Numerical Calculation for Bose-Hubbard Model

Mingyu Xia (夏明宇) and Yue Xiao (肖月) and Lintao Yu (余林涛)
The Lecturer

Group 8 | 2025-12-23

WESTLAKE UNIVERSITY

MODEL INTRODUCTION

Hamiltonian

The Bose-Hubbard model (BHM) gives a description of the physics of interacting spinless Bosons on a lattice[1].

$$\mathcal{H} = -t \sum_{\langle i,j \rangle} \hat{b}_i^\dagger \hat{b}_j + \frac{1}{2} U \sum_i \hat{n}_i (\hat{n}_i - 1) - \mu \sum_i \hat{n}_i. \tag{1.1}$$

Hamiltonian

The Bose-Hubbard model (BHM) gives a description of the physics of interacting spinless Bosons on a lattice[1].

$$\mathcal{H} = -t \sum_{\langle i,j \rangle} \hat{b}_i^\dagger \hat{b}_j + \frac{1}{2} U \sum_i \hat{n}_i(\hat{n}_i - 1) - \mu \sum_i \hat{n}_i. \tag{1.1}$$

Containing

Hopping Term — Superfluid

Bosons move between neighboring sites with amplitude t. The operator $\hat{b}_i^\dagger$ creates a particle at site i, while $\hat{b}_j$ removes one at site j.

Hamiltonian

The Bose-Hubbard model (BHM) gives a description of the physics of interacting spinless Bosons on a lattice[1].

$$\mathcal{H} = -t \sum_{\langle i,j \rangle} \hat{b}_i^\dagger \hat{b}_j + \frac{1}{2} U \sum_i \hat{n}_i(\hat{n}_i - 1) - \mu \sum_i \hat{n}_i. \tag{1.1}$$

Containing

Hopping Term — Superfluid

Bosons move between neighboring sites with amplitude t. The operator $\hat{b}_i^\dagger$ creates a particle at site i, while $\hat{b}_j$ removes one at site j.

On-Site Repulsive Term — Mott-insulation

Bosons on the same site repel with strength U. The term $\hat{n}_i(\hat{n}_i - 1)$ penalizes multiple occupancy, where $\hat{n}_i$ counts particles at site i.

Hamiltonian

The Bose-Hubbard model (BHM) gives a description of the physics of interacting spinless Bosons on a lattice[1].

$$\mathcal{H} = -t \sum_{\langle i,j \rangle} \hat{b}_i^\dagger \hat{b}_j + \frac{1}{2} U \sum_i \hat{n}_i(\hat{n}_i - 1) - \mu \sum_i \hat{n}_i. \tag{1.1}$$

Containing

Hopping Term — Superfluid

Bosons move between neighboring sites with amplitude t. The operator $\hat{b}_i^\dagger$ creates a particle at site i, while $\hat{b}_j$ removes one at site j.

On-Site Repulsive Term — Mott-insulation

Bosons on the same site repel with strength U. The term $\hat{n}_i(\hat{n}_i - 1)$ penalizes multiple occupancy, where $\hat{n}_i$ counts particles at site i.

Chemical Potential Term

Controls the average particle number via external field μ. For $\mu = 5$, the system will keep a fixed density of the mean particle number.

WESTLAKE UNIVERSITY

SSE EXPANSION FOR BHM

Kernel Formula

Applying the Taylor expansion to the partition function

$$Z = \sum_{m=0}^{\infty} \frac{\beta^m}{m!} \sum_{\{i_1,\ldots,i_m\}} \sum_{\{b_1,\ldots,b_m\}} \prod_{k=1}^{m} \langle i_k | -H_{b_k} | i_{k+1} \rangle . \tag{2.1}$$

Kernel Formula

Applying the Taylor expansion to the partition function

$$Z = \sum_{m=0}^{\infty} \frac{\beta^m}{m!} \sum_{\{i_1,\dots,i_m\}} \sum_{\{b_1,\dots,b_m\}} \prod_{k=1}^{m} \langle i_k | -H_{b_k} | i_{k+1} \rangle . \tag{2.1}$$

Expasion Usage Example Taking the order $m = 3$ in 1D-4site model with the parameters

Parameters	t	U	μ	β	N	$\{b_1, b_2, b_3\}$
Values	1	2	5	4	4	$\{[(t, (1, 2)), (U, 3), (\mu, 4)]\}$
Notations	/	/	/	/	Total number of particles in the 4-site	Jumping, repulsion, chemical potential operators

Kernel Formula

Applying the Taylor expansion to the partition function

$$Z = \sum_{m=0}^{\infty} \frac{\beta^m}{m!} \sum_{\{i_1,\ldots,i_m\}} \sum_{\{b_1,\ldots,b_m\}} \prod_{k=1}^{m} \langle i_k | -H_{b_k} | i_{k+1} \rangle . \tag{2.1}$$

Expasion Usage Example Taking the order $m = 3$ in 1D-4site model with the parameters

Parameters	t	U	μ	β	N	$\{b_1, b_2, b_3\}$
Values	1	2	5	4	4	$\{[(t,(1,2)),(U,3),(\mu,4)]\}$
Notations	/	/	/	/	Total number of particles in the 4-site	Jumping, repulsion, chemical potential operators

The state sequences become

1. $|i_1\rangle = |2, 1, 1, 0\rangle$.
2. $|i_2\rangle = |1, 2, 1, 0\rangle$.
3. $|i_3\rangle = |1, 2, 1, 0\rangle$.
4. $|i_4\rangle = |2, 1, 1, 0\rangle = |i_1\rangle$.

WESTLAKE UNIVERSITY

NUMERICAL CALCULATION ANALYSIS: 3 SCHEMES

Scheme A: Heat Bath Update

Basic Principle

Weight-Proportional Sampling: Transition probabilities equal normalized weights $\pi_j = w_j / \sum w_k$.

No rejection step, i.e., always accept according to weights.

Scheme A: Heat Bath Update

Basic Principle

Weight-Proportional Sampling: Transition probabilities equal normalized weights $\pi_j = w_j/\sum w_k$.

No rejection step, i.e., always accept according to weights.

Transition Matrix Structure

For 4 scattering processes (bounce, straight, jump, turn)

$$T_A = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \\ \pi_2 & \pi_1 & \pi_4 & \pi_3 \\ \pi_3 & \pi_4 & \pi_1 & \pi_2 \\ \pi_4 & \pi_3 & \pi_2 & \pi_1 \end{bmatrix}, \quad T_{A_{ii}} = \pi_i.$$

The diagonal term $T_{ii} = \pi_i$ allows state stagnation, then causes high autocorrelation.

Scheme A: Heat Bath Update

Basic Principle

Weight-Proportional Sampling: Transition probabilities equal normalized weights $\pi_j = w_j / \sum w_k$.

No rejection step, i.e., always accept according to weights.

Transition Matrix Structure

For 4 scattering processes (bounce, straight, jump, turn)

$$T_A = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \\ \pi_2 & \pi_1 & \pi_4 & \pi_3 \\ \pi_3 & \pi_4 & \pi_1 & \pi_2 \\ \pi_4 & \pi_3 & \pi_2 & \pi_1 \end{bmatrix}, \quad T_{A_{ii}} = \pi_i.$$

The diagonal term $T_{ii} = \pi_i$ allows state stagnation, then causes high autocorrelation.

Simple but inefficient due to frequent bounce moves.

Scheme B: Minimal Bounce Solution

Basic Principle

Linear Programming Optimization: Minimize total bounce probability $\mathrm{Tr}(T_B) = \sum_i T_{B_{ii}}$ subject to

- Normalization: $\sum_j T_{B_{ij}} = 1$.
- Detailed balance: $w_i T_{B_{ij}} = w_j T_{B_{ji}}$.

Scheme B: Minimal Bounce Solution

Basic Principle

Linear Programming Optimization: Minimize total bounce probability $\mathrm{Tr}(T_B) = \sum_i T_{B_{ii}}$ subject to

- Normalization: $\sum_j T_{B_{ij}} = 1$.
- Detailed balance: $w_i T_{B_{ij}} = w_j T_{B_{ji}}$.

Example Matrix

For weights $[0.2, 0.4, 0.3, 0.1]$

$$T_B = \begin{bmatrix} 0.1 & 0.5 & 0.3 & 0.1 \\ 0.5 & 0.1 & 0.1 & 0.3 \\ 0.3 & 0.1 & 0.1 & 0.5 \\ 0.1 & 0.3 & 0.5 & 0.1 \end{bmatrix}.$$

The diagonal term reduced: $T_{B_{11}} = 0.1$, in comparison to $\pi_1 = 0.2$ in Scheme A.

Scheme B: Minimal Bounce Solution

Basic Principle

Linear Programming Optimization: Minimize total bounce probability $\mathrm{Tr}(T_B) = \sum_i T_{B_{ii}}$ subject to

- Normalization: $\sum_j T_{B_{ij}} = 1$.
- Detailed balance: $w_i T_{B_{ij}} = w_j T_{B_{ji}}$.

Example Matrix

For weights $[0.2, 0.4, 0.3, 0.1]$

$$T_B = \begin{bmatrix} 0.1 & 0.5 & 0.3 & 0.1 \\ 0.5 & 0.1 & 0.1 & 0.3 \\ 0.3 & 0.1 & 0.1 & 0.5 \\ 0.1 & 0.3 & 0.5 & 0.1 \end{bmatrix}.$$

The diagonal term reduced: $T_{B_{11}} = 0.1$, in comparison to $\pi_1 = 0.2$ in Scheme A.

Reduces wasted bounce moves via constrained optimization.

Scheme C: Locally Optimal Algorithm

Basic Principle

Peskun's theorem + Metropolization: Diagonal elements set to zero for all but the largest weight state

$$T_{C_{ii}} = 0 \quad (i \neq \max).$$

Prohibits lingering in low-weight states; Minimizes autocorrelation.

Scheme C: Locally Optimal Algorithm

Basic Principle

Peskun's theorem + Metropolization: Diagonal elements set to zero for all but the largest weight state

$$T_{C_{ii}} = 0 \quad (i \neq \max).$$

Prohibits lingering in low-weight states; Minimizes autocorrelation.

Transition Matrix ($\pi_1 \leq \pi_2 \leq \pi_3 \leq \pi_4$)

$$T_C = \begin{bmatrix} 0 & \frac{\pi_2}{1-\pi_1} & \frac{\pi_3}{1-\pi_1} & \frac{\pi_4}{1-\pi_1} \\ \frac{\pi_1}{1-\pi_2} & 0 & \frac{\pi_3}{1-\pi_2} & \frac{\pi_4}{1-\pi_2} \\ \frac{\pi_1}{1-\pi_3} & \frac{\pi_2}{1-\pi_3} & 0 & \frac{\pi_4}{1-\pi_3} \\ \frac{\pi_1}{1-\pi_4'} & \frac{\pi_2}{1-\pi_4'} & \frac{\pi_3}{1-\pi_4'} & \pi_4' \end{bmatrix}.$$

Only $T_{C_{44}} \neq 0$; All other diagonals are zero.

Scheme C: Locally Optimal Algorithm

Basic Principle

Peskun's theorem + Metropolization: Diagonal elements set to zero for all but the largest weight state

$$T_{C_{ii}} = 0 \quad (i \neq \max).$$

Prohibits lingering in low-weight states; Minimizes autocorrelation.

Transition Matrix ($\pi_1 \leq \pi_2 \leq \pi_3 \leq \pi_4$)

$$T_C = \begin{bmatrix} 0 & \frac{\pi_2}{1-\pi_1} & \frac{\pi_3}{1-\pi_1} & \frac{\pi_4}{1-\pi_1} \\ \frac{\pi_1}{1-\pi_2} & 0 & \frac{\pi_3}{1-\pi_2} & \frac{\pi_4}{1-\pi_2} \\ \frac{\pi_1}{1-\pi_3} & \frac{\pi_2}{1-\pi_3} & 0 & \frac{\pi_4}{1-\pi_3} \\ \frac{\pi_1}{1-\pi_4'} & \frac{\pi_2}{1-\pi_4'} & \frac{\pi_3}{1-\pi_4'} & \pi_4' \end{bmatrix}.$$

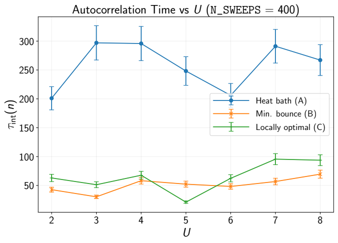
Only $T_{C_{44}} \neq 0$; All other diagonals are zero.

Optimal for balanced weights; Derived from Peskun's ordering theorem.

WESTLAKE UNIVERSITY

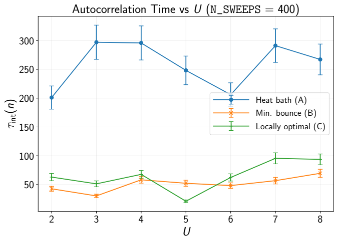
SIMULATION RESULTS

Simulations under different sweep numbers

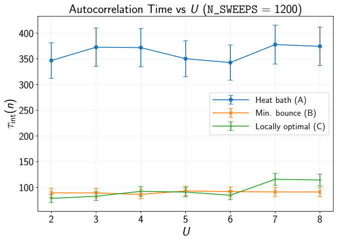


(a) 400 Sweeps

Simulations under different sweep numbers

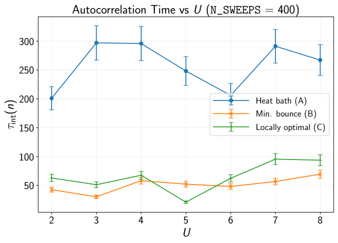


(a) 400 Sweeps

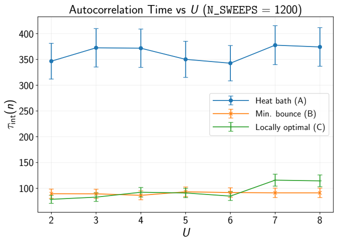


(b) 1200 Sweeps

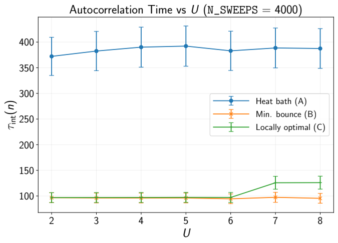
Simulations under different sweep numbers



(a) 400 Sweeps

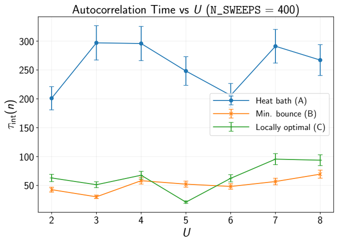


(b) 1200 Sweeps

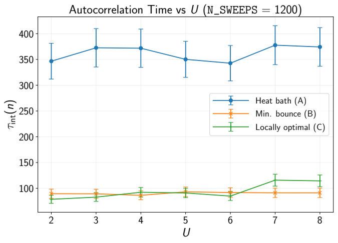


(c) 4000 Sweeps

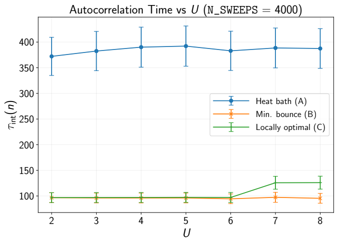
Simulations under different sweep numbers



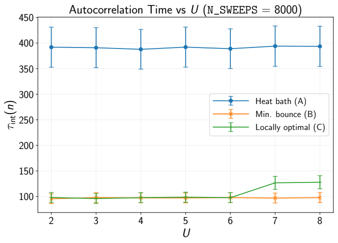
(a) 400 Sweeps



(b) 1200 Sweeps

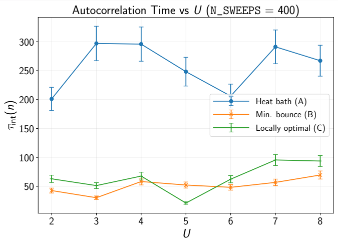


(c) 4000 Sweeps

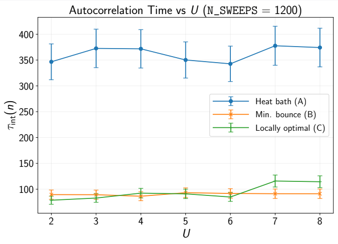


(d) 8000 Sweeps

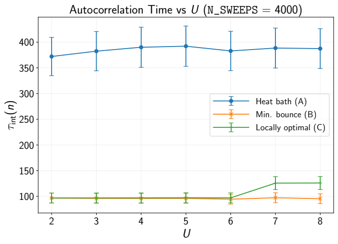
Simulations under different sweep numbers



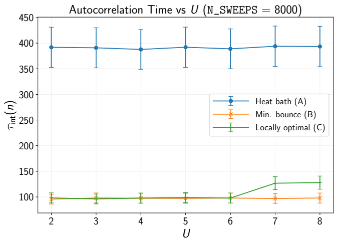
(a) 400 Sweeps



(b) 1200 Sweeps



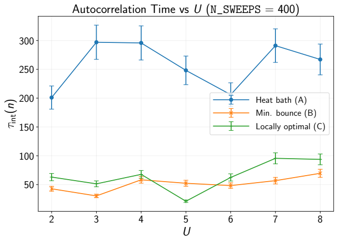
(c) 4000 Sweeps



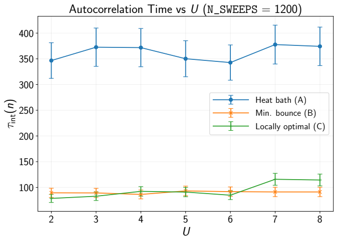
(d) 8000 Sweeps

As U increases, the diagonal weights in the SSE configuration space become dominant. Scheme C shows a slight increase in τ in comparison to B.

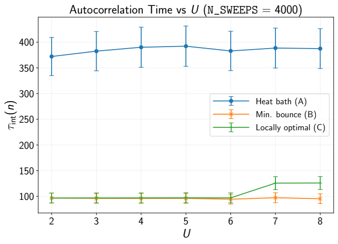
Simulations under different sweep numbers



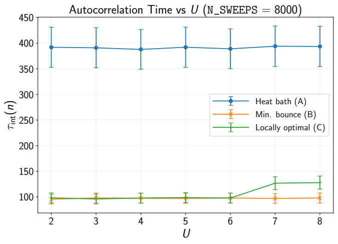
(a) 400 Sweeps



(b) 1200 Sweeps



(c) 4000 Sweeps



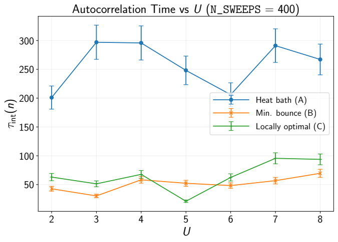
(d) 8000 Sweeps

As U increases, the diagonal weights in the SSE configuration space become dominant. Scheme C shows a slight increase in τ in comparison to B.

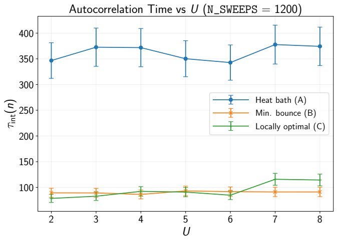
Table. Comparasion under different U

Phases	Superfluid	Transition	Near-Mote
U	$U < 3$	$3 \leq U \leq 6$	$U \to 9$
(A)	$20 \sim 30$	$30 \sim 45$	$45 \sim 50$
(B)	$8 \sim 10$	$10 \sim 20$	$15 \sim 20$
(C)	$7 \sim 9$	$9 \sim 18$	$18 \sim 25$

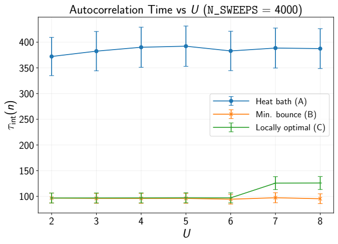
Simulations under different sweep numbers



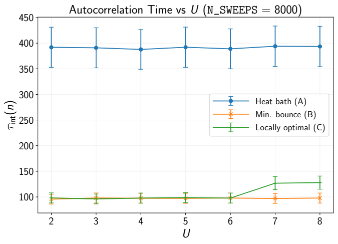
(a) 400 Sweeps



(b) 1200 Sweeps



(c) 4000 Sweeps



(d) 8000 Sweeps

As U increases, the diagonal weights in the SSE configuration space become dominant. Scheme C shows a slight increase in τ in comparison to B.

Table. Comparasion under different U

Phases	Superfluid	Transition	Near-Mote
U	$U < 3$	$3 \leq U \leq 6$	$U \to 9$
(A)	$20 \sim 30$	$30 \sim 45$	$45 \sim 50$
(B)	$8 \sim 10$	$10 \sim 20$	$15 \sim 20$
(C)	$7 \sim 9$	$9 \sim 18$	$18 \sim 25$

A fixed random seed is loaded to ensure the fixed result for every time running.

WESTLAKE UNIVERSITY

CONCLUSION

Conclusion

Heat Bath Update (A) is Consistently Inefficient

Due to the non-zero diagonal elements of the transition matrix (state stagnation), the samples are highly correlated, and $\tau_{\mathrm{int}}(n)$ is 2-3 times that of B/C.

Conclusion

Heat Bath Update (A) is Consistently Inefficient

Due to the non-zero diagonal elements of the transition matrix (state stagnation), the samples are highly correlated, and $\tau_{\text{int}}(n)$ is 2-3 times that of B/C.

Algorithm Selection is Model-Dependent

Low U (weight balance) selects local optima (C), high U (diagonal weight dominance) selects minimum bounce (B).

Conclusion

Heat Bath Update (A) is Consistently Inefficient

Due to the non-zero diagonal elements of the transition matrix (state stagnation), the samples are highly correlated, and $\tau_{\text{int}}(n)$ is 2-3 times that of B/C.

Algorithm Selection is Model-Dependent

Low U (weight balance) selects local optima (C), high U (diagonal weight dominance) selects minimum bounce (B).

Core Design Principle

For efficient algorithms, the condition “Diagonal elements of non-maximum weight states returning to zero” is necessary (both B and C satisfy this), and the remaining degrees of freedom need to be adapted to the weight distribution.

SIMULATION METHOD

Configuration & Process

Key Parameters

- **Lattice**: 1D chain, $L = 64$, $t = 1$, $\mu = 5$.
- **Thermodynamics**: $\beta = L = 64$ (low-T), $U \in [0, 6]$.
- **Algorithm**: A: 16 loops $\times 4$; B/C: 4 loops.
- **Statistics**: 4000 independent chains, each with 1 million steps.
- **Cutoff**: For efficiency, particle number reduced at $U = 3, 8$.

Configuration & Process

Key Parameters

- **Lattice**: 1D chain, $L = 64$, $t = 1$, $\mu = 5$.
- **Thermodynamics**: $\beta = L = 64$ (low-T), $U \in [0, 6]$.
- **Algorithm**: A: 16 loops $\times 4$; B/C: 4 loops.
- **Statistics**: 4000 independent chains, each with 1 million steps.
- **Cutoff**: For efficiency, particle number reduced at $U = 3, 8$.

Method: SSE + Directed Loop

Stochastic Series Expansion maps quantum problem to $(d+1)$D classical graph

$$Z = \mathrm{Tr}\, e^{-\beta H} = \sum_{\alpha} \sum_{n=0}^{\infty} \frac{\beta^n}{n!} \langle \alpha | (-H)^n | \alpha \rangle .$$

Worm algorithm updates via vertex scattering using A/B/C transition matrices.

Configuration & Process

Key Parameters

- **Lattice**: 1D chain, $L = 64$, $t = 1$, $\mu = 5$.
- **Thermodynamics**: $\beta = L = 64$ (low-T), $U \in [0, 6]$.
- **Algorithm**: A: 16 loops $\times 4$; B/C: 4 loops.
- **Statistics**: 4000 independent chains, each with 1 million steps.
- **Cutoff**: For efficiency, particle number reduced at $U = 3, 8$.

Method: SSE + Directed Loop

Stochastic Series Expansion maps quantum problem to $(d+1)$D classical graph

$$Z = \mathrm{Tr}\, e^{-\beta H} = \sum_{\alpha} \sum_{n=0}^{\infty} \frac{\beta^n}{n!} \langle \alpha | (-H)^n | \alpha \rangle .$$

Worm algorithm updates via vertex scattering using A/B/C transition matrices.

Procedure

1. **Burn-in**: Discard first 10% of samples.
2. **Worm updates**: Collect density series $\{n^{(t)}\}$.
3. **Autocorrelation**
$$\tau_{\mathrm{int}}(n) = \frac{1}{2} + \sum_{t=1}^{\infty} \frac{\langle n^{(i+t)} n^{(i)} \rangle - \langle n^{(i)} \rangle^2}{\langle n^{(i)2} \rangle - \langle n^{(i)} \rangle^2} .$$
4. **Average**: Mean over 4000 chains, plot U vs $\tau_{\mathrm{int}}(n)$.

WESTLAKE UNIVERSITY

PYTHON IMPLEMENTATION

Class Initialization and State Representation

The class stores the lattice size L, inverse temperature β, and model parameters. The state is represented by an array of occupation numbers **n**.

```
def __init__(self, L, beta, U, mu, t = 1, method = 'A'):
    self.L = L
    self.beta = beta
    self.U = U
    self.mu = mu
    self.t = t
    self.method = method  # 'A', 'B', 'C'
```

Initialize state: occupation numbers on each site.

```
    self.n = np.zeros(L, dtype = int)
```

Starting density roughly μ/U to reach equilibrium faster.

```
    initial_dens = max(1, int(mu/U + .5)) if U > 0 else 1
    self.n[:] = initial_dens
```

Energy shift: Ensures diagonal weights in SSE remain positive. Calculated based on maximum expected local density.

```
    self.E_shift = .5 * U * 10 * 9 + 20
```

Calculating Vertex Weights

In SSE, the probability of choosing an operator depends on its "weight". Diagonal weights are related to the local energy, while off-diagonal weights are related to the hopping amplitude t.

```python
def get_vertex_weight(self, n1, n2, op_type):
  E1 = .5 * self.U * n1 * (n1 - 1) - self.mu * n1
  E2 = .5 * self.U * n2 * (n2 - 1) - self.mu * n2
  H_diag_val = .5 * (E1 + E2)
```

Branches for the Diagonal operator (1) and Off-diagonal: hopping (2).

```python
  if op_type == 1:
    return max(0, self.E_shift - H_diag_val)
  elif op_type == 2:
    return self.t
  return 0
```

Transition Matrix Schemes I

The core of the "Optimal Monte Carlo" paper is the design of the transition matrix T_{ij}. According to Peskun's theorem, to minimize the autocorrelation time, one should minimize the diagonal elements T_{ii} (the "bounce" or "stagnation" probability).

```python
def solve_greedy_min_bounce(self, weights):
w0, w1, w2 = weights
sw = w0 + w1 + w2
pi = np.array(weights) / sw
p_out = np.zeros(3)
```

- Scheme A: Heat Bath The core of the "Optimal Monte Carlo" paper is the design of the transition matrix T_{ij}. According to Peskun's theorem, to minimize the autocorrelation time, one should minimize the diagonal elements T_{ii} (the "bounce" or "stagnation" probability).
- Scheme B and C: Optimization via Peskun's Theorem Scheme B (Minimal Bounce) and Scheme C (Locally Optimal) aim to set $T_{ii} = 0$ whenever possible. The code implements this using a "greedy" approach or Metropolized Gibbs sampling.

Transition Matrix Schemes II

Metropolized Gibbs strategy: In [2], Using

$$T_{ij}^{MG} = \begin{bmatrix} 0 & \frac{\pi_2}{1-\pi_1} & \frac{\pi_3}{1-\pi_1} & \cdots & \frac{\pi_n}{1-\pi_1} \\ \frac{\pi_1}{1-\pi_1} & 1-\ldots & \frac{\pi_3}{1-\pi_2} & \cdots & \frac{\pi_n}{1-\pi_2} \\ \frac{\pi_1}{1-\pi_1} & \frac{\pi_2}{1-\pi_2} & 1-\ldots & \cdots & \frac{\pi_n}{1-\pi_3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\pi_1}{1-\pi_1} & \frac{\pi_2}{1-\pi_2} & \frac{\pi_3}{1-\pi_3} & \cdots & 1-\ldots \end{bmatrix} = \min\left(\frac{\pi_j}{1-\pi_i}, \frac{\pi_j}{1-\pi_j}\right).$$

```python
for i in [1, 2]:
    term1 = pi[i] / (1 - pi[0]) if (1 - pi[0]) > 1e-9 else 0
    term2 = pi[i] / (1 - pi[i]) if (1 - pi[i]) > 1e-9 else 1
    p_out[i] = min(term1, term2)
current_sum = p_out[1] + p_out[2]
if current_sum > 1:
    p_out[1] /= current_sum
    p_out[2] /= current_sum
    p_out[0] = 0
else:
    p_out[0] = 1 - current_sum
return p_out
```

The Simulation Loop

The run method performs the actual Markov Chain sweeps. Each site is updated based on the calculated transition probabilities.

```python
def run(self, n_sweeps):
  densities = []
  for _ in range(n_sweeps):
    for i in range(self.L):
      n_curr = self.n[i]
      w0 = self.get_vertex_weight(n_curr, n_curr, 1)
      w_plus = self.get_vertex_weight(n_curr, n_curr + 1, 2)
      w_minus = self.get_vertex_weight(n_curr, n_curr - 1, 2) if n_curr > 0
```

```python
      probs = self.solve_scattering([w0, w_plus, w_minus], self.method)
```

Sample the next state.

```python
      r = np.random.rand()
      if r < probs[0]:
        pass # Stay
      elif r < probs[0] + probs[1]:
        self.n[i] += 1
      else:
        self.n[i] -= 1
```

Record average density as the observable.

```python
    densities.append(np.mean(self.n))
  return densities
```

WESTLAKE UNIVERSITY

Thanks for Listening! Any Questions?