STAT 37830 Group Project Checkpoint Report

Paris Hsu, Yifu Wang, Weixin Wang

November 21, 2025

1 Introduction

Monte Carlo methods are fundamental tools for computing integrals that are difficult or impossible to evaluate analytically [2, 3]. Their performance, however, depends critically on the structure of the target distribution and on the design of the sampling algorithm [1]. In problems with strong nonlinearity or multimodality, standard sampling methods may converge slowly, produce highly biased estimates, or fail entirely. This project investigates these issues by comparing four representative Monte Carlo approaches on a single, intentionally difficult test problem: integration over the 1D double-well potential.

The double-well model provides an ideal setting for this study. Its potential function,

$$V(x) = (x^2 - 1)^2, \tag{1}$$

induces a bimodal target distribution

$$p(x; \beta) \propto e^{-\beta V(x)}, \tag{2}$$

with two symmetric modes at $x = \pm 1$. This structure, as shown in Figure 1, creates a large energy barrier that becomes increasingly sharp as β grows, making the model a canonical example where Markov chain Monte Carlo (MCMC) algorithms struggle with mode-sticking and slow mixing, as we will see later in Section 5.2. At the same time, the symmetry of the distribution guarantees that $E[x] = 0$ for all β, giving us a convenient and exact ground truth for quantifying bias.

Using this problem as a unifying benchmark, we systematically evaluate four sampling methods of increasing sophistication:

1. Naive Monte Carlo, which serves as a simple baseline;
2. Importance Sampling, used to explore how proposal distributions affect variance and stability;
3. Metropolis–Hastings, which is expected to fail at high β due to poor mixing between modes;
4. Parallel Tempering, an advanced MCMC technique designed specifically to overcome multimodality.

Our goal is to understand not only how each method performs but why it behaves the way it does. We measure bias, variance, convergence behavior, and computational cost across a range of difficulty settings determined by β. The final product will be a unified, extensible Python toolkit that implements all four methods, along with an analysis module for computing expectations, assessing accuracy, and visualizing sampler behavior.

Through this focused study, we aim to highlight the strengths and limitations of widely used Monte Carlo techniques and to illustrate how problem structure, algorithm design, and tuning choices interact to determine sampling efficiency.

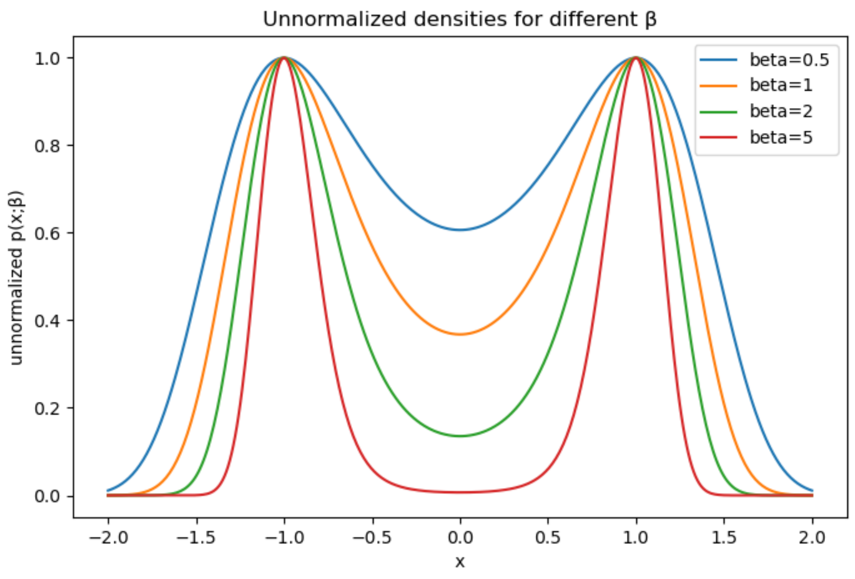

Figure 1: Unnormalized potential function $e^{-\beta V(x)}$ for different β's.

2 Algorithms

2.1 Classical Monte Carlo

Classical Monte Carlo (MC) integration provides a simple baseline for estimating expectations with respect to a target distribution. Given a function $h(x)$ and a distribution with density $f(x)$, s.t. $\int f(x)dx = 1$, the goal is to approximate

$$\mathbb{E}_f[h(X)] = \int h(x)f(x)\,dx. \tag{3}$$

The Classical Monte Carlo method draws N independent samples $X^{(1)}, \dots, X^{(N)}$ from the target distribution and computes the empirical average

$$\hat{I}_N = \frac{1}{N}\sum_{n=1}^{N} h\left(X^{(n)}\right). \tag{4}$$

By the Law of Large Numbers, $\hat{I}_N$ converges to the true expectation as $N \to \infty$, and the estimator has variance $\mathrm{Var}(h(X))/N$.

In our implementation, the user provides two components: a sampling function that returns i.i.d. draws from the target distribution, and a test function $h(x)$ whose expectation is to be estimated. The `estimate` method computes the Monte Carlo estimator $\hat{I}_N$. This method provides a baseline against which we compare the performance of more sophisticated sampling algorithms.

2.2 Importance Sampling

In many problems, drawing samples directly from the target distribution is difficult or computationally expensive. Importance Sampling (IS) provides a way to estimate expectations with respect to a target density $p(x)$ by instead sampling from a more convenient proposal distribution $q(x)$.

Our goal is to compute the expectation

$$\mathbb{E}_p[h(X)] = \int h(x)\, p(x)\, dx, \tag{5}$$

but when sampling from p is impractical, we rewrite the integral by multiplying and dividing by $q(x)$:

$$\mathbb{E}_p[h(X)] = \int h(x)\, \frac{p(x)}{q(x)}\, q(x)\, dx = \mathbb{E}_q[\, h(Y)\, w(Y)\,]\,, \tag{6}$$

where q is a distribution relatively eaiser to sample from, and the importance weight is defined as

$$w(y) = \frac{p(y)}{q(y)}. \tag{7}$$

This identity shows that we can approximate the target expectation using samples $Y^{(1)}, \ldots, Y^{(N)}$ drawn from the proposal distribution:

$$\hat{I}_{\mathrm{IS}} = \frac{\sum_{n=1}^{N} h(Y^{(n)})\, w(Y^{(n)})}{\sum_{n=1}^{N} w(Y^{(n)})}, \tag{8}$$

which is the self-normalized Importance Sampling estimator. This formulation has the advantage that it does not require the normalization constant of $p(x)$.

Moreover, importance sampling can achieve significantly smaller variance than standard Monte Carlo when the proposal distribution is well aligned with the integrand. By reallocating samples toward high-contribution regions, its variance can be bounded in terms of the divergence between the target and proposal, often giving much tighter bounds than the usual $O(1/N)$ rate [7,8].

In our implementation, the user specifies a proposal sampler and its density $q(x)$, along with the unnormalized target density $p(x)$ and test function $h(x)$. The algorithm generates samples from q, computes the corresponding importance weights, and returns weighted estimates of the desired expectation. A companion routine also provides an approximate variance estimate based on the normalized weights. Importance Sampling serves as both a flexible integration tool and a diagnostic for understanding how the choice of proposal distribution affects estimator variance and stability.

2.3 Metropolis-Hastings (MCMC)

The Metropolis-Hastings (M-H) algorithm is a foundational **Markov Chain Monte Carlo (MCMC)** method. M-H generates a chain of correlated samples that, in the long run, is guaranteed to settle into and accurately represent the target distribution, $p(x)$. The core principle ensuring this convergence is called *Detailed Balance.*

At each time step t, the chain evaluates a possible update: it may move to a newly proposed state x', or it may simply remain at its current location x_t. This "propose–then–accept" mechanism allows the algorithm to explore the state space while automatically correcting for biases introduced by the proposal distribution.

The corresponding pseudocode is detailed in Algorithm 1.

2.3.1 The MCMC Trap

This M-H setup applied to the double-well potential presents a canonical **MCMC trap**. When the difficulty parameter β is high, the energy barrier at $x = 0$ becomes exponentially massive. If the chain starts in one mode (e.g., $x \approx 1$), the probability of proposing a step large enough to cross the barrier to $x \approx -1$ is negligible. The chain becomes **stuck in a single mode**, failing to achieve **ergodicity** (the ability to visit all regions). This results in a highly biased estimate; the true value of $E[x]$ is 0, but a stuck chain will erroneously report $E[x] \approx 1$.

Algorithm 1 One step of Metropolis–Hastings

```
1: Input: current state $x_t$, step size $\sigma$
2: Propose candidate $x' = x_t + \mathcal{N}(0, \sigma^2)$
3: Compute acceptance probability:
       $\alpha = \min\left(1, \frac{p(x';\beta)}{p(x_t;\beta)}\right)$
4: Draw $u \sim U[0, 1]$
5: if $u < \alpha$ then
6:     $x_{t+1} \leftarrow x'$
7: else
8:     $x_{t+1} \leftarrow x_t$
9: end if
10: Output: $x_{t+1}$
```

2.4 Parallel Tempering

Parallel tempering, also known as replica exchange Monte Carlo, is studied in the double-well potential, one of two simple free-energy landscapes [4]. By generating hot replicas(low β) and swap them, it can effectively sample multimodal distributions, overcoming the common failure of traditional MCMC methods.

Similarly, we provide the detailed pseudocode in Algorithm 2.

Parallel tempering runs several Metropolis–Hastings chains at different inverse temperatures $\beta_1 < \cdots < \beta_n$. Hot chains explore the state space more freely, while the coldest chain targets the true distribution. At each iteration, every chain performs a Metropolis–Hastings update, followed by a possible swap of states between two *neighboring* temperatures. The swap probability preserves detailed balance across chains, allowing information from the well-mixing hot chains to propagate down to the cold chain. Samples from the coldest chain then approximate the target distribution.

3 Package Structure

Structure of the `src` Package The `src` directory contains the core sampling algorithms used throughout the project. At this stage, two methods have been implemented in `src/scripts.py`:

- **ClassicalMonteCarlo**: A direct Monte Carlo estimator that computes expectations of the form

  $$\mathbb{E}_f[h(X)] = \int h(x)f(x)\,dx$$

  by drawing i.i.d. samples from the target distribution via a user-provided sampler. The class exposes two methods:

  – `estimate(N)`: returns the empirical mean $\frac{1}{N}\sum_{n=1}^{N} h(X^{(n)})$,

- **ImportanceSampling**: An implementation of self-normalized importance sampling for an unnormalized target density $\tilde{p}(x)$. Given a proposal density $q(x)$, samples from q, and a test function $h(x)$, the estimator computes

  $$\hat{I}_{\text{IS}} = \frac{\sum_{n=1}^{N} h(Y^{(n)})\, w(Y^{(n)})}{\sum_{n=1}^{N} w(Y^{(n)})}, \quad w(y) = \frac{\tilde{p}(y)}{q(y)}.$$

  The class provides:

  – `estimate(N)`: returns the self-normalized IS estimator,

Algorithm 2 Parallel Tempering

```
1: Input: inverse temperatures $\beta_1 < \beta_2 < \cdots < \beta_n$, initial states $\{X_i^{(0)}\}_{i=1}^n$, sample size $N$.
2: for $t = 0, 1, \dots, N-1$ do
3:     for $i = 1, \dots, n$ do
4:         Generate $\widetilde{X}_i^{(t+1)}$ by performing one Metropolis–Hastings step
             with current state $X_i^{(t)}$, proposal kernel $q$, and target density $p_{\beta_i}$.
5:     end for
6:     Choose indices $\ell, m \in \{1, \dots, n\}$ with $\ell \neq m$ uniformly at random.
7:     for $i \neq \ell, m$ do
8:         $X_i^{(t+1)} \leftarrow \widetilde{X}_i^{(t+1)}$
9:     end for
10:    Attempt a swap between chains $\ell$ and $m$:
           $$\left(X_\ell^{(t+1)}, X_m^{(t+1)}\right) = \begin{cases} \left(\widetilde{X}_m^{(t+1)}, \widetilde{X}_\ell^{(t+1)}\right), & \text{with probability } a_{\ell,m}, \\ \left(\widetilde{X}_\ell^{(t+1)}, \widetilde{X}_m^{(t+1)}\right), & \text{with probability } 1 - a_{\ell,m}, \end{cases}$$
11:    where
           $$a_{\ell,m} := \min\left\{1, \frac{p_{\beta_\ell}(\widetilde{X}_m^{(t+1)})\, p_{\beta_m}(\widetilde{X}_\ell^{(t+1)})}{p_{\beta_m}(\widetilde{X}_m^{(t+1)})\, p_{\beta_\ell}(\widetilde{X}_\ell^{(t+1)})}\right\}.$$
12: end for
13: Output: the sample $\{X_n^{(t)}\}_{t=1}^N$ from the cold chain $\beta_n$.
```

- **Metropolis-Hastings (MCMC)**: We based our MCMC implementation structure on a guide provided by Mocz [5]. Located in `src/metropolis.py`, this is an implementation of the symmetric M-H sampler using a Gaussian proposal. This is our primary target for demonstrating sampling failure at high β. The module exposes two key routines:
  - `metropolis_hastings`: Generates the MCMC chain of correlated samples. This function is used to demonstrate the issue of high bias at large β values.
  - `tune_step_size`: Provides an automated pre-tuning routine to optimize the proposal's σ for a target acceptance rate (e.g., 40%), maximizing mixing efficiency.
- **Parallel-Tempering**: Located in `src/parallel_tempering.py`, this is an implementation of Parallel Tempering for *Metropolis-Hastings* algorithm (Note that this M-H may not be the same as previous one). This module consists of four functions:
  - `V and log_p`: define the potential function

    $$V(x) = (x^2 - 1)^2$$

    and the log version of target density function

    $$\log(P(x; \beta)) \propto -\beta V(x)$$
  - `generate_betas`: generates an array $\{\beta_{\min} = \beta_1, \beta_2, \dots, \beta_n = \beta_{\max}\}$, assuming that $\beta_i = 1/T_i$, where T_i is one of the temperatures.
  - `parallel_tempering`: apply parallel tempering to $\{\beta_1, \beta_2, \dots, \beta_n\}$ and generate X_i from the Markov Chain of β_n.

  Located in the `src/experiments_PT.ipynb`, it includes the code to evaluate the performance of Paralleling Tempering.

4 Verification and Testing

To ensure the reliability of our simulations, we used the `pytest` framework to write thorough unit tests, primarily focusing on proving that our samplers behave exactly as predicted by theory.

Classical Monte Carlo Tests (`test_for_classicalMC.py`) These tests ensure that the basic Monte Carlo estimator behaves consistently with theoretical expectations:

- **Mean and Second Moment Tests:** Using $X \sim \mathcal{N}(0,1)$, we verify that the empirical estimates of $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$ converge to their known analytical values. This confirms correct implementation of the sampler and averaging routine.
- **Variance Consistency Test:** We compute $\mathbb{V}[X]$ both directly and via the identity $\mathbb{V}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ and check that the estimates agree within sampling noise, ensuring internal numerical consistency.

Importance Sampling Tests (`test_for_IS.py`) These tests validate that the self-normalized importance sampling estimator is implemented correctly for an unnormalized double-well target density. We test the self-normalized Importance Sampling estimator on the double-well target density $p(x) \propto \exp(-\beta(x^2-1)^2)$ with $\beta = 1$. A Gaussian proposal $q = \mathcal{N}(0, 2^2)$ is used to draw samples and compute importance weights $w = p(x)/q(x)$.

- The test estimates the moments $\mathbb{E}[X]$, $\mathbb{E}[X^2]$, and $\mathbb{E}[X^3]$ to verify that the weighting and normalization are implemented correctly.
- A separate plug-in estimator for $\mathbb{V}[X]$ confirms the numerical stability of the variance computation under the IS framework.

Analysis Module Tests (`test_analysis.py`) These tests verify that our core measurement tools are correct:

- **Moment Calculation:** We verified that `compute_expectation` accurately calculates moments ($E[x^n]$) for both **unweighted** MCMC samples (simple average) and **weighted** Importance Samples (self-normalized mean). This confirms the function works for all three current samplers.
- **Metric Calculation:** We tested `calculate_metrics` with known inputs to confirm its ability to correctly compute the experimental **bias** and **sample variance** of an estimator across multiple runs.

Metropolis-Hastings Tests (`test_metropolis.py`) These tests focus on experimentally proving the **MCMC trap**.

- **Smoke Test:** We verified that the `metropolis_hastings` function executes cleanly and returns the correct number of samples after burn-in.
- **Trap Verification Test:** This is the key diagnostic test. We run the sampler with parameters known to cause failure ($\beta = 20.0$, small `step_size`, starting at $x = 1.0$) and assert two critical conditions:
  1. **Sanity Check ($E[x^2]$):** We assert that the estimate for $E[x^2]$ is near the true value (≈ 1.0). This proves the sampler is correctly finding the location of the peak.
  2. **Failure Check ($E[x]$):** We assert that the estimate for $E[x]$ is near 1.0 (and *not* 0.0). This proves that the chain is **stuck** in one mode, thus providing empirical proof of the "MCMC trap" phenomenon we intend to solve.
- **Tuner Plausibility Test:** We verified that the `tune_step_size` function successfully converges and returns a positive, plausible step size within the optimal 40-50% acceptance rate range.

Parallel Tempering Tests (Test_PT.py) It includes the functions that check whether the output of previous functions is what we want:

- `test_V`: check whether `V` output correctly.
- `test_log_p`: check whether `log_p` output correctly.
- `test_generate_betas`: check `generate_betas` whether generate an array with correct length(number of β_i), order and spacing.
- `test_parallel_tempering`: check `parallel_tempering` whether output an array with correct length (number of state t) and under correct distribution.

5 Preliminary Results

5.1 Classical Monte Carlo and Importance Sampling

The provided script implements two separate numerical experiments:

- A `ClassicalMonteCarlo` estimator for expectations under the standard normal distribution $N(0, 1)$;

  For the classical Monte Carlo component, the code samples directly from $N(0, 1)$ and computes empirical estimates of the mean and variance for increasing sample sizes N.

- A `ImportanceSampling` estimator for expectations under the unnormalized double-well target density

  $$p(x) \propto e^{-\beta(x^2-1)^2}. \tag{9}$$

  For the importance sampling component, the script draws samples from a Gaussian proposal distribution $q(x) = N(0, 2^2)$ and applies normalized importance weights to estimate the mean and variance of the double-well target distribution.

The experiment sweeps over a logarithmic grid of sample sizes $N \in [10^2, 10^5]$, records the corresponding estimates, and visualizes the behavior of these estimates as N increases, which is shown in Figure 2.

Taken together, these four panels illustrate how both classical Monte Carlo and importance sampling estimators improve with larger sample sizes, each for its respective target distribution.

As additional sampling modules are introduced, corresponding tests will be added to ensure correctness and compatibility across the package.

5.2 Metropolis-Hastings Sampling

In this numerical experiment, we sample from the unnormalized *double-well target density*

$$p(x) \propto e^{-\beta(x^2-1)^2}. \tag{10}$$

This distribution is bimodal with two local maxima (wells) centered at $x \approx \pm 1$, with the parameter β controlling the sharpness of the modes. The performance of the M-H sampler is highly dependent on its ability to traverse the energy barrier separating these modes, a phenomenon known as *mixing.*

As discussed in Section 1, the Metropolis–Hastings chain can become "trapped" when the energy barrier is sharp, leading to slow mixing or even stagnation. Below we illustrate two representative cases: In the first, when $\beta = 2$ is moderate, the chain can still transition between modes; In the second, for large $\beta = 20$, the chain becomes effectively stuck and fails to explore the state space.

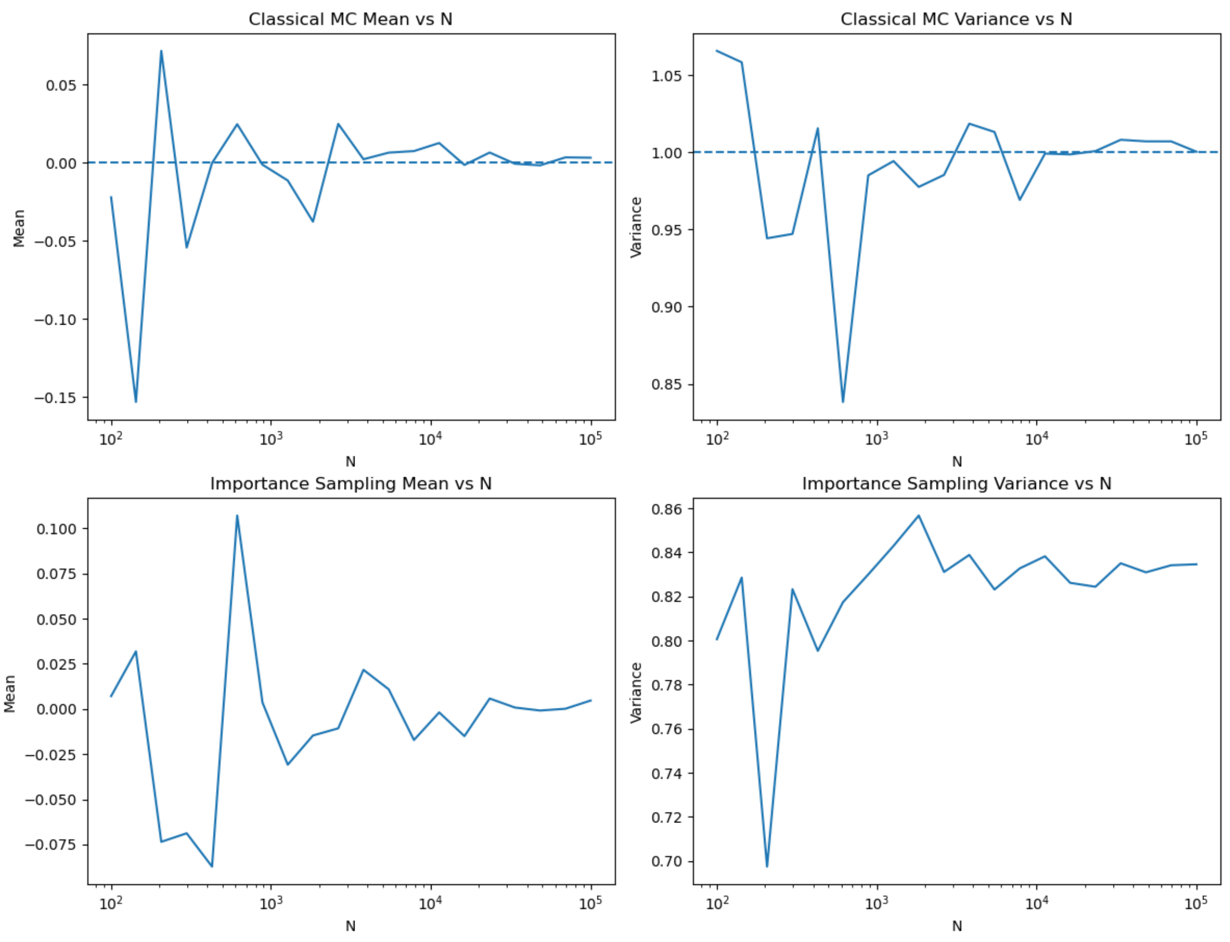

Figure 2: empirical mean and variance of classical Monte Carlo (top row) and importance sampling (bottom row) estimators w.r.t. the sample size.

5.2.1 The "Working" Chain ($\beta = 2$)

For a lower value of $\beta = 2.0$ and an auto-tuned step size, the M-H chain demonstrates **excellent mixing** (Figure 3).

- **Trace:** The chain (top plot) successfully explores both modes, frequently jumping across the barrier between $x \approx -1$ and $x \approx 1$.
- **Histogram:** The MCMC samples (blue histogram, bottom plot) closely match the True Density (red curve), accurately capturing the symmetric bimodal structure.
- **Moments:** The estimates for $\mathbb{E}[x]$ and $\mathbb{E}[x^3]$ correctly converge to zero, confirming the chain has adequately sampled both symmetric modes. All moment estimates are predicted correctly (Table 1).

5.2.2 The "Stuck" Chain ($\beta = 20$)

Table 2 summarizes the Monte Carlo estimates of several moments when the inverse temperature is increased to $\beta = 20$. Consistent with the discussion in Section 1, the Metropolis–Hastings chain exhibits *poor mixing*

Table 1: Experiment Results for $\beta = 2.0$

Moment	**Estimate**	**Naive SEM ($\pm$)**	**True Value**
$\mathbb{E}[x]$	0.0120	0.004131	0.0000
$\mathbb{E}[x^2]$	0.8532	0.002254	0.8521
$\mathbb{E}[x^3]$	0.0122	0.005118	0.0000
$\mathbb{E}[x^4]$	0.9821	0.004412	0.9771

at this large β, becoming trapped in a single mode and failing to explore the full state space. As a result, some moment estimates are badly biased.

Table 2: Moment Estimates of M-H Sampling for $\beta = 20.0$

Moment	**Estimate**	**Naive SEM ($\pm$)**	**True Value**	**Result**
$\mathbb{E}[x]$	0.9899	0.000362	0.0000	Stuck!
$\mathbb{E}[x^2]$	0.9865	0.000711	0.9870	Correct
$\mathbb{E}[x^3]$	0.9893	0.001059	0.0000	Stuck!
$\mathbb{E}[x^4]$	0.9984	0.001416	0.9995	Correct

When the step size is restricted to a small value ($\delta = 0.1$), the chain becomes stuck in the positive mode for the entire duration of the simulation, failing to cross the high-energy barrier separating the two modes, as shown in Figure 4.

- **Trace:** Starting at $x_0 = 1.0$, the chain (top plot) becomes *stuck* in the positive mode for the entire duration of the simulation, failing to cross the now-higher energy barrier to reach the negative mode at $x \approx -1$.
- **Histogram:** The resulting sample histogram (bottom plot) is severely *biased*, only covering the positive mode and failing to reproduce the symmetric, bimodal true density.
- **Moments:** The odd moments, $\mathbb{E}[x]$ and $\mathbb{E}[x^3]$, are incorrectly estimated to be non-zero (≈ 1.0), which is classified as "stuck" because the true value is zero due to symmetry. This highlights the risk of *inaccurate estimates* due to poor mixing and the high-energy barrier effect.

5.3 Parallel Tempering

To evaluate the performance of *Parallel Tempering* and compare the results with above sampling methods, we apply Algorithm 2 to estimate the first moment $\mathbb{E}[x]$ and second moment $\mathbb{E}[x^2]$ of the target probability density function

$$p(X; \beta = 1) = \frac{1}{\int_{-\infty}^{\infty} e^{-(x^2-1)^2} dx} \cdot e^{-(x^2-1)^2}. \tag{11}$$

The corresponding results are shown in Figure 5, comparing the approximated mean/variance against the real mean/variance. From the plots, two features are immediately apparent:

- The Parallel Tempering algorithm exhibits rapid convergence: once the number of states reaches approximately $n \gtrsim 150$, the estimator stabilizes and the variance decreases, as indicated by the increasingly flat behavior of the blue curve.

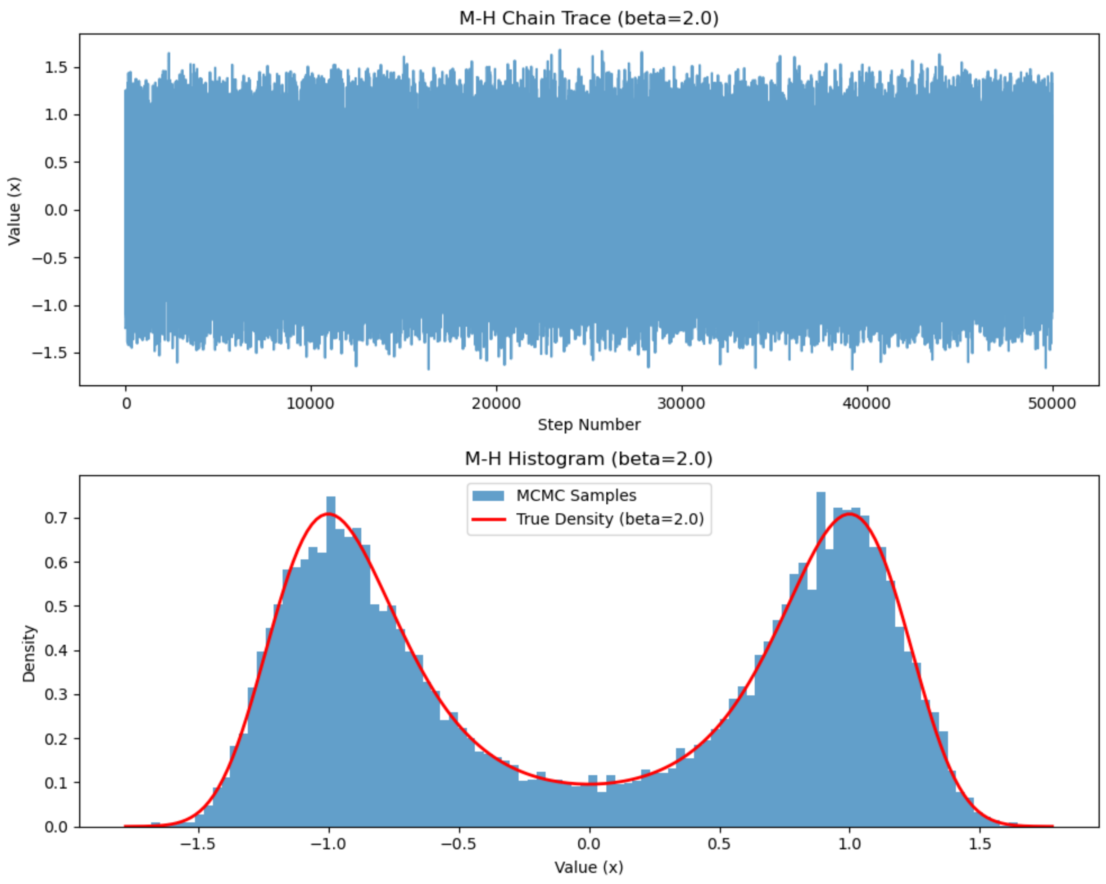

Figure 3: M-H chain trace (top) and histogram (bottom) for the working chain with $\beta = 2$. The chain mixes well and accurately reproduces the bimodal target distribution.

- The accuracy of the estimator improves substantially in this regime. For $n \gtrsim 150$, the blue curve aligns closely with the red reference curve, suggesting that the bias becomes negligible.

Taken together, these observations demonstrate that *Parallel Tempering* provides reliable estimates in settings where standard Metropolis–Hastings struggles due to poor mixing. This indicates that the method is well suited for addressing the more challenging cases where energy barrier is high.

6 Future Work

Several directions remain open for further investigation.

First, it would be valuable to establish theoretical performance guarantees for the sampling methods considered in this study. For example, importance sampling is known to achieve sharper variance bounds than classical Monte Carlo, and analogous bounds for the methods used here would provide a more rigorous justification for their empirical behavior. Moreover, in the present implementation, Metropolis–Hastings and Parallel Tempering are evaluated under different inverse temperature parameters β. For a meaningful quantitative comparison between the two methods, future work should align the temperature schedules so that both algorithms operate under identical β values.

Second, extending the analysis to higher-order moments would yield a more complete characterization of

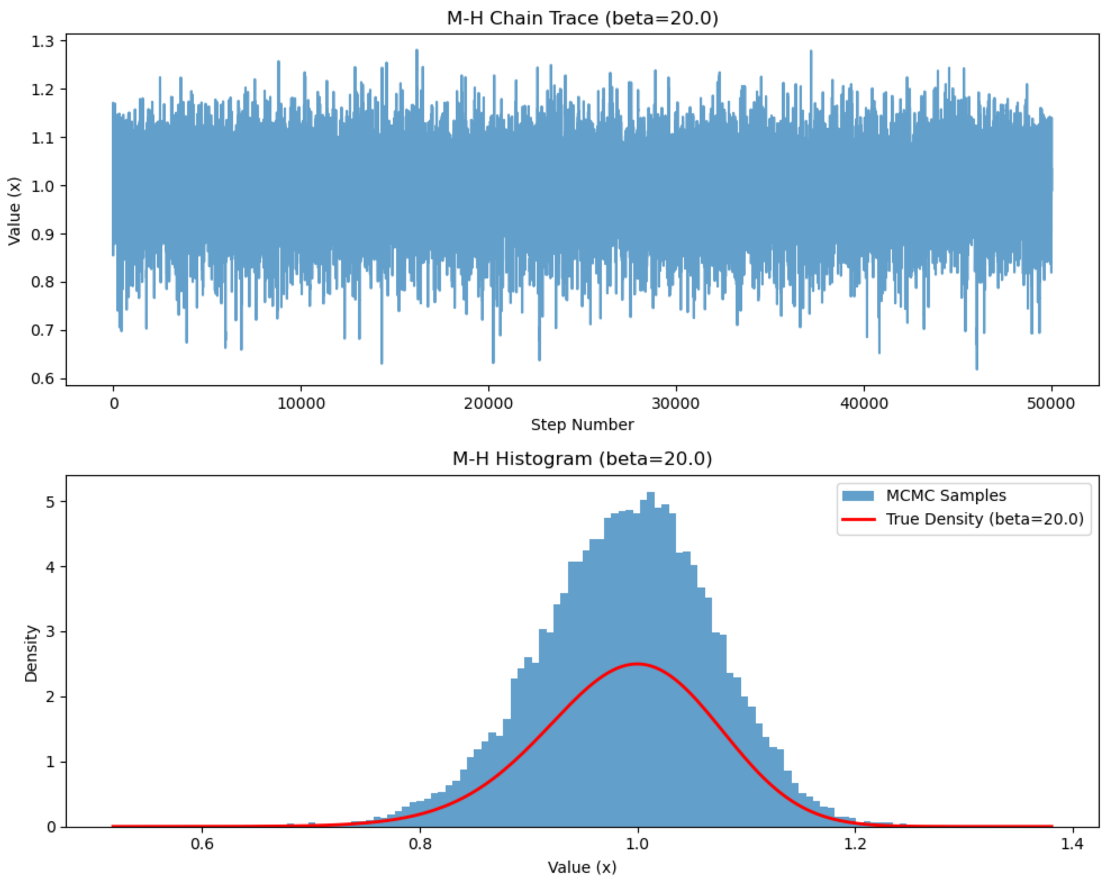

Figure 4: M-H chain trace (top) and histogram (bottom) for the stuck chain with $\beta = 20$. The chain remains confined to a single mode, resulting in a biased sample and incorrect estimation of odd moments.

estimator accuracy beyond the mean and variance.

Third, it would be useful to connect our empirical findings with the broader literature on MCMC efficiency and diagnostics, including work on asymptotic variance, autocorrelation, and general convergence assessment.

Finally, an interesting direction is to study the behavior of these algorithms under more complex target distributions, such as potentials with additional oscillatory structure or multi-well energy landscapes (e.g., triple-well potentials [6]). Such experiments would help assess the robustness of the methods in more challenging sampling regimes.

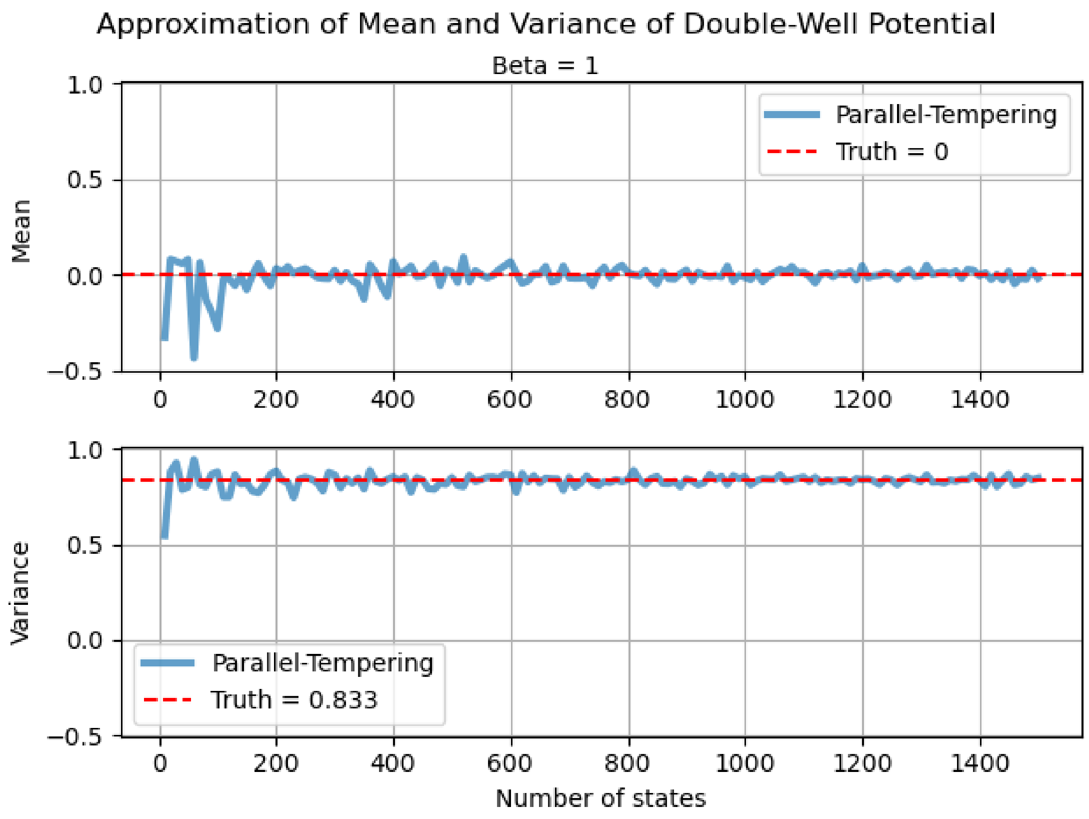

Figure 5: Mean(top) and Variance(bottom) of Double-Well Potential when $\beta = 1$.

References

[1] S. Ferson. What monte carlo methods cannot do. *Human and Ecological Risk Assessment: An International Journal*, 2(4):990–1007, 1996.

[2] W. Janke. Monte carlo methods in classical statistical physics. In *Computational many-particle physics*, pages 79–140. Springer, 2008.

[3] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):386–392, 2014.

[4] J. Machta. Strengths and weaknesses of parallel tempering. *Phys. Rev. E*, 80:056706, Nov 2009.

[5] P. Mocz. Create your own metropolis-hastings markov chain monte carlo algorithm for bayesian inference (with python), 2023.

[6] L. M. Ozherelkova, E. S. Savin, and I. R. Tishaeva. Structural transitions in systems with a triple-well potential. *RUSSIAN*, page 91, 2024.

[7] D. Sanz-Alonso and O. Al-Ghattas. A first course in monte carlo methods, 2024.

[8] S. T. Tokdar and R. E. Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010.