STAT238: Bayesian Statistics May 2nd, 2025

Mini Project # 8: Advanced MCMC Methods

Reece D. Huff

Prompts

This is a mixed coding/reading project. It builds directly on Lab 8. You will be asked to complete the mixing time experiments outlined at the end of the lab, use these to identify a model setting where standard Metropolis-Hastings (MH) algorithms mix slowly, then implement two modifications to the MH algorithm which often speed mixing. These are *simulated tempering* (see BDA 12.3) and *ensemble MCMC* methods. Ensemble methods evolve a population of random walkers together in parallel rather than a single walker.

Model

Background & Notation

The 2-D Ising model considers a collection of atoms arranged in a N-by-N grid. We let s_i represent the magnetic dipole moment of atom $i = 1, 2, \ldots, N^2$. Each atom has spin "up", $s_i = +1$, or spin "down", $s_i = -1$. We let $M : \{+1, -1\}^{N \times N} \to \mathbb{R}$ be the *average spin* across all atoms as it is typically used to measure *magnetization*:

$$M(s) = \frac{1}{N^2} \sum_i s_i.$$

The Ising model assumes the collection of spins is sampled according to a Boltzmann distribution,

$$\mathbb{P}(S = s) \propto \exp\left(-\frac{1}{\tau} E(s)\right) \quad \text{where} \quad E(s) = -\frac{J}{2} \sum_{\langle i,j \rangle} s_i s_j \quad \text{and}$$

- τ is the temperature (formally, the Boltzmann constant times the temperature),
- $E(s)$ is the potential energy stored in the magnetic field when the atoms are arranged with spins s,
- $\langle i, j \rangle$ is the summation over all nearest-neighbor pairs (as such, we divide by 2 to avoid double counting), and
- $J > 0$ is the strength of the magnetic interaction between neighboring atoms.

Suppose that we observed magnetization $m = M(s) + \zeta$ where $\zeta \sim \mathcal{N}(0, \sigma^2)$. Then our joint model is given by

Prior: $p(s) \propto \exp\left(-\frac{1}{\tau} E(s)\right)$
Likelihood: $p(m \mid s) \sim \mathcal{N}(M(s), \sigma^2)$
Posterior: $p(s \mid m) \propto p(m \mid s)\, p(s) \propto \exp\left(-\frac{1}{2}\left(\frac{m - M(s)}{\sigma}\right)^2\right) \exp\left(-\frac{1}{\tau} E(s)\right) \propto \exp\left(\frac{J}{2\tau} \sum_{\langle i,j \rangle} s_i s_j - \frac{1}{2\sigma^2}\left(m - \frac{1}{N^2} \sum_i s_i\right)^2\right)$

Example

Consider an example configuration in Figure 1. The energy of s_i is given by

$$E(s_i) = -Js_i \sum_{j \in \mathcal{N}(i)} s_j = -Js_i(+1-1-1-1) = 2Js_i$$

such that the probability of S_i being $+1$ and -1 is given by

$$\mathbb{P}(S_i = +1) \propto \exp\left(-\frac{1}{\tau}E(s_i = +1)\right) \implies \mathbb{P}(S_i = +1) = \frac{\exp(-2J/\tau)}{\exp(+2J/\tau) + \exp(-2J/\tau)} = \frac{1}{1+\exp\left(-2\left(-\frac{2J}{\tau}\right)\right)}$$

$$\mathbb{P}(S_i = -1) \propto \exp\left(-\frac{1}{\tau}E(s_i = -1)\right) \implies \mathbb{P}(S_i = -1) = \frac{\exp(+2J/\tau)}{\exp(+2J/\tau) + \exp(-2J/\tau)} = \frac{1}{1+\exp\left(2\left(-\frac{2J}{\tau}\right)\right)}$$

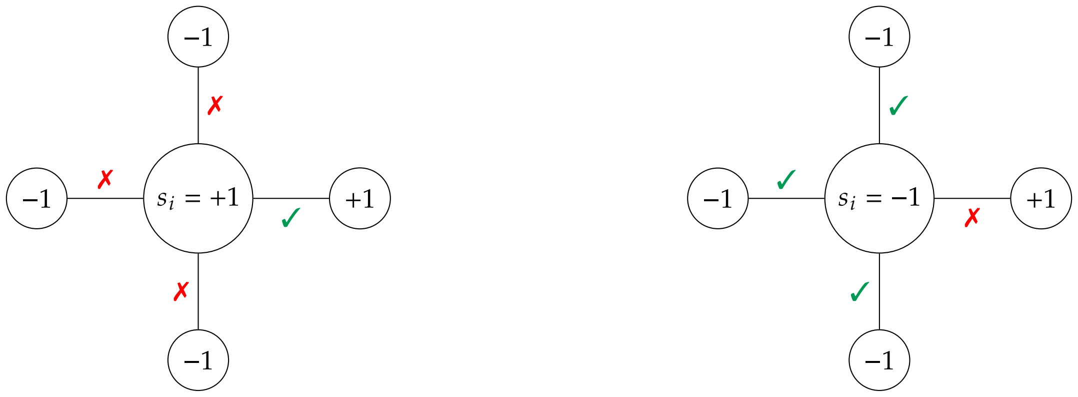

Figure 1: An example configuration of the Ising model. We let ✓ and ✗ represent when s_i and its neighbor are aligned and misaligned, respectively.

Metropolis-Hastings Sampling

Suppose we take $\tilde{p}_*(s) = \mathbb{P}(S = s)$ as our target density and sample from it via the Metropolis algorithm. Let p_r represent a symmetric proposal distribution of sampling a new configuration s' given the current configuration s. Then the acceptance probability is given by

$$a(s') = \min(r, 1) \quad \text{where} \quad r = \frac{\tilde{p}_*(s')\, p_r(s' \to s)}{\tilde{p}_*(s)\, p_r(s \to s')} = \frac{\exp\left(-\frac{1}{\tau}E(s')\right)}{\exp\left(-\frac{1}{\tau}E(s)\right)} = \exp\left(-\frac{1}{\tau}\left[E(s') - E(s)\right]\right) = \exp\left(-\frac{1}{\tau}\left[\Delta E(s, s')\right]\right)$$

We note that since $\tau > 0$, the only candidates s' with an acceptance probability less than 1 are candidates that a proposing a lower energy, i.e., $\Delta E(s, s') > 0$. Additionally, we note that as the temperature increases, the acceptance probability increases. We display this relationship below:

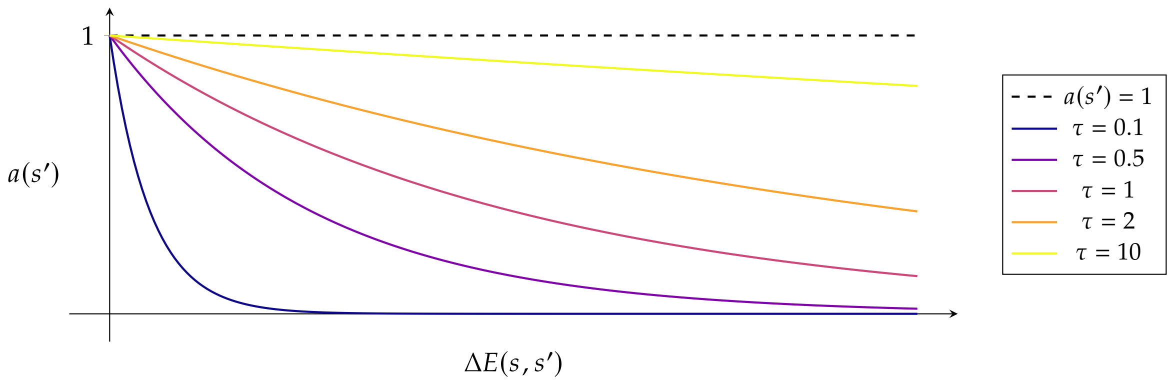

Figure 2: Metropolis-Hastings acceptance probability as a function of change in energy $\Delta E(s, s')$ for varied temperatures.

(a) *Mixing Times for MH in the Ising Model*

Complete the mixing time experiment marked optional at the end of Lab 8 using the Metropolis-Hastings algorithm with Proposal I (independent spin flips). Identify a setting (grid size and temperature) where the process mixes slowly (either, too slowly for MH, or, slowly enough the MH is barely feasible).

Report the results of your experiment. In particular, is mixing slowest near criticality, or at an extreme (very high or very low) temperature?

Experimental Results

- Mixing times are in Figure 3
- Average absolute magnetization, $\mathbb{E}[|M|]$, and estimated critical temperatures τ_{crit} are in Figure 4
- Energy and magnetization samples after warmup are in Figure 5

Discussion

Mixing times: We note that the mixing times largest when the temperature is low. This trend becomes especially clear for the larger grid sizes Figure 3. This is because the low temperature samples are more likely to be stuck in one of the two modes. More on why this is the case is discussed in part (c) 2.

Average absolute magnetization: The average absolute magnetization is largest when the temperature is low. This trend becomes especially clear for the larger grid sizes Figure 4. The critical temperature is estimated to when there is a phase transition in the average absolute magnetization and plotted in Figure 4. All of the estimated critical temperatures hover around the theoretical critical temperature of $\tau_{\text{crit}} = 2.269$ provided in Lab 8. This trend makes sense as when the temperature is low, the spins are more likely to be aligned (either all up or all down).

Energy and magnetization samples: The energy and magnetization samples after warmup are plotted in Figure 5. The nice trend emerges as the temperature changes. For example, when the grid size is 256, the magnetization samples are all close to 1 or -1 when the temperature is low while the hot samples seems to jitter around 0. This trend is also seen in the energy samples. The low temperature samples drive the energy to its minimum (all spins aligned) while the high temperature samples jitter around larger values.

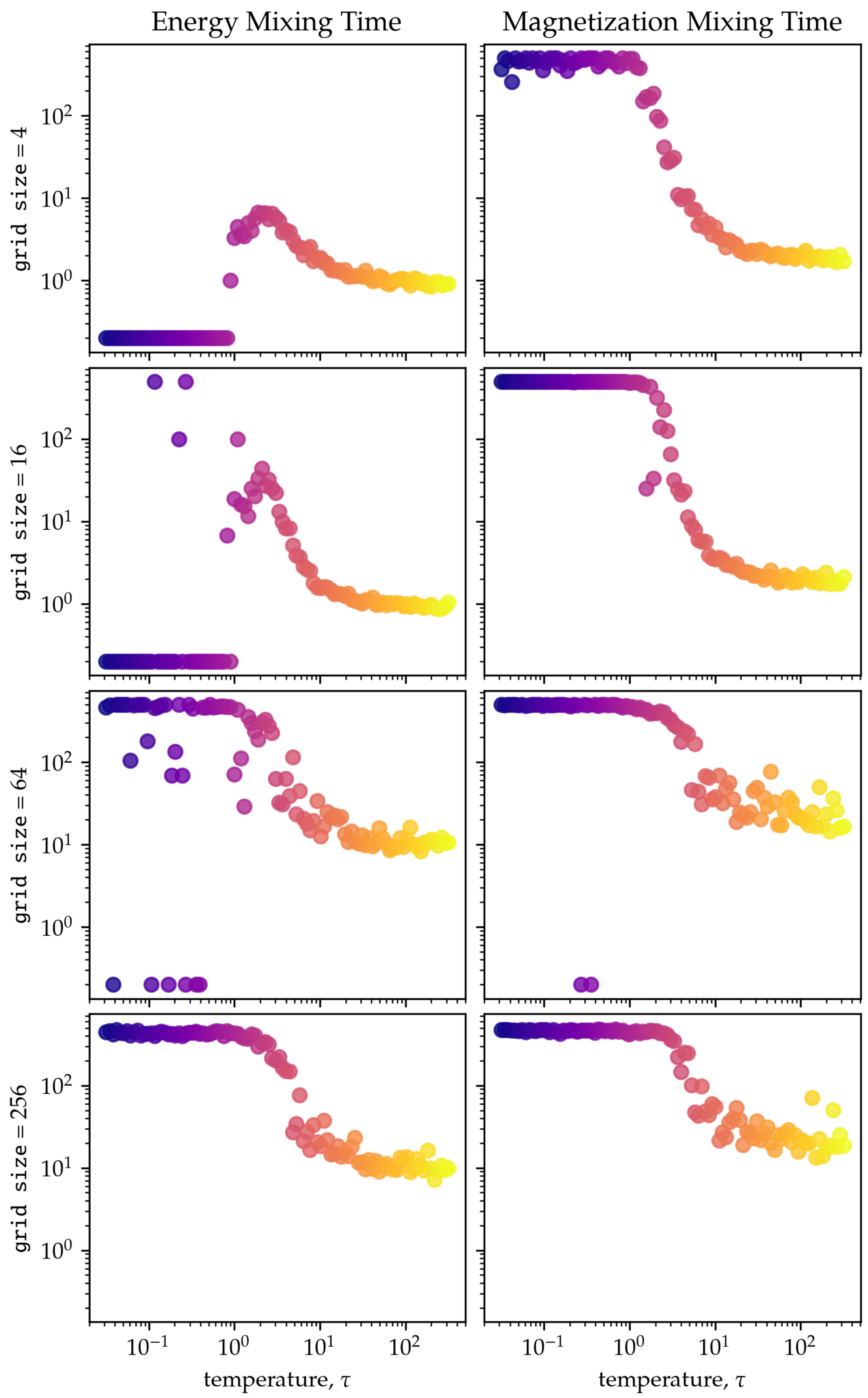

FIGURE 3: Energy $M(\mathbf{s})$ and magnetization $M(\mathbf{s})$ mixing time calculated as total number of samples divided by effective number of samples for grid sizes 4, 16, 64, and 256

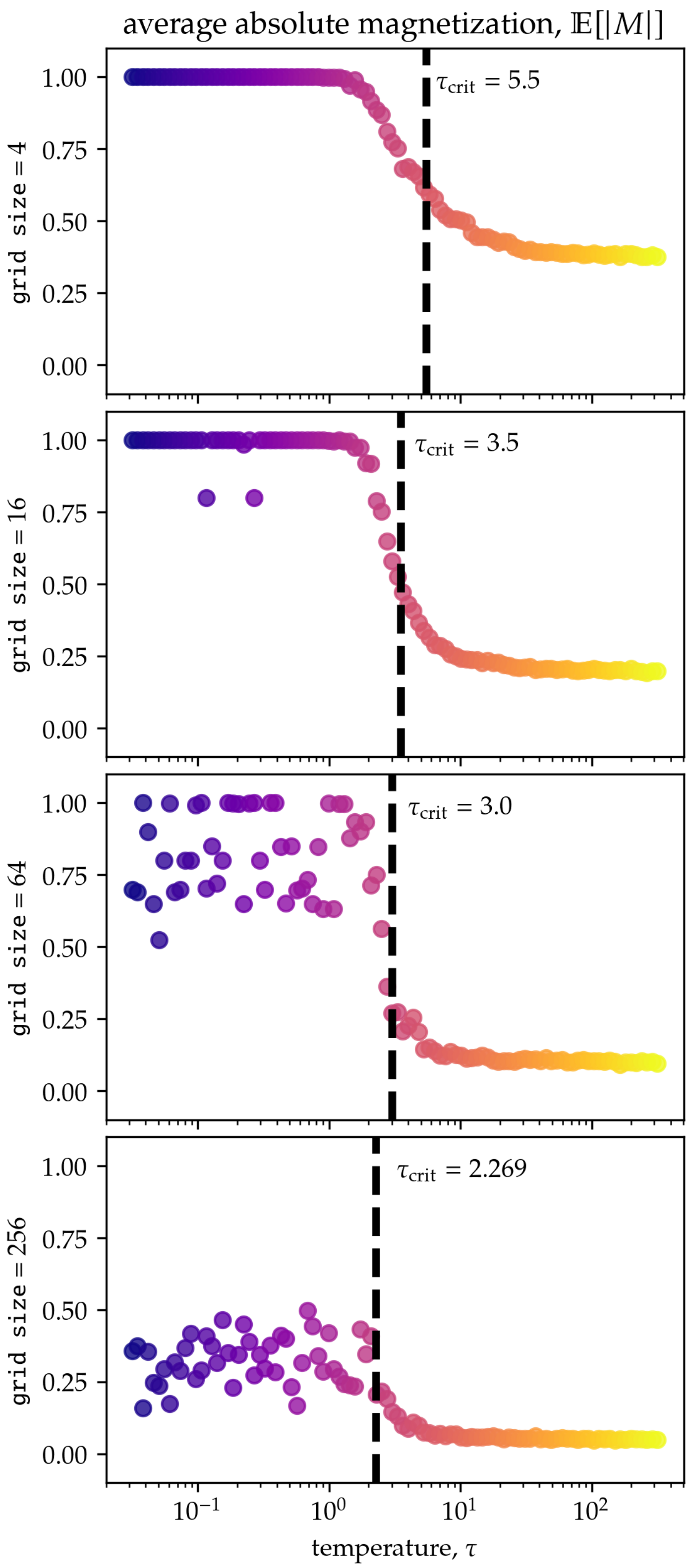

FIGURE 4: Average absolute magnetization, $\mathbb{E}[|M|]$, for grid sizes 4, 16, 64, and 256. The vertical black dashed lines are the estimated critical temperature τ_{crit}

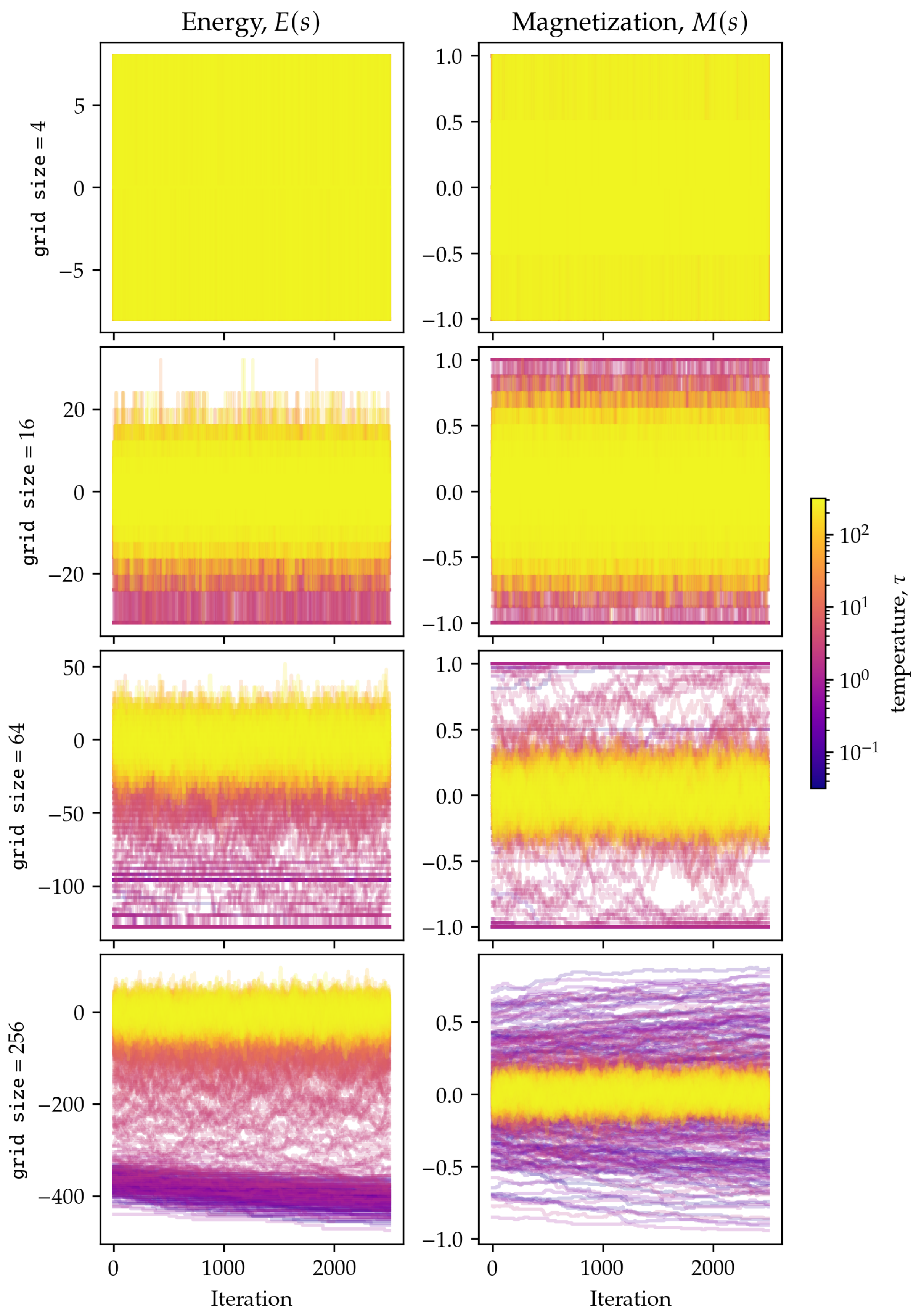

Figure 5: Energy and magnetization samples after burn in for grid sizes 4, 16, 64, and 256

(b) *Simulated Tempering*

1. Read BDA Chapter 12.3 on simulated tempering.

   Suppose our target density is our posterior $p(\theta \mid y)$. For simulated tempering, we define a set of $K+1$ distributions $p_k(\theta \mid y)$ for $k = 0, 1, 2, \ldots, K$ where $p_0(\theta \mid u) = p(\theta \mid y)$ and $p_1, \ldots, p_K$ are distributions with the same basic shape but are tempered for improved opportunities for mixing across modes. As in most cases, we don't have access to the normalized target, but we do have access to the unnormalized target $q_k(\theta)$.

   Simulated tempering works by setting a ladder of unnormalized densities $q_k(\theta)$

   $$q_k(\theta) = p(\theta \mid y)^{1/T_k} p_0(\theta)^{1-1/T_k} \quad \text{for} \quad k = 0, 1, 2, \ldots, K$$

   for a set of 'temperature' parameters $T_k > 0$. We set $T_0 = 1$ such that $q_0(\theta) = p(\theta \mid y)$. Large values of T_k smooth out the target density (e.g., produce less highly peaked mode), going in the limit to some 'base measure' p_0 that is some convenient prechosen distribution with high variance [BDA3]. The simulated tempering algorithm is as follows:

```
Algorithm 1 simulated tempering
Input:
    • $q_k(\theta)$: set of $K$ unnormalized target distributions
    • $k_t$: an integer identifying the distribution used at iteration $t$
    • $p_r$: proposal distribution for sample $\theta^*$ candidates
    • $J$: jumping distribution for sampling $k^*$ candidates
Initialize: $\mathcal{S} \leftarrow \emptyset$

  for $t = 1, 2, \ldots, T$ do
      Draw a candidate sample from the proposal distribution: $\theta^* \sim p_r(\theta_t \to \theta^*)$
      Compute acceptance ratio and acceptance probability:
          $a(\theta^*) = \min(r, 1)$   where   $r = \frac{q_{k_t}(\theta^*)\, p_r(\theta^* \to \theta_t)}{q_{k_t}(\theta_t)\, p_r(\theta_t \to \theta^*)}$      (1)
      if $U \le a(\theta^*)$ where $U \sim \text{Uniform}([0,1])$ then
          Accept candidate $\theta^*$, set $\theta_{t+1} = \theta^*$.
      else
          Reject candidate $\theta^*$, set $\theta_{t+1} = \theta_t$.
      if $k_t = 0$ then Append $\theta_{t+1}$ to $\mathcal{S}$
      Draw a candidate sample from the jumping distribution: $k^* \sim J(k_t \to k^*)$
      Compute acceptance ratio and acceptance probability:
          $a(k^*) = \min(r, 1)$   where   $r = \frac{q_{k^*}(\theta_{t+1})\, J(k^* \to k_t)}{q_{k_t}(\theta_{t+1})\, J(k_t \to k^*)}$      (2)
      if $U \le a(k^*)$ where $U \sim \text{Uniform}([0,1])$ then
          Accept candidate $k^*$, set $k_{t+1} = k^*$.
      else
          Reject candidate $k^*$, set $k_{t+1} = k_t$.
  return $\mathcal{S}$
```

2. Show that, if we apply simulated tempering with p_0 uniform, then the artificial temperature parameter defined in simulated tempering can be absorbed into the choice of temperature in the Ising model. Argue that, simulated tempering for the Ising model can be viewed as a MCMC technique that generates coupled random walks for a series of parallel Ising processes each at a different temperature.

   Suppose we wish to sample from the 2D Ising model where our target distribution is $p_*(s) = \mathbb{P}[S = s] \propto \exp\left(-\frac{1}{\tau}E(s)\right)$ with $E(s) = -\frac{J}{2}\sum_{\langle i,j \rangle} s_i s_j$. When using a uniform $p_0(s) \propto 1$, our unnormalized targets are given by

   $$q_k(s) = p_*(s)^{1/T_k} p_0(s)^{1-1/T_k} \propto \exp\left(-\frac{1}{\tau}E(s)\right)^{1/T_k} = \exp\left(-E(s)\right)^{1/\tau T_k} = \exp\left(-E(s)\right)^{1/\beta_k} \quad \text{for} \quad k = 0, 1, 2, \ldots, K$$

where we define $\beta_k = \tau T_k$ such that the simulated tempering temperature parameter T_k is absorbed into Ising model temperature τ.

We can think of our simulated tempering approach with $\beta_k = \tau T_k$ as follows:

- We initialize K Ising processes with temperatures β_k
- We randomly select $\ell \in \{0, 1, \dots, K\}$ to be the 'leader' of the parallel processes
- At each step...
  (a.) We propose candidate from the leader's tempered target $s' \sim q_\ell(s)$
  (b.) We accept or reject candidate s' via Metropolis-Hastings
  (c.) We propose new leader from $\ell' \in \{0, 1, \dots, K\}$ from temperature jumping distribution
  (d.) We accept or reject new leader ℓ' via Metropolis-Hastings

We use 'leader' as it describes how the sampling will unfold. You can imagine the K Ising processes as a group of walkers *that move together*. They rotate who leads the group by proposing (and accepting or rejecting) a new leader at each step. And importantly each walker only save samples during their steps as the leader. In this way, simulated tempering for the Ising model can be viewed as a MCMC technique that generates coupled random walks for a series of parallel Ising processes each at a different temperature.

3. Adapt your MH algorithm (with independent spin flips) to incorporate simulated tempering. Choose p_0 to be the uniform distribution over spin arrangements. Use your exploratory analysis in part (a) to choose a temperature ladder.

We choose a symmetric jumping distribution such that our acceptance probability in Equation (2) becomes

$$a(k^*) = \min(r, 1) \quad \text{where} \quad r = \frac{q_{k^*}(s_{t+1})\, J(k^* \to k_t)}{q_{k_t}(s_{t+1})\, J(k_t \to k^*)} = \frac{p_*(s_{t+1})^{1/T_{k^*}}}{p_*(s_{t+1})^{1/T_{k_t}}} = \frac{\left(\frac{1}{Z_{k^*}}\right) \tilde{p}_*(s_{t+1})^{1/T_{k^*}}}{\left(\frac{1}{Z_{k_t}}\right) \tilde{p}_*(s_{t+1})^{1/T_{k_t}}} = \left(\frac{Z_{k_t}}{Z_{k^*}}\right) \tilde{p}_*(s_{t+1})^{\frac{1}{T_{k^*}} - \frac{1}{T_{k_t}}}$$

$$r = \left(\frac{Z_{k_t}}{Z_{k^*}}\right) \exp\left(-E(s_{t+1})\right)^{\frac{1}{\beta_{k^*}} - \frac{1}{\beta_{k_t}}} \quad \text{where} \quad Z_k := \int \exp\left(-E(s)\right)^{1/\beta_k} \, \mathrm{d}s$$

It is often more convenient computationally to compute the acceptance probability in log space:

$$a(k^*) = \exp\big(\min\{\log(r), 0\}\big) \quad \text{where} \quad \log(r) = \log(Z_{k_t}) - \log(Z_{k^*}) - \left(\frac{1}{\beta_{k^*}} - \frac{1}{\beta_{k_t}}\right) E(s_{t+1})$$

We select a ladder where $\beta_0 = \tau_{\text{hard}}$ (while normally β_0 is 1, β_0 is temperate the returns the target) corresponds to a difficult temperature (e.g., a cool temperature with a large mixing time) the remaining temperatures $\beta_1, \dots, \beta_K$ include easier temperatures (e.g., hot temperatures with small mixing times). We space the ladder geometrically such that temperatures are clustered around $\beta_0 = 1$.

Tempering Ladder: We select a low temperature target distribution $\tau_{\text{hard}} = 0.1$ that is difficult to sample.

k	0	1	2
T_k	1	10	100
β_k	0.1	1	10

4. Is your algorithm the same as an importance sampling, rejection sampling, or importance resampling procedure that uses a high temperature Ising model as a proposal for a lower temperature target? Discuss the conceptual similarities and differences between this approach and the simulated tempering approach.

Consider simulated tempering with only two temperatures where β_0 corresponds to a low temperature distribution (the target) while β_1 corresponds to a high temperature distribution (proposal). Assuming our simulated tempering algorithm is sufficiently tuned, we will draw about half of samples from the target, cold distribution and half from the proposal, hot distribution. We will then only keep the half corresponding to the target low temperature. Simulated tempering uses an MCMC algorithm to transverse the sample space. Therefore, the stationary distribution of our Markov chain will follow the target distribution (or at least they will in the limit). The tempering allows us to more readily discover modes and therefore may decrease mixing time.

Importance Sampling and Importance Resampling: Recall that importance sampling is a procedure in which we collect a set of m independent samples from the proposal $\{x_1, \dots, x_m\}$. We then calculate the importance of each sample $\tilde{w}(x_j) = \frac{\tilde{p}_*(x_j)}{p_r(x_j)}$. We then select a subset of our samples via importance resampling. Specifically, we sequentially sampling x_j from $\{x_1, \ldots, x_m\}$ with probabilities proportional to the importance weights. This is done without replacement, so we rebalance the importance weights of the remaining samples at each step. Full summaries of importance sampling and importance resampling are provided.

Importance resampling with a high temperature proposal and a low temperature target is quite similar to simulated tempering. We first note a major difference: importance resampling only ever samples from the proposal, high temperature distribution; it never samples from the target, low temperature distribution directly. Consider the case where the proposal is extremely high temperature such that we essentially collect $\{x_1, \ldots, x_m\}$ uniformly at random. Then when we importance resample, we are effectively sampling from the target as $p_r \propto 1$ such that $\tilde{w}(x_j) \propto \tilde{p}_*(x_j)$. So in this sense, we essentially explore the space via the proposal (much like simulated tempering), and then keep important samples with high target, low temperature density.

Rejection Sampling: Recall that rejection sampling is the procedure in which we collect a set of n independent samples. At each step, we draw a candidate from the proposal $Y \sim p_r$ and accept or reject the candidate according to the acceptance probability $a(Y) = \frac{\tilde{p}_*(Y)}{Mp_r(Y)}$ where $M = \max_x \left\{\frac{\tilde{p}_*(x)}{p_r(x)}\right\}$ is a constant to ensure that $\frac{\tilde{p}_*(x)}{Mp_r(x)} \leq 1$ for all x. We repeat until we have accepted n candidates. A full summary of rejection sampling is provided.

Rejection sampling with a high temperature proposal and a low temperature target is quite similar to simulated tempering. Like importance sampling, rejection sampling only ever samples from the proposal, high temperature distribution and never directly from the target. However, we keep samples according to acceptance $\frac{\tilde{p}_*(Y)}{Mp_r(Y)}$. So while we only sample from the high temperature proposal, we are more likely accept candidates that have high-probability on the target, low temperature distribution.

5. Apply your simulated tempering algorithm to the Ising model example you identified as challenging in part (a). Is your simulated tempering method more or less computationally efficient (does it take more or less computational effort to achieve the same effective sample size)? Why or why not?

   We applied our simulated tempering algorithm to the Ising model example we identified as challenging in part (a) ($\tau_{\text{hard}} = 0.1$). For fairness, we ran the similar number of iterations for both the Metropolis-Hastings and simulated tempering algorithms. Specifically, we ran Metropolis-Hastings for 5000 with a burn-in of 2500 samples. We ran simulated tempering for 7500 iterations and used a fairly sparse temperature ladder with $K = 3$ temperatures such that the approximate number of samples from the target was 2500. With these settings, we ran each approach 100 times and calculated the effective sample size (ESS) for both algorithms. We summarize the quantiles of the ESS in Table 1 for both the energy and magnetization.

	Energy N_{eff}					Magnetization N_{eff}				
	2.5%	25%	50%	75%	97.5%	2.5%	25%	50%	75%	97.5%
Metropolis-Hastings	1.17	1.23	1.26	1.36	2.14	1.28	1.33	1.64	5.24	13.59
Simulated Tempering	4.88	23.90	31.23	40.28	54.09	1.89	7.01	11.18	14.41	24.54

TABLE 1: Effective sample size for energy and magnetization using Metropolis-Hastings and simulated tempering after 100 runs.

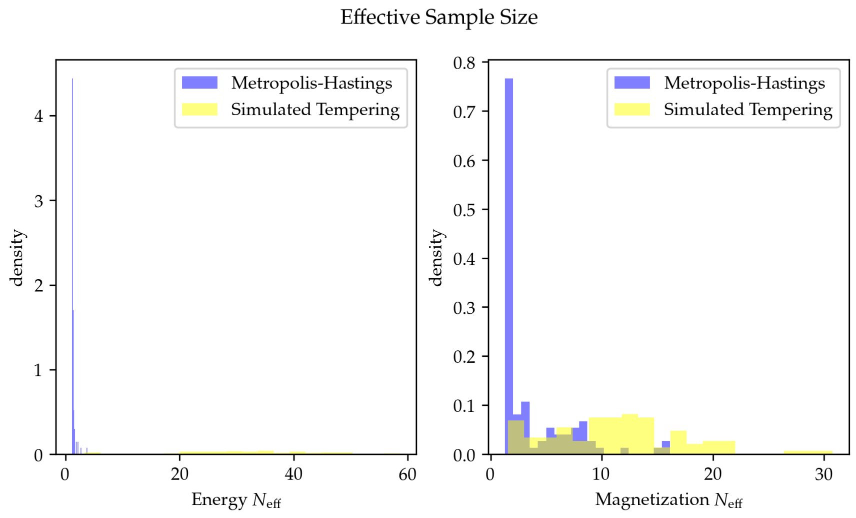

FIGURE 6: Effective sample size for energy and magnetization using Metropolis-Hastings and simulated tempering after 100 runs.

We also plot the samples from the Metropolis-Hastings and simulated tempering algorithms in Figure 7. We see that the Metropolis-Hastings samples often get stuck in either positive or negative magnetization states. This is expected as the Metropolis-Hastings algorithm is not able to easily traverse the sample space. In contrast, the simulated tempering samples are able to traverse the sample space more easily and therefore have a higher effective sample size.

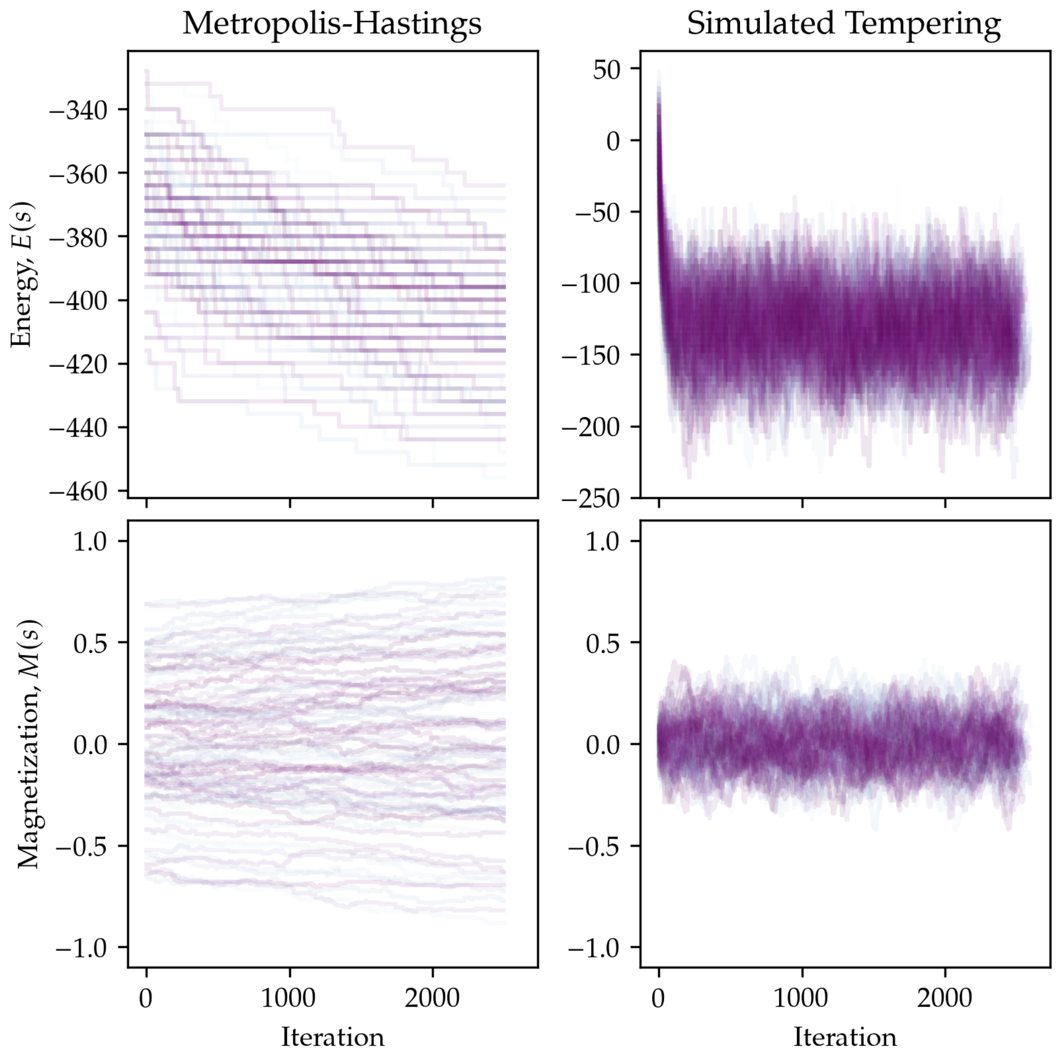

FIGURE 7: Energy and magnetization samples for Metropolis-Hastings and simulated tempering for 100 runs with $\tau_{\text{hard}} = 0.1$.

Importance Sampling and Importance Resampling

Let p_*, $\tilde{p}_*$, and p_r be the target, unnormalized target, and proposal distributions. Let $w(x) = \frac{p_*(x)}{p_r(x)}$ and $\tilde{w}(x) = \frac{\tilde{p}_*(x)}{p_r(x)}$ be the normalized and unnormalized importance weights such that

$$p_*(x) = \left(\frac{p_*(x)}{p_r(x)}\right) p_r(x) = w(x)\, p_r(x) \quad \text{and} \quad \tilde{p}_*(x) = \left(\frac{\tilde{p}_*(x)}{p_r(x)}\right) p_r(x) = \tilde{w}(x)\, p_r(x)$$

$$p_r(x) = \frac{p_*(x)}{w(x)} = \frac{\tilde{p}_*(x)}{\tilde{w}(x)} = \frac{Zp_*(x)}{\tilde{w}(x)} \quad \implies \quad w(x) = \left(\frac{1}{Z}\right)\tilde{w}(x)$$

The goal of importance sampling is to estimate the expectation $\mathbb{E}_{x\sim p_*}[f(x)]$ using n iid samples from the proposal: We have

$$\mathop{\mathbb{E}}_{x\sim p_*}\left[f(x)\right] = \int_{\text{all } x} f(x)\, p_*(x)\, \mathrm{d}x = \int_{\text{all } x} f(x)\, w(x)\, p_r(x)\, \mathrm{d}x = \mathop{\mathbb{E}}_{x\sim p_r}\left[f(x)\, w(x)\right]$$

But we generally don't have access to the normalized target p_* so we therefore don't have access to the importance weights as $w(x) = \frac{p_*(x)}{p_r(x)}$. We will therefore estimate the importance weights as follows

$$\mathop{\mathbb{E}}_{x\sim p_r}[\tilde{w}(x)] = \int_{\text{all } x}\left(\frac{\tilde{p}_*(x)}{p_r(x)}\right) p_r(x)\, \mathrm{d}x = \int_{\text{all } x}\tilde{p}_*(x)\, \mathrm{d}x = Z \quad \implies \quad w(x) = \frac{\tilde{w}(x)}{Z} = \frac{\tilde{w}(x)}{\mathbb{E}_{x\sim p_r}[\tilde{w}(x)]} \approx \frac{\tilde{w}(x)}{\frac{1}{n}\sum_{j=1}^{n}\tilde{w}(x_j)}$$

where $x_j \overset{\text{i.i.d.}}{\sim} p_r$ for $j = 1, \ldots, n$. With this approximation of $w(x)$, we can approximate $\mathbb{E}_{x\sim p_*}\left[f(x)\right]$. We have

$$\mathop{\mathbb{E}}_{x\sim p_*}\left[f(x)\right] = \mathop{\mathbb{E}}_{x\sim p_r}\left[f(x)\, w(x)\right] \approx \mathop{\mathbb{E}}_{x\sim p_r}\left[f(x)\left(\frac{\tilde{w}(x)}{\frac{1}{n}\sum_{j=1}^{n}\tilde{w}(x_j)}\right)\right] \approx \frac{\frac{1}{n}\sum_{j=1}^{n} f(x_j)\, \tilde{w}(x_j)}{\frac{1}{n}\sum_{j=1}^{n}\tilde{w}(x_j)} \quad \text{where } x_j \overset{\text{i.i.d.}}{\sim} p_r$$

Algorithm 2 Importance Resampling

Input: $\tilde{p}_*$ unnormalized target, p_r proposal distribution, and integers $m > n$
Initialize: $\mathcal{S} \leftarrow \emptyset$

```
1 Collect $m$ independent samples from proposal $\mathcal{X} = \{x_1, \ldots, x_m\}$ where $x_j \overset{\text{i.i.d.}}{\sim} p_r$ and $|\mathcal{X}| = m$
2 for $i = 1, 2, \ldots, n$ do
3     Compute and standardize unnormalized importance weights
          $\tilde{w}(x_j) = \frac{\tilde{p}_*(x_j)}{p_r(x_j)}$ for all $j = 1, \ldots, |\mathcal{X}|$ and $\hat{w}(x_j) = \frac{\tilde{w}(x_j)}{\sum_{k=1}^{|\mathcal{X}|}\tilde{w}(x_k)}$ such that $\sum_{j=1}^{|\mathcal{X}|}\hat{w}(x_j) = 1$
4     Sample $x'$ from $\mathcal{X}$ where the probability of sampling $x_j \in \mathcal{X}$ is proportional to the weight $\hat{w}(x_j)$
5     Append $x'$ to $\mathcal{S}$ and remove $x'$ from $\mathcal{X}$
6 return $\mathcal{S}$
```

Rejection Sampling

Let p_*, $\tilde{p}_*$, and p_r be the target, unnormalized target, and proposal distribution such that

$$\tilde{p}_*(x) := \frac{\tilde{p}_*(x)}{Z} \quad \text{where} \quad Z := \int_{\text{all } x} \tilde{p}_*(x)\, \mathrm{d}x \quad \text{and} \quad Mp_r(x) \geq \tilde{p}_*(x) \quad \text{for all } x.$$

The goal of rejection sampling is to collect a set of n independent samples $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ from the target:

Algorithm 3 Rejection Sampling

```
Input: $\tilde{p}_*$ unnormalized target, $p_r$ proposal distribution, and $M^\star = \max_x \left\{\frac{\tilde{p}_*(x)}{p_r(x)}\right\}$
Initialize: $\mathcal{S} \leftarrow \emptyset$

1 while $|\mathcal{S}| < n$ do
2     Draw candidate sample from the proposal distribution: $Y \sim p_r$
3     Compute acceptance probability: $a(Y) = \frac{\tilde{p}_*(Y)}{M^\star p_r(Y)} \in [0, 1]$
4     if $U \leq a(Y)$ where $U \sim \text{Uniform}([0, 1])$ then
5         Append $Y$ to $\mathcal{S}$
6     else
7         Reject $Y$
8 return $\mathcal{S}$
```

It follows that the probability that we accept a candidate sample is

$$\mathbb{P}[Y \text{ is accepted}] = \mathbb{P}[U \leq a(Y)] = \int_{\text{all } x} \mathbb{P}[U \leq a(Y) \mid Y = x]\ \mathbb{P}[Y = x]\, \mathrm{d}x = \int_{\text{all } x} \mathbb{P}[U \leq a(x)]\ \mathbb{P}[Y = x]\, \mathrm{d}x$$

Recall that $\mathbb{P}[U \leq c] = c$ for any $c \in [0, 1]$ implying that $\mathbb{P}[U \leq a(x)] = a(x)$. Therefore, we have

$$\mathbb{P}[Y \text{ is accepted}] = \int_{\text{all } x} \frac{\tilde{p}_*(x)}{Mp_r(x)}\, p_r(x)\, \mathrm{d}x = \frac{1}{M} \int_{\text{all } x} \tilde{p}_*(x)\, \mathrm{d}x = \frac{Z}{M}.$$

To complete, we solve for $M^\star$, the constant that maximizes our acceptance probability. We have

$$M^\star = \underset{M}{\operatorname{argmax}} \left\{\mathbb{P}[Y \text{ is accepted}]\right\} = \underset{M}{\operatorname{argmax}} \left\{\frac{Z}{M}\right\} = \underset{M}{\operatorname{argmax}} \left\{\frac{1}{M}\right\} = \underset{M}{\operatorname{argmin}} \{M\}$$

But M has to satisfy $M \geq \frac{\tilde{p}_*(x)}{p_r(x)}$ so we have that $\min_M \{M\} = \max_x \left\{\frac{\tilde{p}_*(x)}{p_r(x)}\right\}$ implying that $\boxed{M^\star = \max_x \left\{\frac{\tilde{p}_*(x)}{p_r(x)}\right\}}$

(c) *Ensemble Methods*

An ensemble method (or particle filtering method) is an MCMC method that evolves many chains in parallel rather than a single chain. The chains interact, with interaction rules chosen to speed convergence. At any time, an ensemble method consists of a population of particles, each assigned a state that updates according to an MCMC procedure.

For example, we could track the importance weights assigned to each particle had we used the collection of particles as a proposal. Particles with large weights have discovered regions where the target is under-represented by the ensemble, while particles with small weights occupy over-represented regions. This information can be used to move particles from over-represented regions to under-represented regions. In this sense, particles that discover an underexplored region of the target can rapidly recruit other particles to join their neighborhood. In general, these methods do not help the MCMC method discover new modes, but can drastically reduce the mixing time needed to balance mass between, and to fully explore, separate modes.

1. Read Lindsey, Weare and Zhang. *Ensemble Markov Chain Monte Carlo with Teleporting Walkers*. (2022). Briefly summarize the method proposed here, how it shares information between particles/walkers to speed convergence, and some example target distributions where it might be an effective tool.[1]

Lindsey, Weare and Zhang propose an MCMC method where an ensemble of coupled walkers sample from a target distribution [LWZ22]. A summary of the algorithm is below:

Suppose we are given a target (possible unnormalized) distribution $\pi(x)$ on a space $\mathcal{X}$. We wish to sample from the joint target of N walkers, i.e., $\mathbf{x} = (x_1, \dots, x_N) \sim \Pi(\mathbf{x}) = \prod_{i=1}^N \pi(x_i)$. While $(x_1, \dots, x_N)$ are independent on the joint density, the walkers are coupled such that walkers are able to 'teleport' to another walker's location (and thus will not produce N independent chains). Let $p_r(\cdot \to \cdot)$ represent the proposal distribution from we will sample candidates $\mathbf{x}'$. At each step our algorithm, we will accept or reject the candidate according to a Metropolis-Hastings acceptance probability. A full summary of the algorithm is below:

Algorithm 4 Ensemble Markov Chain Monte Carlo with Teleporting Walkers [LWZ22]

Input: $\pi(x)$ target distribution (possibly unnormalized), p_r proposal distribution, fixed N number of random walkers
Initialize: N random walkers with initial positions $\mathbf{x} = (x_1, \dots, x_N) \in \mathcal{X}^N$

```
1 for $t = 1, 2, \dots, T$ do
2     Sample teleporter's arrival index $j \in \{1, 2, \dots, N\}$ uniformly at random and sample $z \sim p_r(x_j \to z)$
3     Sample teleporter's starting index $i \in \{1, 2, \dots, N\}$ (possibly equal to $j$) according to importance weights
```

$$w_i(\mathbf{x}, z) := \frac{p_r(z \to x_i) + \sum_{k \neq i}^N p_r(x_k \to x_i)}{\pi(x_i)} \Bigg/ Z(\mathbf{x}, z) \quad \text{where} \quad Z(\mathbf{x}, z) := \sum_{l=1}^N \frac{p_r(z \to x_l) + \sum_{k \neq l}^N p_r(x_k \to x_l)}{\pi(x_l)} \tag{3}$$

```
4     Construct candidate $\mathbf{x}'$: set $\mathbf{x}' = (x'_1, \dots, x'_N) \leftarrow \mathbf{x} = (x_1, \dots, x_N)$ and teleport walker by overwriting $x'_i \leftarrow z$†
5     Compute acceptance probability and acceptance ratio:
```

$$a(\mathbf{x}') = \min(r, 1) \quad \text{where} \quad r = \frac{\Pi(\mathbf{x}')\, p_r(\mathbf{x}' \to \mathbf{x})}{\Pi(\mathbf{x})\, p_r(\mathbf{x} \to \mathbf{x}')} = \frac{Z(\mathbf{x}, z)}{Z(\mathbf{x}', x_i)} \tag{4}$$

```
6     if $U \leq a(\mathbf{x}')$ where $U \sim \text{Uniform}([0,1])$ then
7         Accept candidate $\mathbf{x}'$, set $\mathbf{x}_{t+1} = \mathbf{x}'$.
8     else
9         Reject candidate $\mathbf{x}'$, set $\mathbf{x}_{t+1} = \mathbf{x}$.
```

†Conceptually, the candidate is the same as the current sample but the walker at x_i is teleported to x_j and takes one step $z \sim p_r(x_j \to z)$.

To appreciate how the algorithm works, let us examine and derive the acceptance probability in Equation (4). The probability of proposing $\mathbf{x}'$ given current sample $\mathbf{x}$ is

$$p_r(\mathbf{x} \to \mathbf{x}') = \mathbb{P}[\text{sampled index } i]\ \mathbb{P}[\text{sampled } z] = w_i(\mathbf{x}, z) \sum_{j=1}^N \mathbb{P}[z \mid x_j]\ \mathbb{P}[\text{sampled index } j] = w_i(\mathbf{x}, z)\, \frac{1}{N} \sum_{j=1}^N p_r(x_j \to z)$$

Noting that $\mathbf{x} = \mathbf{x}'$ for all indices except i where we have $x'_i = z$, our acceptance ratio r is given by

$$r = \frac{\Pi(\mathbf{x}')\, p_r(\mathbf{x}' \to \mathbf{x})}{\Pi(\mathbf{x})\, p_r(\mathbf{x} \to \mathbf{x}')} = \frac{\pi(x'_i)}{\pi(x_i)} \frac{\prod_{k \neq i} \pi(x'_k)}{\prod_{k \neq i} \pi(x_k)} \frac{w_i(\mathbf{x}', x_i)\, \frac{1}{N} \sum_{j=1}^N p_r(x'_j \to x_i)}{w_i(\mathbf{x}, z)\, \frac{1}{N} \sum_{j=1}^N p_r(x_j \to z)}$$

[1]For further reading on related methods see Lu, Lu, and Nolen. *Accelerating Langevin Sampling with Birth-death*. (2019).

$$r = \frac{\pi(x_i')}{\pi(x_i)} \frac{Z(\mathbf{x},z)}{Z(\mathbf{x}',x_i)} \frac{\left(\frac{p_r(x_i \to x_i') + \sum_{k\neq i}^{N} p_r(x_k' \to x_i')}{\pi(x_i')}\right)}{\left(\frac{p_r(z \to x_i) + \sum_{k\neq i}^{N} p_r(x_k \to x_i)}{\pi(x_i)}\right)} \frac{\sum_{j=1}^{N} p_r(x_j' \to x_i)}{\sum_{j=1}^{N} p_r(x_j \to z)} = \frac{Z(\mathbf{x},z)}{Z(\mathbf{x}',x_i)} \frac{p_r(x_i \to z) + \sum_{k\neq i}^{N} p_r(x_k' \to z)}{p_r(z \to x_i) + \sum_{k\neq i}^{N} p_r(x_k \to x_i)} \frac{\sum_{j=1}^{N} p_r(x_j' \to x_i)}{\sum_{j=1}^{N} p_r(x_j \to z)}$$

$$r = \frac{Z(\mathbf{x},z)}{Z(\mathbf{x}',x_i)} \frac{p_r(x_i \to z) + \sum_{k\neq i}^{N} p_r(x_k \to z)}{p_r(x_i' \to x_i) + \sum_{k\neq i}^{N} p_r(x_k' \to x_i)} \frac{\sum_{j=1}^{N} p_r(x_j' \to x_i)}{\sum_{j=1}^{N} p_r(x_j \to z)} = \frac{Z(\mathbf{x},z)}{Z(\mathbf{x}',x_i)} \frac{\sum_{k=1}^{N} p_r(x_k \to z)}{\sum_{k=1}^{N} p_r(x_k' \to x_i)} \frac{\sum_{j=1}^{N} p_r(x_j' \to x_i)}{\sum_{j=1}^{N} p_r(x_j \to z)}$$

$$r = \frac{Z(\mathbf{x},z)}{Z(\mathbf{x}',x_i)} \quad \checkmark$$

Ensemble MCMC can be well suited for multimodal target distributions as the walkers are able to explore the sampling space and jump between modes more easily. Consider the bimodal distribution in Figure 8. Suppose at time t, we randomly select walker x_j and pull candidate $z \sim p_r(x_j \sim z)$ (see pink dot in Figure 8). We next calculate the importance weights. For example, Figure 8 depicts the calculation of $w_i(\mathbf{x}, z) \propto p_r(z \to x_i) + \sum_{k\neq i}^{N} p_r(x_k \to x_i)$ where we sum the connections to x_i highlighted in blue with thicker lines representing higher transition probabilities. The importance weight of x_i be the largest as it has a strong connection to z, strong connections to the other particles, and $\pi(x_i)$ is small. Therefore, we are most likely to select x_i as the teleporting walker. Suppose that we in fact sample x_i and the candidate $\mathbf{x}'$ is accepted. Then x_i would teleport to z and we would keep the bottom row of particles in Figure 8. We note that by teleporting to z, the walker x_i is now closer to mode x_A than to mode x_B. This hightlights how teleportation is able to overcome a deep valley in density and make larger transitions in the sample space.

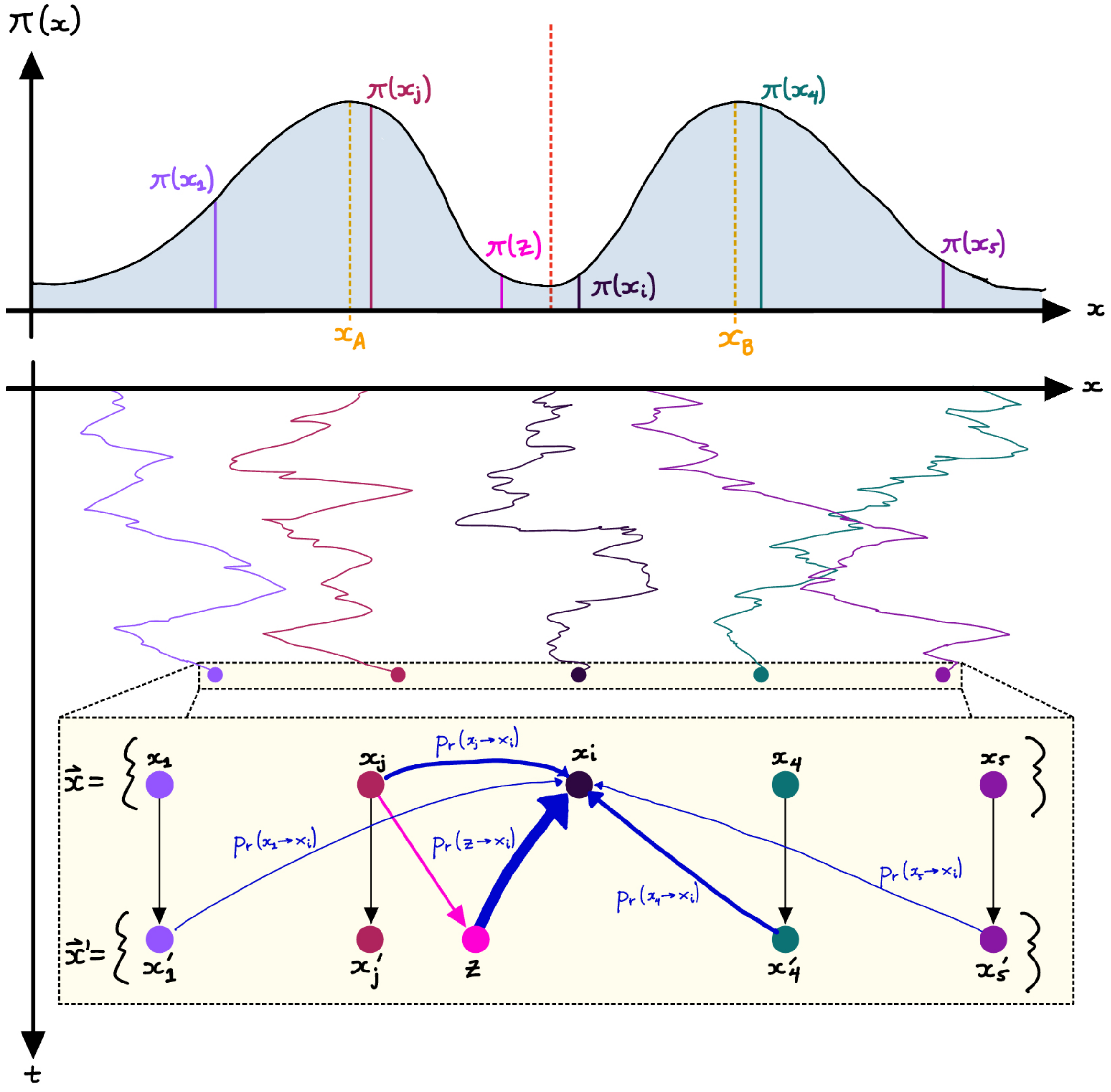

Figure 8: Ensemble MCMC with Teleporting Walkers

2. Propose, in pseudocode, an ensemble particle method for sampling from the Ising model. Are there parameter settings for the Ising model where you think an ensemble method might be useful?[2]

```
Algorithm 5 Ising Model + Ensemble MCMC with Teleporting Walkers [LWZ22]
Input: $p(\mathbf{s}) \propto \exp\left(-\frac{1}{\tau}E(\mathbf{s})\right)$ target distribution (unnormalized), $p_r$ proposal distribution
Initialize: $M$ random particles $\mathbf{S} = \{\mathbf{s}_1, \dots, \mathbf{s}_M\}$ where each $\mathbf{s}_k \in \{+1, -1\}^{N\times N}$ is an $N$-by-$N$ Ising model

for $t = 1, 2, \dots, T$ do
    Choose a particle $\mathbf{s}_j$ uniformly at random from $\mathbf{S}$
    Propose a new configuration $\mathbf{z} \sim p_r(\mathbf{s}_j \to \mathbf{z})$†
    Teleport particle $\mathbf{s}_i$ to $\mathbf{z}$, where $\mathbf{s}_i$ is selected from $\mathbf{S}$ according to weights‡
        $w_i(\mathbf{S},\mathbf{z}) := \frac{\tilde{w}_i(\mathbf{S},\mathbf{z})}{Z(\mathbf{S},\mathbf{z})}$ where $\tilde{w}_i(\mathbf{S},\mathbf{z}) := \frac{p_r(\mathbf{z}\to\mathbf{s}_i) + \sum_{k\neq i}^N p_r(\mathbf{s}_k \to \mathbf{s}_i)}{p(\mathbf{s}_i)}$ and $Z(\mathbf{S},\mathbf{z}) := \sum_{\ell=1}^N \tilde{w}_\ell(\mathbf{S},\mathbf{z})$
    Construct candidate by setting $\mathbf{s}_i = \mathbf{z}$ while keeping all other particles unchanged, $\mathbf{S}' = \{\mathbf{s}_1, \dots, \mathbf{z}, \dots, \mathbf{s}_M\}$
    Compute acceptance probability and acceptance ratio:
        $a(\mathbf{S}') = \min(r, 1)$ where $r = \frac{\Pi(\mathbf{S}')\, p_r(\mathbf{S}' \to \mathbf{S})}{\Pi(\mathbf{S})\, p_r(\mathbf{S} \to \mathbf{S}')} = \frac{Z(\mathbf{S},\mathbf{z})}{Z(\mathbf{S}',\mathbf{s}_i)}$
    if $U \leq a(\mathbf{S}')$ where $U \sim \text{Uniform}([0,1])$ then
        Accept candidate $\mathbf{S}'$, set $\mathbf{S}_{t+1} = \mathbf{S}'$.
    else
        Reject candidate $\mathbf{S}'$, set $\mathbf{S}_{t+1} = \mathbf{S}$.
```

†we use a symmetric proposal where we independently flip all of the dipoles with probability p_{flip}

‡by construction, it is possible that $\mathbf{s}_i = \mathbf{s}_j$ in which case the particle does not teleport, rather it takes a standard Metropolis-Hastings step

Low temperature Ising models is where an ensemble method will be useful as the target is bimodal in magnetization. Without loss of generality, consider the case where our Ising model is initialized with more positive than negative spins such that $M(\mathbf{s}) > 0$. Recall the Metropolis acceptance probability

$$a(s') = \min(r, 1) \quad \text{where} \quad r = \exp\left(-\frac{1}{\tau}\left[E(s') - E(s)\right]\right) = \exp\left(-\frac{1}{\tau}\left[\Delta E(s, s')\right]\right)$$

We note that we will always accept if $\Delta E(s, s') \leq 0$ and we accept with probability $\exp\left(-\frac{1}{\tau}\left[\Delta E(s, s')\right]\right)$ when $\Delta E(s, s') > 0$. In other words, we will always accepted if the proposed energy is less than the current energy. However, especially in the lower temperature regime (see Figure 2), we will infrequently accept candidates with greater energy than the current. This implies that our system is biased towards accepting more and more positive spins driving the system towards almost all positive spins. It would require many steps with large changes in energy to flip the system to negative magnetization. This is where ensemble particle MCMC methods shine. We can initialize N walkers and expect that half fall on the positive magnetization and half on the negative magnetization side ensuring with see both modes. Additionally, teleporting walkers would make jumping between modes easier.

Another simpler strategy for exploring both modes would be flipping all of the spins occasionally as flipping them all results in the same energy $E(-\mathbf{s}) = E(\mathbf{s})$ and flips the magnetization $M(-\mathbf{s}) = -M(\mathbf{s})$.

[2]Think about the symmetry of the model and the number of modes in the target distribution. Are there situations where the target is multimodal and standard MH fails to discover, or balance between, all the modes? Do you need an ensemble method to improve MH in this setting or are there simpler ways to exploit symmetry?

References

[GCS$^+$13] A. Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin, *Bayesian data analysis, third edition*, Chapman & Hall/CRC Texts in Statistical Science, Taylor & Francis, 2013.

[LWZ22] Michael Lindsey, Jonathan Weare, and Anna Zhang, *Ensemble markov chain monte carlo with teleporting walkers*, SIAM/ASA Journal on Uncertainty Quantification **10** (2022), no. 3, 860–885.

mini8_submission

May 2, 2025

```
[ ]: import numpy as np
     np.set_printoptions(precision=4)
     np.random.seed(123)
     import scipy
     import arviz as az

     # My plotting settings
     import os
     FIG_DIR = os.path.join(os.getcwd(), 'figures')
     if not os.path.exists(FIG_DIR):
         os.makedirs(FIG_DIR,exist_ok=True)
     import matplotlib
     import matplotlib.pyplot as plt
     from matplotlib import rcParams
     rcParams['text.usetex'] = True
     rcParams['text.latex.preamble'] = r'''
     \usepackage{newpxtext}
     \usepackage[vvarbb]{newpxmath}
     \usepackage{amsmath}
     \DeclareMathOperator*{\E}{\mathbb{E}}
     \DeclareMathOperator*{\Var}{Var}
     '''
     rcParams["font.family"] = "Palatino"
     rcParams['figure.dpi'] = 300
```

0.0.1 Helper functions from Lab 8

```
[ ]: def initialize_grid(N):
         """
         Initialize an N x N grid with spins randomly set to +1 or -1.

         Each spin s_i is sampled independently from {+1, -1}.
         """
         return np.random.choice([-1, 1], size=(N, N))

     def compute_energy_slow(grid, J=1):
         """
```

```
    Compute the energy of the grid using the Ising Hamiltonian with periodic␣
↪boundaries.

    Mathematical Formulation:
      H(s) = -J * sum_{<i,j>} s_i * s_j,
    with each nearest-neighbor pair counted twice.
    """
    N = grid.shape[0]
    energy = 0
    for i in range(N):
        for j in range(N):
            s = grid[i, j]
            # Nearest neighbors with periodic boundary conditions.
            neighbors = (grid[(i+1) % N, j] +
                         grid[(i-1) % N, j] +
                         grid[i, (j+1) % N] +
                         grid[i, (j-1) % N])
            energy += -J * s * neighbors
    return energy / 2  # Correct for double counting

def compute_energy(grid, J=1):
    neighbors = (
        np.roll(grid,  1, axis=0) + np.roll(grid, -1, axis=0) +
        np.roll(grid,  1, axis=1) + np.roll(grid, -1, axis=1)
    )
    return -J * np.sum(grid * neighbors) / 2

def proposal_independent_slow(grid, flip_prob):
    """
    For each spin in the grid, flip its sign with probability flip_prob.

    This implements a local update with a symmetric proposal.
    """
    new_grid = grid.copy() # set the new grid of spins equal to the old grid of␣
↪spins
    N = grid.shape[0] # extract the side-length
    for i in range(N):
        for j in range(N):
            if np.random.rand() < flip_prob:
                new_grid[i,j] *= -1
    return new_grid

def proposal_independent(grid, flip_prob):
    flips = np.random.rand(*grid.shape) < flip_prob
    return grid * np.where(flips, -1, 1)

def metropolis_step(grid, proposal_func, temperature, **proposal_kwargs):
    """
    Perform one Metropolis-Hastings update step.

    Steps:
      1. Compute current energy H(s).
      2. Generate proposed state s' using proposal_func.
      3. Compute ΔH = H(s') - H(s).
      4. Accept s' with probability: min(1, exp(-ΔH/T)).

    Returns:
      new_grid: the updated configuration.
      accepted: True if the move is accepted; otherwise, False.
    """
    current_energy = compute_energy(grid)

    proposed_grid = proposal_func(grid, **proposal_kwargs)

    proposed_energy = compute_energy(proposed_grid)

    delta_E = proposed_energy-current_energy # compute the change in energy

    r = np.exp( -(1/temperature)*(delta_E) )
    if np.random.rand() <= min(r,1): # implement the MH acceptance rule here
        return proposed_grid, True
    else:
        return grid, False

def run_MH_simulation(initial_grid, proposal_func, temperature, n_steps,␣
 ↪**proposal_kwargs):
    """
    Run the MH simulation for n_steps iterations.

    Records:
      - Energy: H(s)
      - Magnetization: M = (1/N^2) * sum(s)

    Returns:
      grid: Final configuration.
      energies: Array of energy values over iterations.
      magnetizations: Array of magnetization values over iterations.
    """
    grid = initial_grid.copy()
    energies = []
    magnetizations = []
    acceptance_rate = 0 # Initialize acceptance rate

    for step in range(n_steps):
        grid, accepted = metropolis_step(grid, proposal_func, temperature,␣
↪**proposal_kwargs)
        energies.append(compute_energy(grid))
        magnetizations.append(np.mean(grid))
        acceptance_rate += int(accepted) # modify to track the acceptance rate

    acceptance_rate /= n_steps

    # print(f'Acceptance rate: {acceptance_rate:0.3f}')

    return grid, np.array(energies), np.array(magnetizations), acceptance_rate
```

0.0.2 Mixing Times for MH in the Ising Model

The Ising model is often introduced since it provides a useful toy environment for exploration and experimentation. I've outlined an experiment below that could be adapted as part of a mini-project.

Select the fastest method, and use it to explore the mixing time (total number of samples divided by effective number of samples), as a function of gridsize and temperature. I'd suggest using a logarithmic spacing for both, a coarse grid size sampling (say sizes 4, 16, 64, 256), and a fine temperature spacing.

In particular, if you plot the average absolute magnetization (after burn-in) as a function of temperature for each grid size, you should see a transition between a demagnetized and a magnetized state as the material cools. This transition should sharpen, as a function of temperature, as the grid size increases. The critical temperature for the material is, roughly speaking, the temperature at which this transition occurs. We can estimate it hueristically by either selecting the temperature at which the average absolute magnetization crosses 1/2 its peak value (value at absolute zero), or, where it changes the quickest as a function of temperature.

Estimate the critical temperature for each grid size, then make a plot showing the mixing time as a function of grid-size at three different temperatures: near absolute zero, at the critical temperature per grid size, and at a very high temperature. Pick your axis scales appropriately (e.g. linear vs log). How does the mixing time depend on the temperature? Is it slowest at criticality?

0.0.3 Mixing Times

[ ]:
```
# Grid sizes, temperatures, and flip probabilities
grid_sizes = [2, 4, 8, 16]
flip_props = [0.1, 0.1, 0.01, 0.01]
temps      = np.logspace(-1.5, 2.5, 100)

# Simulation parameters
n_steps   = 5000
burn_in   = n_steps//2
n_chains  = 5
chains    = np.arange(n_chains)
temp_crit = 2/(np.log(1+np.sqrt(2)))
```

```
runs = {g:{t:{} for t in temps} for g in grid_sizes}
print('-'*50)
for g, flip_prob in zip(grid_sizes, flip_props):
    for t in temps:
        E_chains     = np.zeros((n_chains,n_steps))
        M_chains     = np.zeros((n_chains,n_steps))
        accept_rates = np.zeros(n_chains)
        for c in chains:
            grid_init = initialize_grid(g)
            grid_final, E, M, accept_rate = run_MH_simulation(
                grid_init, proposal_independent, t, n_steps, flip_prob=flip_prob
            )
            E_chains[c,:]   = E
            M_chains[c,:]   = M
            accept_rates[c] = accept_rate
        E_ess = az.ess(E_chains[:,burn_in:])
        M_ess = az.ess(M_chains[:,burn_in:])
        N_samples_after_burn_in = len(E_chains[:,burn_in:].flatten())
        E_mix_time = N_samples_after_burn_in/E_ess
        M_mix_time = N_samples_after_burn_in/M_ess
        runs[g][t] = {'E': E_chains[:,burn_in:], 'M': M_chains[:,burn_in:],
                      'E_mix_time': E_mix_time , 'M_mix_time': M_mix_time}

        print(f"grid_size={g}, temp={t:0.2f}")
        print(f" N_eff   : energy={E_ess:0.2f}, magnetization={M_ess:0.2f}")
        print(f"Mixing T : energy={E_mix_time:0.2f}, magnetization={M_mix_time:
 ↪0.2f}")
        print(f"Acceptance rates : {accept_rates}")
        print('-'*50)
```

[ ]:
```
# Plot mixing times
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(6, 10), sharex=True,␣
 ↪sharey=True)
plt.subplots_adjust(wspace=0.05, hspace=0.05)

# Define a sequence of colors for different temperatures
cmap   = plt.cm.plasma
colors = cmap(np.linspace(0, 1, len(temps)))
for i, (g, t_runs) in enumerate(runs.items()):
    for t_idx, (t, EM) in enumerate(t_runs.items()):
        avg_abs_magnetization = np.mean(np.abs(EM['M']))
        axes[i,0].scatter(t, EM['E_mix_time'], alpha=0.8, color=colors[t_idx])
        axes[i,1].scatter(t, EM['M_mix_time'], alpha=0.8, color=colors[t_idx])
        axes[i,0].set_ylabel(f"\\texttt{{grid size}} = {int(g**2)}")
        axes[i,0].set_xscale('log')
        axes[i,0].set_yscale('log')
        axes[i,1].set_xscale('log')
        axes[i,1].set_yscale('log')

axes[0,0].set_title("Energy Mixing Time")
axes[0,1].set_title("Magnetization Mixing Time")
axes[-1,0].set_xlabel(r"temperature, $\tau$")
axes[-1,1].set_xlabel(r"temperature, $\tau$")
plt.savefig(os.path.join(FIG_DIR,'mixing_times.png'), bbox_inches='tight')
plt.show()
```

[ ]:
```
# Estimated critical temperates after plotting
critical_temps = [5.5,3.5,3.0,2.269]

# Plot average absolute magnetization
fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(4, 10), sharex=True)
axes = axes.flatten()
plt.subplots_adjust(wspace=0.5, hspace=0.05)

# Define a sequence of colors for different temperatures
cmap   = plt.cm.plasma
colors = cmap(np.linspace(0, 1, len(temps)))
for i, ((g, t_runs), t_crit) in enumerate(zip(runs.items(), critical_temps)):
    for t_idx, (t, EM) in enumerate(t_runs.items()):
        avg_abs_magnetization = np.mean(np.abs(EM['M']))
        label = f"$\\tau={t:0.2f}$" # if c==0 else None
        axes[i].scatter(t, avg_abs_magnetization, alpha=0.8, label=label,␣
 ↪color=colors[t_idx])
    axes[i].axvline(t_crit,color='k',linestyle='--', linewidth=3)
    trans = matplotlib.transforms.blended_transform_factory(axes[i].transData,␣
 ↪axes[i].transAxes)
    axes[i].text(t_crit+1,0.9,rf"$\tau_{{\mathrm{{crit}}}} =␣
 ↪{t_crit}$",transform=trans,ha='left',va='center')
    axes[i].set_ylabel(r"$\mathbb{E}[|M|]$")
    axes[i].set_ylabel(f"\\texttt{{grid size}} = {int(g**2)}")
    axes[i].set_ylim([-0.1,1.1])
    axes[i].set_xscale('log')

axes[0].set_title(r"average absolute magnetization, $\mathbb{E}[|M|]$")
axes[-1].set_xlabel(r"temperature, $\tau$")
plt.savefig(os.path.join(FIG_DIR,'abs_avg_magnet.png'), bbox_inches='tight')
plt.show()
```

[ ]:
```
# Plot energy and magnetization
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(6, 10), sharex=True)
axes = np.expand_dims(axes, axis=0) if axes.ndim==1 else axes
plt.subplots_adjust(wspace=0.25, hspace=0.05)

# Define a sequence of colors for different temperatures
```

```
cmap   = plt.cm.plasma
norm   = matplotlib.colors.LogNorm(vmin=min(temps), vmax=max(temps))
colors = cmap(norm(temps))
for i, (g, t_runs) in enumerate(runs.items()):
    for t_idx, (t, EM) in enumerate(t_runs.items()):
        for c in chains:
            label = f"$\\tau={t:0.2f}$" if c==0 else None
            axes[i,0].plot(EM['E'][c,:], alpha=0.2, label=label,␣
 ↪color=colors[t_idx])
            axes[i,1].plot(EM['M'][c,:], alpha=0.2, label=label,␣
 ↪color=colors[t_idx])
        axes[i,0].set_ylabel(f"\\texttt{{grid size}} = {int(g**2)}")

axes[0,0].set_title("Energy, $E(s)$")
axes[0,1].set_title("Magnetization, $M(s)$")
axes[-1,0].set_xlabel("Iteration")
axes[-1,1].set_xlabel("Iteration")

# Add colorbar of temperatures
sm   = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
cbar = fig.colorbar(sm,ax=axes.ravel().tolist(),
                    orientation='vertical',
                    fraction=0.02,
                    pad=0.04)
cbar.set_label(r'temperature, $\tau$')
plt.savefig(os.path.join(FIG_DIR,'samples.png'), bbox_inches='tight')
plt.show()
```

0.0.4 Simulated Tempering

[ ]:
```
# Calculate normalizing constant via importance sampling
def calc_log_Z_k(E_samps, T_k):
    if len(E_samps)>0:
        return scipy.special.logsumexp(-np.array(E_samps)/T_k) - np.
 ↪log(len(E_samps))
    else:
        return -np.inf

# Simulated tempering step
def tempering_step(grid_t, temp_t, temp_idx, temps, energies_from_each_temp):

    # Random walk proposal
    if temp_idx == 0:
        temp_idx_candidate = temp_idx + np.random.choice([ 0,1])
    elif temp_idx == len(temps)-1:
        temp_idx_candidate = temp_idx + np.random.choice([-1,0])
    else:
        temp_idx_candidate = temp_idx + np.random.choice([-1,1])
    temp_candidate     = temps[temp_idx_candidate]

    # Acceptance ratio
    E_t_plus_1    = compute_energy(grid_t)
    exponent      = (1/temp_candidate) - (1/temp_t)
    log_Z_k_t     = calc_log_Z_k(energies_from_each_temp[temp_t], temp_t)
    log_Z_k_star = calc_log_Z_k(energies_from_each_temp[temp_candidate],␣
 ↪temp_candidate)
    log_r         = log_Z_k_t - log_Z_k_star - exponent*E_t_plus_1

    # Accept or reject
    if np.random.rand() < np.exp(min(log_r,0)):
        return temp_candidate, temp_idx_candidate, True
    else:
        return temp_t, temp_idx, False

# Simulated tempering step
def tempering_step_simple(grid_t, temp_t, temp_idx, temps,␣
 ↪energies_from_each_temp):

    # Random walk proposal
    temp_idx_candidate = np.random.choice([i for i in range(len(temps))])
    temp_candidate     = temps[temp_idx_candidate]
    return temp_candidate, temp_idx_candidate, True

# Simulated tempering sampling
def simulated_tempering(initial_grid, proposal_func, taus, n_steps,␣
 ↪**proposal_kwargs):

    grid              = initial_grid.copy()
    energies_by_temp = {t:[] for t in taus}
    energies          = np.zeros(n_steps)
    magnetizations    = np.zeros(n_steps)
    tau_samples       = np.zeros(n_steps)
    tau_idx           = np.where(taus == 0.1)[0][0]
    tau_t             = taus[tau_idx]
    tau_indices       = []
    n_grid_accepted   = 0
    n_temp_accepted   = 0

    for i in range(n_steps):

        # Perform Metropolis step on s_t
        grid, accepted   = metropolis_step(grid, proposal_func, tau_t,␣
 ↪**proposal_kwargs)
        n_grid_accepted += int(accepted)
        energies_by_temp[tau_t].append(compute_energy(grid))

        # Perform Metropolis step on temperature index
        tau_t, tau_idx, temp_accepted =␣
 ↪tempering_step(grid,tau_t,tau_idx,taus,energies_by_temp)
        n_temp_accepted += int(temp_accepted)

        # Save samples
        energies[i]       = compute_energy(grid)
        magnetizations[i] = np.mean(grid)
        tau_samples[i]    = tau_t
        tau_indices.append(tau_idx)

    # Calculate acceptance rates
    grid_accept_rate = n_grid_accepted/n_steps
    temp_accept_rate = n_temp_accepted/n_steps

    return grid, energies, magnetizations, tau_samples, grid_accept_rate,␣
 ↪temp_accept_rate

# ---------------------------- #
# -------- Parameters -------- #
# ---------------------------- #
n_tests = 100
all_E_ess   = {'MH': [], 'ST':[]}
all_M_ess   = {'MH': [], 'ST':[]}
all_E_samps = {'MH': [], 'ST':[]}
all_M_samps = {'MH': [], 'ST':[]}

for _ in range(n_tests):
    grid_size      = 16
    grid_init      = initialize_grid(grid_size)
    tau_compare    = 0.1
    p_flip_compare = 0.01

    # ---------------------------- #
    # --- Metropolis-Hastings --- #
    # ---------------------------- #
    n_steps    = 5000
    burn_in    = n_steps//2
    n_chains   = 1
    chains     = np.arange(n_chains)

    E_chains     = np.zeros((n_chains,n_steps))
    M_chains     = np.zeros((n_chains,n_steps))
    accept_rates = np.zeros(n_chains)
    for c in chains:
        grid_mh, E_mh, M_mh, accept_rate_mh = run_MH_simulation(
            grid_init.copy(), proposal_independent, tau_compare, n_steps,␣
↪flip_prob=p_flip_compare
        )
        E_chains[c,:]    = E_mh
        M_chains[c,:]    = M_mh
        accept_rates[c] = accept_rate_mh
    E_ess_mh = az.ess(E_chains[:,burn_in:])
    M_ess_mh = az.ess(M_chains[:,burn_in:])
    N_samples_after_burn_in = len(E_chains[:,burn_in:].flatten())
    E_mix_time_mh = N_samples_after_burn_in/E_ess_mh
    M_mix_time_mh = N_samples_after_burn_in/M_ess_mh

    # Print results
    print('-'*50)
    print('-'*12, 'Metropolis-Hastings', '-'*12)
    print('-'*50)
    print(f"grid_size={grid_size}, temp={tau_compare:0.2f}")
    print(f"n_steps={n_steps}, burn_in={burn_in}, n_chains={n_chains},")
    print(f" ==> Number of samples after burn in = {len(E_chains[:,burn_in:].
↪flatten())}")
    print(f"N_eff     : energy={E_ess_mh:0.2f}, magnetization={M_ess_mh:0.2f}")
    print(f"Mixing T : energy={E_mix_time_mh:0.2f},␣
↪magnetization={M_mix_time_mh:0.2f}")
    print(f"Acceptance rates : {accept_rates[0]:0.2f}")
    print('-'*50)

    # ---------------------------- #
    # --- Simulated Tempering --- #
    # ---------------------------- #
    temp_ladder = np.logspace(-1, 1, 3)
    n_iter      = N_samples_after_burn_in*len(temp_ladder) # 1000

    grid_st, E_st, M_st, tau_samples, grid_accept, tau_accept =␣
↪simulated_tempering(
            grid_init.copy(), proposal_independent, temp_ladder, n_iter,␣
↪flip_prob=p_flip_compare
        )

    # Isolate samples from the target
    E_samples = E_st[tau_samples==tau_compare]
    M_samples = M_st[tau_samples==tau_compare]

    # Calculate effective sample size and mixing time
    E_ess_st = az.ess(E_samples)
    M_ess_st = az.ess(M_samples)
    N_samples_at_target = len(E_samples)
    E_mix_time_st = N_samples_at_target/E_ess_st
    M_mix_time_st = N_samples_at_target/M_ess_st

    # Print results
    print('-'*50)
    print('-'*12, 'Simulated Tempering', '-'*12)
    print('-'*50)
    print(f"grid_size={grid_size}, temp={tau_compare:0.2f}")
    print(f"Total number of samples    = {n_iter}")
    print(f"N collected at target dist = {len(E_samples)}")
    print(f"N_eff    : energy={E_ess_st:0.2f}, magnetization={M_ess_st:0.2f}")
    print(f"Mixing T : energy={E_mix_time_st:0.2f},␣
↪magnetization={M_mix_time_st:0.2f}")
    print(f"Acceptance rates : grid={grid_accept:0.2f}, temp={tau_accept:0.2f}")
    print('-'*50)

    # --------------------- #
    # --- Save results --- #
    # --------------------- #
    all_E_ess['MH'].append(E_ess_mh)
    all_M_ess['MH'].append(M_ess_mh)
    all_E_ess['ST'].append(E_ess_st)
    all_M_ess['ST'].append(M_ess_st)
    all_E_samps['MH'].append(E_chains[:,burn_in:].flatten())
    all_M_samps['MH'].append(M_chains[:,burn_in:].flatten())
    all_E_samps['ST'].append(E_samples)
    all_M_samps['ST'].append(M_samples)
```

[ ]:

```
# ---------------------------- #
# ------- Plot results ------- #
# ---------------------------- #
# Plot energy and magnetization
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(6, 6), sharex=True)
axes = np.expand_dims(axes, axis=0) if axes.ndim==1 else axes
plt.subplots_adjust(wspace=0.25, hspace=0.05)

# Define a sequence of colors for different temperatures
cmap   = plt.cm.BuPu
colors = cmap(np.linspace(0, 1, n_tests))
# Plot samples for each approach
for i, (E_mh, M_mh, E_st, M_st) in enumerate(zip(
        all_E_samps['MH'],
        all_M_samps['MH'],
        all_E_samps['ST'],
        all_M_samps['ST'],
    )):
    axes[0,0].plot(E_mh, alpha=0.1, color=colors[i])
    axes[1,0].plot(M_mh, alpha=0.1, color=colors[i])
    axes[0,1].plot(E_st, alpha=0.1, color=colors[i])
    axes[1,1].plot(M_st, alpha=0.1, color=colors[i])

axes[0,0].set_title("Metropolis-Hastings")
axes[0,1].set_title("Simulated Tempering")
axes[0,0].set_ylabel("Energy, $E(s)$")
axes[1,0].set_ylabel("Magnetization, $M(s)$")
axes[1,0].set_ylim([-1.1,1.1])
axes[1,1].set_ylim([-1.1,1.1])
axes[-1,0].set_xlabel("Iteration")
axes[-1,1].set_xlabel("Iteration")
plt.savefig(os.path.join(FIG_DIR,'tempering.vs.metropolis.png'),␣
 ↪bbox_inches='tight')
plt.show()

# Plot effective sample size as overlaid histograms
_, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

# Plot effective sample size for each approach
ax1.hist(all_E_ess['MH'], density=True, alpha=0.5, bins=20,␣
 ↪label='Metropolis-Hastings', color='blue')
ax1.hist(all_E_ess['ST'], density=True, alpha=0.5, bins=20, label='Simulated␣
 ↪Tempering', color='yellow')
ax2.hist(all_M_ess['MH'], density=True, alpha=0.5, bins=20,␣
 ↪label='Metropolis-Hastings', color='blue')
ax2.hist(all_M_ess['ST'], density=True, alpha=0.5, bins=20, label='Simulated␣
 ↪Tempering', color='yellow')
ax1.set_xlabel(r'Energy $N_{\mathrm{eff}}$')
ax2.set_xlabel(r'Magnetization $N_{\mathrm{eff}}$')
ax1.set_ylabel('density')
ax2.set_ylabel('density')
ax1.legend(loc='upper right')
ax2.legend(loc='upper right')
plt.suptitle('Effective Sample Size')
plt.savefig(os.path.join(FIG_DIR,'effective.sample.size.png'),␣
 ↪bbox_inches='tight')
plt.show()

# Calculate percentiles of effective sample size
E_ess_mh = np.percentile(all_E_ess['MH'], [2.5, 25, 50, 75, 97.5])
E_ess_st = np.percentile(all_E_ess['ST'], [2.5, 25, 50, 75, 97.5])
M_ess_mh = np.percentile(all_M_ess['MH'], [2.5, 25, 50, 75, 97.5])
M_ess_st = np.percentile(all_M_ess['ST'], [2.5, 25, 50, 75, 97.5])

# Print LaTeX booktabs table with percentiles
```

```
print(r'Metropolis-Hastings &', rf'{E_ess_mh[0]:0.2f} & {E_ess_mh[1]:0.2f} &␣
 ↪{E_ess_mh[2]:0.2f} & {E_ess_mh[3]:0.2f} & {E_ess_mh[4]:0.2f} &',
        rf'{M_ess_mh[0]:0.2f} & {M_ess_mh[1]:0.2f} & {M_ess_mh[2]:0.2f} &␣
 ↪{M_ess_mh[3]:0.2f} & {M_ess_mh[4]:0.2f} \\')
print(r'Simulated Tempering &', rf'{E_ess_st[0]:0.2f} & {E_ess_st[1]:0.2f} &␣
 ↪{E_ess_st[2]:0.2f} & {E_ess_st[3]:0.2f} & {E_ess_st[4]:0.2f} &',
        rf'{M_ess_st[0]:0.2f} & {M_ess_st[1]:0.2f} & {M_ess_st[2]:0.2f} &␣
 ↪{M_ess_st[3]:0.2f} & {M_ess_st[4]:0.2f} \\')
print('-'*50)
print('MH energy ESS:', E_ess_mh)
print('ST energy ESS:', E_ess_st)
print('MH magnetization ESS:', M_ess_mh)
print('ST magnetization ESS:', M_ess_st)
```