Ch5

May 19, 2025

1 Chapter 5

1.1 Question 3

1.1.1 Author: Ou, Dongwen

[ ]:

```

```

[11]:

```
import numpy as np
import itertools

# ---------------------
#
# ---------------------
J = -0.2          #
h = 0.3           #         +1
rows, cols = 4, 5
N = rows * cols   #      4*5=20

# ---------------------
#         1D   2D
# ---------------------
def get_neighbors(index, rows, cols):
    i, j = divmod(index, cols)  #  1D   2D  index 0~19 1D index
    neighbors = []
    if i > 0: neighbors.append((i - 1) * cols + j)      #
    if i < rows - 1: neighbors.append((i + 1) * cols + j)   #
    if j > 0: neighbors.append(i * cols + (j - 1))      #
    if j < cols - 1: neighbors.append(i * cols + (j + 1))   #
    return neighbors

#
neighbors_list = [get_neighbors(idx, rows, cols) for idx in range(N)]

# ---------------------
# calculate U[x] given a kind of possible x
# ---------------------
def compute_energy(x): # x is in 1D form
```

```python
    interaction_energy = 0.0  # interaction term
    field_energy = 0.0        # single term
    for idx in range(N):
        xi = x[idx]
        for neighbor in neighbors_list[idx]:
            interaction_energy += xi * x[neighbor]  # interaction
        field_energy += xi
    interaction_energy /= 2  #         2
    return J * interaction_energy + h * field_energy

# ---------------------
# enumerate all 2^N possible X
# ---------------------
configs = list(itertools.product([-1, 1], repeat=N))  #       N         [-1, 1]
# itertools.product(...)
# use list(...)


# ---------------------
# Calculate weight sum Z and E[U]
# ---------------------
Z = 0.0   #
EU = 0.0  #

for config in configs:
    Ux = compute_energy(config)       #
    weight = np.exp(-Ux)              #     e^{-U(x)}
    Z += weight                       #
    EU += Ux * weight                 #

expected_U = EU / Z                   #     E[U]
print("True Expected Internal Energy:", expected_U)
```

```
True Expected Internal Energy: -5.647457464068214
```

[ ]:

[25]:

```python
import numpy as np
from tqdm import tqdm
import pandas as pd

# ---------------------
# Gibbs sampling with 1D representation
# ---------------------
def get_neighbors_1d(index, rows, cols):
    """Given a 1D index, return the indices of its neighbors in 1D."""
    i, j = divmod(index, cols)
    neighbors = []
```

```python
    if i > 0: neighbors.append((i - 1) * cols + j)
    if i < rows - 1: neighbors.append((i + 1) * cols + j)
    if j > 0: neighbors.append(i * cols + (j - 1))
    if j < cols - 1: neighbors.append(i * cols + (j + 1))
    return neighbors

def compute_energy_1d(x, neighbors_list, J, h):
    """Compute total energy given a 1D spin configuration."""
    interaction_energy = 0.0
    field_energy = 0.0
    for idx in range(len(x)):
        xi = x[idx]
        for neighbor in neighbors_list[idx]:
            interaction_energy += xi * x[neighbor]
        field_energy += xi
    interaction_energy /= 2  # avoid double-counting
    return J * interaction_energy + h * field_energy

def local_energy_1d(idx, spin_val, x, neighbors_list, J, h):
    """
    """ #    e    product
    neighbor_sum = sum(x[neighbor] for neighbor in neighbors_list[idx])  #
    return J * spin_val * neighbor_sum + h * spin_val                    #

def gibbs_sampling_1d(rows=4, cols=5, iterations=12000, burn_in=2000, J=-0.2, h=0.3):
    N = rows * cols
    x = np.random.choice([-1, 1], size=N)  # initial 1D configuration
    neighbors_list = [get_neighbors_1d(idx, rows, cols) for idx in range(N)]
    energies = []

    for it in tqdm(range(iterations)):
        for idx in range(N):
            # compute unnormalized probabilities for spin = +1 and -1
            e_pos = local_energy_1d(idx, 1, x, neighbors_list, J, h)
            e_neg = local_energy_1d(idx, -1, x, neighbors_list, J, h)
            p_pos = np.exp(-e_pos)
            p_neg = np.exp(-e_neg)
            prob = p_pos / (p_pos + p_neg)
            x[idx] = 1 if np.random.rand() < prob else -1
        if it >= burn_in:
            energies.append(compute_energy_1d(x, neighbors_list, J, h))

    return np.mean(energies), np.std(energies)

# Run for 4x5 and 20x20 grid
mean_4x5, std_4x5 = gibbs_sampling_1d(4, 5)
```

```
mean_20x20, std_20x20 = gibbs_sampling_1d(20, 20)

results_df = pd.DataFrame({
    "Grid Size": ["4x5", "20x20"],
    "Mean Energy": [mean_4x5, mean_20x20],
    "Std Dev": [std_4x5, std_20x20]
})

print(results_df)
```

```
100%|                  | 12000/12000 [00:00<00:00, 13912.90it/s]
100%|                   | 12000/12000 [00:17<00:00, 686.98it/s]

  Grid Size  Mean Energy    Std Dev
0       4x5     -5.70664   2.796847
1     20x20   -143.58190  14.624591
```

1.2 Question 4.

1.2.1 For t=0

$$
\begin{aligned}
& P(z_0|z_1,\theta) \\
\propto\ & P(z_0|\theta)P(z_1|z_0,\theta) \\
\propto\ & \exp\{-\frac{(z_0-\mu_0)^2}{2\eta_0^2}\}\exp\{-\frac{(z_1-b_0-b_1z_0)^2}{2\delta^2}\} \\
=\ & \exp\{-\frac{(z_0-\mu_{z_0})^2}{2\sigma_{z_0}^2}\}
\end{aligned}
$$

where

$$
\mu_{z_0}=\frac{\mu_0\delta^2+b_1(z_1-b_0)}{\delta^2+b_1^2\eta_0^2}\quad \sigma_{z_0}^2=\frac{\eta_0^2\delta^2}{\delta^2+b_1^2\eta_0^2}
$$

1.2.2 For $t=1,\ldots,T-1$

$$
\begin{aligned}
& P(z_t|z_{t-1},z_{t+1},\theta) \\
\propto\ & P(z_t|z_{t-1},\theta)P(z_{t+1}|z_t,\theta) \\
\propto\ & \exp\{-\frac{(z_t-b_1z_{t-1}-b_0)^2}{2\delta^2}\}\exp\{-\frac{(z_{t+1}-b_1z_t-b_0)^2}{2\delta^2}\} \\
\propto\ & \exp\left\{-\frac{1}{2\delta^2}\left[b_1^2z_t^2-2b_1\left(z_{t+1}-b_0\right)z_t+z_t^2-2\left(b_0+b_1z_{t-1}\right)z_t\right]\right\} \\
=\ & \exp\left\{-\frac{b_1^2+1}{2\delta^2}\left(z_t^2-2\times\frac{b_1\left(z_{t+1}-b_0\right)+\left(b_0+b_1z_{t-1}\right)}{b_1^2+1}z_t\right)\right\} \\
\propto\ & \exp\left\{-\frac{b_1^2+1}{2\delta^2}\left(z_t-\frac{b_1\left(z_{t+1}+z_{t-1}\right)-b_1b_0+b_0}{b_1^2+1}\right)^2\right\}
\end{aligned}
$$

then,

$$\mu_{z_t} = \frac{b_0 + b_1(z_{t-1} + z_{t+1}) - b_0 b_1}{1 + b_1^2} \quad \sigma_{z_t}^2 = \frac{\delta^2}{1 + b_1^2}$$

1.2.3 For t=T

$$P(z_T | z_{0:T-1}, \theta) = P(z_T | z_{T-1}, \theta) \sim N(b_0 + b_1 z_{T-1}, \delta^2)$$

then

$$\mu_{z_T} = b_0 + b_1 z_{T-1} \quad \sigma_{z_T}^2 = \delta^2$$