



# Markov Chain Monte Carlo (MCMC) Sampling

 Date: 02-23-2023  
 Time: 20:35

Bipin Koirala

## Table of Contents

- 1. [Motivation](#)
- 2. [Markov Chain Monte Carlo \(MCMC\)](#)
  - 1. [Monte Carlo](#)
    - 1. [On Expected Value and Convergence](#)
  - 2. [Combining Markov Chains and Monte Carlo](#)
    - 1. [Metropolis-Hasting Sampling](#)
    - 2. [Code](#)

## Motivation

We're all familiar with Bayes Theorem which powers many inference models. If we want to estimate the parameter  $\theta$  of a distribution based on the data, we have the following:

$$\mathbb{P}(\theta|data) = \frac{\mathbb{P}(data|\theta) \mathbb{P}(data)}{\mathbb{P}(data)}$$



where,  $\mathbb{P}(data) = \int_{\theta} \mathbb{P}(data|\theta) \mathbb{P}(data) d\theta$ . Suppose there are three unknown parameters  $\alpha, \beta$  and  $\gamma$  for a distribution, then the integral we need to evaluate across the search-space of these parameters becomes:

$$\mathbb{P}(\alpha, \beta, \gamma|data) = \int_{\gamma} \int_{\beta} \int_{\alpha} \mathbb{P}(data|\theta) \mathbb{P}(\alpha, \beta, \gamma) d\alpha d\beta d\gamma \tag{1}$$

These types of integrals are not always analytically tractable. Therefore we need to resort to sampling based technique to estimate the posterior distribution.

## Markov Chain Monte Carlo (MCMC)

This is where MCMC comes in and the main idea is to draw samples large enough so that we can approximate the posterior distribution. This trick avoids the need to explicitly compute multi-dimensional integral (1). These simulation algorithms are called [#Markov-Chain-Monte-Carlo](#) methods and have boosted Bayesian Statistics.

 **Note** 

**Markov Chain:** It is a stochastic model that describes a sequence of events, where the probability of each event depends only on the state of the previous event. They are especially useful for modeling systems that evolve over time, such as weather patterns or stock prices.

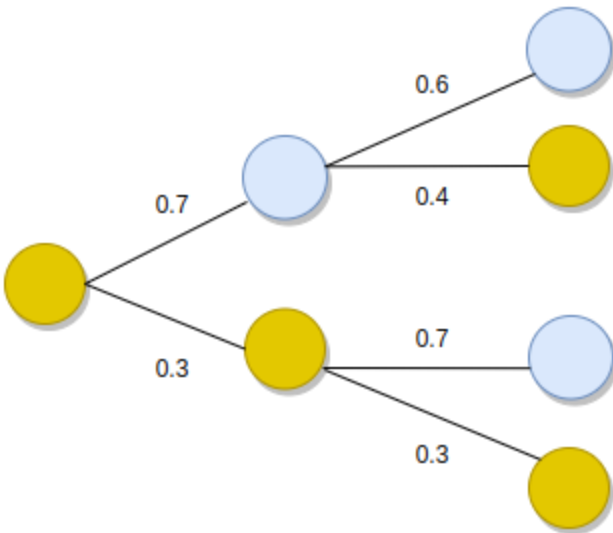


Fig: Markov Chain Model. Colored balls represent state of a system while the edges show the transition probability to the next state.

## Monte Carlo

[#Monte-Carlo](#) integration is a simulation technique to calculate integral as shown in (1). For example, suppose we can to evaluate a 1-D integral;

$$F = \int_a^b f(x) dx \quad (2)$$

The above integral can be approximated by averaging samples of the function  $f(\cdot)$  at uniform random points between the interval  $a$  and  $b$ . Let,  $X_i \sim \text{Unif}(a, b)$ . Then [#Monte-Carlo](#) estimate of the integral is given by;

$$\hat{F}_N = \frac{(b-a)}{N-1} \sum_{i=1}^N f(X_i) \quad (3)$$

Random variable  $X_i$  can be generated using  $X_i = a + \lambda_i(b-a)$  where  $\lambda_i \in \text{Unif}(0, 1)$  is the canonical random number uniformly distributed between 0 and 1. Intuitively, this can be thought of averaging the area of rectangle to get the true area under the curve.

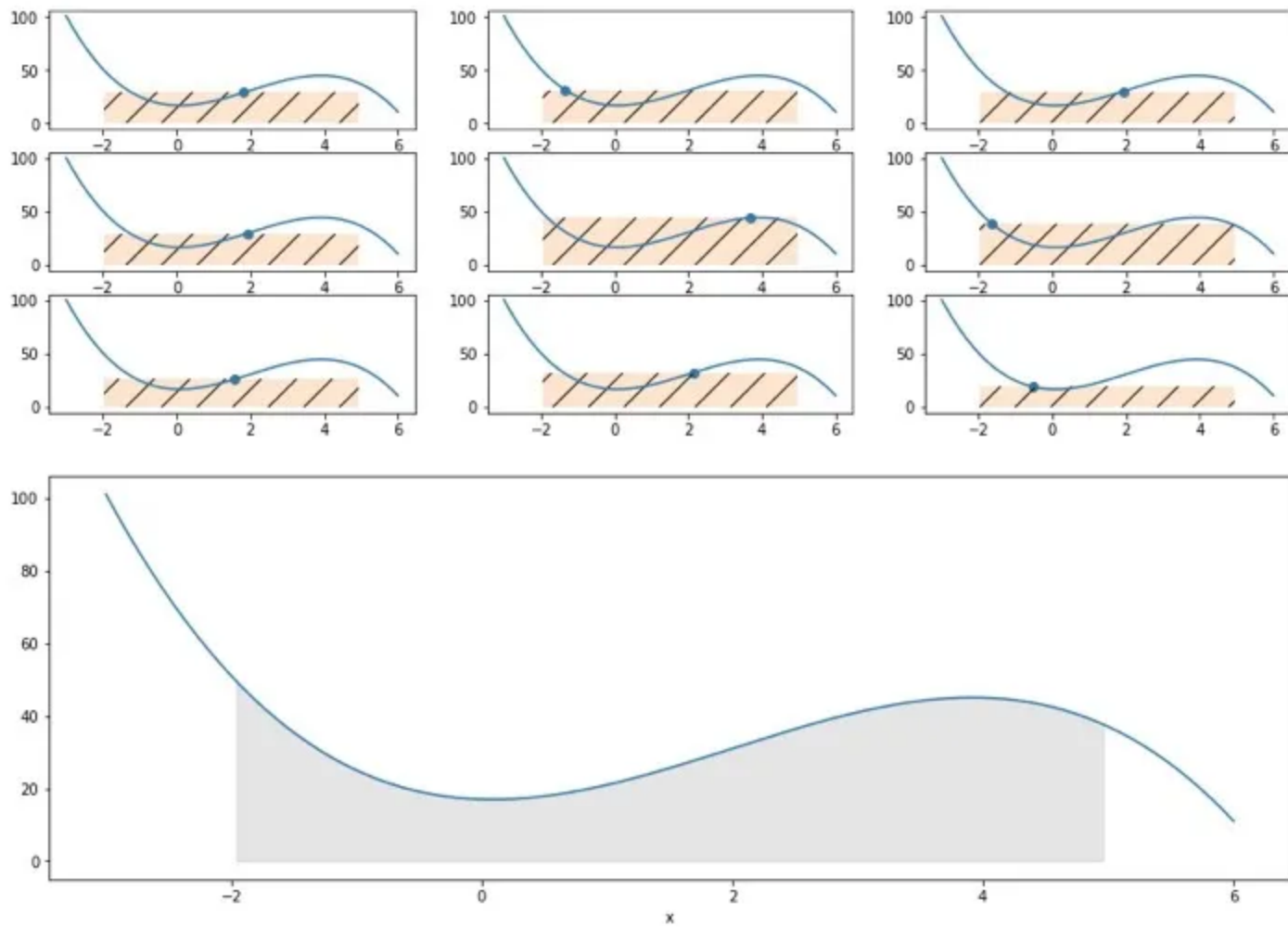


Fig: Monte Carlo estimate in action.

There are many applications where integrations must be performed over many dimensions. For example, phase-space integrals over position and momentum are six dimensional. [#Monte-Carlo](#) integration can be generalized to use random variables sampled from arbitrary PDFs and to compute multi-dimensional integrals such as;

$$F = \int_{\mu(x)} f(x) d\mu(x) \quad (4)$$

In this case, the empirical estimate is given with a slight modification to (3);

$$\hat{F}_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{\text{pdf}(X_i)} \quad (5)$$

## On Expected Value and Convergence

It can be showed that the expected value of  $\hat{F}_N$  is in fact  $F$  i.e.  $\hat{F}_N$  is an unbiased estimator of  $F$ .

$$\begin{aligned} \mathbb{E}[\hat{F}_N] &= \mathbb{E}\left[\frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{\text{pdf}(X_i)}\right] \\ &= \frac{1}{N} \sum_{i=1}^N \mathbb{E}\left[\frac{f(X_i)}{\text{pdf}(X_i)}\right] \\ &= \frac{1}{N} \sum_{i=1}^N \int \frac{f(x)}{\text{pdf}(x)} \text{pdf}(x) dx \\ &= \frac{1}{N} \sum_{i=1}^N \int f(x) dx \\ &= \int f(x) dx \\ \therefore \mathbb{E}[\hat{F}_N] &= F \end{aligned} \quad (6)$$

As we increase the number of samples  $N$ , the estimator  $\hat{F}_N$  gets closer to  $F$ . Due to the Strong Law of Large Numbers, in the limit we can guarantee that we have the exact solution.

$$\mathbb{P}\left(\lim_{N \rightarrow \infty} \hat{F}_N = F\right) = 1 \quad (7)$$

Using the [#Central-Limit-Theorem](#), we can approximate the distribution of the Monte Carlo estimate as a normal distribution with mean  $F$  and variance  $\sigma^2/N$ , where  $\sigma^2$  is the variance of the integrand  $f(x)$  i.e.

$$\hat{F}_N \sim \mathcal{N}\left(F, \frac{\sigma^2}{N}\right) \tag{8}$$

Thus, the standard error of the Monte Carlo estimate can be expressed as:

$$SE = \frac{\sigma}{\sqrt{n}} \tag{9}$$

where  $\sigma$  is the standard deviation of the integrand.

The convergence rate of `#Monte-Carlo` integration is related to the rate at which the standard error decreases as the number of samples increases. From the equation above, we can see that the standard error decreases as the square root of the number of samples used. This means that the convergence rate is relatively slow, particularly for high-dimensional integrals, where a large number of samples may be required to obtain an accurate estimate.

The rate of convergence of Monte Carlo integration is related to the smoothness of the integrand. If the integrand is smooth, meaning it does not vary rapidly, the convergence rate can be faster. On the other hand, if the integrand is irregular or has discontinuities, the convergence rate can be slower.

To improve the convergence rate, we can use variance reduction techniques such as importance sampling or stratified sampling. These techniques involve generating samples in a way that concentrates the sampling in regions where the integrand is expected to be large, which can reduce the variance and improve the convergence rate.

#### Summary

Overall, the convergence of Monte-Carlo integration is determined by a combination of factors, including:

- Number of samples used
- Smoothness of the integrand
- Use of variance reduction techniques

## Combining Markov Chains and Monte Carlo

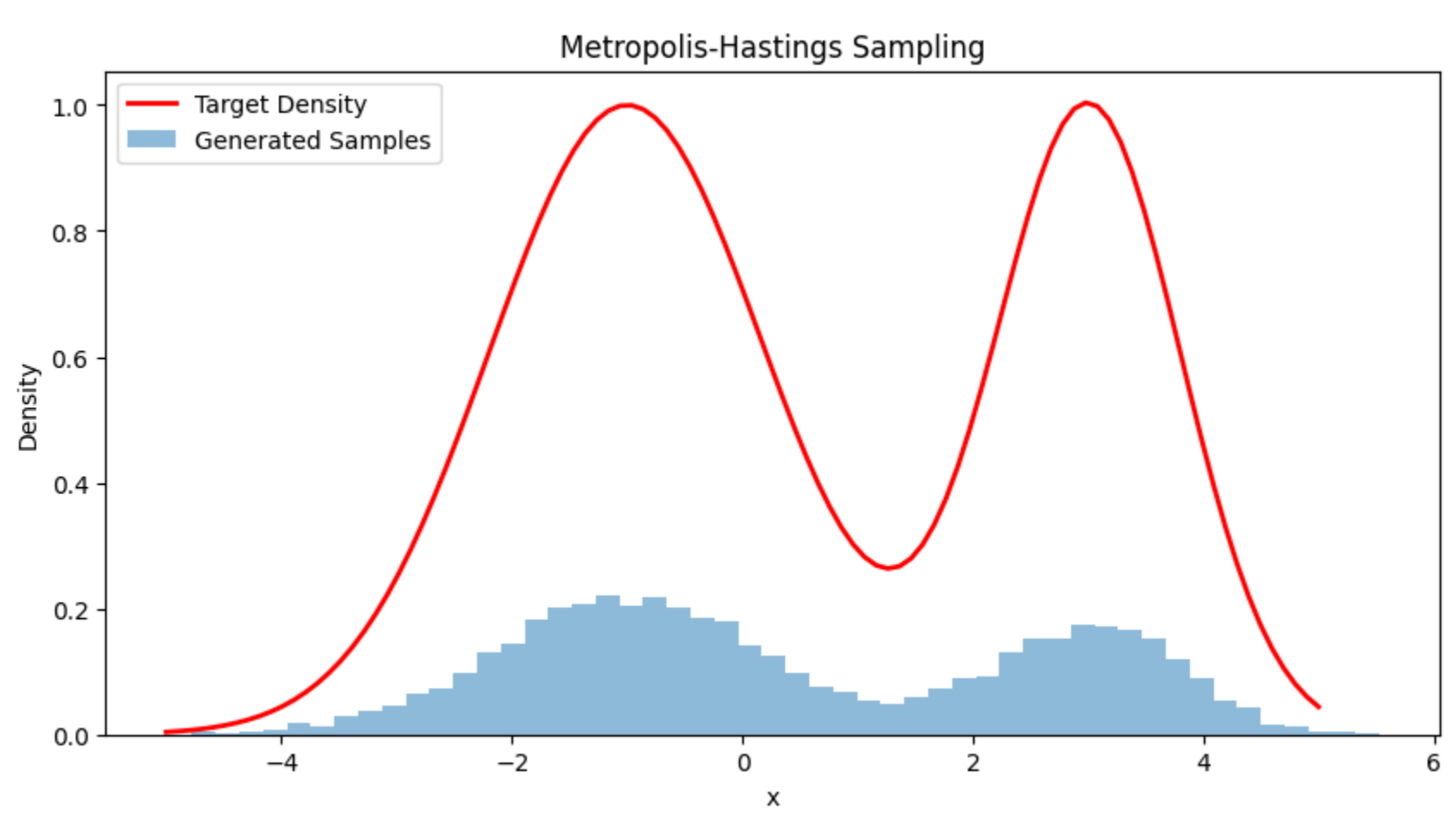
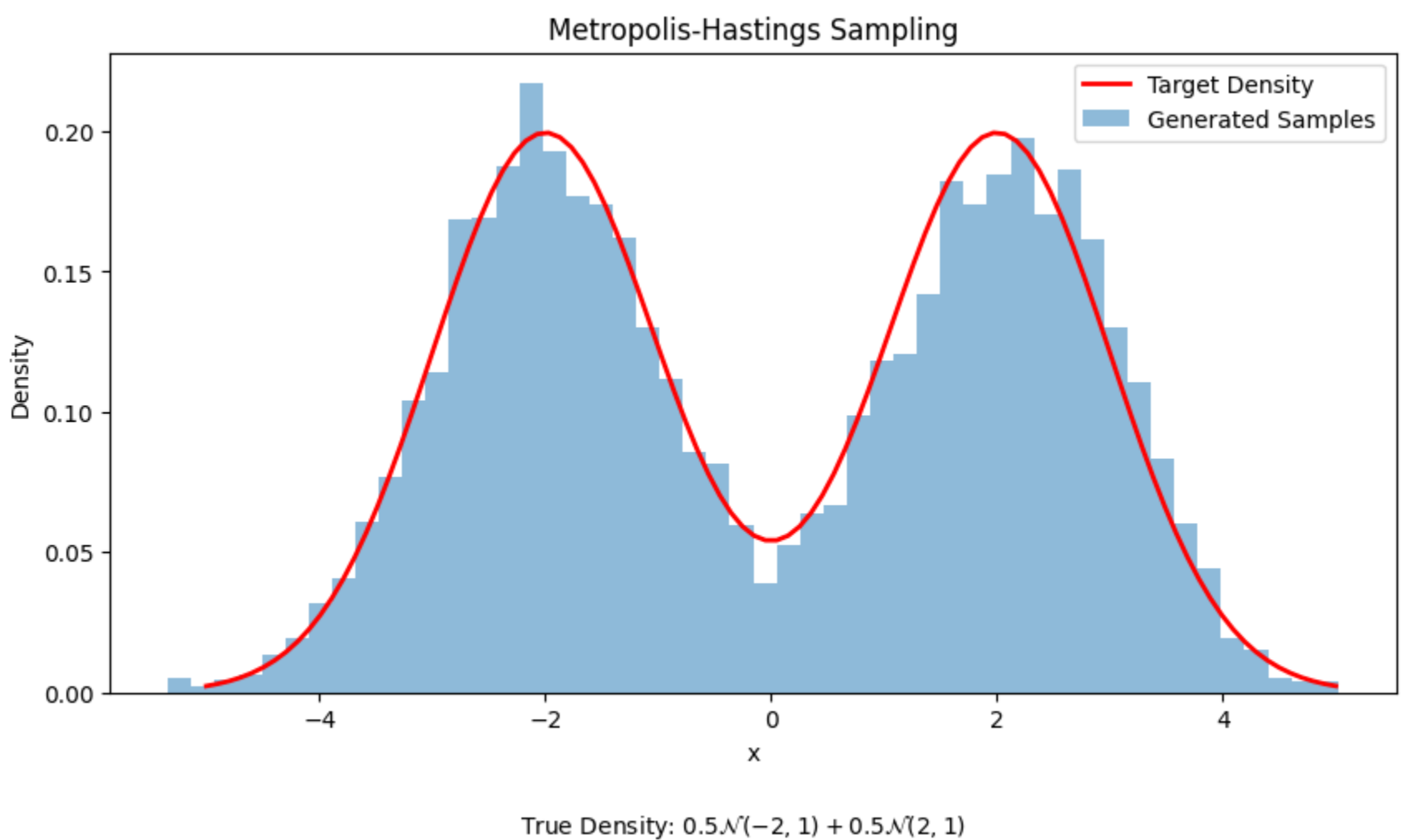
### Metropolis-Hasting Sampling

#### Algorithm

- Initialize,  $X_1 = x_1$  as current state
- $\forall t = 1, 2, \dots$  do:
  - sample proposed state  $x_{t+1}$  from  $Q(x_{t+1}|x_t)$
  - $\alpha = \min\left(1, \frac{\pi(x_{t+1})}{\pi(x_t)} \frac{Q(x_t|x_{t+1})}{Q(x_{t+1}|x_t)}\right)$
  - if  $\alpha \geq \text{Unif}\sim(0,1)$ :  
 $x_t = x_{t+1}$  i.e. current state = proposed state  
 else:  
 $x_t = x_t$  i.e. current state = current state

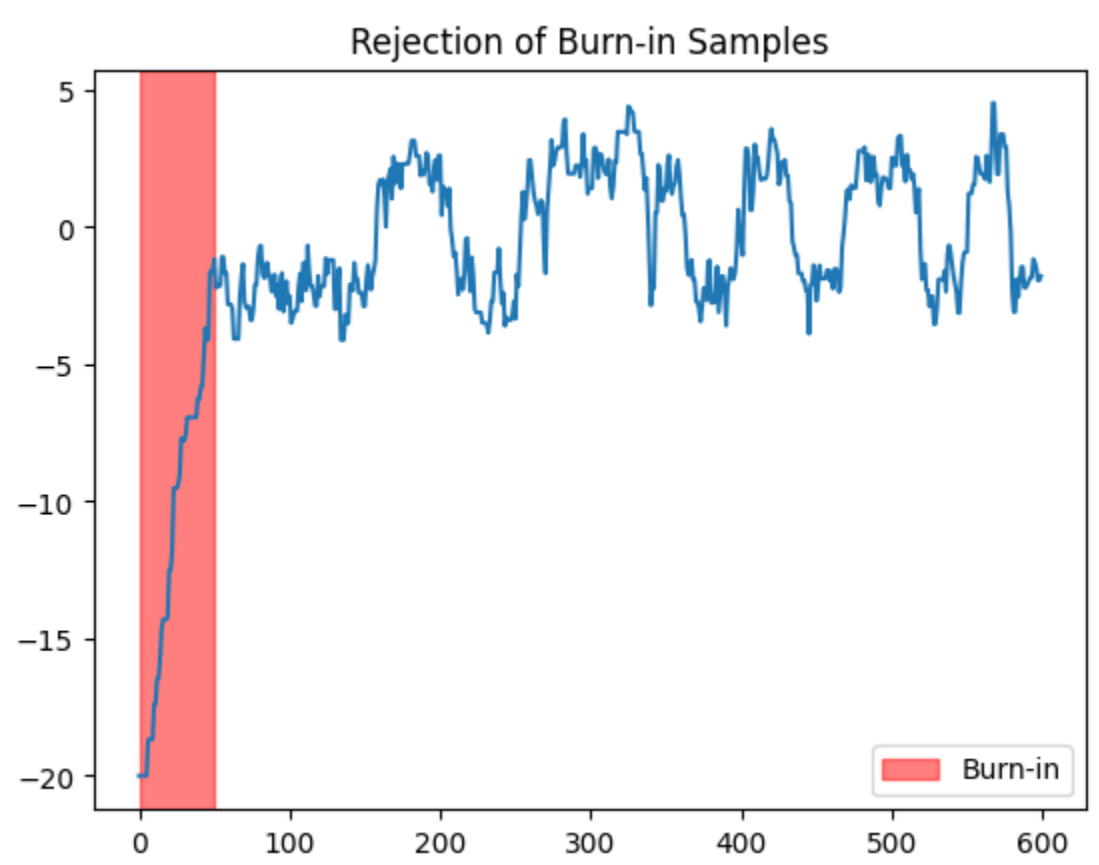
When the proposed distribution  $Q$  is symmetric the hasting ratio

$$\frac{Q(x_t|x_{t+1})}{Q(x_{t+1}|x_t)} = 1 \tag{10}$$



The figure above has true target distribution as:  $\exp\left(-\frac{1}{2}\left(\frac{x-3}{0.8}\right)^2\right) + \exp\left(-\frac{1}{2}\left(\frac{x-1}{0.2}\right)^2\right)$

Furthermore, the samples take some iterations to reach 'stationary' distribution. In practice we reject samples in the beginning of the sampling process. These discarded samples are referred to as the 'burn-in' samples. Below is a visual representation of the phenomenon which typically arises if the initial state is chosen very far from the target distribution.



```
import numpy as np

def metropolis_hastings(target_density, proposal_density, num_samples, initial_state, burn_in=1000):

    """
    Metropolis-Hastings algorithm for sampling from a target density using a proposal density.

    Parameters
    -----
    target_density : function
    The target probability density function.
    proposal_density : function
    The proposal probability density function.
    num_samples : int
    The number of samples to generate.
    initial_state : float or array_like
    The initial state of the Markov chain.
    burn_in : int, optional
    The number of samples to discard at the beginning of the chain as burn-in.

    Returns
    -----
    samples : array_like
    The generated samples.
    acceptance_rate : float
    The acceptance rate of the Metropolis-Hastings algorithm.
    all_samples : float
    """

    samples = [initial_state]
    all_samples = [initial_state]
    num_accepted = 0

    current_state = initial_state

    for i in range(num_samples + burn_in):
        proposed_state = proposal_density(current_state)

        hasting_ratio = proposal_density(current_state) / proposal_density(proposed_state)
        metropolis_ratio = target_density(proposed_state) / target_density(current_state)
        ratio = metropolis_ratio*hasting_ratio

        acceptance_prob = min(ratio, 1)

        if np.random.rand() < acceptance_prob:
            current_state = proposed_state
            num_accepted += 1

        if i >= burn_in:
            samples.append(current_state)

        all_samples.append(current_state)

    acceptance_rate = num_accepted / num_samples
    return samples, acceptance_rate, all_samples
```

```
def gaussian(x):
    return 0.5*(1 / np.sqrt(2 * np.pi * 1**2) * np.exp(-(x + 2)**2 / (2 * 1**2))) + 0.5 *(1 / np.sqrt(2 * np.pi * 1**2) * np.exp(-(x - 2)**2 / (2 * 1**2)))
    #return np.exp(-0.5 * ((x - 3) / 0.8)**2) + np.exp(-0.5 * ((x + 1) / 1.2)**2)

def proposal(x):
    return np.random.normal(x, 1)

target = lambda x: gaussian(x)
proposal = proposal

samples, acceptance_rate, all_samples = metropolis_hastings(target, proposal, 10000, initial_state=0)

print(f"Acceptance rate: {acceptance_rate:.3f}")
print(f"Mean: {np.mean(samples):.3f}")
print(f"Standard deviation: {np.std(samples):.3f}")
```

```
import matplotlib.pyplot as plt
```

```
# Generate samples
samples, acceptance_rate, all_samples = metropolis_hastings(target, proposal, 10000, initial_state=0)

# Plot true target density
x = np.linspace(-5, 5, 100)
plt.figure(figsize=(10,5))
plt.plot(x, target(x), color='red', linewidth=2)

# Plot histogram of samples
plt.hist(samples, bins=50, density=True, alpha=0.5)

plt.xlabel('x')
plt.ylabel('Density')
plt.title('Metropolis-Hastings Sampling')
plt.legend(['Target Density', 'Generated Samples'])
plt.show()
```