Fall 2022: Monte Carlo Methods
Homework 4

NAME: Utkarsh Khandelwal
Net Id: uk2051

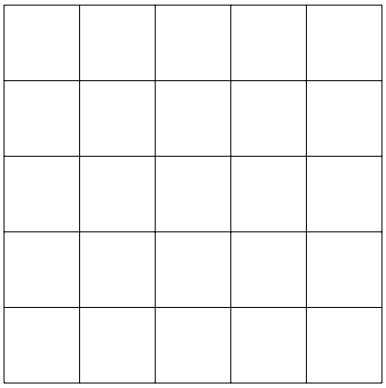
Exercises ask to write a Gibbs Sampler to generate samples of the Ising model and compute the magnetization. The distribution followed by spins are:

$$\pi(\sigma) = \frac{\exp\left(\beta \sum_{i \leftrightarrow j} \sigma_{\vec{i}} \sigma_{\vec{j}}\right)}{Z}$$

And magnetization here is defined as:

$$f(\sigma) = \sum_{i \in Z^{L^2}} \sigma_{\vec{i}}$$

Exercise 39 asks to implement the Gibs Sampling Markov Chain Monte Carlo Algorithm while the **Exercie 41** asks to implement the Metropolis Hastings Markov Chain Monte Carlo Algorithm. I used two separate methods for creating the chain, one is determinisitc sweep to update spin in the lattice and other one is random selection of latice site for update in the Markov Chain.



For the determinisitc sweep, lattice positions are selected from the latice orderly. Start from position $(0, 0)$ then $(0, 1)$ then $(0, 2)$ till $(0, L)$ and then moving to next row $(1, 0)$ and so on and so forth. While for randomly updating the spins, positions are selected based on a random number generated from $U(0, L^2)$.

Below is the algorithm implemented for Gibbs Sampling of N samples

1. Initialie the lattice with spins chosen randomly from discrete values $\{1, -1\}$
2. Calculte the initial magnetization as M_0 and store it. Set $k = 1$
3. Now find the position to update the spin. Position could be selected determinisitcally or randomly. Let the position to be updated is $\vec{i_t}$
4. Store the old spin $\sigma_o = \vec{i_t}$. Now, compute $w_+ = \frac{\exp\left(\beta \sum_{\vec{i} \leftrightarrow \vec{i_t}} \sigma_{\vec{i}}\right)}{\exp\left(\beta \sum_{\vec{i} \leftrightarrow \vec{i_t}} \sigma_{\vec{i}}\right) + \exp\left(-\beta \sum_{\vec{i} \leftrightarrow \vec{i_t}} \sigma_{\vec{i}}\right)}$

5. Generate a bernaulti disrtributed random variable v with probability of success as w_+. If v is True, set the spin as $\sigma_{\overrightarrow{i_t}} = 1$ else set it as $\sigma_{\overrightarrow{i_t}} = -1$ and store $\sigma_n = \sigma_{\overrightarrow{i_t}}$

6. Compute the updated magnetization as $M_k = M_{k-1} - \sigma_o + \sigma_n$ and append it. Here k is the iteration index.

7. Increment k by 1 and repeat step 3,4,5,6 till $N == k$.

Below is the algorithm implemented for Metropolis Hastings of N samples

1. Initialie the lattice with spins chosen randomly from discrete values $\{1, -1\}$

2. Calculte the initial magnetization as M_0 and store it. Set $k = 1$

3. Now find the position to update the spin. Position could be selected determinisitcally or randomly. Let the position to be updated is $\overrightarrow{i_t}$

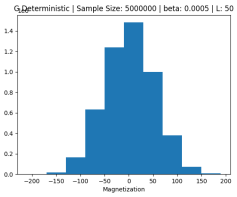
4. Store the old spin $\sigma_o = \sigma_{\overrightarrow{i_t}}$. Filp this spin so $\sigma_n = \text{Flip}(\sigma_o)$. Now, compute $p_{acc} = \min\{1, \exp\left(\beta \sum_{\overrightarrow{i} \leftrightarrow \overrightarrow{i_t}} \sigma_{\overrightarrow{i}} (\sigma_n - \sigma_o)\right)\}$

5. Generate a bernaulti disrtributed random variable v with probability of success as p_{acc}. If v is True, set the spin as $\sigma_{\overrightarrow{i_t}} = \sigma_n$ else do $\sigma_n = \sigma_o$.

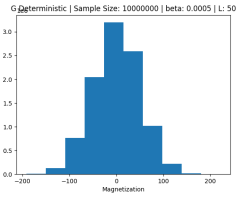
6. Compute the updated magnetization as $M_k = M_{k-1} - \sigma_o + \sigma_n$ and append it. Here k is the iteration index.

7. Increment k by 1 and repeat step 3,4,5,6 till $N == k$.

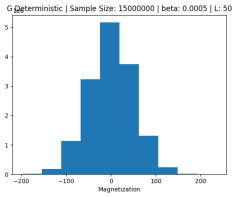
To check the validity of the implementation, histograms of samples were generated. According to the Physics, if the value of β is quite close to 0 the magnetization should be close to 0. Following are the histogram plots for samples generated from Gibbs Sampling 1 and Metropolis Hastings 2 with both Deterministic and Random Update Sequence.



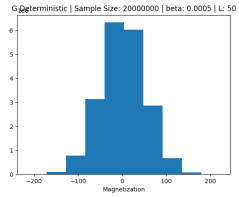
(a) N: 5,000,000 Deterministic Update



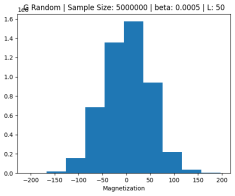
(b) N: 10,000,000 Deterministic Update



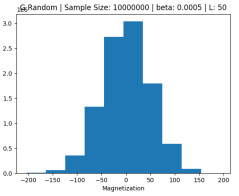
(c) N: 15,000,000 Deterministic Update



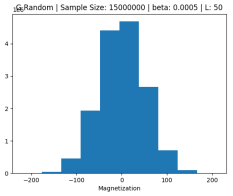
(d) N: 20,000,000 Deterministic Update



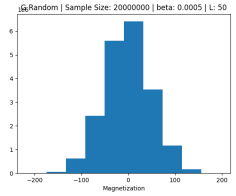
(e) N: 5,000,000 Random Update



(f) N: 10,000,000 Random Update

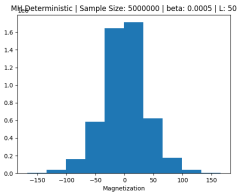


(g) N: 15,000,000 Random Update

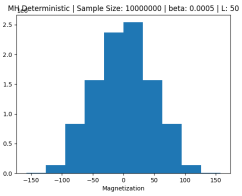


(h) N: 20,000,000 Random Update

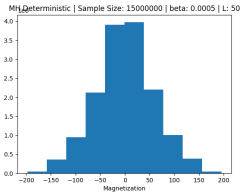
Figure (1) Histograms for magnetization samples generated using Gibbs Sampling



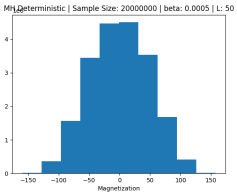
(a) N: 5,000,000 Deterministic Update



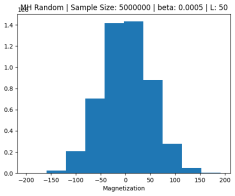
(b) N: 10,000,000 Deterministic Update



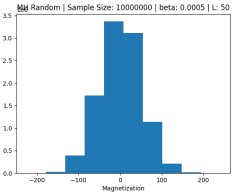
(c) N: 15,000,000 Deterministic Update



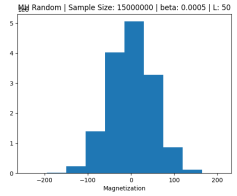
(d) N: 20,000,000 Deterministic Update



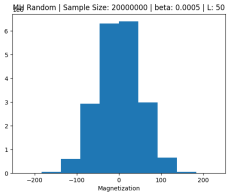
(e) N: 5,000,000 Random Update



(f) N: 10,000,000 Random Update



(g) N: 15,000,000 Random Update



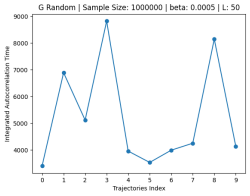
(h) N: 20,000,000 Random Update

Figure (2) Histograms for magnetization samples generated using Metropolis Hastings Sampling

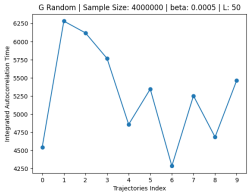
Indeed we can see that the maximum points sampled are having magnetization in the bucket containing 0.

Now, we are well aware of the fact that Markov Chains are conditionally independent and not independent. Hence, Intergrated Auto-Correlation Time (IAT) is used to get statistical value each sample. I used the Python package **emcee** to compute IATs for the magnetization for different values of N for both determinisitc and random update sequence. As IAT are known to be notoriously hard to compute, one way that I used to check the convergance of IAT values by parallely running m $(m = 10)$ Monte Carlo simulation for same value of N and same initial spins. For whatever minimum value of N they converges would be the samples we should be generating to ensure that independent samples are generated from $\pi(\sigma)$. I ran this experiment on CIMS Linux Servers for N starting from 1 million to 20 million with increment of 1 million. Multicore infrastructure of the machines was leveraged by parallelizing the code using the Python's joblib library.

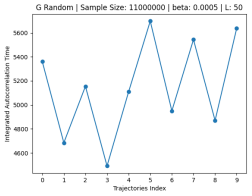
Below are the some of the graphs of IAT values for Gibbs Sampling using Random Update Sequence



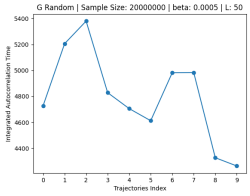
(a) N: 1,000,000 Random Update



(b) N: 4,000,000 Random Update



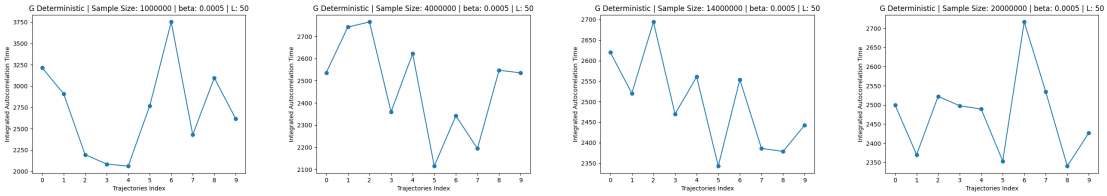
(c) N: 11,000,000 Random Update



(d) N: 20,000,000 Random Update

Figure (3) Graphs for IATs for Gibbs Sampling Algorithm using Random Update Sequence

We can observe from the graphs 3 that IAT values are between 4000 to 9000 for 1 million samples and decreases with increase in the value of N. IAT stabalization between 4600 to 5600 starts at 11 million samples and remains in the same bound even for 20 million samples. To compare it with determinisitc update sequence below graphs were generated.



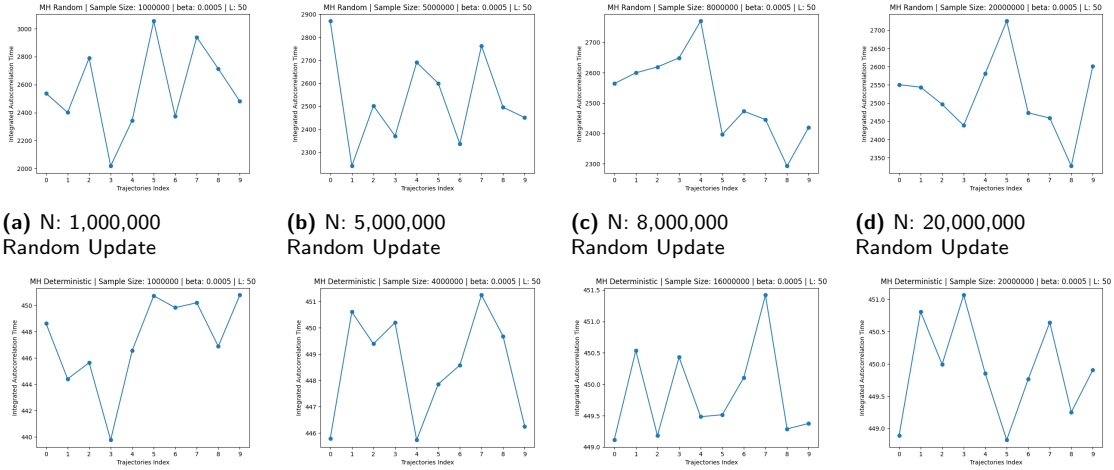
(a) N: 1,000,000 Deterministic Update **(b)** N: 4,000,000 Deterministic Update **(c)** N: 14,000,000 Deterministic Update **(d)** N: 20,000,000 Deterministic Update

Figure (4) Graphs for IATs for Gibbs Sampling Algorithm using Deterministic Update Sequence

Similarly here for determinisitc update sequence 4, we can observe that the IAT values are between 2000 to 3700 for 1 million samples and decreases with the increase in N. This time, IAT stabalization between 2350 to 2700 starts at 14 million and remains in the same bound even for 20 million samples.

So IAT for random update sequence is almost twice as the IAT for determinisitc update sequence, hence it would be wiser to choose determinisitc update sequence for generating samples using Gibbs Sampling Monte Carlo Algorithm.

Same experiment was done for Metropolis Hastings Algorithm. Below are the graphs 5 generated fro IAT values for differnet total samples count and different position update method.

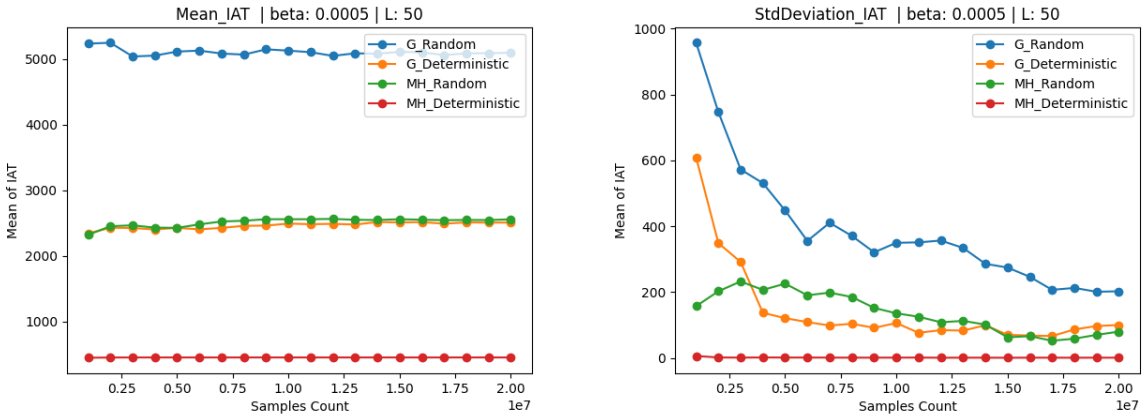


(a) N: 1,000,000 Random Update **(b)** N: 5,000,000 Random Update **(c)** N: 8,000,000 Random Update **(d)** N: 20,000,000 Random Update

(e) N: 1,000,000 Deterministic Update **(f)** N: 4,000,000 Deterministic Update **(g)** N: 16,000,000 Deterministic Update **(h)** N: 20,000,000 Deterministic Update

Figure (5) Graphs for IATs for Metropolis Hastings Sampling Algorithm

Following are the graphs 6 showing the variation of Meat of IAT and Standard Deviation of IAT for various values of N starting from 1 million samples to 20 million samples. We can easily observe that the IAT for Metropolis Hastings with determinisitc update sequence is converged even for 1 million sample count. Second, the standard deviation of IAT keeps on decreasing on increasing the value of N. And, the best method out of four is Metropolis Hasting with determinisitc update sequence then the performance on the basis for IAT value is same for Metropolis Hastings with random update sequenece as for Gibbs Sampling with determinisitc update sequence. And the largest mean value of IAT is for Gibbs Sampling with Random Update Sequence.



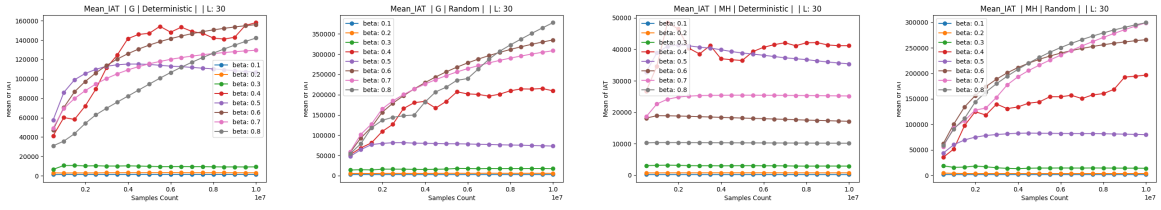
(a) Mean vs Samples Count

(b) Standard deviations vs Samples Count

Figure (6) Means and standard deviations of IATs plotted for different values of N for all four methods

Now, it wouldn't be wrong to infer that the Metropolis Hastings Monte Carlo Algorithm with deterministic update is the best among all four above mentioned methods to generate samples from Ising Model on 2D lattice.

Next, we have been asked to compute IAT for various values of temperature. As temperature is inversely proportional to β, we can check for various values of β and draw conclusion appropriately.

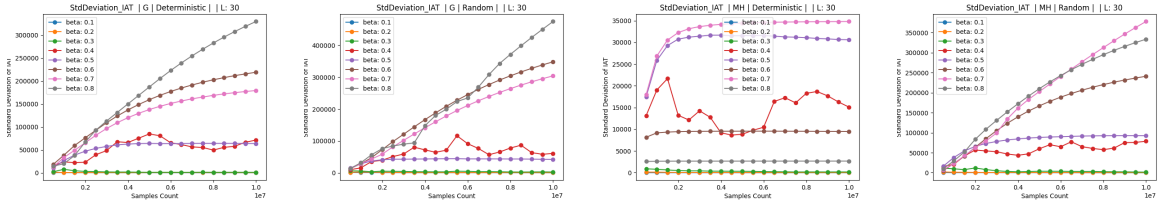


(a) Mean Gibbs Deterministic

(b) Mean Gibbs Random

(c) Mean Metropolis Hastings Deterministic

(d) Mean Metropolis Hastings Random



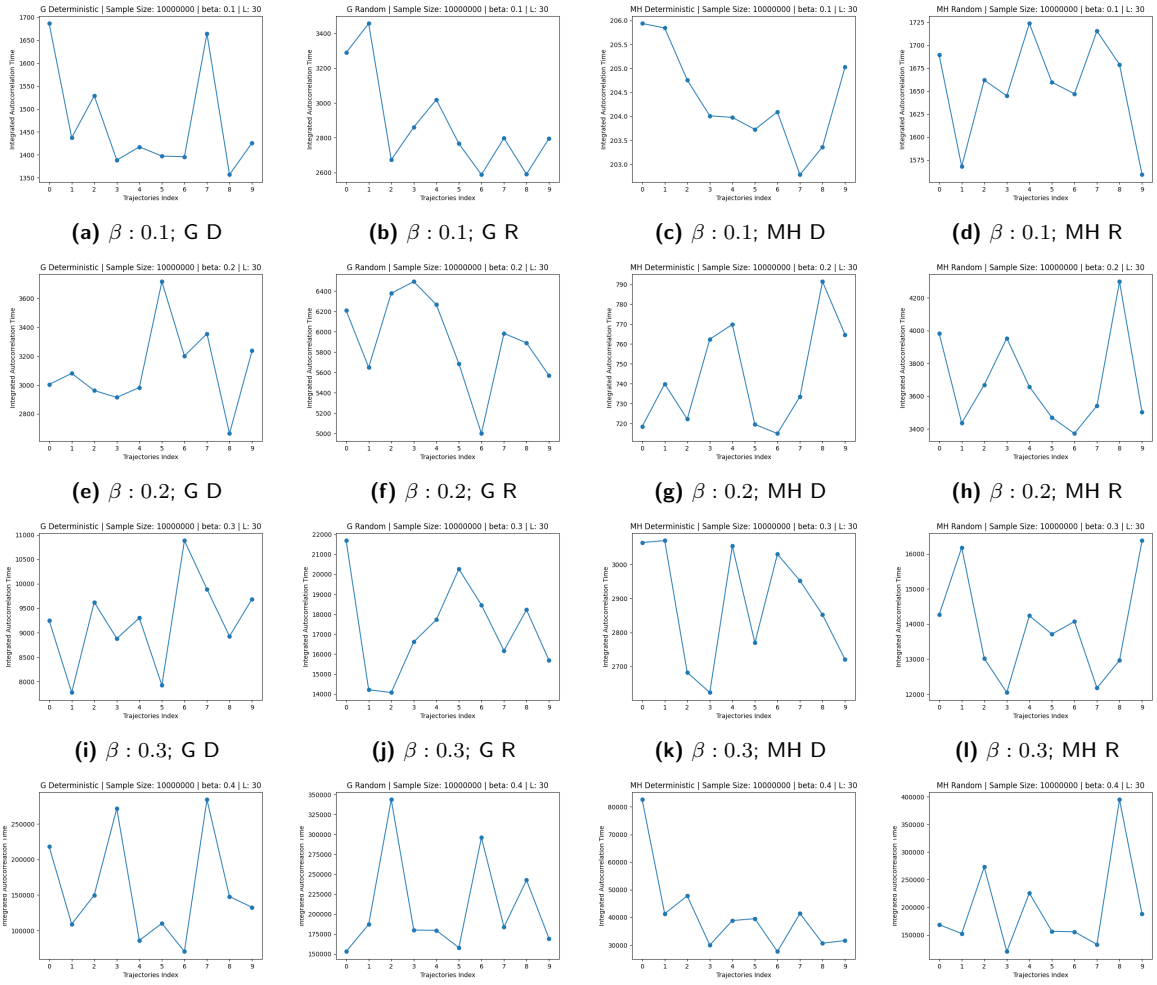
(e) Std Deviation Gibbs Deterministic

(f) Std Deviation Gibbs Random

(g) Std Deviation Metropolis Hastings Deterministic

(h) Std Deviation Metropolis Hastings Random

Figure (7) Means and standard deviations of IATs plotted for different values of beta and for increasing values of N



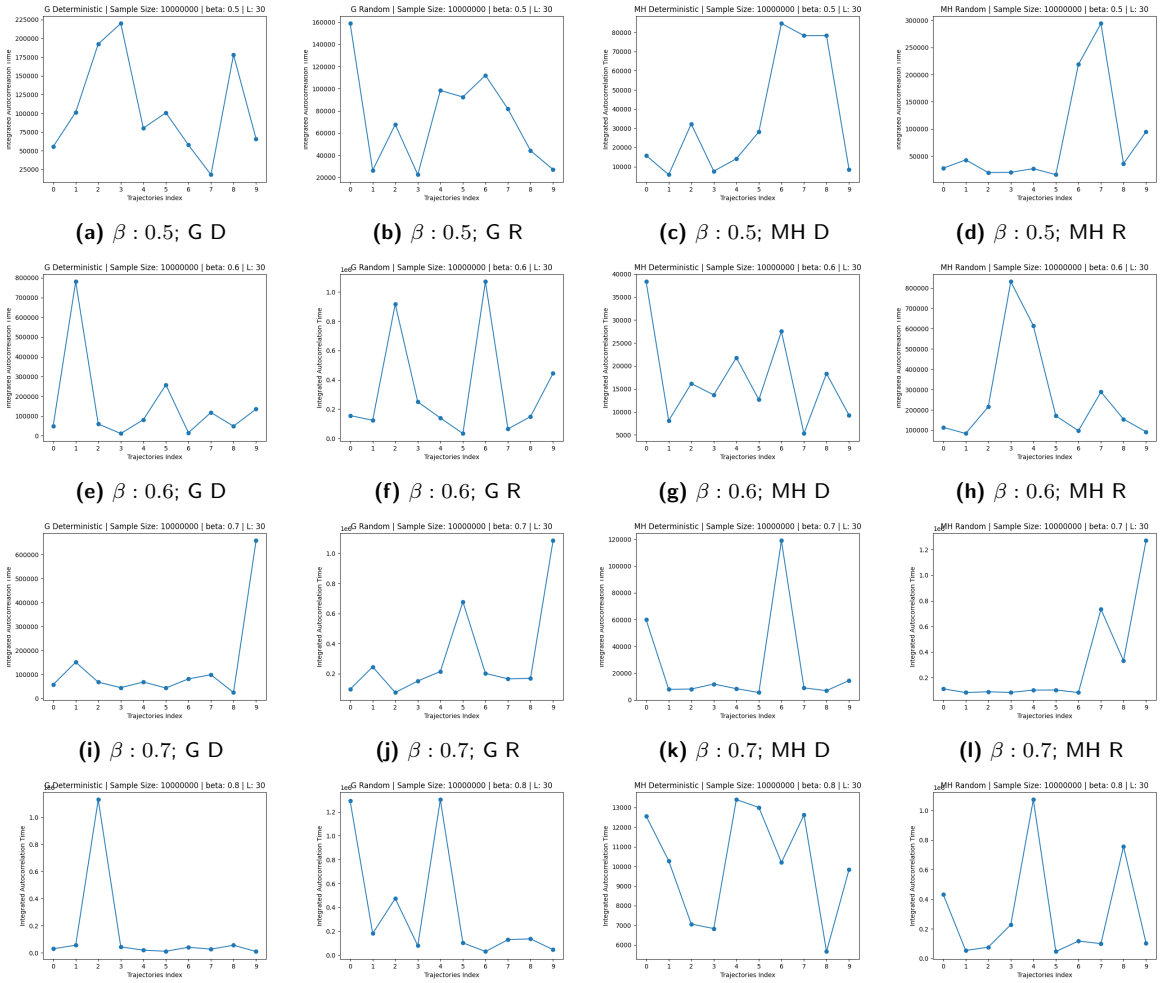
(a) $\beta : 0.1$; G D **(b)** $\beta : 0.1$; G R **(c)** $\beta : 0.1$; MH D **(d)** $\beta : 0.1$; MH R

(e) $\beta : 0.2$; G D **(f)** $\beta : 0.2$; G R **(g)** $\beta : 0.2$; MH D **(h)** $\beta : 0.2$; MH R

(i) $\beta : 0.3$; G D **(j)** $\beta : 0.3$; G R **(k)** $\beta : 0.3$; MH D **(l)** $\beta : 0.3$; MH R

(m) $\beta : 0.4$; G D **(n)** $\beta : 0.4$; G R **(o)** $\beta : 0.4$; MH D **(p)** $\beta : 0.4$; MH R

Figure (8) First half of figure showing IAT for different trajectory for different values of beta for all the methods. Here each column represent different method and going downwards in the image increment the value of beta.



(a) $\beta : 0.5$; G D **(b)** $\beta : 0.5$; G R **(c)** $\beta : 0.5$; MH D **(d)** $\beta : 0.5$; MH R

(e) $\beta : 0.6$; G D **(f)** $\beta : 0.6$; G R **(g)** $\beta : 0.6$; MH D **(h)** $\beta : 0.6$; MH R

(i) $\beta : 0.7$; G D **(j)** $\beta : 0.7$; G R **(k)** $\beta : 0.7$; MH D **(l)** $\beta : 0.7$; MH R

(m) $\beta : 0.8$; G D **(n)** $\beta : 0.8$; G R **(o)** $\beta : 0.8$; MH D **(p)** $\beta : 0.8$; MH R

Figure (9) Second half of figure showing IAT for different trajectory for different values of beta for all the methods. Here each column represent different method and going downwards in the image increment the value of beta.

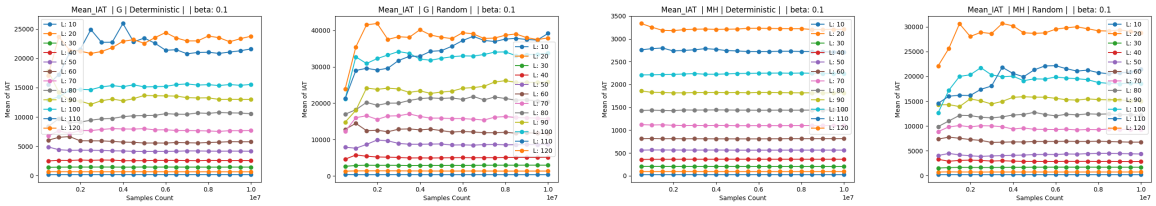
I varied the values of β starting from 0.1 to 0.8 for the different samples of sizes starting from 500,000 to 10,000,000 for a constant L $(L = 30)$. Following are the observations we can easily make from the graphs in 7,8,9:

1. In 6, 7 we can see that as the value of β is IAT values are taking more samples to converge. Most of them are not even converged even at 10 million samples even for Metropolis Hastings Deterministic Update Sequence Algorithm.

2. There is definitely some threashold value of β above which IAT value increase drastically with the β while below the threshold value of β, IAT is samller in magnitude.

3. From these graphs, it is easy to conclude that the value of IAT increase with β for Gibbs Sampling Deterministic Update Sequence and Random Update Sequence and also for Metropolis Hastings Random Update Sequence. While there is one critical value of β in Metropolis Hastings Deterministic Update Sequence that is having maximum IAT value. All other value of β (larger and samller) are having IAT value lesser than the IAT at this critical value.

Next, I varied the value of L starting from $L = 10$ to $L = 120$ with N also increasing from 500,000 to 10,000,000 million samples and $\beta = 0.1$ as fixed. Results can be seen in the following graphs:

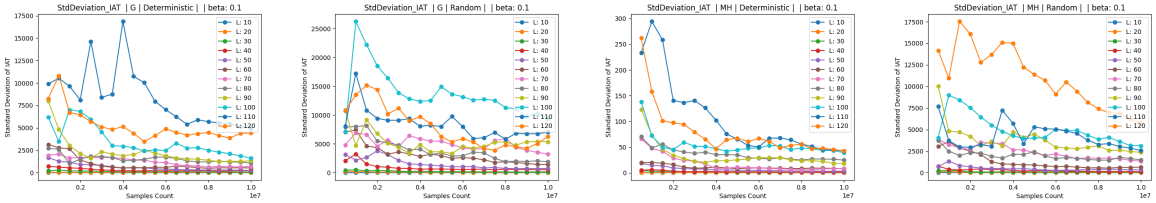


(a) Mean Gibbs Deterministic

(b) Mean Gibbs Random

(c) Mean Metropolis Hastings Deterministic

(d) Mean Metropolis Hastings Random



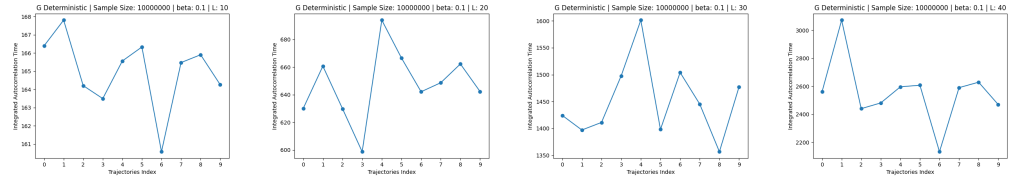
(e) Std Deviation Gibbs Deterministic

(f) Std Deviation Gibbs Random

(g) Std Deviation Metropolis Hastings Deterministic

(h) Std Deviation Metropolis Hastings Random

Figure (10) Means and standard deviations of IATs plotted for different values of L and for increasing values of N

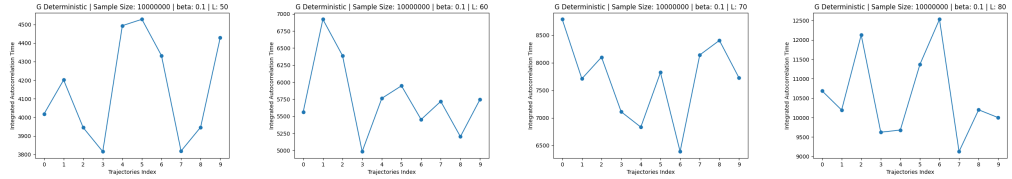


(a) L:10; Gibbs Deterministic

(b) L:20; Gibbs Deterministic

(c) L:30; Gibbs Deterministic

(d) L:40; Gibbs Deterministic

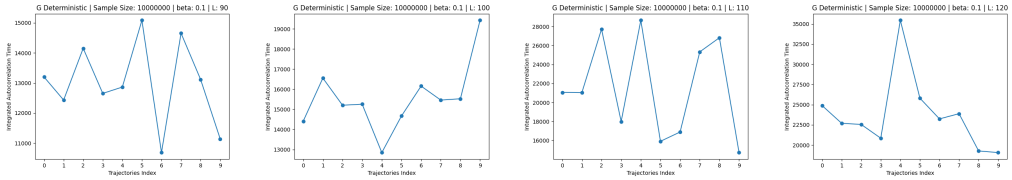


(e) L:50; Gibbs Deterministic

(f) L:60; Gibbs Deterministic

(g) L:70; Gibbs Deterministic

(h) L:80; Gibbs Deterministic



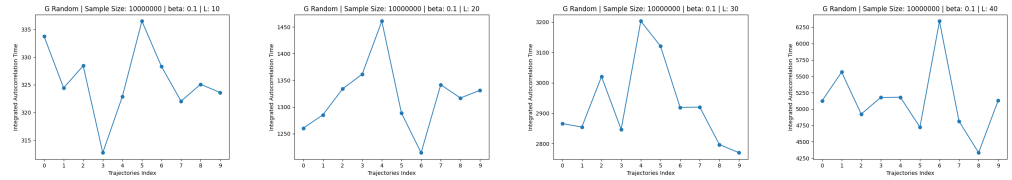
(i) L:90; Gibbs Deterministic

(j) L:100; Gibbs Deterministic

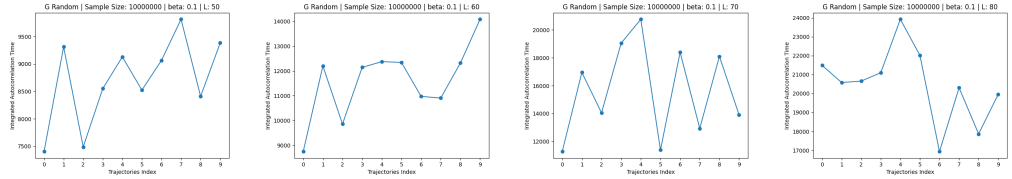
(k) L:110; Gibbs Deterministic

(l) L:120; Gibbs Deterministic

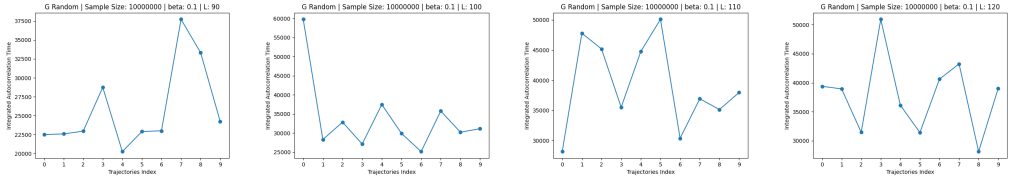
Figure (11) Graphs for values of IAT for different values of L for Gibbs Sampling Deterministic Update



(a) L:10; Gibbs Random **(b)** L:20; Gibbs Random **(c)** L:30; Gibbs Random **(d)** L:40; Gibbs Random

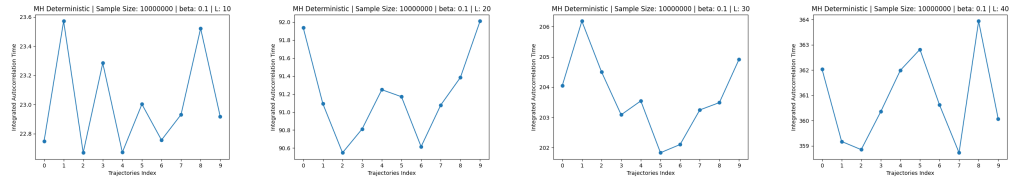


(e) L:50; Gibbs Random **(f)** L:60; Gibbs Random **(g)** L:70; Gibbs Random **(h)** L:80; Gibbs Random

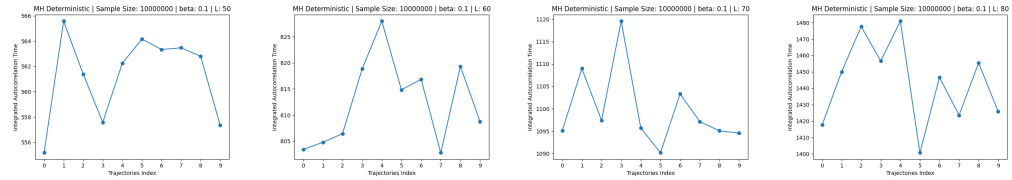


(i) L:90; Gibbs Random **(j)** L:100; Gibbs Random **(k)** L:110; Gibbs Random **(l)** L:120; Gibbs Random

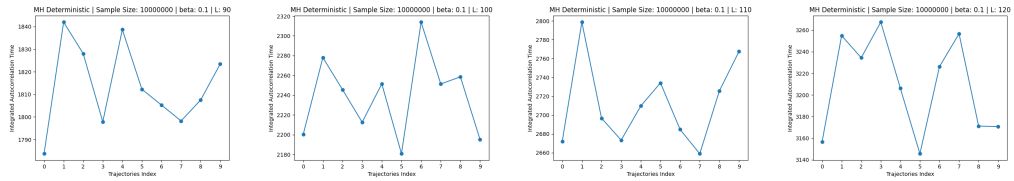
Figure (12) Graphs for values of IAT for different values of L for Gibbs Sampling Random Update



(a) L:10; MH Deterministic **(b)** L:20; MH Deterministic **(c)** L:30; MH Deterministic **(d)** L:40; MH Deterministic

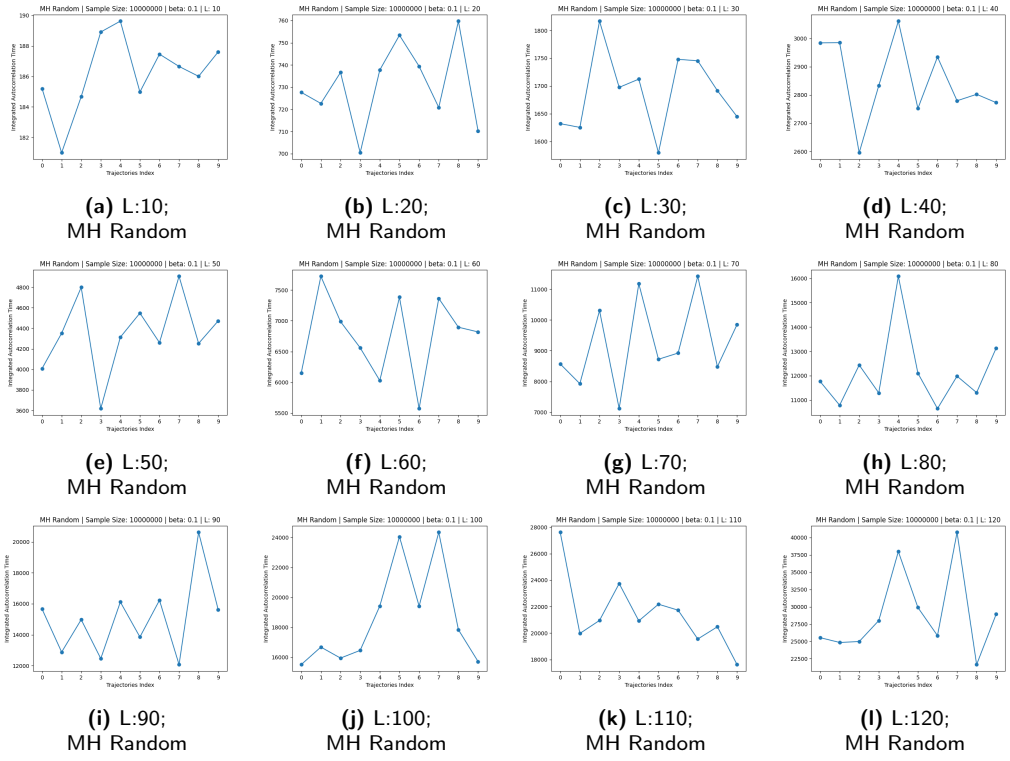


(e) L:50; MH Deterministic **(f)** L:60; MH Deterministic **(g)** L:70; MH Deterministic **(h)** L:80; MH Deterministic



(i) L:90; MH Deterministic **(j)** L:100; MH Deterministic **(k)** L:110; MH Deterministic **(l)** L:120; MH Deterministic

Figure (13) Graphs for values of IAT for different values of L for Metropolis Hastings Sampling Deterministic Update



(a) L:10; MH Random **(b)** L:20; MH Random **(c)** L:30; MH Random **(d)** L:40; MH Random

(e) L:50; MH Random **(f)** L:60; MH Random **(g)** L:70; MH Random **(h)** L:80; MH Random

(i) L:90; MH Random **(j)** L:100; MH Random **(k)** L:110; MH Random **(l)** L:120; MH Random

Figure (14) Graphs for values of IAT for different values of L for Metropolis Hastings Sampling Random Update

Following are the observations, we can make from the graphs in 10, 11, 12 ,13, 14

1. Even for incresing dimension of the Lattice, Metropolis Hastings with Deterministic Update Sequence is having the samllel standard deviation among all four possibilities.
2. We can easily infer from Gibs Samplign Deterministic Update graphs 11, the IAT values, scales up quadratically with the change in value of L. For instance for (L, meanIAT) pair are: $(10, 164)$, $(20, 640)$, $(30, 1400)$ and $(40, 2600)$.
3. Similar is the case for all the other three methods, the mean IAT sacles up quadratically with the change in value of L.

Moving on to **Exercise 43**, this question asks us to implement Jarzynski Method for Ising model for both with resampling and without resampling. Here, the distributions to generate samples from are changing with the sample index. We are given

$$\pi_k = \pi^{\frac{k}{N}}$$

Here, N is the maximum sample count and k is the index of the sample with are generating. This method can be imagined as the Metropolis Hastings Scheme at each index with different probability distribution.

Below is the algorithm implemented for Jarzynski Method of N samples

1. Initialie the lattice with spins chosen randomly from discrete values $\{1, -1\}$ and create M copies of it.
2. Calculte the initial magnetization as M_0 and store it And itntial weights of M copies as $1/M$. Set $k = 1$

3. Now find the position to update the spin. Position is selected determinisitcally. Let the position to be updated is $\overrightarrow{i_t}$

4. Store the old spin $\sigma_o = \sigma_{\overrightarrow{i_t}}$. Filp this spin so $\sigma_n = \text{Flip}(\sigma_o)$. Now, compute $p_{acc} = \min\{1, \exp\left(\frac{\beta k}{N} \times \sum_{\overrightarrow{i} \leftrightarrow \overrightarrow{i_t}} \sigma_{\overrightarrow{i}}(\sigma_n - \sigma_o)\right)\}$

5. Generate a bernaulti disrtributed random variable v with probability of success as p_{acc}. If v is True, set the spin as $\sigma_{\overrightarrow{i_t}} = \sigma_n$ else do $\sigma_n = \sigma_o$.

6. Compute the change in Hamiltonian, new weights, and update the magnetization.

7. Increment k by 1 and repeat step 3,4,5,6 till $N == k$.

8. When $k == N$, get all the M copies of last sample generated in each chain, get the magnetization and multiply them by appropriate weights.

Below is the code snippet for the Algorithm for the initialization and update of spins:

```
def InitializeLattice(self, inputInitValues):
  initializedValues = inputInitValues
  self.initSpins[:] = initializedValues
  self.SetInitialHamiltonian()
  partialSum = np.sum(self.initSpins, 2)
  partialSum = np.sum(partialSum, 1)
  self.Magnetization[:, 0] = partialSum
  self.Weights[:, 0] = self.Weights[:, 0] / np.sum(self.Weights[:, 0])
  self.GetUpdateSequence()

def UpdateSpins(self, x_pos, y_pos, index, resample=False):
  oldSpin = self.initSpins[:, x_pos, y_pos]
  nSum, finalSpin = self.GinalFinalSpinToUpdate(x_pos, y_pos, index)
  deltaSpinChange = finalSpin - oldSpin
  self.initSpins[:, x_pos, y_pos] = finalSpin
  delta_Hamiltonian = deltaSpinChange * nSum
  self.hamiltonian[:, 0] = self.hamiltonian[:, 0] + delta_Hamiltonian
  omega_num = np.exp((self.beta / self.N) * self.hamiltonian)
  omega_num = np.reshape(omega_num, (omega_num.shape[0],))
  self.Weights[:, index] = self.Weights[:, index - 1] * omega_num
  self.Weights[:, index] = self.Weights[:, index] / np.sum(self.Weights[:,
                                        index])
  self.Magnetization[:, index] = self.Magnetization[:, index - 1] +
                                        deltaSpinChange
  if (resample == True):
    self.DoMultinomialResampling(index)
```

Following is the acceptance probability:

$$p_{acc}(X^k, Y^{k+1}) = \min\left\{1, \exp\left(\frac{\beta k}{N} \times \sum_{\overrightarrow{i} \leftrightarrow \overrightarrow{i_t}} \sigma_{\overrightarrow{i}}(\sigma_n - \sigma_o)\right)\right\}$$

Here, $\overrightarrow{i_t}$ is the position where we are checking for spin flip. And σ_n is the spin after flip and σ_o is the spin before flip. So, transition operator for the index $k+1$ can be written as

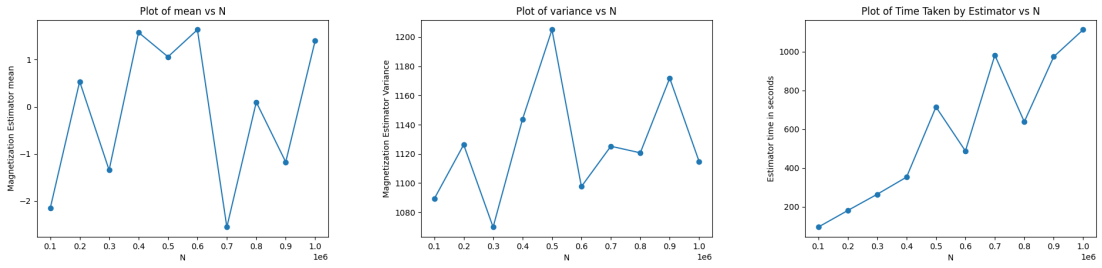
$$\tau_{k+1} f = f(x) p_{rej}(x) + \int f(y) q(y|x) p_{acc}(x, y) dy$$

here, $p_{rej}(x) = \int (1 - p_{acc}(x, z)(dz|x))$ and y is the spin at $\overrightarrow{i_t}$ after the flip and x is before the filp.

$$\tau_{k+1} f = f(x) p_{rej}(x) + \int f(y) q(y|x) \min \left\{ 1, \exp \left(\frac{\beta k}{N} \times \sum_{\overrightarrow{i} \leftrightarrow \overrightarrow{i_t}} \sigma_{\overrightarrow{i}}(y - x) \right) \right\} dy$$

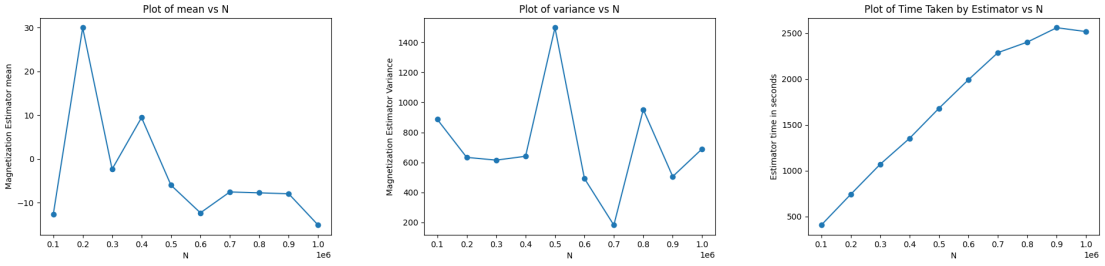
Below is the code snippet resampling of the samples using numpy choice function.

```
def DoMultinomialResampling(self, index):
    out = np.random.choice(self.M, self.M, replace=True, p=self.Weights[:, index]
                                                            )
    newWeights = np.ones((self.M, ))
    newSpins = self.initSpins[out, :, :]
    newHamiltonin = self.hamiltonian[out, :]
    newMagnetization = self.Magnetization[out, index]
    newWeights = newWeights / self.M
    self.initSpins = newSpins
    self.hamiltonian = newHamiltonin
    self.Magnetization[:, index] = newMagnetization
    self.Weights[:, index] = newWeights
    return
```



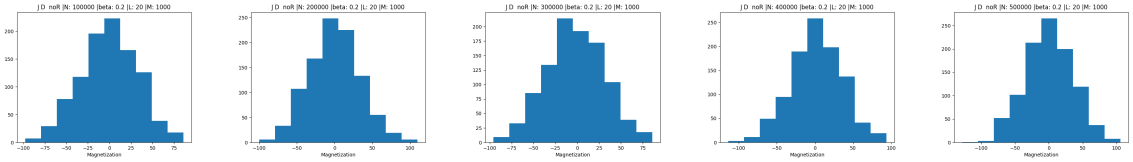
(a) Mean vs N **(b)** Standard deviations vs N **(c)** Time taken vs N

Figure (15) Means, standard deviations and time taken without resampling for Jarzynski Method for $\beta = 0.2$, $L = 20$ and $M = 1000$

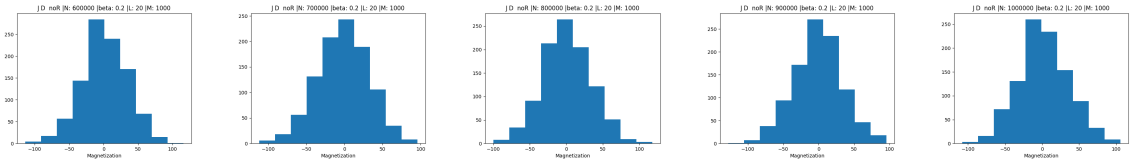


(a) Mean vs N **(b)** Standard deviations vs N **(c)** Time taken vs N

Figure (16) Means, standard deviations and time taken with resampling for Jarzynski Method for $\beta = 0.2$, $L = 20$ and $M = 1000$

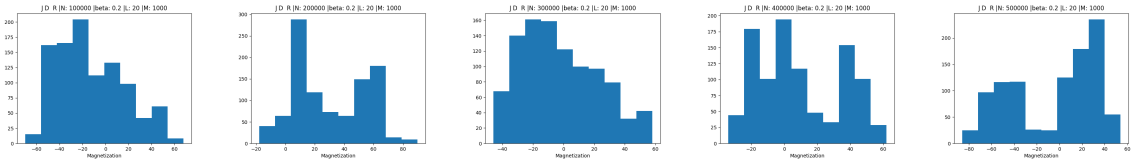


(a) N: 10000 **(b)** N: 20000 **(c)** N: 30000 **(d)** N: 40000 **(e)** N: 50000

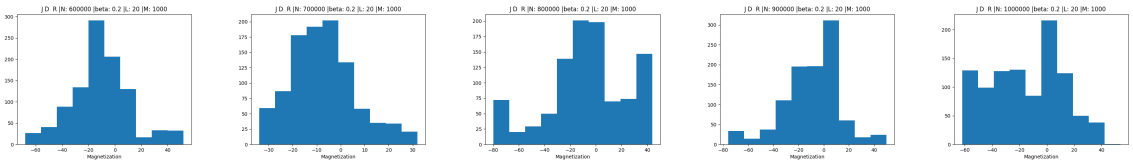


(f) N: 60000 **(g)** N: 70000 **(h)** N: 80000 **(i)** N: 90000 **(j)** N: 100000

Figure (17) Histograms Jarzynski Method for increasing N without Resampling



(a) N: 10000 **(b)** N: 20000 **(c)** N: 30000 **(d)** N: 40000 **(e)** N: 50000



(f) N: 60000 **(g)** N: 70000 **(h)** N: 80000 **(i)** N: 90000 **(j)** N: 100000

Figure (18) Histograms Jarzynski Method for increasing N with Resampling

We know that the magnetization for $\beta = 0.2$ is not zero. Hence, the Jarzynski Method with resampling 18 is much more closer to the reality than without resampling 17. Also, on increasing the value of N and M, Jarzynski method without resampling is going will converge to correct value of magnetization. It is just resampling keeps samples with more weights and move ahead.

From the graphs in 15 and 16 variance increases with both N while incase of resampling variance almost stays the same for the increasing value of N. Moreover the standard deviation with resampling is samller as compared to standard deviation without resampling.Considering the time taken by the estimator as the measure of effort we can see that time taken with resampling is almost twice the time taken without resampling but it increases linearly $O(N)$. Hence, for larger values of N with resampling would only take linearly twice the time which shouldn't be problem.

Comparing between Jarzynski Method with the Gibbs and Metropolis Hasting Algorithm, can be done by comparing the variance of the estimator for same values of $beta$ and L. For Gibbs and Metropolis Hastings variance can be computed as the variance of M independent trajectory each of size N. For various values of N, these estimates can be computed and analyzed for the bettwe performing model.