controversial research in medicine on the prostate cancer blood test. Four other algorithms are discussed later, with their advantages and disadvantages: Gibbs, Metropolis-adjusted Langevin rule, Hamiltonian, and Slice Sampling. All four have different ways of sampling probability distributions and I highlight in what cases each one ought to be used in. The algorithm that I used in the later sections is called *emcee*, written for Python, and developed by Foreman-Mackey et al. (2013).

2 BAYESIAN INFERENCE

2.1 Terminology

Let's begin our discussion of MCMC with an introduction of Bayesian inference. In essence, Bayesian inference is about using Bayes' theorem to allow us to update the probability of our hypothesis based on information we have available to us about the particular problem in question. In a more rigorous definition, Bayesian inference allows us to derive a posterior distribution function based on a prior and a likelihood function (See Joyce (2019)). Let's dissect what all of this means.

Bayes' theorem states that

$$p(B \mid A) = \frac{p(A \mid B)\, p(B)}{p(A)}\,, \tag{1}$$

or in plain English; the probability of event "B" given event "A" is true is the probability of A given B is true, multiplied by the probability of A, and divided by the probability of B. The word "given" refers to the | in the equation. These probabilities are called "conditional probabilities" – the chance of something happening if something else has already happened or is true. The terms in Bayes' theorem also have their own nomenclature, so let's define them and explain what they are. The MCMC algorithms that we will discuss later will make use of this terminology since they're all built upon the same foundation – Bayesian inference.

The term $p(B)$ from Eq. 1 is called the "prior." This is the term that would account for all background knowledge that we could have about a particular event or probability. The term $p(A \mid B)$ is called the "likelihood." The most general definition of this term is that if we have an unknown quantity B given another unknown A, it would be the probability of B given A. A more precise definition is difficult to formulate. Among statisticians, there is an epistemological debate over how inductive inference should be done using data and each major school of thought; Frequentist, Bayesian, Likelihoodist, and Akaike Information Criterion, have their own definition, as ex-

plained further by Bandyopadhyay & Forster (2011). To avoid this long-standing debate, let's keep the previous general definition of the likelihood. The next term to define is $p(A)$. This is generally referred to as a "marginal probability." With regards to a simple problem, it would have a similar definition as $p(B)$ in that it is the probability of observing A or having A be true. In more complex problems, the denominator of Bayes' theorem is generally regarded as a normalization constant for the probability $p(B \mid A)$. It is the sum of all the probabilities of all the ways that A can be obtained or can be true. The last term to define is $p(B \mid A)$. This is the posterior distribution function – a the conditional probability of our result given the prior information we know about the problem combined with the likelihood probability and normalized using the marginalized probability. Whenever we learn something new about the problem we're investigating, we update the prior and sample the likelihood to obtain a new posterior. This is the basis of Bayesian inference in a nutshell.

2.2 Doctor's visit

Now that we have the basic form of Bayes' theorem from Eq. 1 and the terminology to work with, let's go through an interesting and illustrative example. The inspiration for the following problem was from Lynch (2007), chapter 3. Suppose a man over the age of 50 visits his family physician for a yearly check-up. As per routine, for men who are over the age of 50, the physician administers a prostate cancer test that is known to be 90% accurate – that is, the test will produce a positive result for someone who presents with prostate cancer 90% of the time. The test comes out positive. The man is shocked, but he regains his composure because he knows about Bayesian statistics. He asks his doctor what the false positive rate of the test administered is and the physician tells him that it is, on average, 20%. Relieved, he calms down and the doctor explains that the man will have to undergo biopsy so they can be certain. Why was the man relieved? Well, let's find out!

According to the Canadian Cancer Society Society (2018), the incidence rate of prostate cancer for men is 110.4 for every 100,000 males. That works out to a probability of prostate cancer $p(\text{cancer}) = 0.001104 \approx 0.0011$ for men. This is actually the last key piece of information needed to calculate the probability that the man has prostate cancer. You might be wondering why we are even doing this in the first place: the man obtained a positive result for his prostate cancer test, which is 90% accurate, so that must be the chance of him having cancer, right? Interestingly enough, not exactly. The accuracy of the test is defined as the probability that the test will produce a positive result given that an individual has prostate cancer, or $p(+ \mid \text{cancer})$. What we're looking for is the probability that the man has prostate cancer given the positive result, or $p(\text{cancer} \mid +)$. The formula we should use in this case is Bayes' theorem:

$$p(\text{cancer} \mid +) = \frac{p(+ \mid \text{cancer})\, p(\text{cancer})}{p(+)} \,. \tag{2}$$

We know $p(+ \mid \text{cancer}) = 0.9$ and $p(\text{cancer}) = 0.0011$, but what is $p(+)$? As mentioned earlier, it could be defined as the sum of all the different ways that a positive test result could be obtained. In general, it is given as

$$p(A) = \sum_{B_i \in S_B} p(A \mid B_i)\, p(B_i) \,, \tag{3}$$

where S_B is the set of all possible events that could lead to a positive result. This summation can also be represented as an integral over all of B-space, if B-space is a continuous parameter space, but this example is defined as a discrete sample space, so we will use the summation. Why? Because there are only two different ways that the prostate cancer test could produce a positive result. The man either has prostate cancer and the test result is positive, which would be a true positive and we know that is $p(+ \mid \text{cancer}) = 0.9$, or the man does not have prostate cancer and the test result is positive, which would be a false positive and we now know that is $p(+ \mid \text{no cancer}) = 0.2$. Now, let's calculate $p(+)$.

$$\begin{aligned} p(+) &= \sum_{B_i \in S_B} p(+ \mid B_i)\, p(B_i) \,, \\ p(+) &= p(+ \mid \text{cancer})\, p(\text{cancer}) + p(+ \mid \text{no cancer}) p(\text{no cancer}) \,, \\ p(+) &= (0.9)(0.0011) + (0.2)(1 - 0.0011) \,, \\ p(+) &= 0.20077 \approx 0.2 \,. \end{aligned} \tag{4}$$

Therefore, the probability that the man has cancer given the positive test result is

$$\begin{aligned} p(\text{cancer} \mid +) &= \frac{p(+ \mid \text{cancer})\, p(\text{cancer})}{p(+)} \,, \\ p(\text{cancer} \mid +) &= \frac{(0.9)(0.0011)}{0.2} \,, \\ p(\text{cancer} \mid +) &= 0.00495 \approx 0.5\%. \end{aligned} \tag{5}$$

This is why the man was relieved. He quickly did the math in his head and found out that he only has a 0.5% chance of having prostate cancer given the positive test result. How is this possible though? The reason for the seemingly suspicious result is two-fold: (i) the incidence rate of prostate cancer is relatively low in the population – $p(\text{cancer}) = 0.0011$, whereas (ii) the probability of a false positive is relatively high – $p(+ \mid \text{no cancer}) = 0.2$. Another way to look at this is that if we took a population of 100,000 men, on average there would be 110 of them who have prostate cancer, but 20,000 who would have been obtained a positive test result. That means that there's only a $110/20000 = 0.0055$ probability of the test diagnosing a true positive. This is the reason why the result seems suspicious.

Now, having done the test once, let's say that the man wants to redo the test. If he performs the test again, instead of using the incidence rate of prostate cancer, he would now use his previous posterior value of $\approx 0.5\%$. This is the essence of Bayesian statistics – every time that we learn something new or we have new evidence about our event, we include it in our prior. The man tested positive for prostate cancer once, which would increase his chances of having prostate cancer compared to the general population. Let's say that he obtains a positive result again. Using the new prior, the man's probability of having prostate cancer would be

$$p(\text{cancer} \mid +) = 0.0223 \approx 2\%. \tag{6}$$

The man's probability of actually having prostate cancer has increased once again, as we would expect. Now, in the spirit of MCMC, the man could continue to take the test to see what result he obtains. By chance, the test will sometimes come out negative, so if the man takes the test enough times, assuming his physician is also very patient, he would eventually approach the true probability

of having cancer. He could also use a different prior for his probability of cancer – say if the doctor palpated a tumor. That would greatly increase his chances of having prostate cancer and we could include that in the prior. Though, in the first method, the process of repeating the test and recomputing the posterior probability is the what our MCMC algorithms do. From a Bayesian perspective, we begin with some prior probability for some event, and we update this prior probability with new information to obtain a posterior probability. The posterior is then used as the new prior in future tests. Rather than beginning each new hypothesis from a new and ignorant perspective, we rely on previous *a priori* information to test the hypothesis we're exploring.

3 A PROBLEM IN MEDICINE

The prostate cancer test is a blood test for the Prostate-Specific Antigen (PSA) as a concentration in ng/mL. PSA is a protein produced by the prostate gland, in both normal and malignant cells. Since its discovery by Wang et al. (1979), it has been an active tool for diagnosing prostate cancer based on its concentration in the body. (See Catalona et al. (1991) and Brawer et al. (1992)) Until 2008, most doctors encouraged yearly PSA screenings for men age 50+, until more was discovered about the dangers of overdiagnosing. Ever since, a number of organizations began to caution against routine screening. Currently, there is somewhat of a consensus that a PSA level of 4.0 ng/mL is predictive for more testing and biopsy, while a level of 10.0 ng/mL is predictive of a >50% chance of prostate cancer, as seen in Fig. 1 (See Barry (2001) and Thompson et al. (2004)).

The reason I'm currently focusing on prostate cancer is twofold: (i) to show the versatility and applicability of MCMC methods and (ii) my grandfather unfortunately passed away due to prostate cancer – so it's a topic that I wanted to explore a bit more in depth. In regards to the first point, to showcase the power of MCMC and its most commonly used algorithm, Metropolis-Hastings, I initially decided that I was going to find any data that is available from researchers who have worked on the topic of prostate cancer. After rifling through the entangled web of almost-fully inaccessible research that exists on MEDLINE and PubMED, I eventually found some research that I could gain access to, some of which I have already referenced, but none of the authors had published the data they worked with. So, to remedy the situation, I took the currently accepted literature numbers that exist on what researchers think the probability of prostate cancer is based on the concentration of PSA and I created my own set of data.

TABLE 1. ESTIMATED PROBABILITY OF PROSTATE CANCER IN MEN WITH NORMAL FINDINGS ON DIGITAL RECTAL EXAMINATION, ACCORDING TO THE PROSTATE-SPECIFIC ANTIGEN LEVEL.*

PROSTATE-SPECIFIC ANTIGEN LEVEL	PROBABILITY OF PROSTATE CANCER
ng/ml	%
0–2.4	Uncertain
2.5–4.0	12–23†
4.1–10.0	25
>10.0	>50

*Data are from Catalona et al.,[11,12] Lodding et al.,[13] Djavan et al.,[14] Babaian,[15] and Babaian et al.[16]

†This range is derived from three studies[11,13,16] that reported probabilities of 12, 22, and 23 percent; the lowest estimate is from a population-based study.[13]

Figure 1. Table from Barry (2001). References used to generate the table are mentioned therein.

3.1 The Data

To describe the probability of prostate cancer given the concentration of PSA, the model that seemed to be used the most, and the model that would fit our problem the best, was the sigmoid function model. A general form of a sigmoid function is given by

$$y = \frac{1}{1 + e^{-(mx+b)}}\,, \tag{7}$$

where m and b are variables that are allowed to vary. To recreate the literature numbers seen in Fig. 1, I used the following numbers in the sigmoid function: $m_{\text{true}} = 0.5$, and $b_{\text{true}} = -5$. Then, to test how well the MCMC algorithm can deal with large uncertainties, I added random errors to the probability of prostate cancer. In Fig. 2, the x-axis is the concentration of PSA in ng/mL, the y-axis is the probability of prostate cancer, and δy are the errors in the probability of prostate cancer. The y and δy were generated in the following way:

(i) Generate the true fit, y, from m_{true}, and b_{true};
(ii) Generate an array of random errors for δy of up to 20%;
(iii) Add a fractional amount of random errors, $f_{\text{true}} = 0.5$, to y.

So in summary, the y values were moved up to 50% away from their initial value using f_{true}, and random errors of up to 20% were generated for δy. After all these modifications, the data set in its final form appears as shown in Fig. 2. After introducing a lot of error into the data, some points fell below 0% and many points above 100% with respect to the probability of cancer, so for correctness' sake I renamed the y-axis to be the "relative probability of prostate cancer." The three important values that we want to recover accurately from our MCMC test will be $m_{\text{true}} = 0.5$, $b_{true} = -5$, and $f_{true} = 0.5$.

3.2 Metropolis-Hastings Algorithm

The most commonly used algorithm for MCMC is the Metropolis-Hastings algorithm. A good general review of samplers like this are written in Goodman & Weare (2010). It is, in its essence, an algorithm that explores our parameter space randomly, with a preference towards exploring higher values of probability distributions. Before we continue, we need to modify Bayes' theorem slightly since now we're going to be working with data and a continuous parameter space, rather than individual events like before. Generally, Bayes' theorem for a continuous parameter space is written as

$$p(m, b, f \mid x, y, \delta y) \propto p(m, b, f)\, p(y \mid x, \delta y, m, b, f)\,, \tag{8}$$

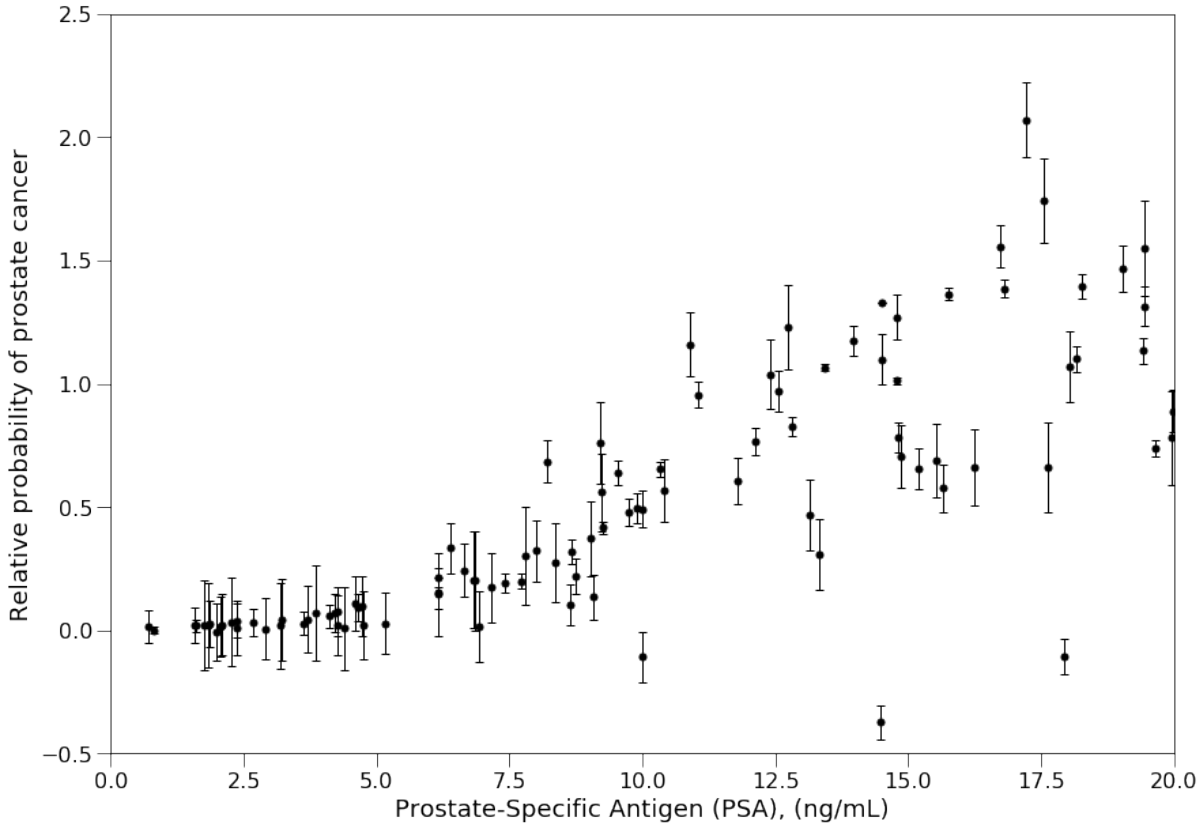

Figure 2. Data set generated based upon the current literature numbers of the relative probability of prostate cancer, y-axis, given the concentration of PSA in ng/mL, x-axis.

where m, b, and f are the variables or parameters we would like to explore using MCMC, x is the independent data, y is the dependent data, and δy is the error in y. The first term, $p(m, b, f \mid x, y, \delta y)$, is the posterior, $p(y \mid x, \delta y, m, b, f)$ is the likelihood and $p(m, b, f)$, is the prior. You might be wondering where the marginal probability in the denominator is. With regards to continuous parameter space, that term is generally either really difficult to figure out or frankly impossible. Hence, we can define our posterior distribution function to be “proportional" to the prior multiplied by the likelihood. The marginal probability in the denominator of Eq. 1 acts as a normalizing factor in this new modified version of Bayes’ theorem so that our posterior is calculated as a proper probability distribution. However, if we take the ratio of two posteriors, which is what we will be doing, then the denominators are not required. That’s why we can get away with not using that term in our definition.

Figure 3. A visualization for the set-up of the Metropolis-Hastings algorithm. Here we have a target distribution that we would like the algorithm to reach. x_t is the initial sample state and x' is the new sample state.

One of the calculations done in the actual coding aspect of running the algorithm is that we want to calculate the natural logarithm of the prior and the likelihood function. While the prior is easy to implement as it is a constraint on the variables, the likelihood function is a bit different. For our purposes, the likelihood function is in the form of a gaussian. The reason is that the likelihood is what is used by the algorithm to decide how to take steps. If we define a gaussian, then the algorithm has a chance to take a random step in either direction. A way to imagine it is for each point sampled in Fig. 3, a gaussian is generated around that sample point and the next sample point is taken at some distance away, related to the standard deviation of the gaussian. If we take the natural logarithm of the likelihood, we obtain

$$\ln p(y \mid x, \delta y, m, b, f) = -\frac{1}{2}\sum_n \left[\frac{(y_n - \text{model}(x_n, m, b))^2}{s_n^2} + \ln(2\pi s_n^2)\right], \quad (9)$$

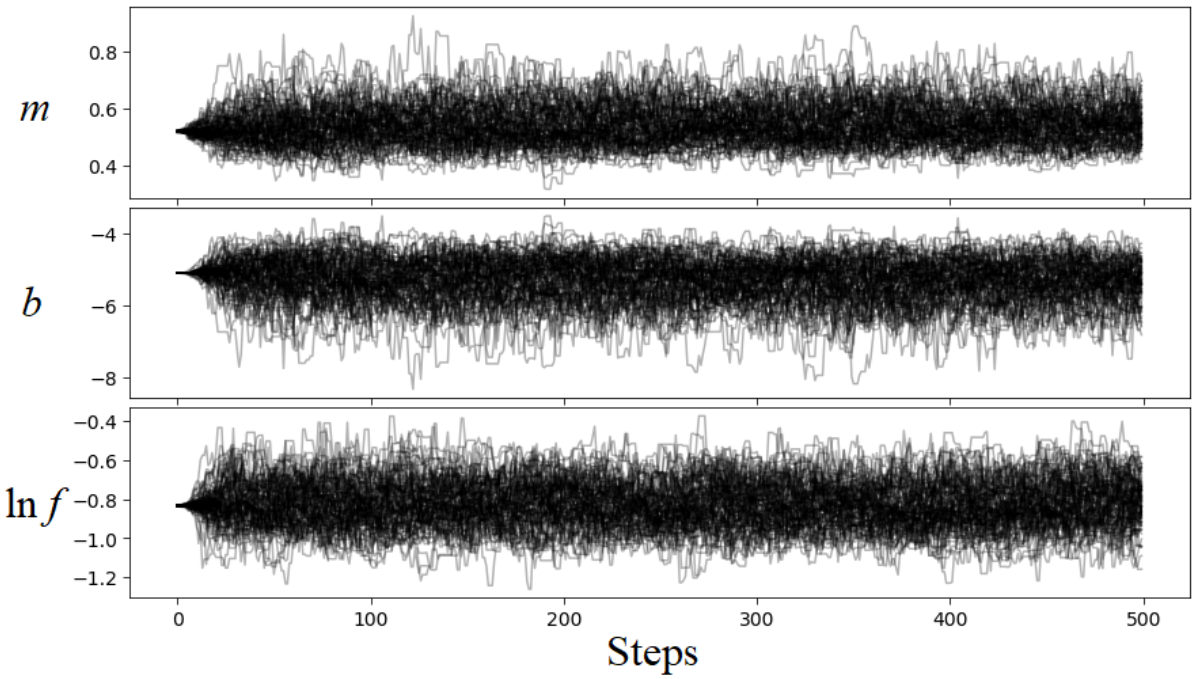

Figure 4. The position of each walker in each parameter space is mapped against the number of steps in the overall chain. The walkers have all fully explored their respective parameter spaces by approximately step number 30.

where the model is given by Eq. 7, and

$$s_n^2 = \delta y_n^2 + f^2(mx_n + b)^2 \ . \tag{10}$$

The variable f that we're including in our MCMC analysis is a fractional amount of error added to the model at its value. This would, in theory, represent a bias in our sample that would grow larger with increasing certainty of the probability of prostate cancer. In theory, this should also be uncorrelated from the other two variables, m and b, since they're not associated with how much random error is added to the model.

Let's now begin with Metropolis-Hastings. First, we draw a sample state from our prior, x_t, where $x_t = \{m^0, b^0, f^0\}$. What this means is that initially, for each variable, the algorithm draws a random variable from the probability distribution of the prior based on the prior information, or constraints, we have on the variables and the likelihood. Now, let's say that there is a target distribution function that we would like to find. This target distribution would be related to the data, generally speaking, and would be ideally what the MCMC algorithm converges to as the highest probability distribution given a certain range of parameters. After our first calculation of the posterior, we can then sample that posterior distribution function for new sample states. The next sample state, let's call it x', would have samples values for the parameters from the updated prior, which would be our previous posterior. As you can see, this is analogous to the earlier example of the man retaking the prostate cancer test and updating the prior with the new posterior. For visualization of the set-up so far, see Fig. 3.

The next step of the algorithm calculates the ratio between the posterior of the new sample state, x', to the posterior of the previous state, x_t. This ratio is called the "acceptance ratio" and it's calculated in the following way:

$$\alpha = \frac{p(x')}{p(x_t)} \ . \tag{11}$$

Following the calculation of α, the algorithm computes a random value $u \in [0, 1]$. This is where the name "acceptance ratio" comes from: the algorithm compares the ratio of the posteriors to this randomly generated number u and goes through an "accept/reject" statement that either allows the algorithm to move to this new sampled state (accept) or to deny this new sampled state and to sample another one:

- for $u \leq \alpha$, accept sample state $\longrightarrow$ $x_{t+1} = x'$,
- for $u > \alpha$, reject sample state $\longrightarrow$ $x_{t+1} = x_t$.

Each iteration of this algorithm is called a "step." The reason for that is that the algorithm begins in a random location in parameter space based on the prior and likelihood. It then "steps" in a random direction, which is what the acceptance of the new sample state is. Due to this behaviour, each instance of the algorithm is colloquially referred to as a "walker." These infinitesimal walkers wander randomly in parameter space, sampling the proposed distribution for new sample states, accepting and rejecting steps based on these two rules. Now, if you remind yourself of Fig. 3, what you'll notice is that what I proposed was that the new sample state x' has a higher posterior value than the posterior of the previous state x_t. That would make $\alpha > 1$ and since $u \in [0, 1]$, $\alpha > u$ *always*. In this case, the algorithm will *always* accept sample states that have a higher posterior value, or higher up the target distribution, if you will. In contrast, if we take a step in the opposite direction, then our $\alpha \in [0, 1]$. In this case, the chance of accepting or rejecting the step depends on how small the acceptance ratio is and what the random number generated for u is. If the walkers are allowed to take a lot of steps, the algorithm will preferentially keep them in areas close

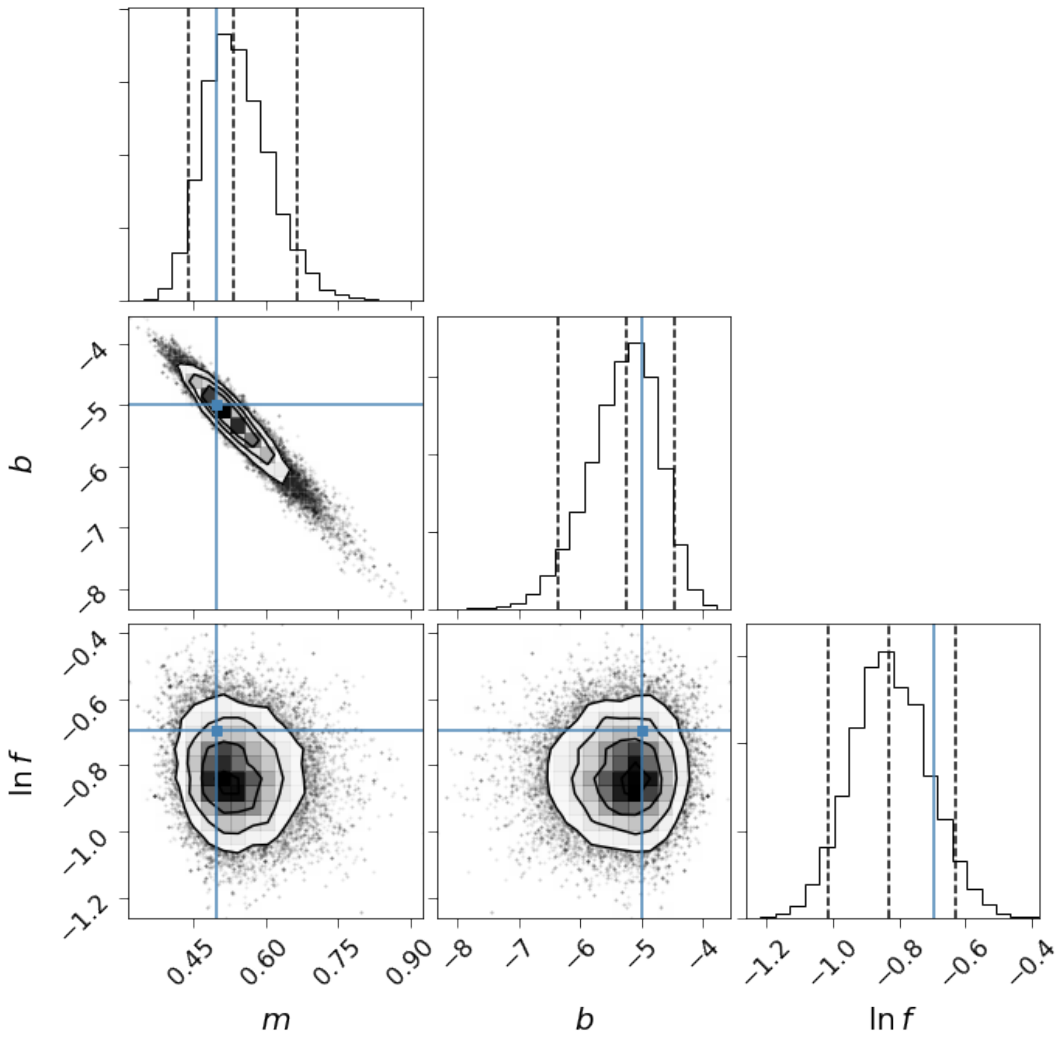

Figure 5. A summary of all the walkers from the Metropolis-Hastings algorithm. The “diagonal" with the three histograms is generated by taking binning the walk pattern of all the walkers, with each variable defined on both the x- and y-axis. The blue lines represent the true values defined earlier in the problem. The vertical black dashed lines are the following quantiles: 5%, 50%, and 95%, i.e. the confidence interval probability of the outer black dashed lines is 90%, which also corresponds to a regular standard deviation $\sigma = 1.64$. The plots below the histogram diagonal show the covariances between the variances.

to the target distribution while only sometimes letting them explore lower probability areas of parameter space.

The benefit of this sampling method is that while walkers tend to congregate in regions of high probability, they are in essence walking randomly in the parameter space. There is a check that we can do to make sure that the walkers have converged properly onto what they believe is the target distribution. Fig. 4 shows the position of each walker with respect to the number of steps it has taken. The walkers start in a small distribution around some initial values either randomly sampled from the prior and likelihood, or as initial guesses manually inputted into the algorithm. The walkers then wander and explore the full parameter space. There is a phase called the “burn-in" phase, which is difficult to define quantitatively, but it is a period of time from when the walkers begin to walk to when their exploration of their parameter space looks more or less uniform. We can see that in Fig. 4, by about step number 30, it seems all three parameters have fully explored their parameter space and are now wandering within those distributions.

One of the disadvantages of using the Metropolis-Hastings algorithm is that it is highly dependent on the step-size. While it might be easy to change the step sizes, setting the step-size too large would result in too many rejections of new sample states, while setting the step-size too small would lead to a poor (or slow) exploration of the parameter space. That considered, the reason that Metropolis-Hastings is the most popular algorithm to use is that it is simple to implement and is great for sampling from correlated parameter space.

3.3 THE MCMC SOLUTION

Since the walkers within the first 30 steps are biased and not representative of the full parameter space, we can remove the points up until then from each of the walkers in Fig. 4. After these steps are removed, we can bin the remaining, now homogeneous, data from all the walkers into one summary plot called a “corner" plot. Developed by Foreman-Mackey (2016), the corner plot is a beautiful visualization of multidimensional analyses in which the multiple projections of covariances between variables are displayed. Not only

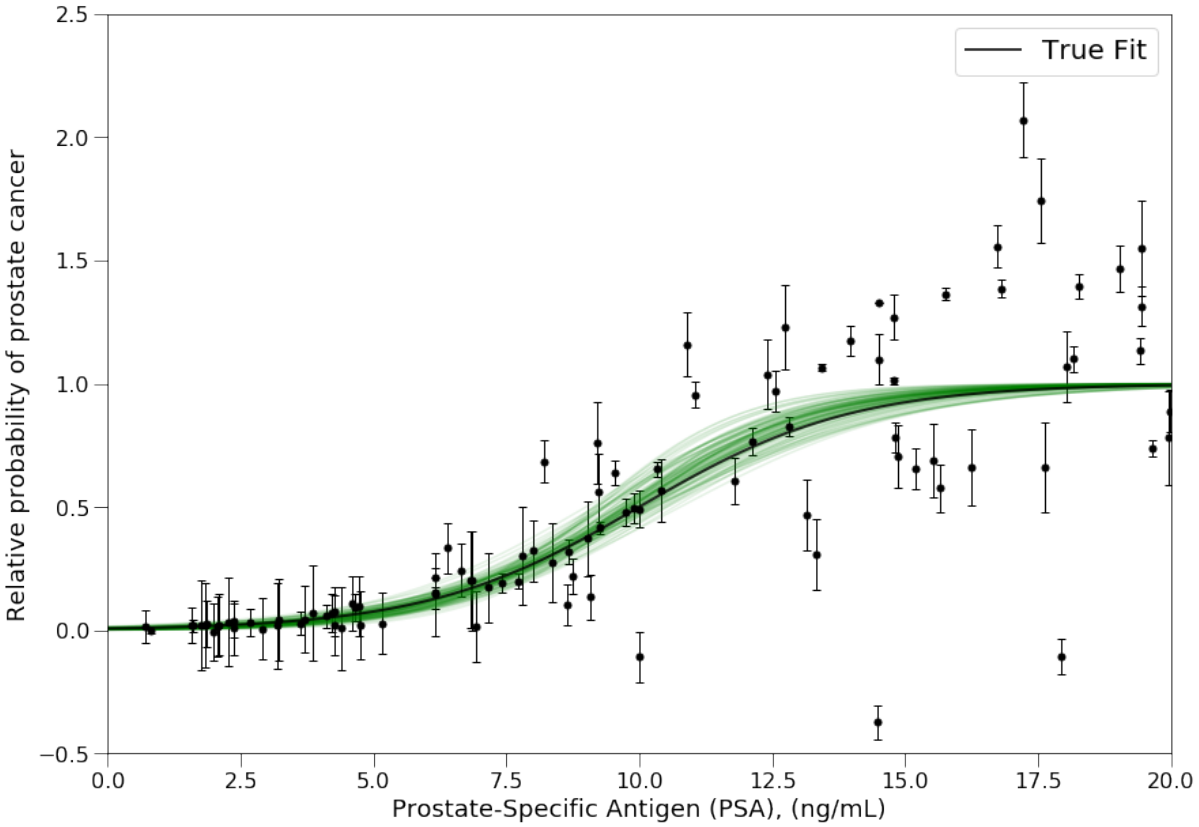

Figure 6. The data set is plotted with the true fit, black line, defined by our m_{true} and b_{true}, as well as with 100 randomly sampled fits, green lines, using the binned values of m and b from the MCMC algorithm.

that, but the histograms with respect to each variable is displayed no top of the covariances, so plot acts like a table for variables with each entry being either a covariance between two variables, or a histogram if the variables match up. I produced a corner plot with respect to the prostate cancer data in Fig. 5. The reason I chose to use the 5%, 50%, and 95% quantiles is mostly because I wanted to stick true to the literature I was using. It's uncommon to use those limits in Physics, but they seem to be the most common numbers when describing statistics for at least this specific type of problem in Medicine. There are two things I'd like to draw attention to from the corner plot: (i) there is a highly negative correlation between the variables m and b, and the reason for that is because if we want our model from Eq. 7 to be low for low concentrations of PSA and vice versa, then one of either m or b has to be negative, and (ii) there is no correlation between the fractional random error amount added to the y-data and the variables themselves, as expected. The great thing about these covariance plots is that we can not only test to see what is the best value of each variable with respect to the target distribution, but we can also see how correlated each variable is to every other one, which might reveal degeneracies in the model.

Finally, we can see how well the Metropolis-Hastings fared by sampling our binned values of m and b in the histograms and generating fits based on them. In Fig. 6, I've plotted 100 randomly sampled fits from the values we obtained from the Metropolis-Hastings algorithm in green and compared them to the true fit used to generate the data. Despite the large errors randomly added to the data to try and confuse the algorithm, we can see that the fit was reproduced within reasonable error. The quantitative results are shown in table below.

	m	b	f
True values	0.5	-5	0.5
MCMC values	$0.54^{+0.13}_{-0.09}$	$-5.25^{+0.80}_{-1.10}$	$0.44^{+0.10}_{-0.07}$

Now that we have explored the Metropolis-Hastings algorithm in detail, discussed how it works, and seen how it fares against data with moderate amounts of error, let's discuss what other algorithms exist and why.

4 MCMC METHODS

With the most common method already covered with an example, let's explore some of the other algorithms that exist and why we would need to use other methods.

4.1 Gibbs Algorithm

In the prostate cancer example, we had a model defined for what our target distribution *ought* to look like. The Metropolis-Hastings algorithm used that piece of information to create and update a posterior from which it could continue to draw random states. Unfortunately, there are some cases in which it may be difficult to directly sample the posterior. In those cases, it would be better to use the Gibbs algorithm. The number of dimensions of a problem, or rather the number of parameters explored by MCMC, highly impact the "accept/reject" statement in the algorithm, as well as target

distribution sampling. Both of those become exponentially more difficult, especially if the target distribution is multi-modal.

Gibbs sampling was named after the physicist Josiah Gibbs, described first by Geman & Geman (1984). This algorithm, in the most basic form, is a subset of Metropolis-Hastings, however one key difference is that this algorithm requires the ability to be able to sample the conditional distributions of each variable directly, rather than a single proposal distribution for each variable. Given that this is possible, the algorithm begins by drawing a sample state $x_0 = \{m^0, b^0, f^0\}$ from a proposed distribution $p(m, b, f)$, be it the prior multiplied by the likelihood, or a manually inputted initial guess. To calculate the next sample state, we follow the steps below:

$$\begin{aligned} &\text{for } t \text{ in } [1 : \text{steps}] : \\ &\quad m^t \sim p(m \mid b^{t-1}, f^{t-1}) \\ &\quad b^t \sim p(b \mid m^t, f^{t-1}) \\ &\quad f^t \sim p(f \mid m^t, b^t) \end{aligned} \tag{12}$$

What the algorithm does is that for the next sample state, it will sample the conditional probability of the first variable m^t based on the two previous variables, b^{t-1} and f^{t-1}, however when the algorithm moves to update the second variable b^t, it samples the conditional probability of b given the new value m^t and the old value f^{t-1}. The algorithm completes the new sample state by sampling the conditional probability of f given both new values of m^t and b^t. The pattern then repeats, for the next sample state, $t + 1$. Within the first several steps, there would intrinsically be a bias depending in which order the variables are being updated, but as for any random sampling algorithm that are in this paper, if the sampling is done for long enough, those differences will be negligible.

Let us use χ as a general form for the variables so that χ could mean either m, b, or f from our earlier example. The advantage of using Gibbs is that if the conditional distributions $p(\chi_i \mid \chi_{j\neq i})$ are available to sample, the algorithm essentially reduces the dimensionality of the problem by using a joint distribution. This lowering of the dimensionality allows the algorithm to be more efficient, especially because there is no need the go through an accept/reject statement – all new sample states are accepted. Also, the Gibbs algorithm is often used in cases where you can't define a good proposal distribution to begin with, so you give more power to the data to dictate where the algorithm samples.

Unfortunately, a major disadvantage is that if we have highly correlated variables, then the algorithm has a hard time exploring the parameter space. The reason for that is that the algorithm cannot take "diagonal" steps. Since it samples the conditional probability of two variables, it depends on one to be fixed so it can sample the other. You can therefore imagine that if two variables are highly correlated, then their correlation will appear somewhat like the correlation between m and b in Fig. 5. If the algorithm cannot take diagonal steps in this case, then it will poorly explore the parameter space.

4.2 Metropolis-adjusted Langevin Algorithm

The Metropolis-adjusted Langevin algorithm, from here on the Langevin algorithm, was initially introduced by Roberts & Tweedie (1996). While very similar to Metropolis-Hastings, the Langevin algorithm uses a proposal distribution from which it generates a sample state, as before, but it's also used when calculating the acceptance ratio. The full Markov chain can be defined as $\{x_t,\ t \in \mathbb{N}\}$, where at every step $t \in \mathbb{N}$, where $\mathbb{N}$ is the total number of steps taken

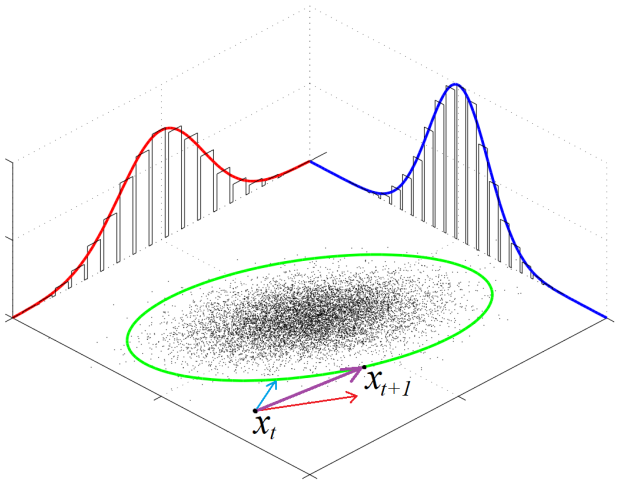

Figure 7. The Metropolis-adjusted Langevin algorithm. Starting from sample state x_t, the step taken in a random direction is colored red. The Langevin term $q(x, x')$ forces the walker in the direction of the blue arrow. The total step taken is therefore the purple arrow, labeled as x_{t+1}.

by the walkers. If x_t is given, we can generate a new candidate x'_{t+1} from a proposed density distribution function $q(x_t, ...)$. The algorithm calculates the acceptance ratio of the new sample state x' and it is

$$\alpha = \frac{p(x'_{t+1})q(x'_{t+1}, x_t)}{p(x_t)q(x_t, x'_{t+1})} \,. \tag{13}$$

The acceptance of α is based on the same reason: a random value $u \in [0, 1]$ is generated after which if $\alpha > u$, the the new sample state is accepted, and vice versa. For the Langevin algorithm, $q(x, x')$ is a normal density distribution represented as

$$N\left(x + \frac{\sigma^2}{2}\Delta \log p(x_j)\,, \sigma^2 D\right) \,, \tag{14}$$

where σ is a step-size greater than 0, and D is a $n \times n$ symmetric and real matrix. The details of how $q(x, x')$ is chosen or why it's form is the way it is is really complicated. The math and reasoning behind $q(x, x')$ was initially developed by Roberts & Tweedie (1996), and the later improved by at least both Roberts & Rosenthal (2001) and Durmus et al. (2017). Since the statistics get very complex, I won't cover them in this paper, but I can describe the result of this algorithm. The algorithm is somewhat illustrated in Fig. 7.
Allowing the factor of $q(x, x')$ to be involved in the acceptance ratio the way that it is allows for the random walk algorithm to "nudge" the random walker in the direction of where it believes the target distribution is. The reason why $q(x, x')$ "knows" the correct direction is because it is an evaluation of the gradient of the log density of the current state: $\nabla \log p(x)$. So, if it knows in which direction the gradient of the parameter space that the walker is currently in is increasing, then it can nudge the walker in that direction, since that's the direction that it *should* be going. Generally this is scaled by some factor, depending on how strong you want this "nudge" to be. This is what I attempted to illustrate in Fig. 7. While the red arrow is the random step, the blue arrow is the "nudge" in the right direction. The total step taken, therefore, is the sum of both arrows, which is closer to the target distribution than it would have been if the blue arrow was not used.

The advantage of using the Langevin algorithm is that it is well known to have better convergence onto target distributions than any of the random walk algorithms covered so far. Though the immediate disadvantage of using a term that forces the walker to move with the gradient of the log density is exactly how much power to give that term. Current research is still not in consensus about what the scaling rate of the $q(x, x')$ term should be. Initially the first somewhat agreed upon scaling rate was suggested by Kennedy & Pendleton (1991), but later on was shown to be limited. The work by Ottobre et al. (2017) has explored how to lower the convergence time by proposing different scaling rates or if any other steps should be included in the Langevin algorithm, such as "irreversible proposals." All things considered, if the problem we're investigating seems to converge very slowly, one of the algorithms that we could try is the Langevin algorithm to help nudge it in the right direction.

4.3 Hamiltonian Monte Carlo Algorithm

The Hamiltonian Monte Carlo is, in a way, like the Metropolis-Hastings algorithm, except the way that the proposed distributions are generated and sampled are much more akin to physical systems that have energy and momentum. If we compare all the previous algorithms so far, one of the things they have in common is that they all, in their own respective way, try to figure out a step-by-step method of being able to sample a proposed distribution in way that allows them to sample the "peak" of a distribution more than the rest of the parameter space. That *is* what we want – the positions where this distribution is highest in its probability. Let's now take this imaginary probability distribution and inverse it, so as to look like a "potential well", where the minimum of this potential would correspond to the maximum likelihood of the target distribution. If we now take a fictitious particle, basically a walker from previous algorithms since it's generated in a random location, and allow it to have a kinetic energy in some random direction, what will happen if this particle interacts with the well? Well, if we treat this like a physical system, then it will lose potential energy and gain kinetic energy as it falls inwards. It will feel a force that pushes it towards the minimum potential well which is supposed to be an analog to gravity. As this particle travels, every so often we can provide it with a random amount of kinetic energy in any random direction. This way, it will preferentially explore the potential well, but it will also have excess energy to explore the outer reaches every so often. This is essentially the method of the Hamiltonian algorithm.

Initially developed by Radford (1993) and then later improved by Radford (2012), the Hamiltonian is defined as the sum of the potential energy $U(x)$ and the kinetic energy $K(\mathbf{p})$, where x is a sample state, or a random position of the particle in parameter space, and p is the "momentum" of the fictitious particle. The Hamiltonian is therefore

$$H(x, p) = U(x) + K(p) \, . \tag{15}$$

Rather than sampling the proposed distribution directly, say $p(x)$, the Hamiltonian algorithm samples the canonical distribution of $p(x, p) = p(x)p(p)$. Since we're treating this like a physical system, where the point of the matter is keep the Hamiltonian constant, we also have to assume that our target probability distribution will behave similar to problems generally seen in statistical mechanics. The form of the probability should therefore follow the form of

$$p(x, p) \propto e^{-H(x,p)} \, . \tag{16}$$

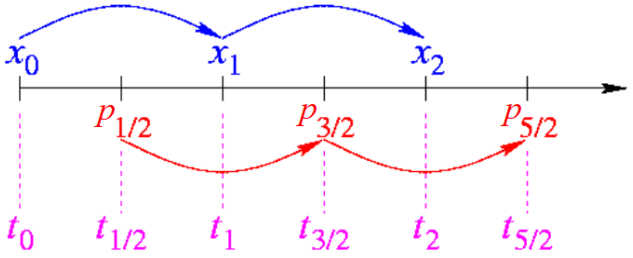

Figure 8. A severely simplified schematic of the leap-frog algorithm. The positions and momenta and time t, as well as the positions at time $t - \Delta t$, are used to predict the positions of $t + \Delta t$. After being calculated, the next iteration happens at $t + \Delta t$.

One of the problems that arises from setting up the problem as a real system is that now the time over which we evolve our physical system is discretized. To remedy this problem, the Hamiltonian algorithm uses something called the "leap-frog" algorithm, which uses the positions and momenta at time t and the positions at time $t - \Delta t$ to predict the positions at time $t + \Delta t$. The actual formulas are shown below, in consecutive order:

$$\begin{aligned} p_i\left(t + \frac{\Delta}{2}\right) &= p(t) - \frac{\Delta}{2}\frac{\partial}{\partial x_i}U(x(t)) \, , \\ x_i\left(t + \Delta\right) &= x(t) + \Delta p_i\left(t + \frac{\Delta}{2}\right) \, , \\ p_i\left(t + \Delta\right) &= p_i\left(t + \frac{\Delta}{2}\right) - \frac{\Delta}{2}\frac{\partial}{\partial x_i}U(x(t + \Delta)) \, . \end{aligned} \tag{17}$$

For an over-simplification of the leap-frog mechanism, see Fig. 8. Following leap-frog is the accept/reject step, which is identical to the Metropolis-Hastings algorithm, though the general form of the probability is the one shown in Eq. 16. Therefore, the acceptance probability ratio has the form of

$$\alpha = \frac{p(x', p')}{p(x, p)} = \frac{e^{-H(x',p')}}{e^{-H(x,p)}} \, . \tag{18}$$

The major benefit of using the Hamiltonian algorithm arise from its ability to cover large distances in high-dimensional space due to the way the algorithm keeps the Hamiltonian constant. In a physical system, this would be related to energy conservation, but in terms of the algorithm, it allows the fictitious particle to travel far before switching directions, allowing the parameter space to be sampled to a larger degree. The unfortunate drawback of this algorithm is that it is poor at sampling distributions with isolated local minima. In the case where the minima are far apart, the fictitious particle would not generally have enough of a random energy kick to cross over the energy barrier to another minima.

4.4 Slice Sampling Algorithm

One of the steps in fine-tuning the Metropolis-Hastings algorithm is to use a good likelihood function, with a step size that would allow the algorithm to sample the distribution appropriately. If the step size is too small, then the algorithm may get stuck in a local minima, while if the step size is large, there is a possibility of missing a small feature entirely. Slice sampling was developed to counteract this problem by being built to adjust the step-size autonomously. Radford (2003), who initially developed the algorithm, described

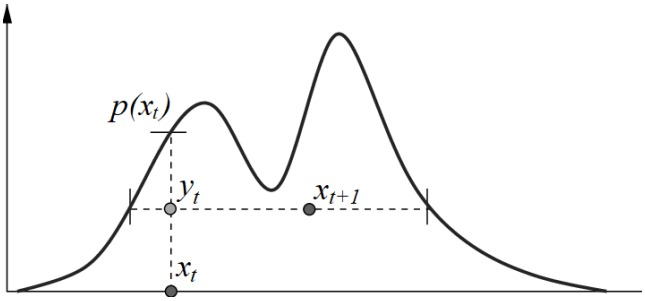

Figure 9. A visual representation of how the slice sampling algorithm works. x is the variable that we are sampling in our parameter space and y is an auxiliary variable generated to allow us to sample the distribution using slice sampling.

it as "easy to implement" and "simple to fine-tune", though other authors like Royle et al. (2013) seem to disagree. What is safe to assume is that, like every other algorithm so far, it has its benefits and drawbacks – which I'll discuss in a bit. The basic idea is that any distribution can be sampled by sampling points uniformly under the proposed distribution function. The algorithm step-by-step is as follows:

1. Draw variable x_t from proposed distribution $p(x)$.
2. Sample auxiliary variable, $y_t \in (0, p(x_t))$, vertically.
3. Slice the curve horizontally at y_t. (19)
4. Draw variable x_{t+1} uniformly from horizontal slice.
5. Repeat steps 2 to 4.

An example of this process is shown in Fig. 9. The reason the method was developed in this way is because sampling a distribution $p(x)$ is the same as uniformly sampling the points underneath the curve of the distribution. To show why this is so, let's take what we already know: we use a condition with which we sample the auxiliary variable y uniformly, where $0 < y_t < p(x_t)$, or that the auxiliary variable is between the 0 and the proposed distribution. Technically that's incorrect – we don't know what the normalized posterior distribution function is. Radford (2003) explains this in much more depth during his derivation of this algorithm, but essentially the condition is $0 < y_t < \hat{p}(x_t)$, where $\hat{p}(x)$ is the not yet normalized posterior distribution function and

$$p(x) = \frac{\hat{p}(x)}{Z}, \tag{20}$$

where Z is a normalization constant defined as $Z = \int \hat{p}(x)dx$. This term is the same as the usual denominator term seen in Bayes' theorem as shown in Eq. 1.

To bring this back, the point I made was that sampling a distribution $p(x)$ is the same as uniformly sampling the points underneath a curve. Sampling the points underneath the curve would be given by the joint distribution $p(x, y)$, for which the points follow the constraint $0 < y < p(x)$ and then getting rid of all the y values. So when we sample this joint distribution, we sample

$$p(x, y) = \begin{cases} \frac{1}{Z}, & \text{if } 0 < y < \hat{p}(x) \\ 0, & \text{otherwise} \end{cases}. \tag{21}$$

The above statement is valid because

$$p(x) = \int p(x, y)dx = \int_0^{\hat{p}(x)} \frac{1}{Z} du = \frac{\hat{p}(x)}{Z}. \tag{22}$$

The above arguments are explained in full detail by Radford (2003), but the point of bringing them up here was to show how the unknown normalization constant Z is used in this algorithm.

This algorithm benefits from the fact that the only thing we need to input is a proposal distribution function that is proportional to the true target distribution. This algorithm works best for single-variable cases, though it is possible to implement it for multivariate too. The algorithm would have to sample each variable by itself, meaning that the conditional probabilities between parameters would be required. To update variable x_i, we would need the conditional probability $p(x_i \mid x_{j\neq i})$, where $x_{j\neq i}$ are the values for all the other parameters.

The biggest drawback this algorithm is in its inability of sampling multimodal probability distributions well, especially if they are far from each other. The reason this is a drawback is that the algorithm needs to define bounds on the horizontal slice. In its most basic form, the interval chosen is equal to the horizontal slice length underneath the $p(x)$, which ignores the potential for any modes outside the local parameter space the algorithm is in. Several schemes have been proposed by Radford (2003), though none can fully solve the problem: (i) the algorithm could pick an interval that's "double" the size of the horizontal slice defined under the curve, or (ii) the algorithm could randomly pick an initial size for the horizontal slice and then scale it up by some factor. Both of these could potentially find multimodal points of interest that are relatively close to each other in parameter space, but it's not a universal solution as many modes could exist outside the ranges covered. Increasing the horizontal slice over which the algorithm samples also decreases its efficiency – so we might as well use a different algorithm at that point. It's possible that an accept/reject mechanism, such as the one that exists in the Metropolis-Hastings algorithm, could potentially solve this issue, but even this has been shown to be fruitless in works like in Dittmar (2013). Other methods have been proposed to fix this issue, such as "elliptical slice sampling," but this is the slice sampling algorithms' biggest drawback.

5 CONCLUSIONS

In this paper we discussed five different Markov Chain Monte Carlo methods. The most-common and widely used algorithm, Metropolis-Hastings, was used as a demonstration on a problem that currently exists in Medicine – the controversial prostate cancer blood tests. We found that this algorithm quickly converged onto the expected results and within reasonable error (± 1 standard deviation). The Gibbs algorithm was shown to be a useful algorithm if sampling from the proposed posterior distribution is difficult. The advantage of this method is that it uses conditional probabilities to sample from, so the dimensionality of the problem is reduced and the algorithm becomes more efficient. The Metropolis-adjusted Langevin algorithm was shown to be useful in situations where the convergence was slow. It nudges the walkers in the right direction so that over time, they will be preferentially pushed towards the target distribution through the calculation of the gradient of the log density function. The Hamiltonian algorithm showed how we could use physical problems to help us solve computational problems. We inversed the probability distribution functions to form potential wells over which fictitious particles (essentially walkers) could explore.

We defined a Hamiltonian for a system and kept it constant as we evolved it over time. The last algorithm discussed was Slice sampling. This algorithm focused on sampling the target distribution by uniformly sampling underneath it with the help of an auxiliary variable. The benefit of this mechanism is that we only only need to input a proposal distribution function that is proportional to the target distribution. The algorithm then samples the space well enough to derive the true target distribution from that. This summary covered the most common algorithms that someone may decide to use, but there are many more algorithms that exist online. Hopefully, this explained the basics of what one would need to know before diving into the world of Monte Carlo analysis.

REFERENCES

Bandyopadhyay P., Forster M., 2011, Philosophy of Statistics

Barry M. J., 2001, The New England Journal of Medicine, 344, 1373

Brawer M. K., Chetner M. P., Beatie J., Buchner D. M., Vessella R. L., Lange P., 1992, The Journal of Urology, 147, 841

Catalona W., Smith D., Ratliff T., Dodds K., Coplen D., Yuan J., Petros J., Andriole G., 1991, The New England Journal of Medicine, 324, 1156

Dittmar D., 2013, Slice Sampling, doi:10.1.1.399.1614

Durmus A., Roberts G. O., Vilmart G., Zygalakis K. C., 2017, The Annals of Applied Probability, 27, 2195

Foreman-Mackey D., 2016, The Journal of Open Source Software, 24

Foreman-Mackey D., Hogg D. W., Lang D., Goodman J., 2013, Publications of the Astronomical Society of the Pacific, 125, 306

Geman S., Geman D., 1984, IEEE Transactions on Pattern Analysis and Machine Intelligence, 6, 721

Goodman J., Weare J., 2010, Communications in Applied Mathematics and Computational Science, 5, 65

Joyce J., 2019, in Zalta E. N., ed., , The Stanford Encyclopedia of Philosophy, spring 2019 edn, Metaphysics Research Lab, Stanford University

Kennedy A. D., Pendleton B., 1991, Nuclear Physics B Proceedings Supplements, 20, 118

Lynch S. M., 2007, Introduction to Applied Bayesian Statistics and Estimation for Social Scientists. Springer-Verlag New York, doi:10.1007/978-0-387-71265-9

Ottobre M., Pillai N. S., Spiliopoulos K., 2017, arXiv e-prints, p. arXiv:1702.01777

Radford N. M., 1993, Probabilistic Inference Using Markov Chain Monte Carlo Methods

Radford N. M., 2003, The Annals of Statistics, 31, 705

Radford N. M., 2012, arXiv e-prints, p. arXiv:1206.1901

Roberts G. O., Rosenthal J. S., 2001, Statistical Science, 16, 351

Roberts G. O., Tweedie R. L., 1996, Bernoulli, 2, 341

Royle J. A., Chandler R. B., Gazenski K. D., Graves T. A., 2013, Ecology, 94, 287

Society C. C., 2018, Canadian Cancer Statistics 2018, `cancer.ca/Canadian-Cancer-Statistics-2018-EN`

Thompson I. M., et al., 2004, The New England Journal of Medicine, 350, 2239

Wang M. C., Valenzuela L. A., Murphy G., Chu T., 1979, Investigative Urology, 17, 159

This paper has been typeset from a TEX/LATEX file prepared by the author.