ZJU Computational Physics: Homework #5

Due on Monday, December 23, 2024

Github: https://github.com/NAKONAKO4/ZJU-computational-physics-NAKO

NAKO

Problem 1

MD: Approach to Equilibrium

a. 可以看到能量和动量都是守恒的，且动量基本为 0（10^{-14} 数量级）；同时计算得到的压强与理想气体压强差值相对误差较小。）

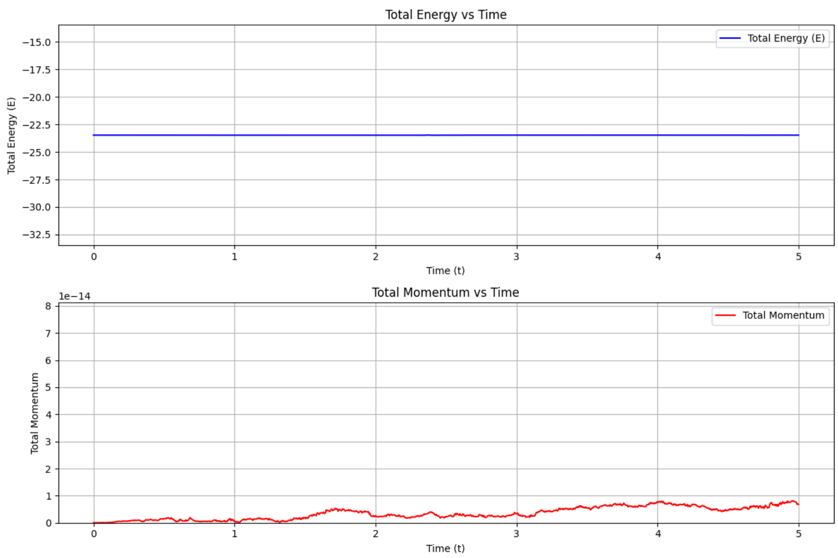


图 1: 能量和动量随时间变化

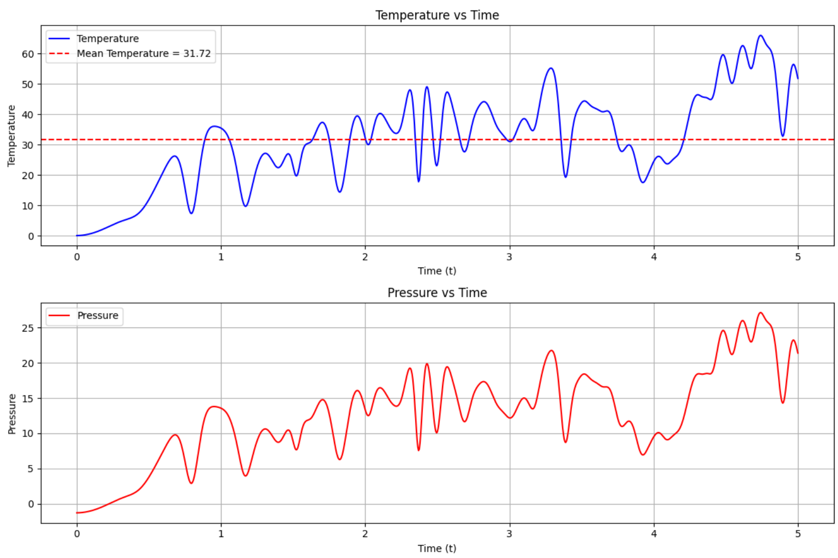


图 2: 温度和压强随时间变化

Problem 1 continued on next page...

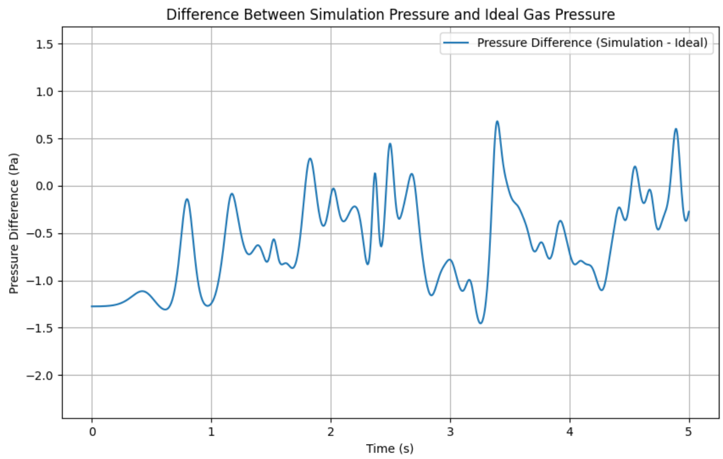


图 3: 压强计算值与理想气体公式计算得到压强差值

b. 可以看到能量和动量都是守恒的，且动量基本为 0（10^{-14} 数量级））

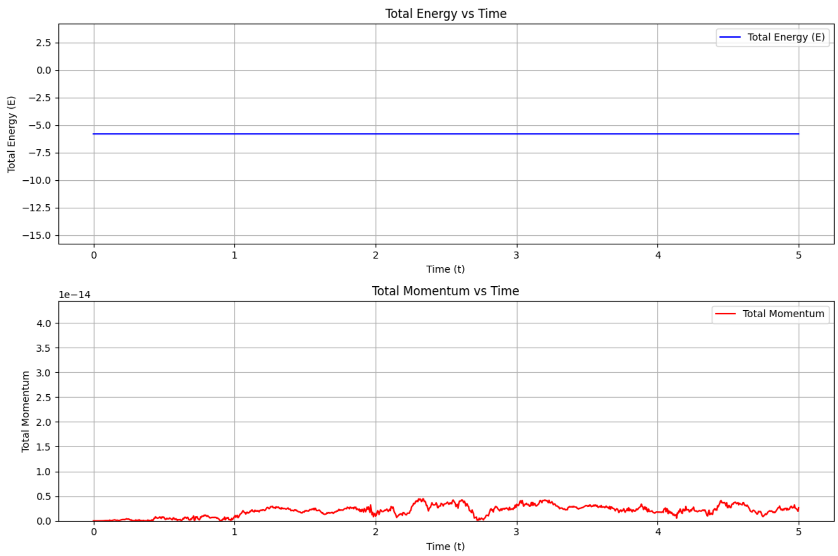


图 4: 能量和动量随时间变化

Problem 1 continued on next page...

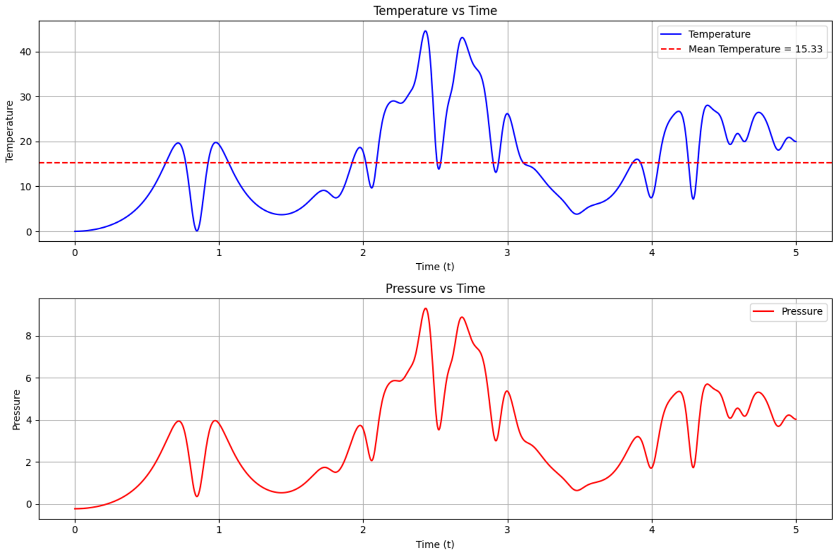
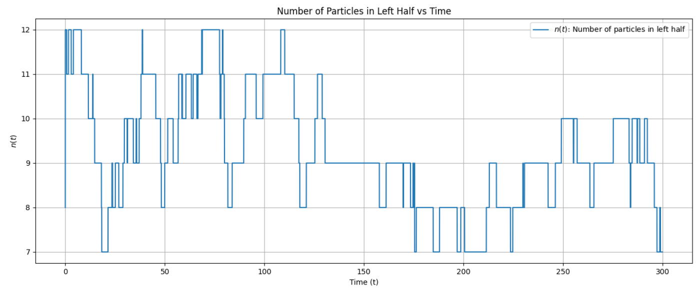
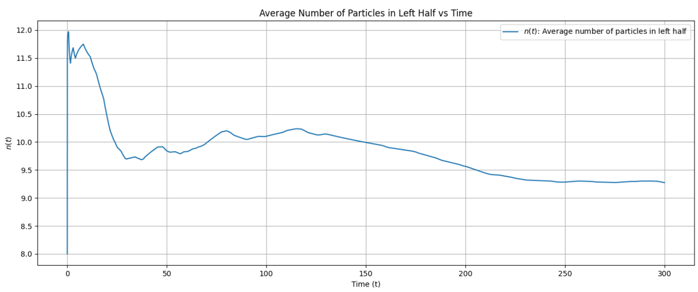


图 5: 温度和压强随时间变化

c. 可以看到接近平衡态时，左边粒子数均值会趋近于总粒子数的一半 $N/2 = 9$）





d. 可以看到能量和动量都是守恒的，且动量基本为 0（10^{-14} 数量级）；同时计算得到的压强与理想气体压强差值相对误差较小。）

Problem 1 continued on next page...

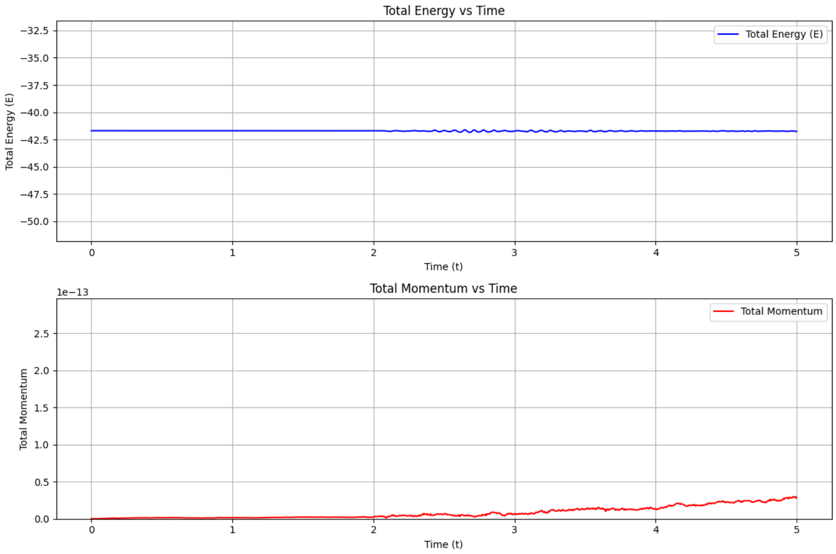


图 6: 能量和动量随时间变化

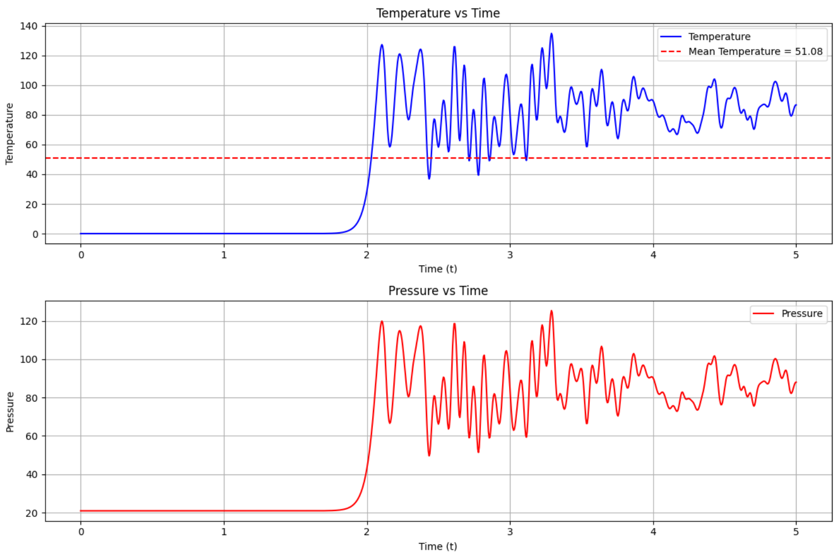


图 7: 温度和压强随时间变化

Problem 1 continued on next page...

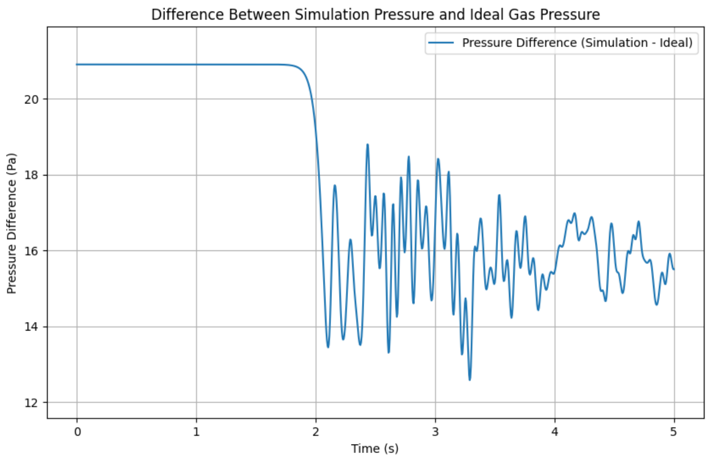


图 8: 压强计算值与理想气体公式计算得到压强差值

e. 可以看到能量和动量都是守恒的，且动量基本为 0（10^{-14} 数量级）；同时计算得到的压强与理想气体压强差值相对误差较小。）

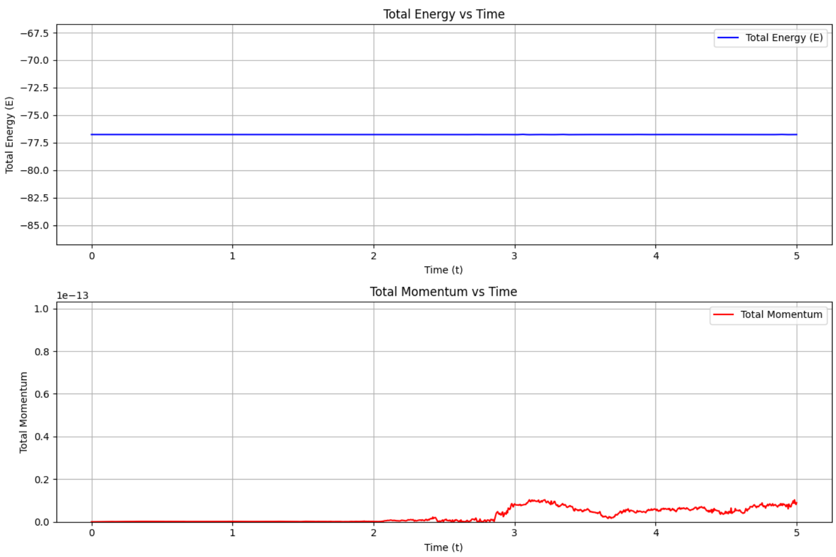


图 9: 能量和动量随时间变化

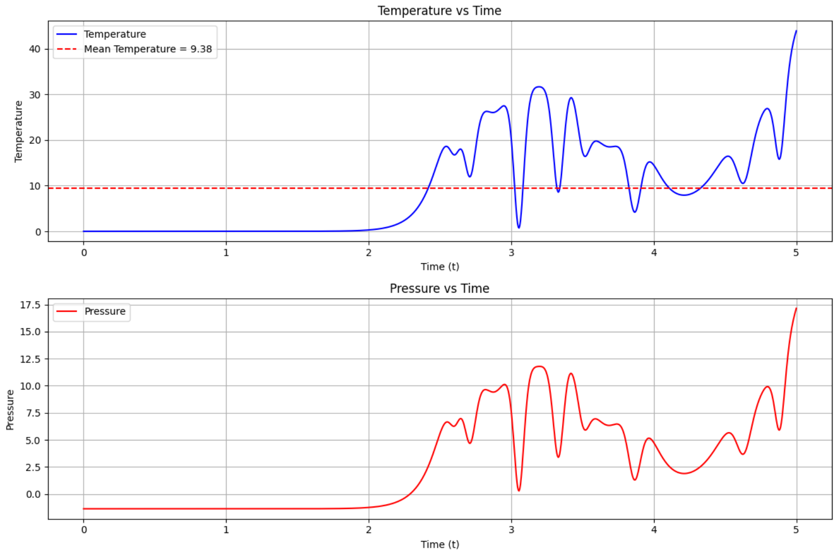


图 10: 温度和压强随时间变化

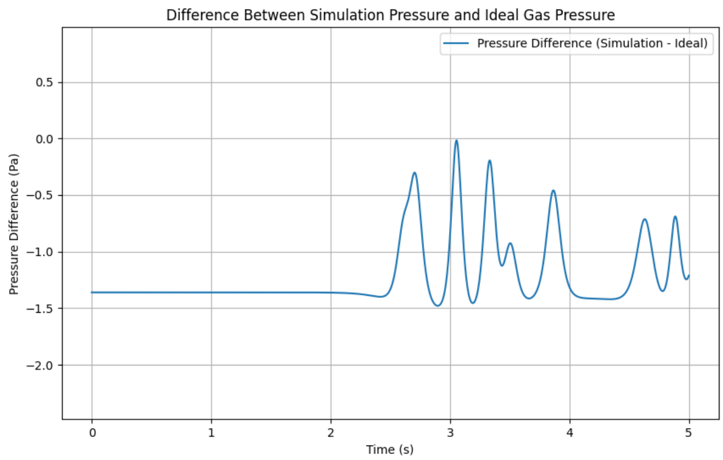


图 11: 压强计算值与理想气体公式计算得到压强差值

abde 题代码如下，对于不同的初始条件只需要更改代码中的初始条件设置即可。c 题代码见本代码的后一部分

```
import numpy as np
import matplotlib.pyplot as plt

sigma = 0.1
epsilon = 1
mass = 6.69e-26
tau = sigma * np.sqrt(mass / epsilon)
k_B = 1.38e-23
```

Problem 1 continued on next page...

```python

dt = 0.005
dt2 = dt**2
N = 18
Lx, Ly = 6.0, 6.0
k_B = 1.38e-23
temperature = 100
m = 1
area = Lx * Ly

def create_lattice(N, Lx, Ly):
    positions = np.zeros((N, 2))
    num_rows = int(np.sqrt(N))
    num_cols = int(np.ceil(N / num_rows))
    x_spacing = Lx / num_cols
    y_spacing = Ly / num_rows

    index = 0
    for row in range(num_rows):
        for col in range(num_cols):
            if index >= N:
                break
            x_pos = (col + 0.5) * x_spacing
            y_pos = (row + 0.5) * y_spacing
            positions[index] = [x_pos, y_pos]
            index += 1
    return positions

def init_velocity(N, temperature, mass):
    stddev = np.sqrt(k_B * temperature / mass)
    velocities = np.random.normal(0, stddev, (N, 2))
    velocities -= np.mean(velocities, axis=0)
    return velocities

positions = create_lattice(N, Lx, Ly)
velocities = init_velocity(N, temperature, m)

x, y = positions[:, 0], positions[:, 1]
vx, vy = velocities[:, 0], velocities[:, 1]
print(vx)
ax = np.zeros(N)
ay = np.zeros(N)

plt.figure(figsize=(6, 6))
plt.scatter(x, y, c='blue', label='Particles', s=100)
plt.xlim(0, Lx)
plt.ylim(0, Ly)
```

Problem 1 continued on next page...

```python
plt.gca().set_aspect('equal', adjustable='box')
plt.xlabel('X Position (Å)')
plt.ylabel('Y Position (Å)')
plt.title('Particle Positions in 6x6 Box')
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.legend()
plt.show()


def check_momentum(vx, vy):
    vx -= np.sum(vx) / N
    vy -= np.sum(vy) / N
    return vx, vy

def pbc(pos, L):
    return (pos + L) % L

def separation(ds, L):
    return ds - L * np.round(ds / L)

def force(dx, dy):
    r2 = dx**2 + dy**2
    if r2 == 0:
        return 0, 0, 0
    rm2 = 1.0 / r2
    rm6 = rm2**3
    f_over_r = 24 * rm6 * (2 * rm6 - 1) * rm2
    fx = f_over_r * dx
    fy = f_over_r * dy
    pot = 4.0 * (rm6**2 - rm6)
    return fx, fy, pot

def accel(x, y, ax, ay):
    ax.fill(0)
    ay.fill(0)
    pe = 0.0
    virial = 0.0
    for i in range(N - 1):
        for j in range(i + 1, N):
            dx = separation(x[i] - x[j], Lx)
            dy = separation(y[i] - y[j], Ly)
            fx, fy, pot = force(dx, dy)
            ax[i] += fx
            ay[i] += fy
            ax[j] -= fx
            ay[j] -= fy
            pe += pot
```

Problem 1 continued on next page...

```python
            virial += dx* fx + dy* fy
    return pe, virial

def verlet(x, y, vx, vy, ax, ay):
    x += vx * dt + 0.5 * ax * dt2
    y += vy * dt + 0.5 * ay * dt2
    x = pbc(x, Lx)
    y = pbc(y, Ly)
    vx += 0.5 * ax * dt
    vy += 0.5 * ay * dt
    pe, virial = accel(x, y, ax, ay)
    vx += 0.5 * ax * dt
    vy += 0.5 * ay * dt
    ke = 0.5 * m * np.sum(vx**2 + vy**2)
    return ke, pe, virial

def compute_momentum(vx, vy):
    px = np.sum(m * vx)
    py = np.sum(m * vy)
    total_momentum = np.sqrt(px**2 + py**2)
    return total_momentum

vx, vy = check_momentum(vx, vy)
ke = 0.5 * np.sum(vx**2 + vy**2)
pe, virial = accel(x, y, ax, ay)
total_momentum = compute_momentum(vx, vy)
print(f"{'time':>6} {'E':>12} {'Momentum':>12} {'T':>12} {'P':>12}")

t_values = []
T_values = []
E_values = []
P_values = []
momentum_values = []
pressure_diff_values = []

t = 0.0
while t < 5.0:
    d=2
    ke, pe, virial = verlet(x, y, vx, vy, ax, ay)
    total_energy = ke + pe
    total_momentum = compute_momentum(vx, vy)
    pressure = (0.5 * virial) / area

    T = (2 * ke*epsilon) / ((d * N-d) * k_B)
    pressure = (N * k_B * T / (area*(sigma**2))) + (virial / (d * area))
    P_ideal = (N * k_B *T) / (area*(sigma**2))

```

Problem 1 continued on next page...

```python
    pressure_diff = pressure - P_ideal

    t_values.append(t)
    E_values.append(total_energy)
    P_values.append(pressure)
    momentum_values.append(total_momentum)
    pressure_diff_values.append(pressure_diff)
    #print(f"{t:6.2f} {total_energy:12.4f} {total_momentum:12.4f} {2*ke*mass*(1.57e2)**2/ (N * k_B):12.4e} {pressure:12.4e}")
    T_values.append(T*1.65e-21)
    t += dt

plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.plot(t_values, E_values, label='Total Energy (E)', color='blue')
plt.ylim(np.min(E_values)-10, np.max(E_values)+10)
plt.xlabel('Time (t)')
plt.ylabel('Total Energy (E)')
plt.title('Total Energy vs Time')
plt.grid(True)
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(t_values, momentum_values, label='Total Momentum', color='red')
plt.ylim(0,10*np.max(momentum_values))
plt.xlabel('Time (t)')
plt.ylabel('Total Momentum')
plt.title('Total Momentum vs Time')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.plot(t_values, T_values, label='Temperature', color='blue')
T_mean = np.mean(T_values)
plt.axhline(y=T_mean, color='red', linestyle='--', label=f'Mean Temperature = {T_mean:.2f}')
plt.xlabel('Time (t)')
plt.ylabel('Temperature')
plt.title('Temperature vs Time')
plt.grid(True)
plt.legend()

```

Problem 1 continued on next page...

```
plt.subplot(2, 1, 2)
plt.plot(t_values, P_values, label='Pressure', color='red')
plt.xlabel('Time (t)')
plt.ylabel('Pressure')
plt.title('Pressure vs Time')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(t_values, pressure_diff_values, label="Pressure Difference (Simulation - Ideal)")
plt.ylim(np.min(pressure_diff_values)-1,np.max(pressure_diff_values)+1)
plt.xlabel('Time (s)')
plt.ylabel('Pressure Difference (Pa)')
plt.title('Difference Between Simulation Pressure and Ideal Gas Pressure')
plt.legend()
plt.grid(True)
plt.show()
```

c 题代码如下

```
import numpy as np
import matplotlib.pyplot as plt

sigma = 3.4e-10
epsilon = 1.65e-21
mass = 6.69e-26
tau = sigma * np.sqrt(mass / epsilon)
k_B = 1.38e-23

dt = 0.005
dt2 = dt**2
N = 18
Lx, Ly = 12.0, 6.0
k_B = 1.38e-23
temperature = 100
m = 1
area = Lx * Ly

def create_lattice(N, Lx, Ly):
    positions = np.zeros((N, 2))
    num_rows = int(np.sqrt(N))
    num_cols = int(np.ceil(N / num_rows))
    x_spacing = Lx / num_cols
```

Problem 1 continued on next page...

```python
    y_spacing = Ly / num_rows

    index = 0
    for row in range(num_rows):
        for col in range(num_cols):
            if index >= N:
                break
            x_pos = (col + 0.5) * x_spacing
            y_pos = (row + 0.5) * y_spacing
            positions[index] = [x_pos, y_pos]
            index += 1
    return positions

def init_velocity(N, temperature, mass):
    stddev = np.sqrt(k_B * temperature / mass)
    velocities = np.random.normal(0, stddev, (N, 2))
    velocities -= np.mean(velocities, axis=0)
    return velocities

positions = create_lattice(N, Lx, Ly)
velocities = init_velocity(N, temperature, m)

x, y = positions[:, 0], positions[:, 1]
vx, vy = velocities[:, 0], velocities[:, 1]
ax = np.zeros(N)
ay = np.zeros(N)

plt.figure(figsize=(6, 6))
plt.scatter(x, y, c='blue', label='Particles', s=100)
plt.xlim(0, Lx)
plt.ylim(0, Ly)
plt.gca().set_aspect('equal', adjustable='box')
plt.xlabel('X Position (Å)')
plt.ylabel('Y Position (Å)')
plt.title('Particle Positions in 6x6 Box')
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.legend()
plt.show()


def check_momentum(vx, vy):
    vx -= np.sum(vx) / N
    vy -= np.sum(vy) / N
    return vx, vy

def pbc(pos, L):
    return (pos + L) % L
```

Problem 1 continued on next page...

```python
def separation(ds, L):
    return ds - L * np.round(ds / L)

def force(dx, dy):
    r2 = dx**2 + dy**2
    if r2 == 0:
        return 0, 0, 0
    rm2 = 1.0 / r2
    rm6 = rm2**3
    f_over_r = 24 * rm6 * (2 * rm6 - 1) * rm2
    fx = f_over_r * dx
    fy = f_over_r * dy
    pot = 4.0 * (rm6**2 - rm6)
    return fx, fy, pot

def accel(x, y, ax, ay):
    ax.fill(0)
    ay.fill(0)
    pe = 0.0
    virial = 0.0
    for i in range(N - 1):
        for j in range(i + 1, N):
            dx = separation(x[i] - x[j], Lx)
            dy = separation(y[i] - y[j], Ly)
            fx, fy, pot = force(dx, dy)
            ax[i] += fx
            ay[i] += fy
            ax[j] -= fx
            ay[j] -= fy
            pe += pot
            virial += dx * fx + dy * fy
    return pe, virial

def verlet(x, y, vx, vy, ax, ay):
    x += vx * dt + 0.5 * ax * dt2
    y += vy * dt + 0.5 * ay * dt2
    x = pbc(x, Lx)
    y = pbc(y, Ly)
    vx += 0.5 * ax * dt
    vy += 0.5 * ay * dt
    pe, virial = accel(x, y, ax, ay)
    vx += 0.5 * ax * dt
    vy += 0.5 * ay * dt
    ke = 0.5 * m * np.sum(vx**2 + vy**2)
    return ke, pe, virial

```

Problem 1 continued on next page...

```
def compute_momentum(vx, vy):
    px = np.sum(m * vx)
    py = np.sum(m * vy)
    total_momentum = np.sqrt(px**2 + py**2)
    return total_momentum

n_t_values = []
left_half = Lx / 2

t = 0.0
while t < 300.0:
    ke, pe, virial = verlet(x, y, vx, vy, ax, ay)

    n_t = np.sum(x < left_half)
    n_t_values.append(n_t)

    t += dt
n_t_values = n_t_values[0:60000]

t_values = np.arange(0, 300.0, dt)
print(len(t_values))

n_t_mean = []

for i in range(len(t_values)):
    n_t_mean.append(np.mean(n_t_values[0:i]))

plt.figure(figsize=(16, 6))
plt.plot(t_values, n_t_values, label='$n(t)$: Number of particles in left half')
plt.xlabel('Time (t)')
plt.ylabel('$n(t)$')
plt.title('Number of Particles in Left Half vs Time')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(16, 6))
plt.plot(t_values, n_t_mean, label='$n(t)$: Average number of particles in left half')
plt.xlabel('Time (t)')
plt.ylabel('$n(t)$')
plt.title('Average Number of Particles in Left Half vs Time')
plt.legend()
plt.grid(True)
plt.show()

print(f"Time-averaged number of particles in left half: {n_t_mean:.2f}")
```

Problem 2

Ising Model

a. 可以看到 T_c 随着材料扩大，逐渐逼近无限大系统的相变温度，大约在 2.35 附近

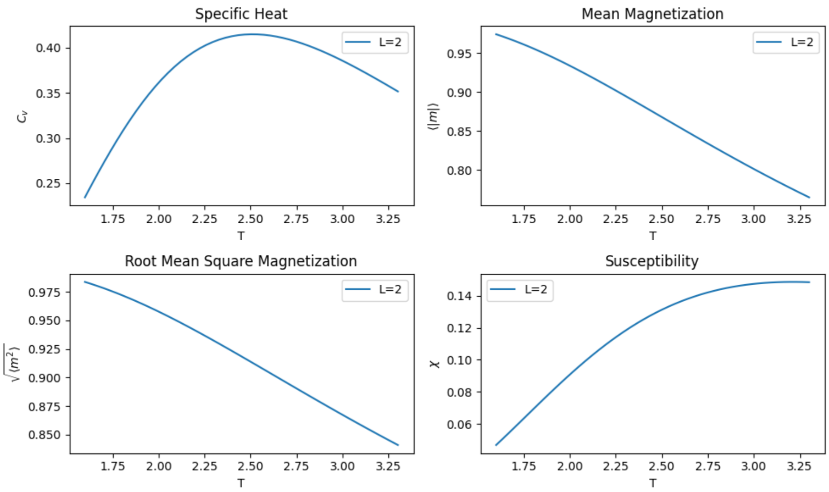


图 12: L=2

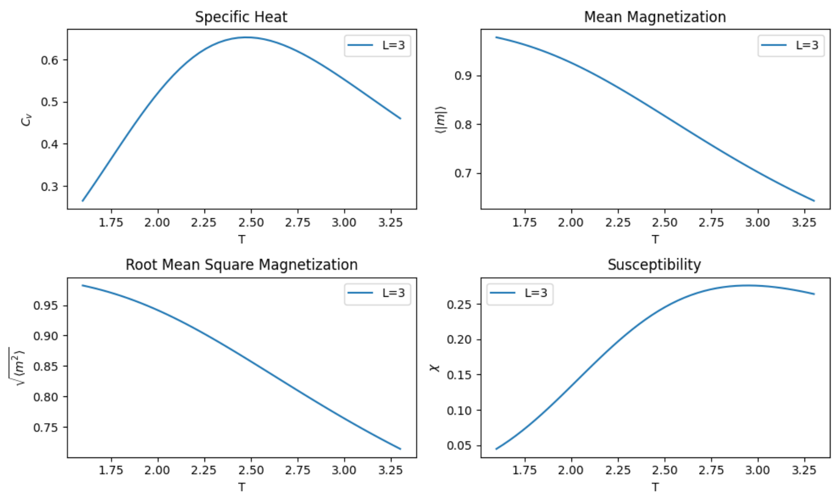


图 13: L=3

Problem 2 continued on next page...

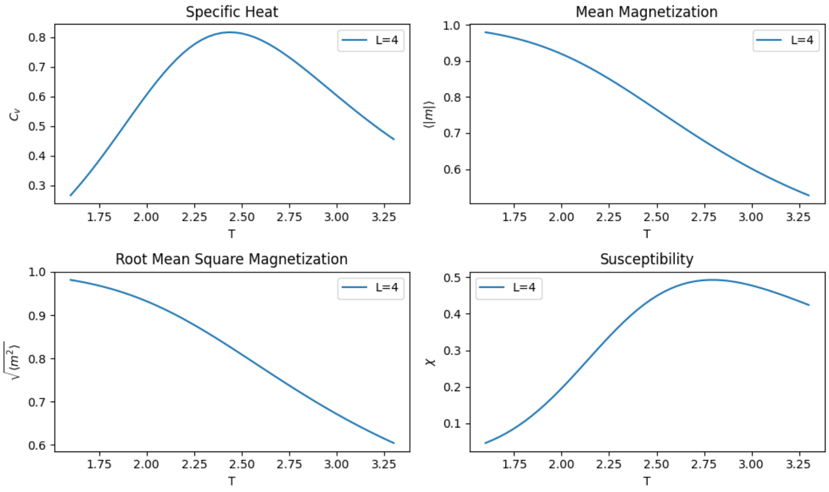


图 14: L=4

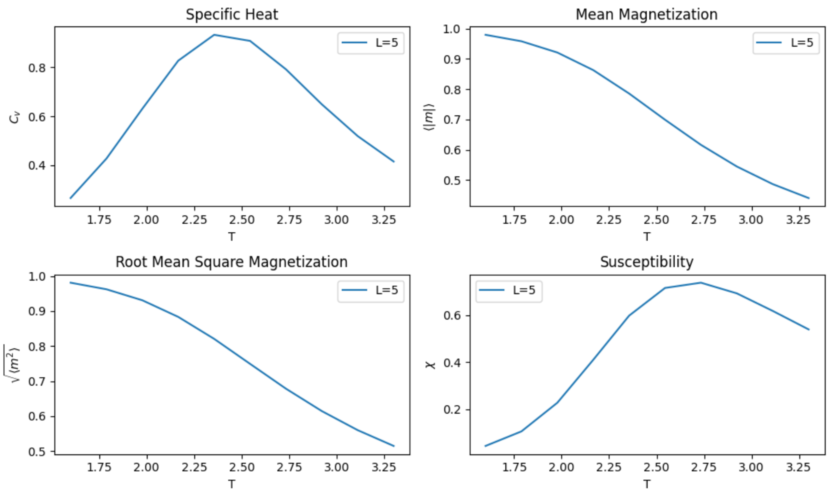


图 15: L=5

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import product

J = 1.0
L_values = [2,3,4,5]
T_range = np.linspace(1.6, 3.3, 10)


def compute_ising_properties(L, T_range):

    N = L * L
    print(1)
    states = np.array(list(product([-1, 1], repeat=N)))
```

Problem 2 continued on next page...

```
    print(states)
    E_values = []
    M_values = []

    for state in states:
        state = state.reshape(L, L)
        E = 0
        M = np.sum(state)
        for i in range(L):
            for j in range(L):
                E -= J * state[i, j] * (state[(i + 1) % L, j] + state[i, (j + 1) % L])
        E_values.append(E)
        M_values.append(M)
    E_values = np.array(E_values)
    M_values = np.array(M_values)

    results = {"Cv": [], "m_mean": [], "sqrt_m2": [], "m_max": [], "chi": []}
    for T in T_range:
        beta = 1 / T
        Z = np.sum(np.exp(-beta * E_values))
        E_mean = np.sum(E_values * np.exp(-beta * E_values)) / Z
        E2_mean = np.sum((E_values ** 2) * np.exp(-beta * E_values)) / Z
        M_mean = np.sum(np.abs(M_values) * np.exp(-beta * E_values)) / Z
        M2_mean = np.sum((M_values ** 2) * np.exp(-beta * E_values)) / Z
        M_max = np.max(np.abs(M_values))

        Cv = (E2_mean - E_mean ** 2) * beta ** 2 / N
        chi = (M2_mean - M_mean ** 2) * beta / N

        results["Cv"].append(Cv)
        results["m_mean"].append(M_mean / N)
        results["sqrt_m2"].append(np.sqrt(M2_mean) / N)
        results["m_max"].append(M_max / N)
        results["chi"].append(chi)
    return results


for L in [1]:
    results = compute_ising_properties(5, T_range)
    plt.figure(figsize=(10, 6))

    # 绘制 Cv
    plt.subplot(2, 2, 1)
    plt.plot(T_range, results["Cv"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$C_v$")
    plt.title("Specific Heat")
```

Problem 2 continued on next page...

```
    plt.legend()

    # 绘制 <|M|>
    plt.subplot(2, 2, 2)
    plt.plot(T_range, results["m_mean"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\langle |m| \\rangle$")
    plt.title("Mean Magnetization")
    plt.legend()

    # 绘制 SQRT(<M^2>)
    plt.subplot(2, 2, 3)
    plt.plot(T_range, results["sqrt_m2"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\sqrt{\\langle m^2 \\rangle}$")
    plt.title("Root Mean Square Magnetization")
    plt.legend()

    # 绘制 CHI
    plt.subplot(2, 2, 4)
    plt.plot(T_range, results["chi"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\chi$")
    plt.title("Susceptibility")
    plt.legend()

    plt.tight_layout()
    plt.show()
```

b. 可以看到 T_c 随着材料扩大逼近大约 2.25 附近的过程，其中 L=128 的图片由于材料较大，在有限计算资源下难以达到接近平衡态的状态，进而出现多个峰值，可以估计出真实峰值应该接近 2.25。

蒙特卡洛模拟参数：10000 步 MC 来达到平衡态，之后 5000 步进行测量。由于 L=64、L=128 的材料尺寸很大，在这个 MC 模拟参数下运算一次需要很长时间，且计算资源有限，因此没有设置更大的模拟参数，如果计算资源足够可以设置更大参数，得到更准确结果。

Problem 2 continued on next page...

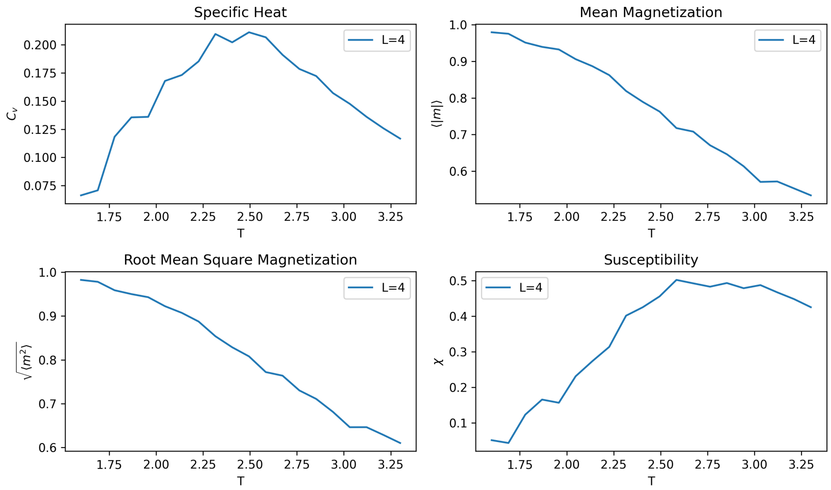


图 16: L=4

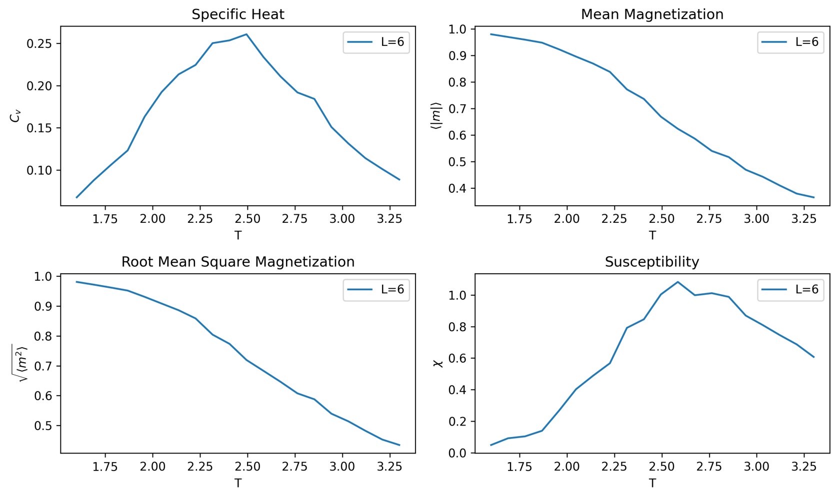


图 17: L=6

Problem 2 continued on next page...

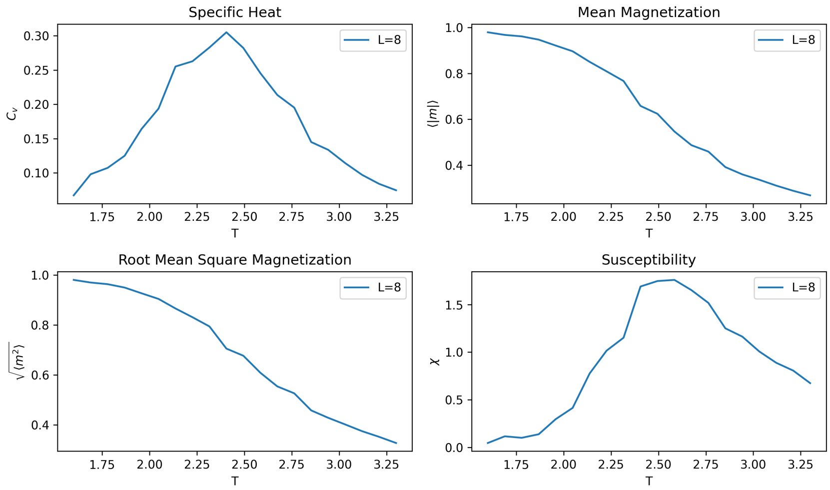


图 18: L=8

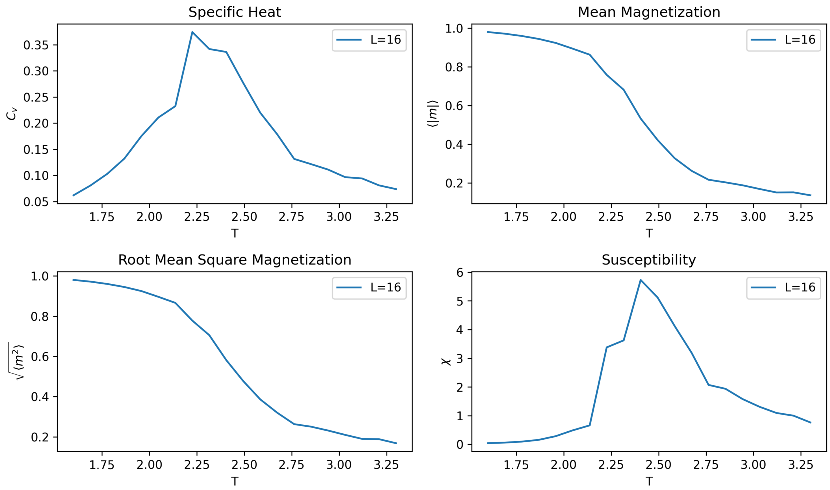


图 19: L=16

Problem 2 continued on next page...

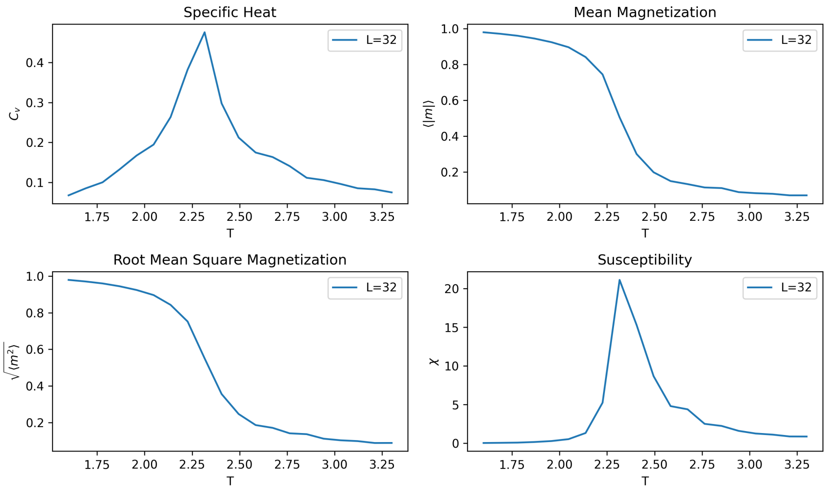


图 20: L=32

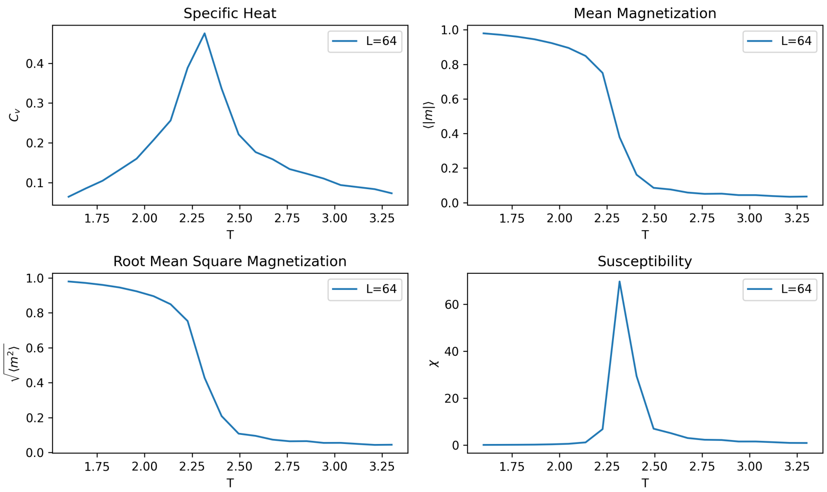


图 21: L=64

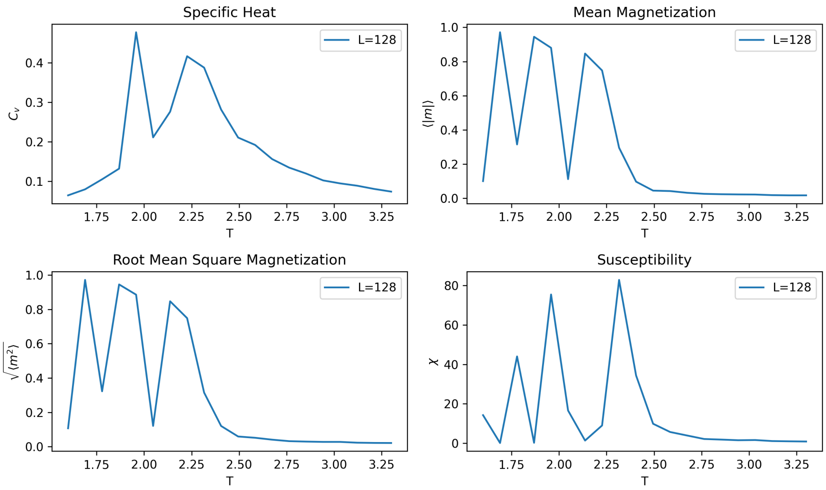


图 22: L=128

```python
import numpy as np
import matplotlib.pyplot as plt

J = 1.0
L_values = [4, 6, 8, 10, 16, 32, 64, 128]
T_range = np.linspace(1.6, 3.3, 20)
MCS = 100
measurements = 50


def metropolis_step(spins, beta, L):

    for _ in range(L * L):
        i, j = np.random.randint(0, L), np.random.randint(0, L)
        dE = 2 * J * spins[i, j] * (
                spins[(i + 1) % L, j] + spins[i, (j + 1) % L] +
                spins[(i - 1) % L, j] + spins[i, (j - 1) % L]
        )
        if dE <= 0 or np.random.rand() < np.exp(-beta * dE):
            spins[i, j] *= -1


def monte_carlo_simulation(L, T_range, MCS, measurements):

    N = L * L
    results = {"Cv": [], "m_mean": [], "sqrt_m2": [], "m_max": [], "chi": []}

    for T in T_range:
        beta = 1 / T
        spins = np.random.choice([-1, 1], size=(L, L))
```

```python
        E_values = []
        M_values = []

        for _ in range(MCS):
            metropolis_step(spins, beta, L)

        for _ in range(measurements):
            metropolis_step(spins, beta, L)
            E = -J * np.sum(spins * (
                    np.roll(spins, 1, axis=0) +
                    np.roll(spins, 1, axis=1)
            )) / 2
            M = np.sum(spins)
            E_values.append(E)
            M_values.append(M)

        E_values = np.array(E_values)
        M_values = np.array(M_values)
        E_mean = np.mean(E_values)
        E2_mean = np.mean(E_values ** 2)
        M_mean = np.mean(np.abs(M_values))
        M2_mean = np.mean(M_values ** 2)

        Cv = beta ** 2 * (E2_mean - E_mean ** 2) / N
        chi = beta * (M2_mean - M_mean ** 2) / N
        m_max = np.max(np.abs(M_values)) / N

        results["Cv"].append(Cv)
        results["m_mean"].append(M_mean / N)
        results["sqrt_m2"].append(np.sqrt(M2_mean) / N)
        results["m_max"].append(m_max)
        results["chi"].append(chi)

    return results


for L in L_values:
    print(f"Running simulation for L = {L}...")
    results = monte_carlo_simulation(L, T_range, MCS, measurements)

    plt.figure(figsize=(10, 6))

    # 绘制 Cv
    plt.subplot(2, 2, 1)
    plt.plot(T_range, results["Cv"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$C_v$")
```

```
    plt.title("Specific Heat")
    plt.legend()

    # 绘制 <|m|>
    plt.subplot(2, 2, 2)
    plt.plot(T_range, results["m_mean"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\langle |m| \\rangle$")
    plt.title("Mean Magnetization")
    plt.legend()

    # 绘制 sqrt(<m^2>)
    plt.subplot(2, 2, 3)
    plt.plot(T_range, results["sqrt_m2"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\sqrt{\\langle m^2 \\rangle}$")
    plt.title("Root Mean Square Magnetization")
    plt.legend()

    # 绘制 chi
    plt.subplot(2, 2, 4)
    plt.plot(T_range, results["chi"], label=f"L={L}")
    plt.xlabel("T")
    plt.ylabel("$\\chi$")
    plt.title("Susceptibility")
    plt.legend()

    plt.tight_layout()
    plt.show()
```