Mini Project 3: MCMC for Geometric Brownian Motion Model Mathematical Foundation for Scientific Computing FTN0469

Li Ju

November 30, 2023

1 Introduction

The Geometric Brownian Motion (GBM) model, a cornerstone in the field of financial mathematics, is extensively employed for modeling the dynamics of various financial instruments, particularly stock prices. GBM was popularized by the seminal work of Osborne [1959], who first proposed modeling stock prices as Brownian motion. GBM is characterized by a continuous stochastic process in which the logarithm of the asset price follows a Brownian motion with drift, encapsulated by the stochastic differential equation

$$dX_t = \mu X_t dt + \sigma X_t dW_t$$

where X_t represents the value at time t, μ the drift coefficient, σ the volatility coefficient, and dW_t the increment of a Wiener process.

Metropolis-Hastings algorithm is one the most popular tools for the estimation of parameters for the GBM model. This algorithm, a classic example of Markov Chain Monte Carlo (MCMC) methods, addresses the inherent complexity encountered in Bayesian inference for stochastic processes. Due to the non-linear and non-Gaussian nature of GBM, neither direct sample nor analytical solutions of the posterior distribution of parameters like μ and σ is feasible. However, the Metropolis-Hastings algorithm facilitates the generation of a sequence of sample values from the posterior distribution, even when its form is unknown or complicated to compute directly. By iteratively proposing new parameter values and accepting or rejecting them based on a calculated acceptance ratio, this algorithm constructs a Markov chain that converges to the desired posterior distribution, enabling efficient and effective parameter estimation [Hastings, 1970].

In this project, we implement the Bayesian inference for parameters of the GBM model using the Metropolis-Hastings algorithm. We discuss implementation details in Section 2 and report experiment results in Section 3. The main codes are deferred to the appendix and full codes can be found at GitHub.

2 Implementation

2.1 Data Generaiton

The stochastic differential equation for geometric Brownian motion (GBM) model is given by

$$dX_t = \mu X_t dt + \sigma X_t dW_t \tag{1}$$

From Itô's Lemma, we have the analytical solution for GBM model as

$$X_t = X_0 \exp\left((\mu - \frac{1}{2}\sigma^2)t + \sigma W_t\right) \tag{2}$$

With $x_0 = 1$, $\mu = 0.1$, $\sigma = 0.3$, and $T = 100.0$, $N = 1000$ samples are generated from Equation 2. Here $\mu = 0.1$ and $\sigma = 0.3$ are both moderate values to provide significant enough randomness. With $N = 1000$,

a significant number of data points can allow for more precise estimation of parameters. Here $\Delta t = 0.1$ is small enough to capture the short-term variations due to volatility $\sigma = 0.3$ while still being large enough to avoid noise that might be present in high-frequency data. The generated data is denoted by $D = \{x_i\}_{i=1}^N$.

2.2 Likelihood Function

The discritized form for the analytical solution of GBM model is given by

$$X_{t+\Delta t} = X_t \exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}Z\right) \tag{3}$$

where Z denotes a standard gaussian variable and Δt denotes the sampling rate.

Then we have

$$\ln\left(\frac{X_{t+\Delta t}}{X_t}\right) = (\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}Z \tag{4}$$

$$\sim \mathcal{N}((\mu - \frac{1}{2}\sigma^2)\Delta t, \sigma^2\Delta t) \tag{5}$$

For samples $\{X_i\}_{i=1}^N$ with sampling rate Δt, we have the following *negative* log likelihood function:

$$-\mathcal{L}(\mu, \sigma) = -\ln\left(\prod_{i=1}^{N-1} \frac{1}{\sigma\sqrt{2\pi\Delta t}} \exp\left(-\frac{(r_i - (\mu - \frac{1}{2}\sigma^2)\Delta t)^2}{2\sigma^2\Delta t}\right)\right) \tag{6}$$

$$= (N-1)\ln\sigma + \frac{1}{2\sigma^2\Delta t}\sum_{i=1}^{N-1}(r_i - (\mu - \frac{1}{2}\sigma^2)\Delta t)^2 + \text{const.} \tag{7}$$

where $r_i \coloneqq x_{i+1} - x_i, i \in \{N-1\}$.

2.3 Maximum Likelihood Estimation

First we test the log-likelihood function with the data with maximum likelihood estimator $\hat{x} = \arg\max_x \mathcal{L}(x)$. We solve the optimizaiton problem numerically with Newton's method. The implementation is deferred to Appendix A.1. To Tackle the accuracy of maximum likelihood estimation, for a fixed time interval $\Delta t = 0.1$, we increase the number of samples N from 100 to 10000.

2.4 Bayesian Inference

With the likelihood function, then we infer posterior distributions for $\hat{\mu}$ and $\hat{\sigma}$, applying Metropolis-Hasting algorithm.

Prior Distributions In this problem, μ and σ are our target parameters to be inferred. For $p(\mu)$, a natural choice is the standard gaussian, i.e. $p(\mu) = \mathcal{N}(0, 1)$. For $p(\sigma)$, considering that σ is non-negative, we use gamma distribution with $\alpha = 1$ and $\beta = 1$, denoted by $p(\sigma) = \mathcal{G}(1, 1)$.

Proposal Probability Here we choose the Gaussians for the proposal probability for μ: $q(\mu_2|\mu_1) = \mathcal{N}(\mu_1, 10^{-2})$ and log gaussian for σ: $q(\sigma_2|\sigma_1) = \mathcal{LG}(\sigma_1, 10^{-4})$.

For σ, the gaussian distribution is symmetric and unbounded, allowing μ to take any real value. The variance of 0.01 indicates the scale of steps around the current value of μ_1. This can ensure local exploration yet explore the parameter space with a proper trade-off.

For μ, the log gaussian is used (i.e. $p(\ln\sigma_2|\sigma_1) = \mathcal{N}(\ln\sigma_1, 10^{-4})$. Since σ must be positive, the log gaussian proposal distribution inherently respects this constraint and can also explore the parameter space efficiently.

Also, it's beneficial to have a symmetric proposal distribution for σ and μ to simplify the acceptance ratio calculation, since they canceled out for $x_1 \to x_2$ and $x_2 \to x_1$ and we only need to compare $\pi(x_1)$ and $\pi(x_2)$, where $\pi(x) \propto p(x|D)$.

Acceptance Criterion The Metropolis-Hastings acceptance criterion is given by

$$\alpha(\theta_1, \theta_{2c}) = \min\left(1, \frac{\pi(\theta_{2c})q(\theta_1|\theta_{2c})}{\pi(\theta_1)q(\theta_{2c}|\theta_1)}\right) \tag{8}$$

where $\pi(\theta) = p(\theta)\exp(\mathcal{L}(\theta))$.

In the context of this problem, this criterion can simplified as follows:

$$\ln \alpha(\mu_1, \sigma_1, \mu_{2c}, \sigma_{2c}) = \min\left(0, \underbrace{\ln p(\mu_{2c}) + \ln p(\sigma_{2c}) + \mathcal{L}(\mu_{2c}, \sigma_{2c})}_{\ln p(\mu_{2c}, \sigma_{2c}|D)} - \underbrace{(\ln p(\mu_1) + \ln p(\sigma_1) + \mathcal{L}(\mu_1, \sigma_1))}_{\ln p(\mu_1, \sigma_1|D)}\right)$$

2.5 Algorithm

With the likelihood function, prior and proposal distributions defined, we can implement the Metropolis-Hastings algorithm to infer parameters of the geometric Brownian motion model. The pseudo code of the algorithm is given as follows. The implementation of the method is deferred to Appendix A.2.

Algorithm 1 Metropolis-Hastings for Estimating μ and σ

Compute log returns of data as R
Set number of iterations $I_{\max}$
Initialize μ^0 and σ^0
for $i = 1$ to $I_{\max}$ **do**
    $\mu' \leftarrow$ sample from $\mathcal{N}(\mu^{i-1}, 10^{-2})$ ▷ Propose a new μ
    $\sigma' \leftarrow$ sample from $\mathcal{LN}(\sigma^{i-1}, 10^{-4})$ ▷ Propose a new σ
    $L' \leftarrow \mathcal{L}(\mu', \sigma', R)$ ▷ Compute log likelihood of (μ', σ')
    $L \leftarrow \mathcal{L}(\mu^{i-1}, \sigma^{i-1}, R)$ ▷ Compute log likelihood of $(\mu^{i-1}, \sigma^{i-1})$
    $P'_\mu \leftarrow \text{NormLogPdf}N(\mu'|0, 1)$ ▷ Compute log pdf of μ prior for μ'
    $P'_\sigma \leftarrow \text{GammaLogPdf}N(\sigma'|1, 1)$ ▷ Compute log pdf of σ prior for σ'
    $P_\mu \leftarrow \text{NormLogPdf}N(\mu|0, 1)$ ▷ Compute log pdf of μ prior for μ
    $P_\sigma \leftarrow \text{GammaLogPdf}N(\sigma|1, 1)$ ▷ Compute log pdf of σ prior for σ
    $a \leftarrow \exp(L' + P'_\mu + P'_\sigma - L - P_\mu - P_\sigma)$ ▷ Compute the probability of accepting the new sample
    $p \leftarrow$ random uniform number between 0 and 1
    if $p < a$ **then**
        $\mu^i \leftarrow \mu'$
        $\sigma^i \leftarrow \sigma'$ ▷ Move
    else
        $\mu^i \leftarrow \mu^{i-1}$
        $\sigma^i \leftarrow \sigma^{i-1}$ ▷ Stay
    end if
end for

3 Results & Discussion

In this section, we report our experiment results, including the estimation with maximum likelihood estimator, discussion of the measurement, compare the posterior samples with different numbers of collected data points, and the choices of prior distributions and proposal distributions.

3.1 MLE Results

First we estimate the target parameters μ and σ of the GBM model with maximum likelihood estimation and the results are visualized in Figure 1. As observed, the estimation error for μ exhibits a marked decrease as the number of data points increases from $N = 10^2$ to 10^4, after which the error stabilizes and shows

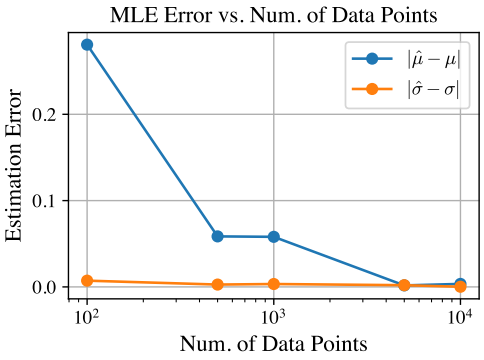

Figure 1: Estimation error of the parameters $\hat{\mu}$ and $\hat{\sigma}$ using Maximum Likelihood Estimation (MLE) as a function of the number of data points.

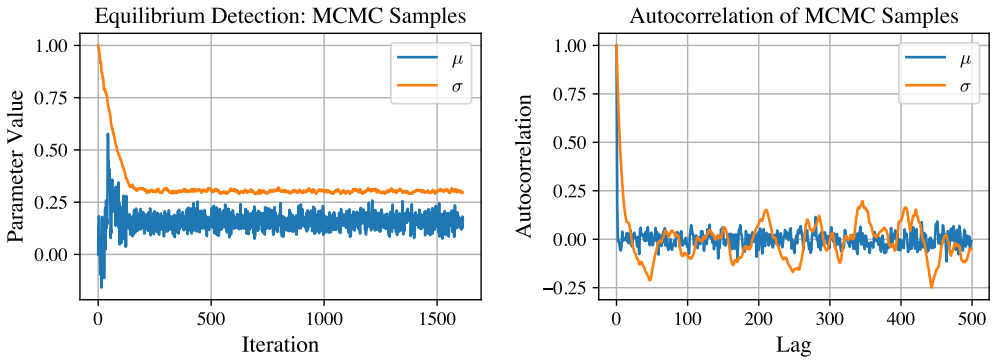

Figure 2: Analysis of MCMC samples. **Left:** Convergence of parameter estimates. **Right:** Autocorrelation in MCMC samples for GBM parameter estimation.

minimal improvement. In contrast, the estimation error for σ remains consistently low across the number of data points, demonstrating a robust estimation of the volatility coefficient even with fewer data points. Overall, the MLE process appears to provide precise estimations for both parameters, particularly when the sample size is sufficiently large.

3.2 Measurement

In the analysis of the Markov Chain Monte Carlo samples for Bayesian inference of the GBM model parameters, two key aspects were investigated: the detection of equilibrium in the sampling process and the evaluation of sample autocorrelation. The results for each study are visualized in Figure 2 on the left and the right panel, respectively.

The left panel showcases the convergence behavior of the parameters μ and σ over iterations. As can be seen, both parameters reach a state of equilibrium within approximately 300 iterations (burn-in period). The right panel presents the autocorrelation of the MCMC samples for both parameters. The autocorrelation for μ drops sharply within 50 lags and oscillates around zero thereafter, indicating low serial correlation and suggesting that samples become effectively independent at a relatively short lag. The autocorrelation for σ demonstrates a similar pattern, with a slightly higher degree of serial correlation initially, yet also tending towards zero.

The obtained results affirm the validity of our implementation and provide insights into the burn-in period and the requisite number of samples for accurate parameter inference within the specified GBM model framework.

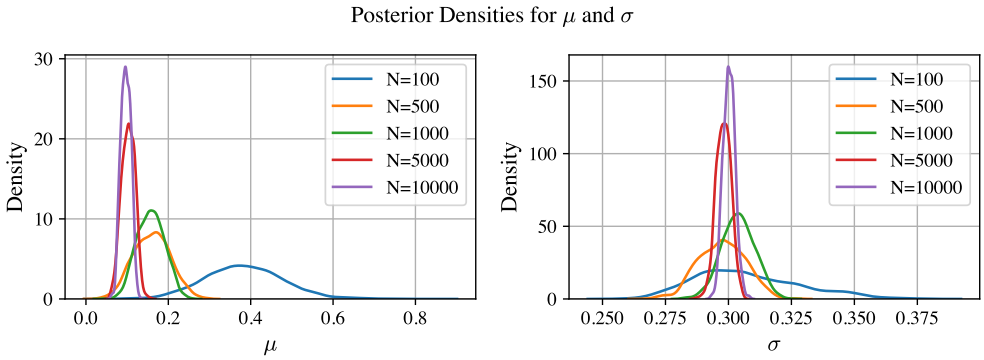

Figure 3: Kernel density estimates of posterior distributions for μ and σ with varying sample sizes.

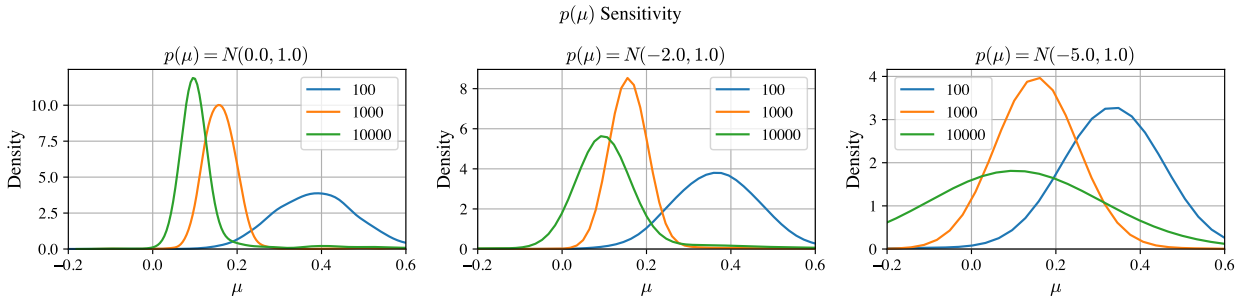

Figure 4: Sensitivity of posterior distributions for μ to different prior distributions across sample sizes.

3.3 Posterior Distribution

The Kernel Density Estimates (KDE) of the posterior distributions for the GBM model parameters μ and σ are illustrated in Figure 3, with the number of data points increasing from $N = 100$ to 10^4.

It is observed that the distributions for both parameters become increasingly concentrated around the true values of $\mu = 0.1$ and $\sigma = 0.3$, as the number of data points grows. Notably, with $N = 10^4$, the posterior of μ tightly clusters around the true value, signifying a high degree of estimation confidence. A similar refinement is observed for σ, where the posterior density shows a marked peak at the true value with the largest sample size, indicating a significant enhancement in the precision of the parameter's estimation. These trends underscore the important role of sample size in achieving precise and confident parameter estimation within Bayesian inference frameworks.

3.4 Choosing Prior and Proposal Distributions

Then we tackle the effects of the inference for different choices of prior distributions and proposal distributions.

Prior Three different prior distributions for μ is chosen as $p_1(\mu) = \mathcal{N}(0, 1)$, $p_2(\mu) = \mathcal{N}(-2, 1)$ and $p_3(\mu) = \mathcal{N}(-5, 1)$, to represent prior distributions of different degrees of correctness, given that our true parameter is known as $\mu = 0.1$. The experiments have been repeated with different numbers of data points with $N \in \{100, 1000, 10000\}$.

The first plot demonstrates the influence of a prior centered around the true value, $\mathcal{N}(0, 1)$, which yields posteriors that rapidly converge towards $\mu = 0.1$ as the sample size increases, illustrating a high degree of alignment between the prior beliefs and the true parameter value. In contrast, the second and third plots reveal the impact of priors centered further from the true value, $\mathcal{N}(-2, 1)$ and $\mathcal{N}(-5, 1)$, respectively. With

these priors, the posteriors are initially biased towards the prior means; however, as the number of data points grows, the posteriors shift towards the true μ, with the largest sample size showing a significant movement towards 0.1. These results illustrate the importance of prior selection in Bayesian inference, particularly when dealing with smaller datasets, as the prior can substantially influence the posterior distribution, potentially leading to biased estimates if the prior is not well-calibrated to the true parameter values.

Proposal Distribution To tackle the effectss of different proposal distributions for the parameter σ, three proposal distributions were examined: Uniform $\mathcal{U}(0,1)$, Gamma $\mathcal{G}(1,1)$, and log gaussian $\mathcal{LG}(0,10^{-4})$.

It is observed that all proposal distributions provided a similar estimation expectation for σ and μ, indicating the robustness of the choices of proposal distribution. However, there exists significant differences of acceptance rate, ranging from 0.001 to 0.149. This experiment highlights the impact of the choice of proposal distribution on the efficiency of the sampling process and ultimately on the precision of parameter estimation.

Prop. Dist.	Acc. Rate	Estimation Expectation
$\mathcal{U}(0,1)$	0.001	$(0.104, 0.298)$
$\mathcal{G}(1.0,1.0)$	0.020	$(0.101, 0.299)$
$\mathcal{LG}(0,10^{-4})$	0.149	$(0.102, 0.299)$

Table 1: Comparison of acceptance rates and estimation expectations for different proposal distributions.

3.5 Conclusion

In conclusion, we implement Metropolis-Hastings algorithm to do Bayesian inference for parameters of GBM model and report the experiment results. We analyzed the equilibrium and autocorrelation in MCMC samples and examined the effects of prior and proposal distribution choices on posterior estimates. These experiments collectively provide evidence supporting the reliability of Bayesian methods and the MCMC algorithms for parameter estimation in GBM models.

References

Maury FM Osborne. Brownian motion in the stock market. *Operations research*, 7(2):145–173, 1959.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

A Code

A.1 MLE Estimation

```
# script to apply MLE to synthetic GBM data
import jax.numpy as jnp
import jax
from jax import jit, grad, vmap
import matplotlib.pyplot as plt


def get_data(params, delta_t, n, key):
    mu, sigma = params[0], params[1]
    z = jax.random.normal(key, shape=(n,))
    x = 1.0
    ret = [x]
    for i in range(1, n):
        inc = (mu-0.5*(sigma**2))*delta_t + sigma*jnp.sqrt(delta_t)*z[i]
        x *= jnp.exp(inc)
        ret.append(x)
    return jnp.array(ret)


# define loss function for a single data point
def loss_fn_sample(params, r, delta_t):
    mu, sigma = params[0], params[1]
    t1 = jnp.log(sigma)
    t2 = (r-(mu-0.5*(sigma**2))*delta_t)**2 / (2*(sigma**2)*delta_t)
    return t1+t2


# vectorize the loss function
loss_fn_samples = vmap(loss_fn_sample, in_axes=(None, 0, None))


@jax.jit
# compute the mean loss
def loss_fn(params, rs, delta_t):
    return jnp.mean(loss_fn_samples(params, rs, delta_t))


grad_fn = grad(loss_fn)


# define a newton step
@jax.jit
def newton_step(rs, params, delta_t):
    grad = grad_fn(params, rs, delta_t)
    hess = jax.hessian(loss_fn)(params, rs, delta_t)
    params -= jnp.linalg.inv(hess)@grad
    loss = loss_fn(params, rs, delta_t)
    return params, loss
```

```
# True Parameters
true_params = jnp.array([0.1, 0.3])
delta_t = 0.1

# Generating the GBM data
key = jax.random.PRNGKey(0)
key, subkey = jax.random.split(key)


ns = [100, 500, 1000, 5000, 10000]
errors = []

for n in ns:
    data = get_data(true_params, delta_t, n, key=subkey)
    # compute the log returns
    log_returns = jnp.log(data[1:]/data[:-1])

    # initialize the parameters
    params = jnp.array([0.2, 0.2])
    num_steps = 50

    # run the newton steps
    for i in range(num_steps):
        params, loss = newton_step(log_returns, params, delta_t)

    # compute estimation error
    error = jnp.abs(params-true_params)
    errors.append(error)

errors = jnp.array(errors)
# plot the results
plt.plot(ns, errors[:, 0], '-o', label='$|\hat{\mu}-\mu|$')
plt.plot(ns, errors[:, 1], '-o', label='$|\hat{\sigma}-\sigma|$')
plt.xlabel('Num. of Data Points')
plt.ylabel('Estimation Error')
plt.title('MLE Error vs. Num. of Data Points')
plt.xscale('log')
plt.legend()
plt.savefig('gbm_mle.pdf', dpi=300)
plt.close()
```

A.2 Bayesian Inference

```
# script to apply Bayesian inference to synthetic GBM data
import jax.numpy as jnp
import jax
from jax import jit, vmap
from jax.scipy.stats import norm, gamma


def get_data(params, delta_t, n, key):
    mu, sigma = params
    z = jax.random.normal(key, shape=(n,))
    x = 1.0
```

```
    ret = [x]
    for i in range(1, n):
        inc = (mu-0.5*(sigma**2))*delta_t + sigma*jnp.sqrt(delta_t)*z[i]
        x *= jnp.exp(inc)
        ret.append(x)
    return jnp.array(ret)


# define the prior logpdfs
def prior_logpdf(params):
    mu, sigma = params
    mu_logpdf = norm.logpdf(mu, 0.0, 1.0)
    sigma_logpdf = gamma.logpdf(sigma, 1.0)
    return mu_logpdf + sigma_logpdf


# define the loglikelihood
def loglikelihood_point(params, log_return, delta_t):
    mu, sigma = params
    t1 = -jnp.log(sigma)
    t2 = -(log_return-(mu-0.5*(sigma**2))*delta_t)**2 / (2*(sigma**2)*delta_t)
    return t1+t2


# vectorize the loglikelihood function
loglikelihood_points = vmap(loglikelihood_point, in_axes=(None, 0, None))


# define the sum of loglikelihoods
def loglikelihood(params, log_returns, delta_t):
    ret = jnp.sum(loglikelihood_points(params, log_returns, delta_t))
    return ret


# define the posterior logpdf
def posterior_logpdf(params, log_returns, delta_t):
    ret = prior_logpdf(params) + \
        loglikelihood(params, log_returns, delta_t)
    return ret


def compute_probability(new_params, params, log_returns, delta_t):
    log_alpha = posterior_logpdf(new_params, log_returns, delta_t) - \
        posterior_logpdf(params, log_returns, delta_t)
    return jnp.exp(log_alpha)


@jit
def mcmc_step(params, log_returns, delta_t, key):
    muk, sigmak, uk = jax.random.split(key, 3)
    # propose a step

    new_mu = jax.random.normal(muk)*0.1 + params[0]
    new_sigma = params[1] * jnp.exp(jax.random.normal(sigmak)*0.01)
```

```
    new_params = jnp.array([new_mu, new_sigma])

    # compute the acceptance probability
    alpha = compute_probability(
        new_params, params, log_returns, delta_t)

    # accept or reject
    u = jax.random.uniform(uk)
    return jax.lax.cond(
        u < alpha,
        lambda x: (new_params, True),
        lambda x: (params, False), None)


def mcmc(params, log_returns, delta_t, key, num_steps):
    samples = [params]
    keys = jax.random.split(key, num_steps)
    for idx, key in enumerate(keys):
        params, accepted = mcmc_step(
            params, log_returns, delta_t, key)
        if accepted:
            samples.append(params)
    return jnp.array(samples)


def main():
    # True Parameters
    true_params = jnp.array([0.1, 0.3])
    delta_t = 0.1
    n = 5000

    # Generating the GBM data
    key = jax.random.PRNGKey(0)
    key, subkey = jax.random.split(key)

    data = get_data(true_params, delta_t, n, key=subkey)

    # compute the log returns
    log_returns = jnp.log(data[1:]/data[:-1])

    num_steps = int(1e4)
    # start from expectation of prior
    start = jnp.array([0., 1.])

    key, subkey = jax.random.split(key)
    samples = mcmc(start, log_returns,
                   delta_t, subkey, num_steps)

    burn_in = 500
    samples = samples[burn_in:]
    print(f"Expectation of Posterior: {jnp.mean(samples, axis=0)}")
```

```
if __name__ == "__main__":
    main()
```