MARKOV CHAINS AND THE METROPOLIS ALGORITHM

HARSHA ANAND

Abstract

The Metropolis Algorithm and its more general counterpart, the Metropolis Hastings Algorithm, are useful in modeling complicated distributions. These algorithms are based on Markov Chains, stochastic models describing the probability between transitioning from various states where the probability of each transition depends only on the previous state. This paper first goes over the key properties of such Markov Chains to provide the reader with enough background knowledge to understand the algorithm part of the paper. Then the paper goes over the Metropolis and Metropolis algorithms and covers the impact of many factors within the simulation which play a crucial role in its efficiency to model the desired target distribution.

Date: September 2023.

Contents

List of Figures

1. Introduction

In the constantly expanding realm of statistical modeling, the ability to accurately and efficiently simulate complex probability distributions is very challenging. This challenge has inspired the creation of many algorithms to help model these complex distributions. One of these algorithms developed was the Metropolis Algorithm, created by Nicholas Metropolis in 1953. The algorithm was first used in the paper *Equation of State Calculations by Fast Computing Machines.* [2] The paper deals with the problem of simulating configurations of a model particle system represented by two dimensional rigid spheres and then calculating the properties of interest (such as energy or density) of these configurations. Rather than choosing configurations of the rigid sphere model randomly, then weighing them with the Boltzmann factor $\exp(-E/kT)$, where E is the energy, T is the temperature, and k is the Boltzmann's constant, configurations were chosen with a probability given by the Boltzmann factor and then weighed evenly. This change made the sampling method focus more on the low-energy configurations, which resulted in improved convergence. The earliest version of the Metropolis Algorithm was used to choose configurations with the probability given by the Boltzmann factor. It went as following: 1.) each candidate configuration is generated by a symmetric proposal distribution based on the previous configuration; 2.) If the energy of the new configuration is lower than the energy of the previous configuration, the move is always accepted. Otherwise, the move is accepted with the probability given by the Boltzmann factor. If the move is rejected, the previous configuration will be counted again as the current configuration. 17 years after the Metropolis Algorithm was first used, W.K. Hastings extended the algorithm to a more general case, as the prior Metropolis Algorithm assumed a symmetric proposal distribution. This more general version of the algorithm, the Metropolis Hastings Algorithm, also works with asymmetric proposal distributions. The underlying foundation of both the Metropolis and the Metropolis Hastings Algorithm are based purely on Markov Chains. A Markov Chain is a mathematical model which describes the transitions between its various states in a stochastic manner. The key defining property of a Markov Chain is that its probability of transitioning from one state to another depends only on the previous state. This property is often described as "memoryless" as the transition between states is independent to all previous states except the one state that precedes the future state. The Metropolis Algorithm is a Markov Chain Monte Carlo (MCMC) method. It constructs a Markov Chain with a stationary distribution (we will go more on about what this is later) equal to the desired target distribution. This allows the Metropolis Algorithm to be very useful in modeling complicated target distributions accurately and efficiently. But there are many factors such as step size, the starting value of your simulation, burn in periods, etc. which you will have to know how to control in order to have an efficient simulation.

Here is quick walk through of the topics covered by this paper. We first start with key definitions and properties regarding Markov Chains, including the Markov Property, transition matrices, stationary distributions, irreducibly, periodicity, the Fundamental Theorem of Markov Chains and many more topics related to Markov Chains. Then we move on to the Metropolis and Metropolis-Hastings algorithm section of the paper. Here, you will read about how and why these algorithms work, the importance of controlling key factors such as step size, burn in periods and the starting value of the algorithm, a simple hill climb algorithm to find maximal vertices, and effective sample size (ESS).

2. Markov Chains

Definition 2.1: Markov Chain. A Markov Chain is a mathematical model that details the stochastic transitions between various elements or states of a set X. The probability of these transitions between various states only depend on the current state.

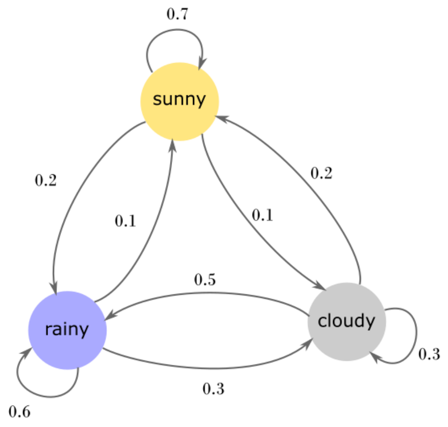

Figure 1. A simple weather system represented as a Markov Chain.

Definition 2.2: Markov Property. Suppose we have a set of states $S = (s_{n+1}, s_n, \ldots, s_0)$. All Markov chains follow the Markov Property, described as:

For any $n \geq 1$,

$$P(X_{n+1} = s_{n+1} | X_n = s_n, \ldots, X_0 = s_0) = P(X_{n+1} = s_{n+1} | X_n = s_n) = P(s_{n+1}, s_n).$$

In other words the probability of entering any future state s_{n+1} depends only on the current state s_n and none of the previous states "visited".

Definition 2.3: Transition Matrix. A transition matrix is a stochastic matrix whose (i, j) entry represents the probability of transitioning from s_i to s_j. This probability is denoted as p_{ij} or $P(i, j)$ and is called a **transition probability**.

Definition 2.4: First Passage Time Probabilities. We define the first-time passage probabilities by

$$p_{ij}^{(n)} = P(X_n = j; X^k \neq j, 0 < k < n-1 | X_0 = i);\ i, j \in \mathbb{Z}.$$

We can denote the expected return time of this distribution as

$$\mu_{ij} = \sum_{n=1}^{\infty} n p_{ij}^{(n)}.$$

Theorem 2.5: Chapman Kolmogorov Equation

$$p_{ij}^{n} = \sum_{k \in \mathbb{Z}} p_{ik}^{(m)}\ p_{kj}^{(n-m)}, \text{ for all } m \in \mathbb{N} \cup 0 \tag{2.1}$$

Proof. [3] By law of total probability:

$$p_{ij}^{(n)} = P(X_n = j|X_0 = i) = \sum_{k\in\mathbb{Z}} P(X_n = j, X_m = k|X_0 = i)$$

$$= \sum_{k\in\mathbb{Z}} P(X_n = j|X_m = k, X_0 = i)P(X_m = k|X_0 = i).$$

Now, by the Markovian property:

$$= \sum_{k\in\mathbb{Z}} P(X_n = j|X_m = k)P(X_m = k|X_0 = i) = \sum_{k\in\mathbb{Z}} p_{ik}^{(r)}\, p_{kj}^{(m-r)}.$$

The Chapaman-Kolmogrov equation allows us to denote our t-step transition probabilities as a matrix of t-step transistion probabilities:

$$P^n = p_{ij}^{(m)}.$$

Definition 2.6: Stationary Distribution Part 1. Consider an initial probability distribution ϕ_i s.t $\phi_i = P(X_0 = i), i \in \mathbb{Z}$. Also, $\sum\limits_{i\in\mathbb{Z}} \phi_i = 1$.

Distribution π_j is defined to be stationary if it satisfies:

$$\pi_j = \sum_{i\in\mathbb{Z}} \phi_i p_{ij}. \tag{2.2}$$

Example 2.7: Simple Weather System. Consider a weather system with only two possible states: sunny and rainy. Every day the weather can change based on certain transition probabilities. On a sunny day there is a probability p that it is rainy the next day, and on a rainy day there is a probability q that it is sunny the next day.

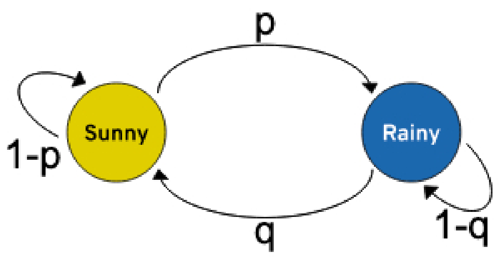

Figure 2. Example 2.7's weather system represented by a Markov Chain.

Let $X = s, r$, and let $(X_0, X_1, \ldots)$ be the sequence of weather states on Sunday, Monday, ... The rules of the weather system indicate that if we set

$$P = \begin{pmatrix} P(s,s) & P(s,r) \\ P(r,s) & P(r,r) \end{pmatrix} = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix},$$

then $(X_0, X_1, \ldots)$ is a Markov chain with transition matrix P. Notice that the top row of the transition matrix is the conditional probability distribution of X_{t+1} given that $X_t = s$. Likewise, the bottom row is the conditional probability distribution of X_{t+1} given that X_t $= r$.

Assume the weather is sunny on Sunday. On Monday, the weather has probability p changing to rainy and probability $1 - p$ staying sunny. In conditional probability terms:

$$P(X_1 = r|X_0 = s) = 1 - p, P(X_1 = s|X_0 = s) = p. \tag{2.3}$$

On Tuesday, considering the two possibilities for X_1:

$$P(X_2 = s | X_0 = s) = (1-p)(1-p) + pq \tag{2.4}$$

and

$$P(X_2 = r | X_0 = s) = (1-p)p + p(1-q). \tag{2.5}$$

Writing equations like (2.4) and (2.5) would be very inefficient for days further along in the future. Fortunately for us there is a much more efficient way to store our distribution information, being in a row vector:

$$\mu_t := (P(X_t = s|X_0 = s), P(X_t = r|X_0 = s)).$$

Since the weather started of as sunny on Sunday, we can now write this as $\mu_0 = (1, 0)$, while (2.3) can be written as $\mu_1 = \mu_0 P$.

To reach the distribution of the next day, you must multiply the current distribution by P. We now get

$$\mu_t = \mu_{t-1} P \text{ for all } t \geq 1.$$

Further more, for any initial distribution μ_0, after t days in the weather system we can say

$$\mu_t = \mu_0 P^t \text{ for all } t \geq 0.$$

But how does this distribution μ_t behave long term?

Definition 2.8: Stationary Distribution Part Two. A stationary distribution is a row vector that represents the long term probabilities of being in each state of a Markov chain. This row vector is non-negative, sums up to 1 and is not changed by the operation of a transition matrix P on it. Thus, it satisfies

$$\pi P = \pi. \tag{2.6}$$

Here, π is a left eigenvector of the transition matrix P with an eigenvalue of 1. We can now find the stationary distribution of Markov chains using this equation. But by using linear algebra we can tell that such an equation either has a unique solution or infinitely many solutions. Since we want a unique stationary distribution, we will have to verify if the Markov chain meets certain conditions. So what are these properties a Markov chain must have to have a unique stationary distribution?

3. Properties of Markov Chains.

Definition 3.1: Recurrence and Transience. We say that a state i is **recurrent** if

$$\sum_{n=1}^{\infty} f_{ij}^n = 1.$$

In other words, a state is recurrent if whenever we leave that state, the probability of returning to that same state in the future is 1.

We say the state is **transient** if

$$\sum_{n=1}^{\infty} f_{ij}^n < 1.$$

In other words, a state is transient if whenever we leave that state, there is a possibility of never returning to that same state.

A recurrent state i is **positive-recurrent** if $\mu_{ii} < \infty$ and is **null-recurrent** if $\mu_{ii} = \infty$.

Definition 3.2: Irreducibility of Markov Chains. A Markov chain is **irreducible** if for every pair of states i, j there is a n s.t. $p^n_{i,j} > 0$. In other words, for a Markov chain to be irreducible every state has to be interconnected, and there has to be no isolated state(s) that cannot be reached from each other.

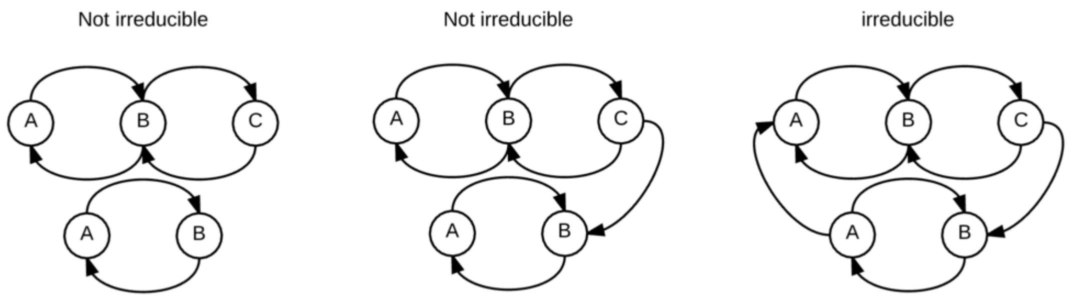

Figure 3. Some examples of irreducible and reducible Markov Chains

Definition 3.3: Periods and Periodicity of states. The **period** of some state $s_i \in X$ is $h(s_i) = \gcd(t \in \mathbb{Z}^+ | p^n_{s_i,s_i} > 0)$. In other words, from s_i, the Markov chain can return back to s_i in periods of $d(s_i)$.

This value of $d(s_i)$ is used to determine periodicity. s_i is **periodic** if $d(s_i) > 1$ and is **aperiodic** if $d(s_i) = 1$. We can relate this to Markov chains too; A Markov chain is periodic if every state is periodic. Otherwise it is aperiodic.

But checking that the gcd of the times that a chain returns back to its original state can be inefficient, considering the fact that you would have to do it for every state in the chain until you find a $t = 1$. Fortunately, given an irreducible Markov chain if there is at least one self-transition $P(s_i, s_i)$ in the chain, then the chain is aperiodic. Also, a chain is aperiodic if and only if there exists a positive integer n such that all entries in the transition matrix P^n are strictly positive.

Theorem 3.4: Fundamental Theorem Of Markov Chains. The Fundamental Theorem of Markov Chains is that for any irreducible, aperiodic, and positive-recurrent Markov Chain there exists a unique stationary distribution $\{\pi_j, j \in \mathbb{Z}\}$ s.t $\forall i \in \mathbb{Z}$.

Proof. [3] Since our Markov Chain is irreducible, aperiodic and positive-recurrent, we know that $\forall i \in \mathbb{Z}, \pi_j = \lim_{n\to\infty} p^n_{ij} > 0$. Likewise as π_j is a probability distribution $\sum_{n=1}^{\infty} f^n_{ij} < 1$. We know that $\forall m$

$$\sum_{i=0}^{m} p^m_{ij} \leq \sum_{i=0}^{\infty} p^m_{ij} \leq 1.$$

Now taking the limit:

$$\lim_{m\to\infty} \sum_{i=0}^{m} p^m_{ij} = \sum_{i=0}^{\infty} \pi_j = 1.$$

This means that for any M

$$\sum_{i=0}^{M} \pi_j \leq 1.$$

Now lets use the *Chapman-Kolmogorov equation (Theorem 2.5).* :

$$p_{ij}^{(m+1)} = \sum_{i=0}^{\infty} p_{ik}^{m}\, p_{kj} \geq \sum_{i=0}^{M} p_{ik}^{m}\, p_{kj}$$

Now we take the limit as $m \to \infty$. Remember though as $m \to \infty$, $M \to \infty$ too.

$$\lim_{m\to\infty} p_{ij}^{(m+1)} \geq \lim_{m\to\infty} \sum_{i=0}^{\infty} p_{ik}^{m}\, p_{kj}$$

$$\pi_j \geq \sum_{i=0}^{\infty} \pi_k\, p_{kj} \tag{3.1}$$

Notice inequality 3.1 looks very similar to equation 2.2. We must just now search for a contradiction to prove the equality. Summing over this inequality we arrive at

$$\sum_{j=0}^{\infty} \pi_j > \sum_{j=0}^{\infty}\sum_{i=0}^{\infty} \pi_k\, p_{kj}.$$

Rewriting this inequality we get

$$\sum_{j=0}^{\infty} \pi_j > \sum_{i=0}^{\infty} \pi_k \sum_{j=0}^{\infty} p_{kj}.$$

$\sum\limits_{j=0}^{\infty} p_{kj}$ represents the sum of transition probabilities from state k to all possible states j in our Markov Chain. Since our chain is irreducible, this means that this summation will be equal to 1.

$$\sum_{j=0}^{\infty} \pi_j > \sum_{i=0}^{\infty} \pi_k$$

Both of these summations are equal to 1, making this the contradiction of our original inequality we are looking for. This means equality must hold in inequality 3.1.

$$\pi_j = \sum_{i=0}^{\infty} \pi_k\, p_{kj}$$

Thus we arrive at equation 2.2, meaning a unique stationary distribution does exist.

Example 3.5: Simple Weather System Continued. Continuing on with example 2.6, let us now compute the stationary distribution for the weather system with our new found knowledge. Before we can use Theorem 2.7, we must first see if the Markov chain representing the weather system is irreducible and aperiodic.

Since the chain only consists of two states that are always connected to each other, the weather system is irreducible. Since the chain is irreducible, we now just have to find at least one self-transition in the chain to prove that the chain is also aperiodic. Both the sunny and rainy states have transition probabilities of $1-p$ and $1-q$ to stay at the current state respectively, making there two possible self-transitions. Therefore, this condition is met and thus the weather system is aperiodic too.

Since the weather system is irreducible and aperiodic we can now apply Theorem 3.3 to find the unique stationary distribution π. To find π we must solve the equation $\pi P = \pi$. After a little linear algebra you arrive at the results:

$$\pi(s) = \frac{q}{p+q},\ \pi(r) = \frac{p}{p+q}.$$

Before we move on to the Metropolis Algorithms, we have to learn one more key property of Markov chains.

Definition 3.6: Reversibility. A Markov chain is called reversible when its stationary distribution π and transition matrix P satisfy

$$\pi(x)P(x,y) = \pi(y)P(y,x).$$

In simpler terms, a reversible Markov chain "looks the same" regardless if we run it forward or backwards in time. This property is used much more than just for mathematical curiosity, as in future sampling algorithms involving Markov chains, reversibility is a crucial property for Markov chains to have.

4. Metropolis Algorithm and the Hill Climb Algorithm

Introduction 4.1. Given some Markov chain with state space X and some arbitrary stationary distribution, can the chain be modified so that the new chain has a stationary distribution π? Fortunately for us the Metropolis algorithm accomplishes this.

Definition 4.2: Metropolis Algorithm. Suppose we have some Ψ that is a symmetric transition matrix reversible with respect to the uniform distribution on X. We now will modify transitions made according to Ψ to obtain a chain with stationary distribution π.

The new chain is formed through an algorithm of accepting and rejecting new states. First, when the chain is at state x, a potential move to a new state is generated from the proposal distribution $\Psi(x,\cdot)$. Let this proposed new state be y. This move to y is accepted with the probability $a(x,y)$ and is rejected with the probability 1-$a(x,y)$. If the move to y is rejected the chain stays at its current state x. Rejecting moves to new states slows the chain down, which intuitively seems to reduce its efficiency, but is still needed to reach the specific stationary distribution π. We will discuss how to choose the acceptance ratio a(x,y) soon, but below is the transition matrix $P(x,y)$ for our new chain:

$$P(x,y) = \begin{cases} \Psi(x,y)a(x,y), \text{ if } y \neq x \\ 1 - \sum\limits_{z:z\neq x} \Psi(x,z)a(x,z), \text{ if } y = x. \end{cases}$$

Keep in mind that this transition matrix $P(x,y)$ can also be written as the transition kernel $q(y|x)$. This transition kernel q has a stationary distribution π if

$$\pi(x)\Psi(x,y)a(x,y) = \pi(y)\Psi(y,x)a(y,x) \tag{4.1}$$

for all $x \neq y$. Since we know Ψ is symmetric, equation 4.1 only holds if

$$b(x,y) = b(y,x),$$

where $b(x,y) = \pi(x)a(x,y)$. Since the acceptance probability function $a(x,y)$ is a probability, it satisfies $a(x,y) \leq 1$, the function b must satisfy these conditions:

$$b(x,y) \leq \pi(x), b(x,y) = b(y,x) \leq \pi(y).$$

A suitable choice for $a(x,y)$ that satisfies the detailed balance condition is:

$$a(x,y) = \min\left(1, \frac{\pi(y)}{\pi(x)}\right).$$

So our new state y has probability $a(x,y)$ of being accepted, and probability $1 - a(x,y)$ of being rejected.

Let's quickly recap what we have learned about the Metropolis Algorithm before moving on to its proof. The Metropolis Algorithm's purpose is to construct a Markov chain with a given stationary distribution π from a symmetric transition matrix Ψ that is reversible with respect to the uniform distribution on X. This algorithm works as follows:

1. Start at some initial state x in the state space X.

2. Generate a candidate for the new state y using a symmetric proposal distribution.

3. Calculate the acceptance ratio $a(x,y) = \min\left(1, \frac{\pi(y)}{\pi(x)}\right)$. Then accept the move to state y with probability $a(x,y)$ or stay at state x with the probability $1 - a(x,y)$.

4. Continue this process for a sufficient number of iterations to ensure convergence to the desired stationary distribution π.

Proof 4.3: Metropolis Algorithm Proof Of Convergence.
Lemma 1 If $\pi_i P(X_{n+1} = j | X_n = i) = \pi_j P(X_{n+1} = i | X_n = j)$ for all i, j, (in other words reversible), then π is a stationary distribution of P.

Proof of Lemma 1 If π satisfies the above condition then:

$$\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j.$$

so $\pi = \pi P$, which makes π a stationary distribution by definition.

Now we just have to show that the Markov chain resulting from the Metropolis Algorithm is reversible with respect to the desired target distribution $\pi(x)$. Once we show this we will have proved that $\pi(x)$ is the stationary distribution to the Markov chain by *Lemma 1*.

Obviously $\pi_i P(X_{n+1} = j | X_n = i) = \pi_j P(X_{n+1} = i | X_n = j)$ is met if $i = j$, so let us only consider when $i \neq j$. Also remember for the Metropolis Algorithm we assume that our proposal distribution Q is symmetric, leaving our acceptance ratio as $\frac{\pi_j}{\pi_i}$.

$$P(X_{n+1} = j | X_n = i) = q_{ij} \min\left(1, \frac{\pi_j}{\pi_i}\right) = \min\left(q_{ij}, \frac{\pi_j q_{ij}}{\pi_i}\right)$$

Let us now multiply both sides by π_i:

$$\pi_i P(X_{n+1} = j | X_n = i) = \min\left(q_{ij}\pi_i, q_{ij}\pi_j\right).$$

Similarly we arrive at:

$$P(X_{n+1} = i | X_n = j) = \min\left(q_{ji}\pi_i, q_{ji}\pi_j\right).$$

Since we are using the Metropolis Algorithm, we assume our proposal distribution Q is symmetric, meaning $q_{ij} = q_{ji}$. Therefore the Markov Chain resulting from the algorithm is reversible, meaning our target distribution $\pi(x)$ is indeed the stationary distribution of the chain.

Example 4.4: Simple Sampling Using Metropolis Algorithm. To further illustrate the Metropolis algorithm, let us sample from the simple exponential distribution:

$$\pi(x) = \exp(-x)(x \geq 0).$$

Obviously there would be other much simpler ways to sample from this distribution, but we are just using this example to illustrate the purpose of the Metropolis algorithm.

Our first step is to start at some initial x. Let's say we start at $x = 3$. Now using a symmetric proposal distribution of $N(0, 1)$ we can obtain candidates for the proposed new state y. Once you have generated a candidate you can use the probability ratio

$$a = \min\left(1, \frac{\pi(y)}{\pi(x)}\right) = \min\left(1, \frac{\exp(-y)}{\exp(-x)}\right)$$

to decide whether to accept or reject this candidate state. We now generate a random number u from a uniform distribution between 0 and 1. If $u \leq a$ we accept the candidate as the next state. If $u > a$ we reject the candidate as the next state and stay at our current state $x = 3$. Now if we keep repeating this process of generating, accepting and rejecting samples from our proposal model for a long enough time we should get an accurate sample of our exponential distribution. Utilizing some code to iterate through the Metropolis algorithm 10,000 times, look at Figure 4 for the plot of the values visited by the algorithm.

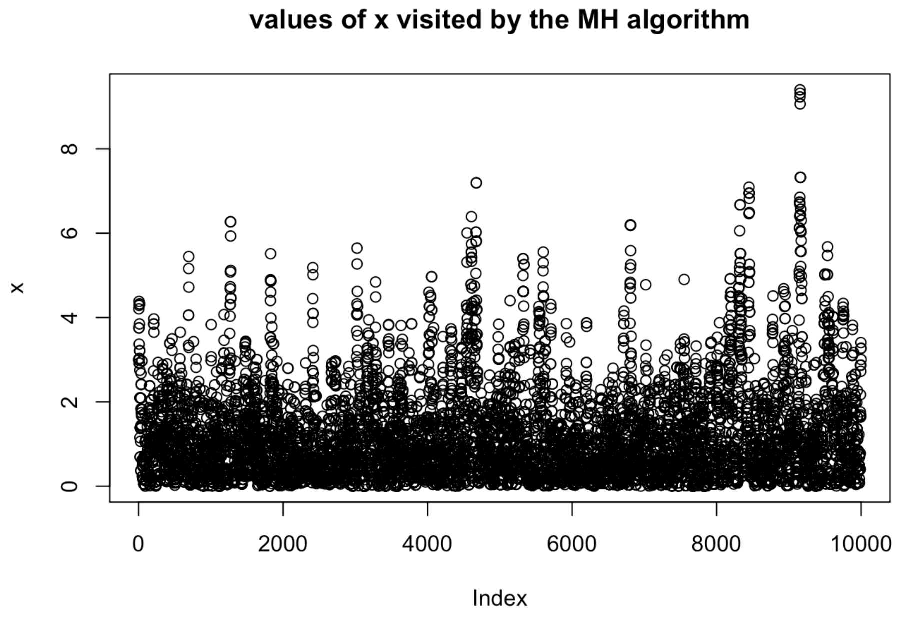

Figure 4. [1] plot of the values visited by the algorithm

Look at Figure 5 for the histogram of our values visited by the Metropolis algorithm. After 10,000 iterations our sample of values generated by the Metropolis algorithm provides a relatively close fit to the exponential distribution.

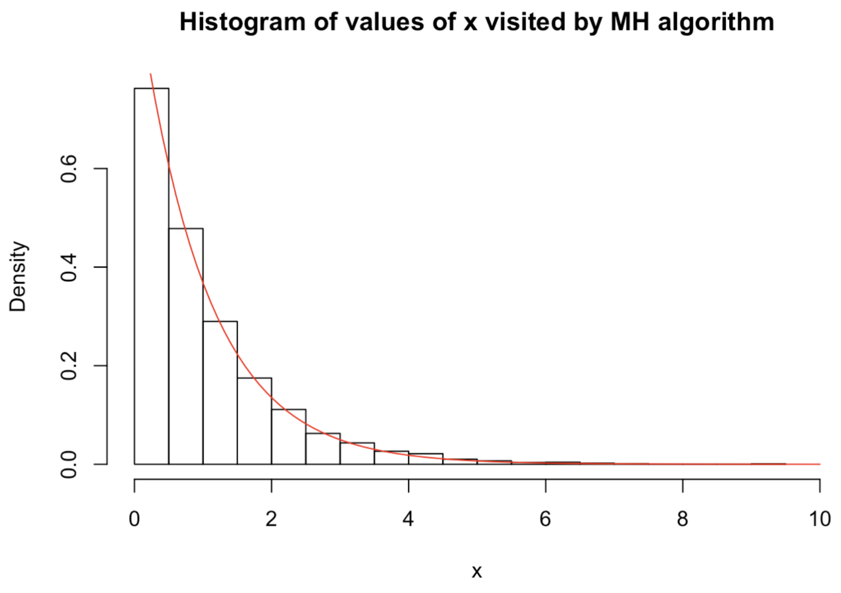

Figure 5. [1] histogram of the values visited by the Metropolis Algorithm

Remark 4.5. One of the key features of the Metropolis chain is that it only depends on the ratios $\frac{\pi(x)}{\pi(y)}$. Many times $\pi(x)$ has the form $\frac{h(x)}{Z}$, where the function $h : X \to [0, \infty)$ is known and $Z = \sum\limits_{x \in X} h(x)$ is a normalizing constant. It is often times difficult to compute Z, especially when X is large. But since the Metropolis chain only depends on the ratio $\frac{h(x)}{h(y)}$, you do not need to compute Z to simulate the sampling of the chain. Optimization chains such as the one in Example 4.4 are examples of this type.

Definition 4.6 Hill Climb Algorithm. Let f be a a real-valued function defined on the vertex set X of a graph. In many applications it is desirable to find a vertex x where $f(x)$ is maximal. But if the domain X is too large, then the search would be very inefficient.

The hill climb algorithm attempts to find the maximum values of f. Here is the process it follows: when at state x, if there is at least one neighboring state y satisfying $f(y) > f(x)$, the algorithm will move to a neighboring state with the largest value of f. But what if the algorithm is stuck at a local maxima of f?

The hill climb algorithm provides a solution to this: it randomizes the moves so that instead of eventually always getting stuck at a local maxima, there is some probability that the algorithm moves to lower states.

Let's fix $\lambda \geq 1$ and define

$$\pi_\lambda(x) = \frac{\lambda^{f(x)}}{Z(\lambda)},$$

where $Z(\lambda) = \sum_{x \in X} \lambda^{f(x)}$ is the normalizing constant. Since $\pi_\lambda(x)$ is increasing in $f(x)$, the probability $\pi_\lambda(x)$ favors vertices x for which $f(x)$ is large.

If $f(y) < f(x)$, the Metropolis chain accepts a transition $x \to y$ with the probability $\lambda^{-[f(x)-f(y)]}$. As $\lambda \to \infty$, the metropolis chain now resembles the optimal hill climb.

5. Metropolis Hastings Algorithm

Introduction 5.1. *4.2*'s Metropolis Algorithm can only be used when the initial transition matrix is symmetric. So naturally when the initial transition matrix is not symmetric a new metropolis algorithm is needed.

Definition 5.2: Metropolis Hastings Algorithm. For an irreducible transition matrix Ψ and some probability distribution π on X, the new Metropolis algorithm is executed as following. When at state x, a new state y is generated from $\Psi(x, \cdot)$. The chain then moves to the new state y with probability

$$a(x,y) = \min\left(1, \frac{\pi(y)\Psi(y,x)}{\pi(x)\Psi(x,y)}\right),$$

and remains at the state x with the complementary probability. The transition matrix P for this chain is

$$P(x,y) = \begin{cases} \Psi(x,y)a(x,y), \text{ if } y \neq x \\ 1 - \sum_{z: z \neq x} \Psi(x,z)a(x,z), \text{ if } y = x. \end{cases}$$

Proof 5.3: Metropolis Hastings Algorithm Proof. Last time when we showed why the Metropolis Algorithm works, we had gotten to assume that our proposal distribution $Q(y|x)$ was symmetric. Now with the more general Metropolis Hastings Algorithm this condition is not necessary.

In order to prove that the stationary distribution is our desired target distribution $\pi(x)$ we must prove the same *Lemma 1* as last time. The only thing different in our proof will be the acceptance ratio, as we cannot assume that $q_{ij} = q_{ji}$.

$$P(x_{n+1} = j | X_n = i) = q_{ij} \min\left(1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right) = \min\left(q_{ij}, \frac{\pi_j q_{ji}}{\pi_i}\right)$$

Now multiplying both sides by π_i, we arrive at

$$\pi_i P(X_{n+1} = j | X_n = i) = \min\left(\pi_i q_{ij}, \pi_j q_{ji}\right).$$

Similarly we arrive at:

$$\pi_j P(x_{n+1} = i | X_n = j) = \min\left(\pi_i q_{ij}, \pi_j q_{ji}\right).$$

Since *Lemma 1* is met, we can therefore conclude that the desired target distribution $\pi(x)$ is indeed the stationary distribution of the Markov Chain formed by the algorithm.

6. Improving The Efficiency of Simulations

Definition 6.1: Step Size in Metropolis Algorithm. All though the Metropolis algorithm is guaranteed to asymptotically converge to the target distribution, practically speaking you will never have an infinite amount of time to run your sampler. Therefore we should be only interested in maximizing the efficiency of our sampler in a finite sample. This is where the step size of our transition kernel comes into play.

Remember the Metropolis algorithm starts off by sampling some θ_0 from some arbitrary proposal distribution π. Then, for each iteration t: we generate a candidate θ'_t by sampling it from our symmetric proposal kernel q such that $q(\theta_t|\theta_{t-1}) = q(\theta)$. Like in our last example, most of the time this proposal kernel is a normal distribution centered at the previous value and has some arbitrary step size σ: $N(\theta_t, \sigma)$.

Choosing this step size σ is crucial to having an efficient simulation. Remember this step size is the standard deviation of the proposal model generating the candidate states. If σ is low in value, there will be less variance between the states, and a high proportion of candidates are accepted. But the downside of a low step size would mean it takes us a long time to move through the posterior space. By contrast, if σ is large in value, there will be more variance between the states. All though this would allow us to travel through the posterior space more quickly, this would also result in us rejecting a higher proportion of candidates, resulting in a relatively inefficient simulation. The goal is to get this σ somewhere in the middle so that we propose values that tend to be accepted with reasonable frequency and we can still move a relatively long distance in posterior space.

Let's see the effects of different step sizes on the efficiency of a finite sample Metropolis algorithm.

Example 6.2: Simulation with different step sizes. In our last example we used the Metropolis algorithm to sample from a relatively simple 1D exponential function. Let's level things up a bit and use a more complicated target distribution in 2D space:

$$p(x, y) = \exp\left(-x^2 - y^2\right) + \exp\left(-(x-4)^2 - (y-4)^2\right).$$

Let us set the domain for our simulation to the square set by the four vertices of (-5,5); (-5, 10); (10,0) and (10,5). Figure 6 is the probability density distribution of our target.

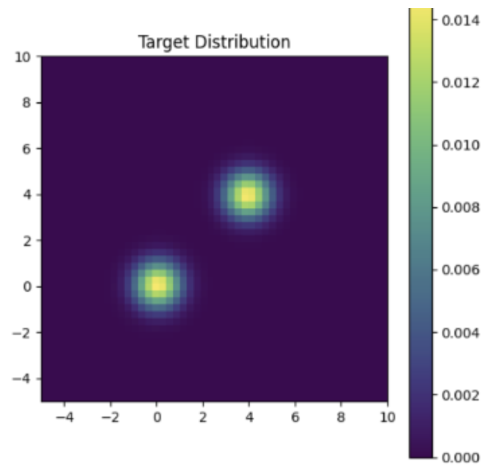

Figure 6. Probability density distribution of our target distribution

For our proposal distributions, let's use normal models again:

$$x' \sim \mathcal{N}(x, \sigma),$$
$$y' \sim \mathcal{N}(y, \sigma).$$

Now using the help of python, let us run five simulations with different step sizes of $\sigma = 0.25, 0.5, 1, 2, 4$ to see the impact step size has on the efficiency of the algorithm. We will also be running 10,000 iterations for each simulation. Figures 7-11 are the resulting probability density distributions of the values visited by the five different simulations.

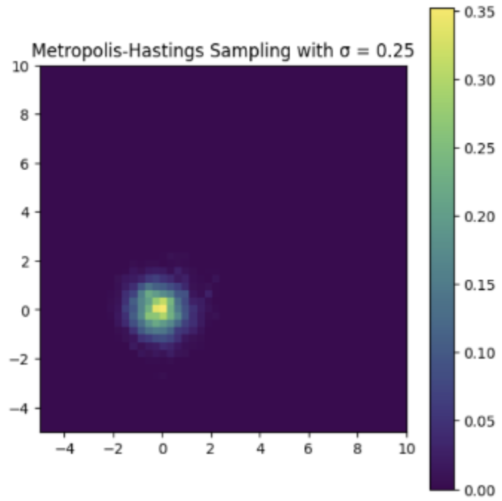

Figure 7. Probability density distribution with step size of 0.25

With the step size only at 0.25, we can see that though the simulation got a pretty accurate representation of one of the modes of the target distribution, it failed to discover the second mode in total. This is not the ideal step size.

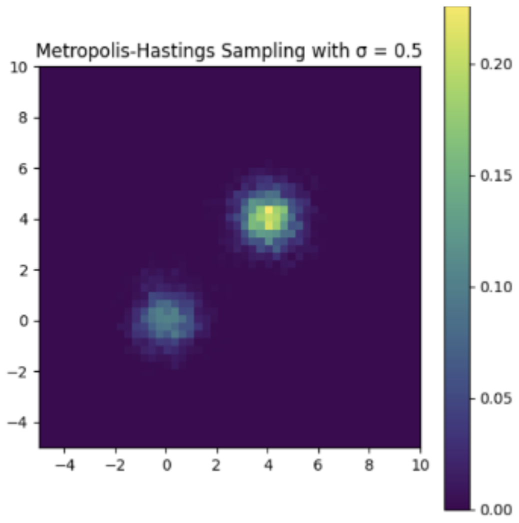

Figure 8. Probability density distribution with step size of 0.5

With the step size increased to 0.5, we can see that now the simulation has discovered the general area of a second mode while maintaining an accurate representation of the first mode. This is still far from our ideal step size, though, as we barely have a notion of the probability distribution of the second mode.

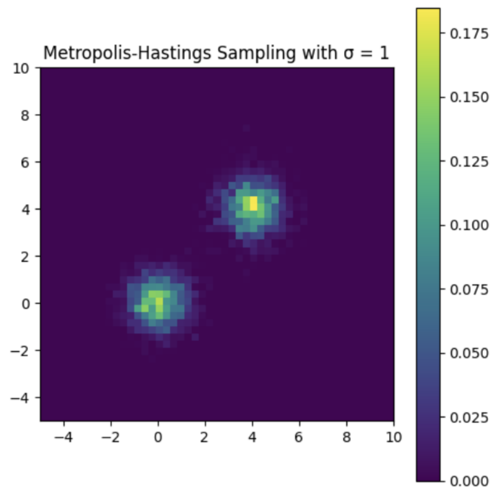

Figure 9. Probability density distribution with step size of 1

With the step size increased to 1, we can now see how close our probability distribution of the values visited by the algorithm is compared to the actual target distribution. Though there is more noise in one of the modes of our sampling distribution compared to the last two simulations, we have a much more accurate distribution overall due to the quality of the second mode. This may be the ideal step size for our simulation.

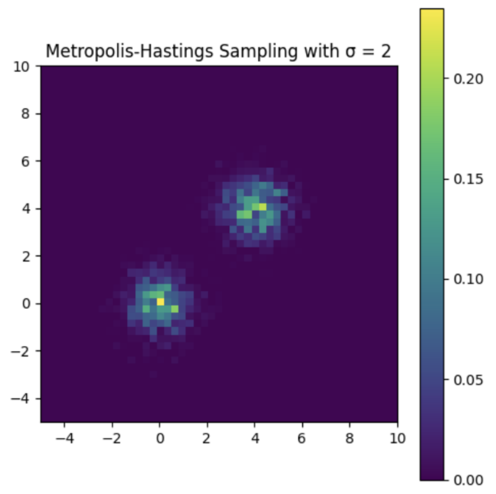

Figure 10. Probability density distribution with step size of 2

With the step size increased to 2, we can now see our sampling distribution has deteriorated in accuracy. All though we still have a relatively accurate representation of our target distribution, there is much more noise in both modes of our distribution compared to our last simulation. This is not the ideal step size for our simulation.

With the step size all the way increased to 4, we can see now our sampling distribution has deteriorated even more in accuracy. Now we only have a general notion of what our

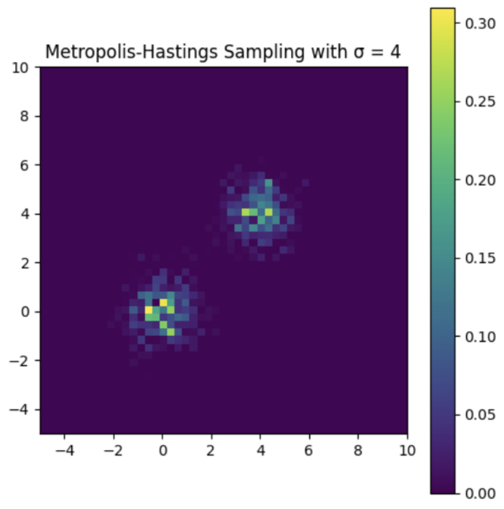

Figure 11. Probability density distribution with step size of 4

target distribution looks like, as there is even more sound in both modes. This is not the ideal step size for our simulation.

As you can deduct from these five simulations, step size plays a crucial role to conducting an efficient Metropolis algorithm. When our step size was too small we only had an accurate representation of one of the modes of the target distribution; the algorithm did not have enough iterations to accurately discover the other mode. In contrast when our step size was too large, though we had a good notion of where the two modes were located at, there were much more noise in both of the modes, reducing the accuracy of our sampling distribution. But when we found a step size that "was just right", we found ourselves a relatively accurate sampling distribution with only 10000 iterations.

Another thing to keep in mind for Metropolis algorithm simulations is that the step size plays a less important role as the number of iterations of our simulation increases. This makes sense intuitively, as hypothetically with an infinite amount of steps, no matter the step size, the algorithm will have a fully accurate representation of the target distribution. Take a look at Figures 12 through 16 for the new distributions of the various different step sizes.

Even with a step size of 0.25, with the number of iterations increased all the way to 500,000, we can see that our simulation has successfully found the second mode of the target distribution, and provides us with a relatively accurate sampling distribution.

We get another relatively accurate representation of our target distribution with the step size of 0.5 and get a nearly perfect representation with the step size set at 1. But when the step size is set at 2, we can now see a bit more noise in both of the modes. The distribution still stays relatively accurate though. When the step size set at 4, we can see a noticeable amount of noise in both of the modes, but the sampling distribution is still relatively accurate compared to our target distribution.

Through this example, I hope you are now familiar with the effects of different step sizes on a Metropolis algorithm and the relationship between the importance of a good step size and the number of iterations you have to run the simulation.

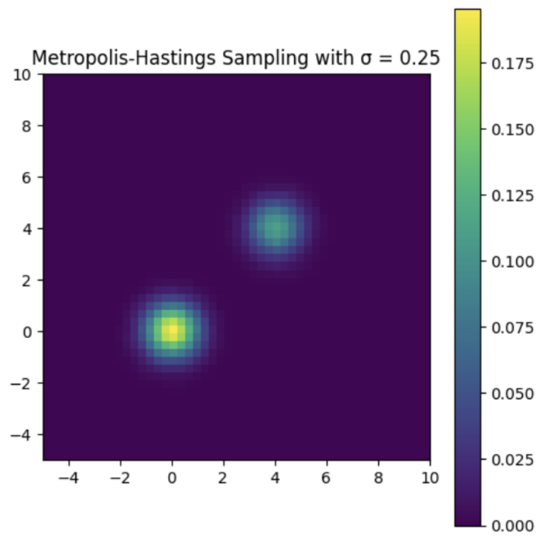

Figure 12. Probability density distribution with step size of 0.25 with 500,000 iterations

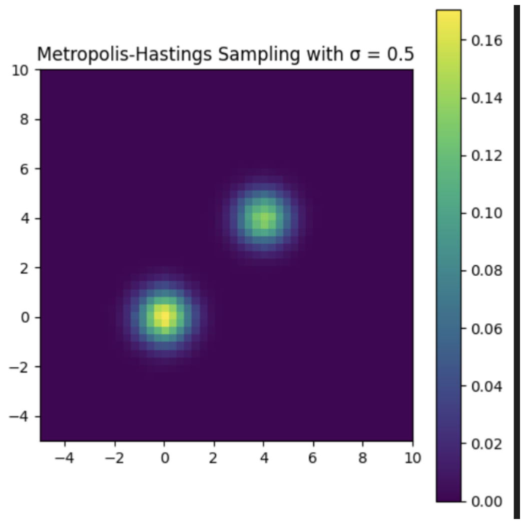

Figure 13. Probability density distribution with step size of 0.5 with 500,000 iterations

Remark 6.3: The Importance of a Good Starting Value. Many times in a Metropolis algorithm the starting value of the simulation is often far away from where the target distribution is actually concentrated at. Due to this, given a finite number of iterations in the simulation, it would not be wrong to consider the initial bias caused by this set back. To deal with this bias, a burn-in period is implemented, where the first N samples being discarded, such that N is large enough that the chain has reached its stationary regime by this time. Let's see an example of this:

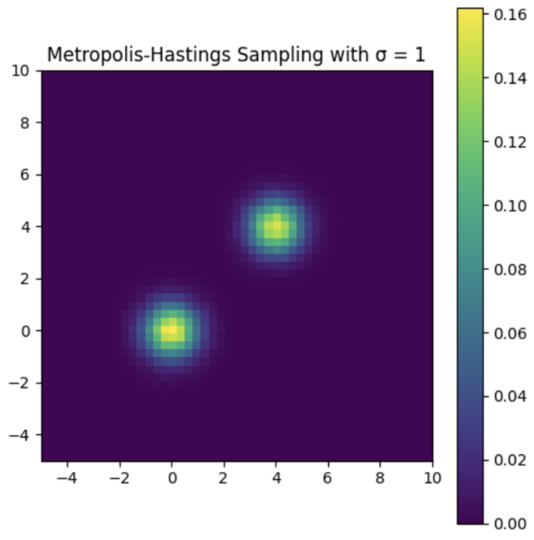

Figure 14. Probability density distribution with step size of 1 with 500,000 iterations

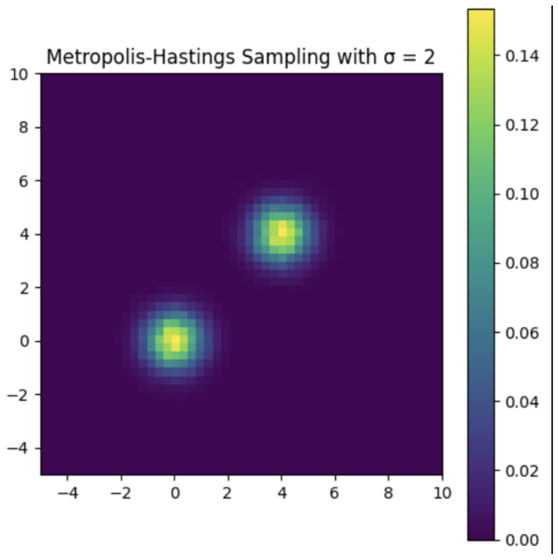

Figure 15. Probability density distribution with step size of 2 with 500,000 iterations

Example 6.4: Burn-in Example. Let's try to sample from a Gaussian distribution centered at x = 3:

$$f(x) = \frac{1}{0.5\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-3}{0.5}\right)^2\right)$$

For our first simulation, let us use a bad starting value of x = -2, where we know our target distribution is not concentrated at. For our second simulation, let us use a good starting value of x = 3, where we know our target distribution is concentrated at. Our step size and the number of iterations for both simulations are kept constant.

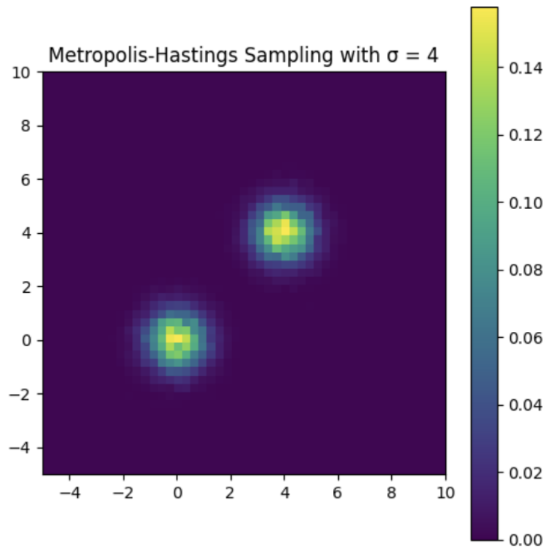

Figure 16. Probability density distribution with step size of 4 with 500,000 iterations

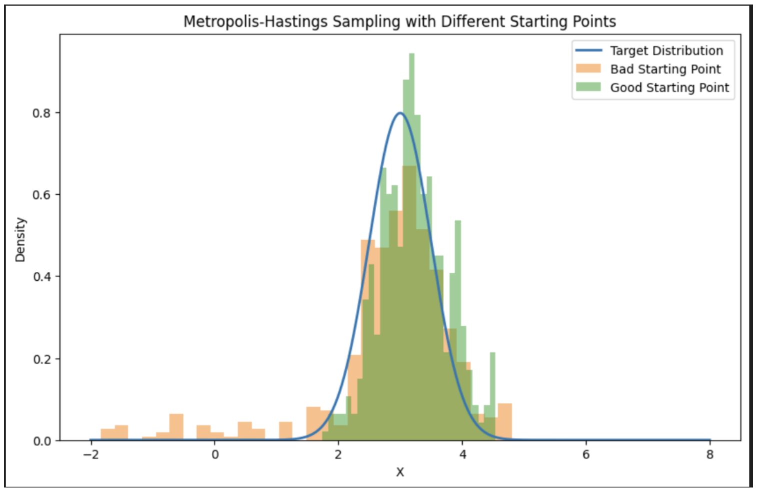

Figure 17. Sampling distributions with different starting points

As you can see from Figure 17, the simulation with a good starting point has a much more accurate representation of the target distribution. Keep in mind though, for most simulations done you will not know where your target distribution is concentrated at, making the burn-in period crucial. Let's see try to find a burn-in period for our bad starting point simulation.

In order to estimate a good burn-in period for our simulation, we must look at the time-series of our simulations.

As you can see from Figure 18 and Figure 19 there is significant lag for the first several iterations of the simulation when we started far from the true mean of the target distribution compared to when we started closer to the true mean. We can already estimate a decent value for our burn-in period size, but let us get a more accurate number using an Autocorrelation plot.

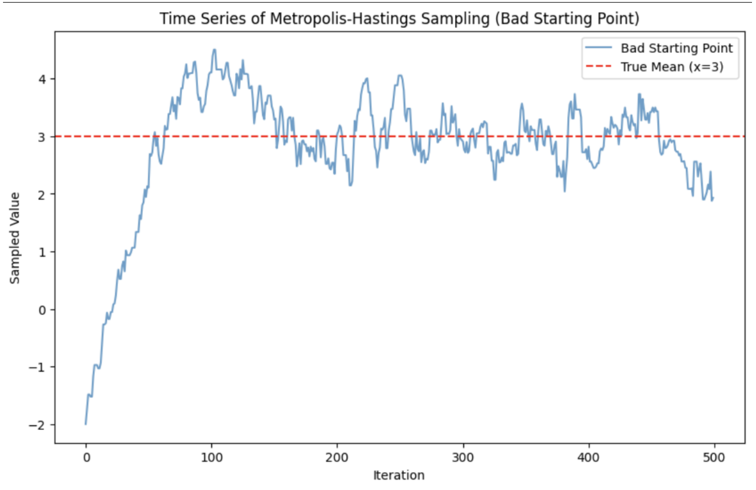

Figure 18. Time series for our bad starting point simulation

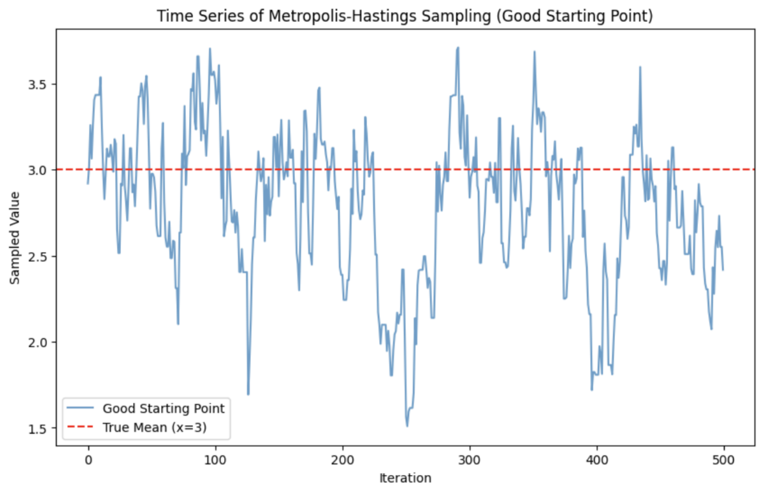

Figure 19. Time series for our good starting point simulation

Looking at our autocorrelation plots (Figure 20), the bad starting point simulation decreases in autocorrelation relatively slowly, meaning that the samples of the respective simulations are correlalated with each other. This indicates that the chain is not mixing well. We must now look for the point in the autocorrelation plot where the autocorrelation values begin to stabilize or plateau. This may be a little tough due to our small number of iterations, but $x = 325$ would not be a bad estimate for this value. Now if you rerun this algorithm with the same starting point you know to discard the samples up to iteration number 325 in order to minimize the bias of your simulation.

Definition 6.5: Effective Sample Size For Metropolis Algorithm Another approach to determine the sample size required to run a Metropolis algorithm is to obtain the autocorrelation values at different lags in order to calculate an effective sample size (ESS). This ESS factors out the effect of autocorrelation, calculating the number of independent samples

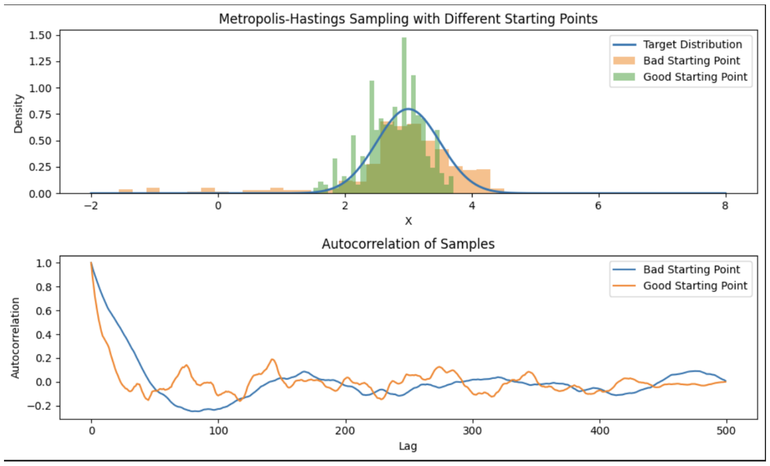

Figure 20. Autocorrelation plots with both simulations

that you have in your simulation. This is the formula for ESS:

$$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} ACF(k)}$$

where $ACF(k)$ is the autocorrelation value at lag k.

Since we can not calculate an infinite sum, it is typical to only calculate the sum up to the point where $ACF(k)$ is small enough, in many cases $ACF(k) < 0.05$. If you are using the Metropolis algorithm to determine a statistic of a distribution that depends on the less dense parts of the distribution, an ESS value of 10,000 is adequate. For a statistic that depends on the more dense parts of the distribution, like a median, then the ESS of your simulation can be much smaller.

Since the standard error of the mean for a random sample of size n is $\frac{s}{\sqrt{n}}$, given s is the standard deviation of the sample, we can simply replace n with the ESS to get the Markov Chain Standard Error (MCSE):

$$MCSE = \frac{s}{\sqrt{ESS}}$$

where s is the standard deviation of the sample produced by the Metropolis algorithm. The $MCSE$ will give an estimate of the accuracy of the estimated mean of the distribution that is calculated using the Metropolis algorithm.

7. Further Questions

This paper goes over the basics of improving the efficiency of Metropolis Algorithm simulations. If you want to go more in depth into improving the efficiency of your simulations it would be wise to look into the rates of convergence of your algorithms. Reading the paper *Exact Convergence Analysis of the Independent Metropolis-Hastings Algorithms* by Guanyang Wang would be a good place to start. You should also learn how to use Python's matplotlib to generate the autocorrelation plots, time series, etc, you will need to analyze the efficiency of your simulations. I used matplotlib to generate the plots visualizing efficiency in many of this paper's examples.

Markov Chains and the Metropolis Algorithms have many potential applications. In statistical physics and chemistry, Markov Chains and the Metropolis Algorithm are used to simulate the behavior of particles in various different physical systems, one being the Ising model for ferromagnetism. In statistics, the Metroplis Hastings Algorithm is also used to help approximate posterior distributions of parameters in Bayesian models. Markov Chains are also used to simulate the spread of contagious diseases in the field of Epidemiology. Markov Chains and the Metropolis Algorithms have more applications in many different fields, such as machine learning, financial modelling and image processing.

Acknowledgements

The author would like to thank Professor Rubinstein-Salzedo for many helpful discussions.

References

[1] Daniel T. Kaplan. *Introduction to the Metropolis-Hastings Algorithm*. Online resource. Year, e.g., 2023. URL: https://stephens999.github.io/fiveMinuteStats/MH_intro.html.

[2] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092.

[3] Aaron Plavnick. "The Fundamental Theorem Of Markov Chains". In: (2008). Online document. URL: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2008/REUPapers/Plavnick.pdf.