Computational Physics (PHYS6350)

Lecture 19: Problems in statistical physics

- Markov chain and Metropolis algorithm
- Ising model

Reference: Chapter 10 of *Computational Physics* by Mark Newman

April 4, 2023

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Thermodynamic averages

For a system in statistical equilibrium at given temperature, the probability that the system is in microstate *i* is given by the Boltzmann formula:

$$P(E_i) = \frac{e^{-\beta E_i}}{Z}$$

$\beta = 1/(k_B T)$ is the inverse temperature E_i is the energy in state i $Z = \sum_i e^{-\beta E_i}$ is the partition function.

Main interest typically lies in calculating the average of various physical observables.
For an arbitrary quantity *X* it reads

$$\langle X \rangle = \sum_i X_i P(E_i)$$

How to calculate <X>?

Thermodynamic averages and importance sampling

One possible way to estimate <X> is to sample each microstate uniformly at random, calculate X_i and accept with a weight proportional to $P(E_i)$. If we have N samples, the estimate for <X> reads

$$\langle X \rangle = \frac{\sum_{k=1}^{N} X_k P(E_k)}{\sum_{k=1}^{N} P(E_k)} = \frac{\sum_{k=1}^{N} X_k e^{-\beta E_k}}{\sum_{k=1}^{N} e^{-\beta E_k}}$$

This method does not require the evaluation of the partition function Z.

However, the method is not very efficient because it will typically sample states that do not contribute much to the final result due large penalty from the Boltzmann factor $e^{-\beta E_k}$

Importance sampling: Sample the microstates directly from Boltzmann distribution $P_i \propto e^{-\beta E_i}$

$$\langle X \rangle = \sum_i X_i P(E_i) \simeq \frac{1}{N} \sum_{k=1}^{N} X_k$$

Markov chain method

How to pick states from $P_i = e^{-\beta E_i}/Z$?

Markov chain method:

- Iterative procedure
- Move from state i to state j -> Transition probability T_{ij}

$$\sum_j T_{ij} = 1$$

- Choose T_{ij} such that

$$\frac{T_{ij}}{T_{ji}} = \frac{P_j}{P_i} = \frac{e^{-\beta E_j}/Z}{e^{-\beta E_i}/Z} = e^{-\beta(E_j - E_i)}$$

- No need to compute the partition function Z

If state *i* is drawn from Boltzmann distribution $P_i = e^{-\beta E_i}/Z$ the probability to have state *j* in next step is

$$\sum_i T_{ij} P_i = \sum_i T_{ji} P_j = P_j \sum_i T_{ji} = P_j$$

Metropolis algorithm

Metropolis algorithm is a way to simulate the Markov chain such that $\frac{T_{ij}}{T_{ji}} = \frac{P_j}{P_i} = \frac{e^{-\beta E_j}/Z}{e^{-\beta E_i}/Z} = e^{-\beta(E_j - E_i)}$

Suppose we can make M moves from state i to a new state j – *a move set.*

- Pick a candidate next state *j* uniformly at random (the probability is 1/M).
- Calculate the energy E_j of the candidate state j and compare it to the energy E_i of the current step *I*
  - If $E_j < E_i$, the move is unconditionally accepted.
  - If $E_j > E_i$, the move is accepted with a probability

$$P_a = e^{-\beta(E_j - E_i)}$$

Consider the transition probabilities for the case $E_j > E_i$

- *i->j:*

$$T_{ij} = \frac{1}{M} e^{-\beta(E_j - E_i)}$$

- *j->i:*

$$T_{ji} = \frac{1}{M}$$

therefore $\frac{T_{ij}}{T_{ji}} = \frac{P_j}{P_i} = \frac{e^{-\beta E_j}/Z}{e^{-\beta E_i}/Z} = e^{-\beta(E_j - E_i)}$

Ideal gas in a finite volume

Recall the energy states of a particle in box of length L

Solving the *Schroedinger equation* $-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) = E\psi(x)$ with boundary conditions $\psi(0) = \psi(L) = 0$

Energy levels: $E_n = \frac{\pi^2\hbar^2}{2mL^2}n^2, \qquad n = 1, 2 \ldots$ **3D:** $E_{n_x,n_y,n_z} = \frac{\pi^2\hbar^2}{2mL^2}(n_x^2 + n_y^2 + n_z^2), \qquad n_x, n_y, n_z = 1, 2, \ldots$

Ideal gas:

$$E = \sum_{i=1}^{N} E_{n_x^{(i)},n_y^{(i)},n_z^{(i)}}$$

$n_x^{(i)}, n_y^{(i)}, n_z^{(i)}$ enumerate the microstates

The probability to have a particular state is given by the Boltzmann distribution $P \propto e^{-\beta E}$

Metropolis algorithm: pick a random particle and change one of its energy indices by ± 1

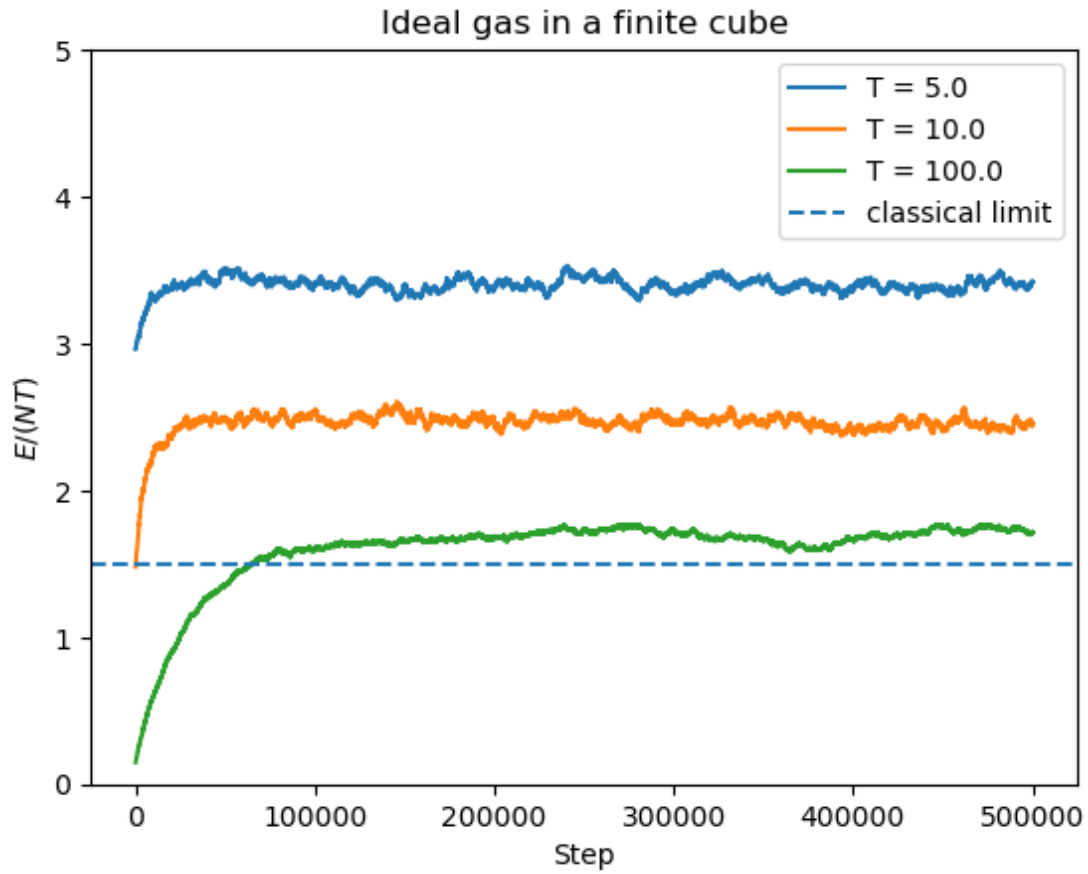
Ideal gas in a finite volume

```python
# Simulates the ideal gas of N particles at temperature T
# by performing Markov chain steps using Metropolis algorithm
# Returns an array energies normalized by the number of particles times the temperature
def simulateIdealGas(T, N, steps, periodicBC):
    # Initialization
    n = np.ones([N,3],int)
    E = 0
    for i in range(N):
        E += En(n[i], periodicBC)

    # Energy per particle normalized by T
    eplot = [ E / (N * T) ]

    for k in range(steps):
        # Choose the particle
        i = np.random.randint(N)
        # Choose the component
        j = np.random.randint(3)
        tn = n[i].copy()
        # Choose the direction
        if (np.random.rand() < 0.5):
            tn[j] += 1
        else:
            tn[j] -= 1

        # If n becomes negative, by symmetry set it to positive (periodic BC)
        if (tn[j] == -1 and periodicBC):
            tn[j] = 1

        # Avoid n = 0 states if not periodic BC
        if (tn[j] == 0 and not periodicBC):
            tn[j] = 1

        # Energy difference
        dE = En(tn, periodicBC) - En(n[i], periodicBC)

        if (np.random.rand() < np.exp(-dE/T)):
            n[i,j] = tn[j]
            E += dE

        eplot.append(E / (N * T))

    return eplot
```

```python
# Calculate energy of a particle in a state n = (nx,ny,nz)
# periodicBC: apply periodic BC
def En(n, periodicBC):
    nx = n[0]
    ny = n[1]
    nz = n[2]
    factor = 0.5
    if (periodicBC):
        factor = 2.
    return factor * np.pi**2 * (nx**2 + ny**2 + nz**2)
```

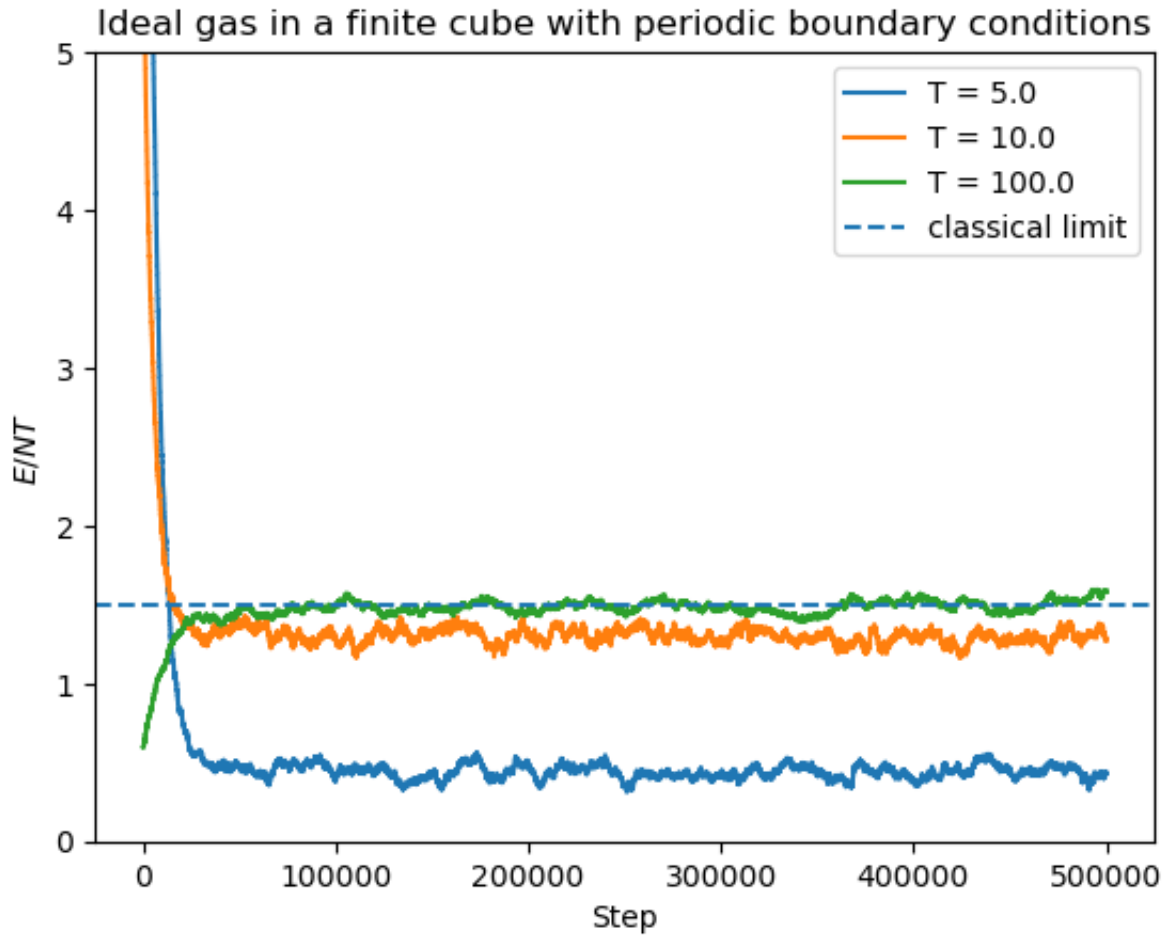
Ideal gas in a finite volume

```
N = 1000
steps = 500000
periodicBC = False
Ts = [5., 10., 100.]
```



Ideal gas in a finite volume with periodic boundary conditions

Periodic boundary conditions: $\psi(x) = \psi(x + L)$ ⟹ $E_{n_x,n_y,n_z} = \frac{2\pi^2\hbar^2}{mL^2}(n_x^2 + n_y^2 + n_z^2), \qquad n_x, n_y, n_z = 0, 1, \ldots$



2D Ising model

Ising model represents a system of spins (magnetic dipoles) on a lattice.

Without external magnetic field, the energy reads

$$E = -J \sum_{<ij>} s_i s_j$$

J > 0: ferromagnetic, sum over neighbors only

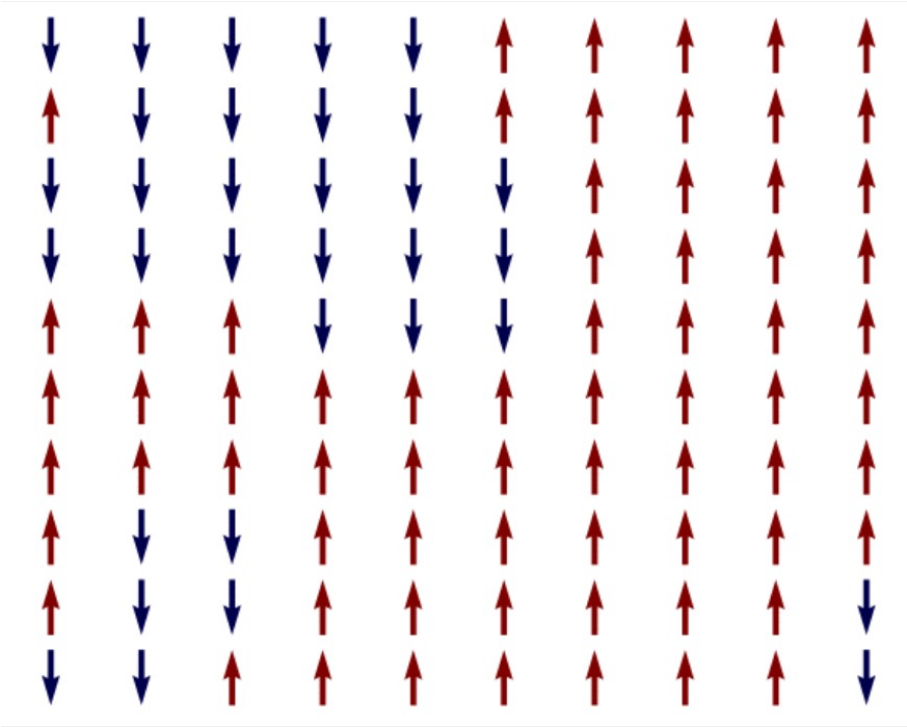
Magnetisation:

$$M = \sum_i s_i$$

Below the Curie temperature

$$\frac{k_B T_C}{J} = \frac{2}{\ln(1+\sqrt{2})}$$

exhibits spontaneous magnetization |M|>0



2D Ising model

Metropolis algorithm for 2D Ising model:

- At each step have spin configuration s_i
- Randomly pick a spin i and flip its orientiation, $s_i -> -s_i$
- Calculate the energy difference

$$\Delta E = 2J \sum_j s_i s_j$$

- Accept new state with probability

$$P_a = e^{-\Delta E/T}$$

2D Ising model

Metropolis algorithm for 2D Ising model:

- At each step have spin configuration s_i
- Randomly pick a spin i and flip its orientiation, s_i-> -s_i
- Calculate the energy difference

$$\Delta E = 2J \sum_j s_i s_j$$

- Accept new state with probability

$$P_a = e^{-\Delta E/T}$$

```python
# Simulates the 2D Ising system of NxN spins at temperature T
# by performing Markov chain steps using Metropolis algorithm
# Returns arrays energies and magnetizations at each step
def simulateIsing(T, N, steps):
    spins = -1 + 2 * np.random.randint(0, high = 2, size=(N,N))

    E = IsingE(spins)
    M = IsingM(spins)

    # Energy
    eplot = [ E ]
    # Magnetisation
    Mplot = [ M ]

    for k in range(steps):
        # Pick the lattice site randomly
        i = np.random.randint(N)
        j = np.random.randint(N)

        # Energy change from flipping the site
        dE = IsingdEflip(spins, i, j)

        # Flip the spin with some probability
        if (np.random.rand() < np.exp(-dE/T)):
            spins[i,j] = -spins[i,j]
            E += dE
            M += 2 * spins[i,j]

        eplot.append(E)
        Mplot.append(M)

    return eplot, Mplot
```

2D Ising model

Simulate $T = 1 < T_C$ several times

20x20 system

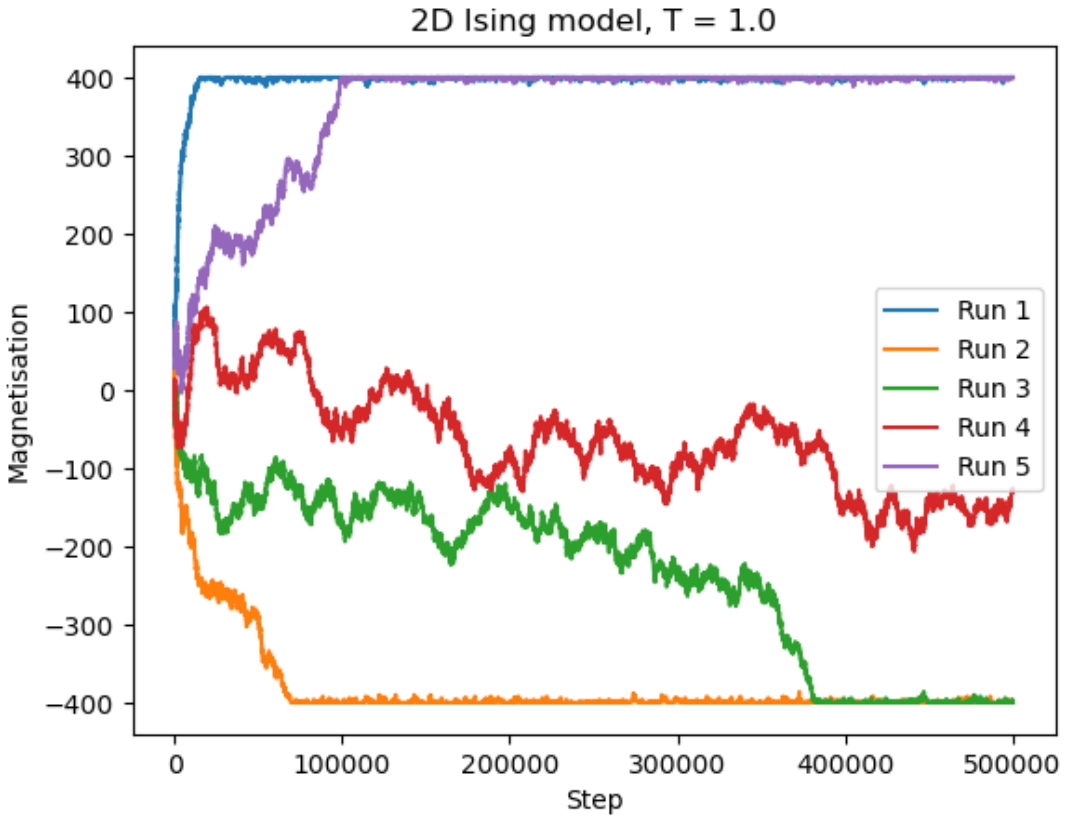
```
N = 20
steps = 500000
periodicBC = True
Temperature = 1.
Ts = np.empty(5)
Ts.fill(Temperature)
eplots = []
Mplots = []

for T in Ts:
    resE, resM = simulateIsing(T, N, steps)
    eplots.append(resE)
    Mplots.append(resM)
```

2D Ising model

Simulate $T = 1 < T_C$ several times

20x20 system

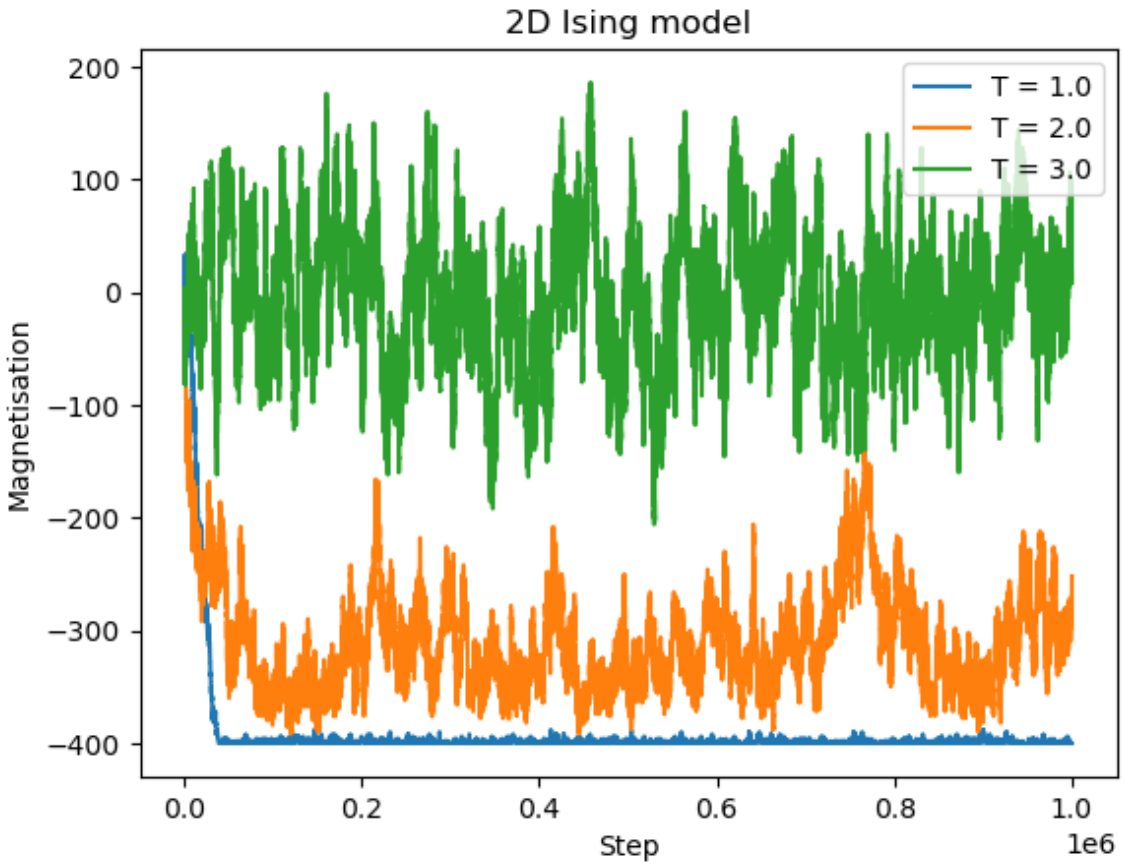
```
N = 20
steps = 500000
periodicBC = True
Temperature = 1.
Ts = np.empty(5)
Ts.fill(Temperature)
eplots = []
Mplots = []

for T in Ts:
    resE, resM = simulateIsing(T, N, steps)
    eplots.append(resE)
    Mplots.append(resM)
```



Spontaneous magnetization!

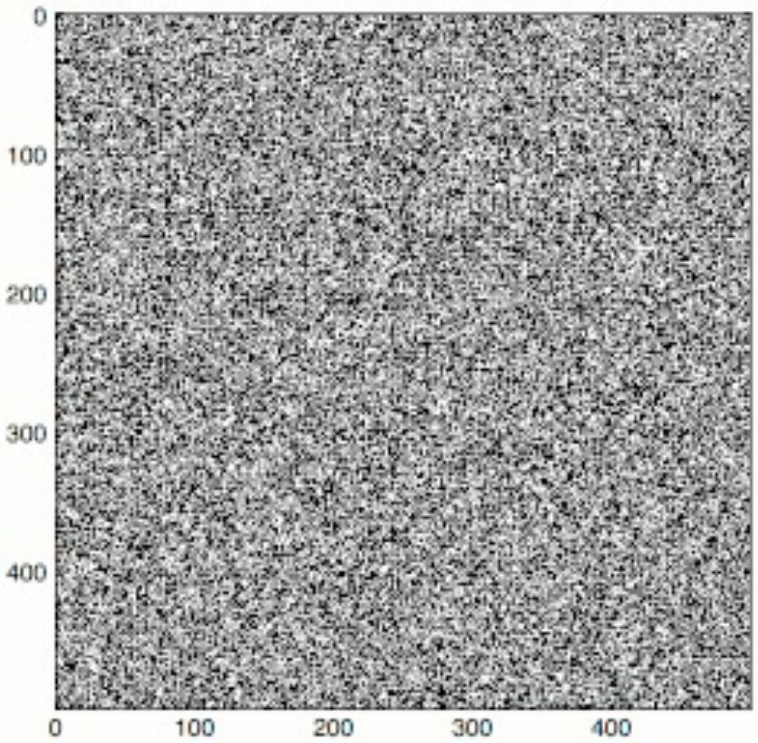
2D Ising model

Try different temperatures



2D Ising model

Large system (500x500)



Credit: Wikipedia