Ising Spin-Lattice Model

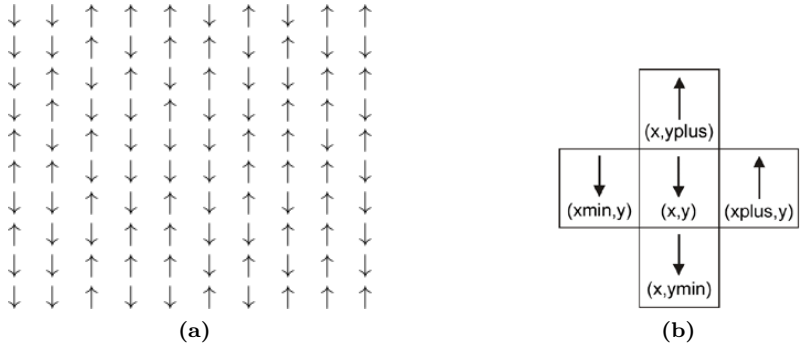
Molecular Statistics

2014

1 Introduction

1.1 The Ising Model

The Ising model is a mathematical representation of a magnetic solid. The Ising model decribes the interaction of atomic magnetic dipoles (or spins) with its neighbors. In this project, you will implement and simulate a 2D square-lattice Ising model. In the 2D Ising model, magnetic spins are arranged on a 2D lattice (see Fig. 1.1a). Magnetic spin can be up or down (represented as 1 or -1).



(a) (b)

Figure 1.1: (a) An example of illustrating a 10x10 Ising spin lattice. (b) Illustrating the spin system (x, y) and its neighbours used to calculate the interaction energy.

A Pythonic representation of a 5x5 spin-system could for instance be a 2D list or a 2D numpy-array with the following contents:

```
lattice = [[ 1. -1.  1. -1.  1.],
           [-1.  1. -1.  1. -1.],
           [ 1. -1. -1.  1.  1.],
           [-1. -1.  1.  1. -1.],
           [ 1.  1. -1. -1.  1.]]
```

An easier way is to use the `np.random.randint()` function to generate a grid.

1.2 Energy of the 2D Ising lattice

Spins interact only with the four neighboring spins. The interaction energy between two neighbors is given by:

$$\epsilon_{ij} = -J\ S_i\ S_j \tag{1.1}$$

The value of S is either 1 or -1, corresponding to a spin-up or spin-down state. J is the interaction-constant. The total energy of a spin lattice is thus:

$$E_{\text{total}} = \sum_{ij}^{\text{neighbours}} \epsilon_{ij} = -J \sum_{ij}^{\text{neighbours}} S_i\ S_j \tag{1.2}$$

where the sum runs over all unique neighboring pairs of magnetic dipoles.

The spin or the element in position i will interact with four other elements. The spin immediately above, below, to the right and to the left of that position as shown in figure 1.1b. The interaction energy `e` between these spins can be calculated in Python as:

```
e  = -J * lattice[x][y] * lattice[x][yplus]
e += -J * lattice[x][y] * lattice[x][ymin]
e += -J * lattice[x][y] * lattice[xplus][y]
e += -J * lattice[x][y] * lattice[xmin][y]
```

To simulate a real lattice, and not just a NxN system, we will be using periodic boundary conditions. If an element is located on an edge we will make the element interact with the first element on the other side. In the example above, the element with indexes (3, 0) will interact with (3, 4), (3, 1), (2, 0) and (4, 0).[1]

If $J > 0$ the lowest energy configuration of the system is when all spins are aligned and have the same value. This is called *ferro-magnetism*. We set $J = 1$ in the following, so all energies reported are in units of J.

1.3 The Metropolis-Hastings algorithm

The Metropolis-Hastings (MH) algorithm is a Monte Carlo algorithm that can be used to describe a probability distribution of the possible states of a system. In this project, you will use the MH algorithm to determine the Boltzmann distribution of the 2D Ising model.

A system is in a state K_n. A random Monte Carlo step then perturbs the system into a new proposed state, K_{n+1}. The energy difference between the two states is $\Delta E = E_{n+1} - E_n$. The probability of accepting the new state is then defined according to the Boltzmann distribution as:

$$P = \min\left[1, \exp\left(-\frac{\Delta E}{k_{\mathrm{B}}T}\right)\right] \tag{1.3}$$

If $\Delta E < 0$, the acceptance probability is $P = 1$, which means that new states with a lower energy are *always* accepted. Conversely, states with $\Delta E > 0$ are only accepted with the probability described above. Whether or not a move is accepted is evaluated using random numbers. Python code for evaluating this could be:

```
for n in range(n_steps):
    # Flip a spin
    dE = ...
    if dE < 0:
        # Accept change
    elif np.exp(-dE / (kB * T)) >= np.random.random():
        # Accept change
    else:
        # Reject change
        # Revert system back to old state
```

If the current energy is saved after each iteration in the loop, the distribution of the recorded energies corresponds to the Boltzmann Distribution (if the simulation is long enough).

It is common to set $k_{\mathrm{B}} = 1$ so that all temperatures reported are in units of k_{B}.

[1] Remember that Python indexing start at 0.

1.4 Physical properties

If you run a simulation as described in the previous chapter, you will obtain a probability distribution of the energy corresponding to the Boltzmann distribution. The analytical expression for the Boltzmann distribution is:

$$\frac{N_i}{N} = \frac{1}{Z} g_i \exp\left[\frac{-E_i}{k_\mathrm{B}T}\right] \tag{1.4}$$

Where $\frac{N_i}{N}$ is the probability to observe the energy state E_i which is g_i degenerate. Z is the partition function. Evaluation of the analytical expression is usually infeasible for large systems. In these cases, Monte Carlo is a powerful tool to describe the Boltzmann distribution.

1.4.1 Heat capacity

The heat capacity describes how much the energy of the system changes with respect to a change in temperature. For our system it can be defined as:

$$C_v = \left(\frac{J}{k_\mathrm{B}T}\right)^2 \left(\langle E^2\rangle - \langle E\rangle^2\right) \tag{1.5}$$

$\langle E\rangle$ and $\langle E^2\rangle$ are Boltzmann averages of the energy and *squared* energy. If you save the energy (and squared energy) after each iteration step throughout the simulation and take the average, you will obtain the correct values. The energy E is **not** the total energy, but the energy *per* spin, i.e. $E = \frac{E_\mathrm{total}}{L^2}$, where L is the width of the lattice.

1.4.2 Overall magnetization

The magnetization is the average spin of the system, and is defined as;

$$M = \frac{1}{L^2} \sum_i S_i \tag{1.6}$$

where L is the width of the lattice. At a low temperature M will be close to either 1 or -1, depending on how the spins align. Conversely, all spins are random in the high temperature limit, and the overall magnetization will be close to 0.

1.4.3 Magnetic susceptibility

The magnetic susceptibility describes how the magnetization changes with respect to a change in temperature. This can be calculated as

$$\chi = \frac{J}{k_\mathrm{B}T} \left(\langle M^2\rangle - \langle M\rangle^2\right) \tag{1.7}$$

where $\langle M\rangle$ and $\langle M^2\rangle$ are calculated as averages over a simulation.

1.4.4 Critical temperature

The temperature $T_c = 2.269$ (the *critical temperature*) is the highest temperature at which there can be a non-zero magnetization. The system undergoes a phase transition if the temperature is changed across T_c, from being an ordered state to a disordered state.

1.5 PyPy

PyPy is an alternative implementation of Python that functions similarly to Numba, which you have previously used via Anaconda. Using PyPy instead of regular Python gives you around a factor of 6 increase in speed, but it doesn't fully support NumPy or Matplotlib. You can get around this by using `lists` instead of `ndarrays`, the `random` module instead of `numpy.random` and saving the simulation observables to a file and plotting it later. Using PyPy isn't strictly necessary, but can help reduce your simulation time. You can install it by executing the following in the terminal:

```
sudo apt-get install pypy
```

Then execute your python code with `pypy <program.py>` instead of with Python.

2 Assignment

2.1 The 2D grid

The central mathematical entity in your program is the 2D lattice. Come up with a way to define a square 2D list (or Numpy array) containing randomly distributed values of -1 or 1.

2.2 Calculating total energy

In order to calculate the total energy of a system you have to sum up all unique pairs of interactions. Make a function called `calc_energy()` that takes the lattice as argument:

```
def calc_energy(lattice):
    energy = 0.0
    for x in range(len(lattice)):
        for y in range(len(lattice)):
            # Calculate xplus, xmin, yplus and ymin here
            # See fig. 2
            energy = energy + lattice[x][y] * ...
    return energy
```

See section 1.2 for hints on calculating the energy. Test your code. An LxL lattice with all spins aligned should have the energy $-2L^2$. Make sure you do not count the same interaction twice.

2.3 Random Monte Carlo move (spin flip)

Monte Carlo (MC) works by randomly perturbing a system (via a *Monte Carlo move*). The simplest MC move in the 2D Ising model is a spin flip of a randomly chosen dipole in the lattice. A random move could for instance be carried out like this:

```
x = np.random.randint(lattice_size)
y = np.random.randint(lattice_size)

grid[x][y] *= -1.0
```

This, of course, has to be carried out once every iteration.

2.4 Implementing the Metropolis-Hastings algorithm

An example of how to implement the MH algorithm is:

```
lattice = ...
for n in range(n_steps):
    e_old = calc_energy(lattice):
    # Flip a spin
    e_new = calc_energy(lattice):
    dE = e_new - e_old
    if dE <= 0:
        # Accept the change
    elif np.exp(-dE / T) >= np.random.random():
        # Accept the change
    else:
        # Reject the change
        # Revert system back to old state
        #   i.e. flip the same spin back.
```

2.5 Incremental energy calculation

If you followed the example in the last section, you calculated the energy by summing up all contributions twice in order to calculate the energy difference, ΔE. It is, however, only necessary to calculate the energy of the parts of the system that change (you only change the energy interaction between four spins). This will make your program run orders of magnitudes faster.

Change your code so you only calculate the total energy once, right after you initialize the grid. During the simulation you can then add ΔE to the energy every time a new state is accepted. Test your implementation by printing and comparing the energy calculated throughout the simulation with the energy you get if you use you `calc_energy(lattice)` function.

This step is required for a program that is fast enough to run simulations of adequate lengths.

2.6 Calculating the heat capacity

After every iteration step save the current energy and the square of the current energy. When a simulation at a given temperature is over, calculate the mean energy and mean squared energy. Use these to calculate the heat capacity.

Test your code by running a simulation with a 20x20 matrix, 100,000 steps and $T = 1.5$. You should get a C_v around 0.0004 to 0.0005. The $\langle E \rangle$ should be around -1.95 and $\langle E^2 \rangle$ around 3.8.

Remember to run a burn-in period of at least 100,000 steps before you start to record the average energy and squared energy. Run your simulation a couple of times to account for statistical variation in the result.

Note that the heat capacity has a divergence around $T = 2.269$. See Fig. 2.2. If you run a simulation around this temperature your results will vary quite a bit.

2.7 Calculate overall magnetization

The magnetization is the mean spin of the system. After each iteration in your Monte Carlo simulation calculate the magnetization and save it. If you have stored your lattice in a 2D numpy-array or a 2D list, you can use the following numpy function to calculate the magnetization:

```
1 M = np.mean(lattice)
```

This is, however, not the fastest way of calculating the magnetization. Just as you did with the energy difference, you can calculate the change in magnetization each iteration and add this to the magnetization. This will be much faster. Test your code by running a simulation at a low temperature ($T < 1$) and a high temperature ($T > 3.0$). In the low temperature limit you should get $\langle M \rangle = \pm 1$ and zero in the high temperature limit - see Fig. 2.3.

2.8 Calculate the magnetic susceptibility

After each iteration, save the magnetization and squared magnetization. When your simulation is over, calculate the mean values and use these to calculate the magnetic susceptibility. If you run a 20x20 lattice for 100,000 steps at $T = 1.5$ you should get χ values between $5 \cdot 10^{-5}$ and $8 \cdot 10^{-5}$. There will be some variation in the numbers you get, so run the simulation a few times and record the average. Again, remember that a proper burn-in period is required.

The magnetic susceptibility has a divergence around $T_c = 2.269$ - see Figure 2.4. If you run your simulation around this value, the results will vary quite a bit.

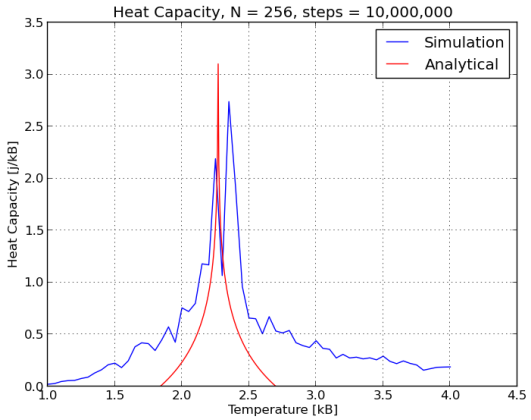


Figure 2.2: The heat capacity derived from a series of simulations using a system of 256x256 spins and 10,000,000 steps, compared to an analytical approximation around the critical temperature.

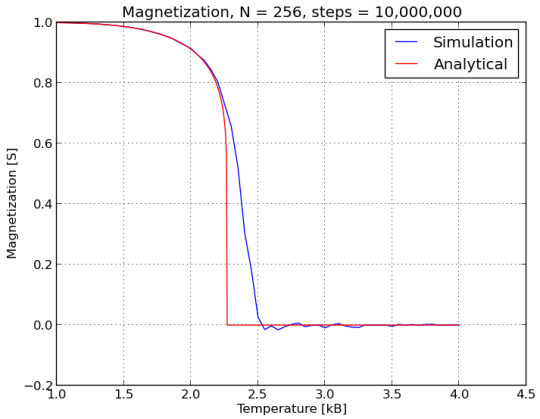


Figure 2.3: The magnetization derived from a series of simulations using a system of 256x256 spins and 10,000,000 steps, compared to the analytical solution.

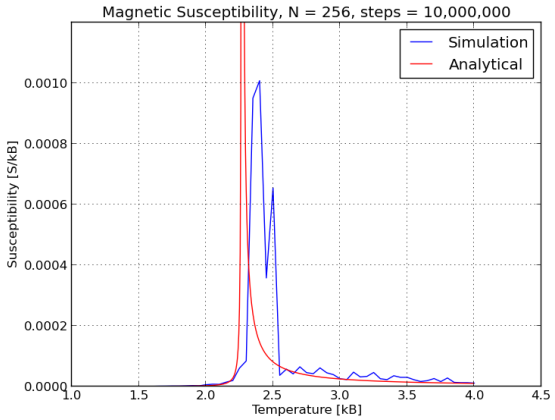


Figure 2.4: The magnetic susceptibility derived from a series of simulations using a system of 256x256 spins and 10,000,000 steps, compared to an analytical approximation.

3 Data

In addition to a brief presentation of the theory and your implementation of the 2D Ising mode, you have to discuss the following results:

1. Investigate how the size of your lattice, the burn-in period before you record E and M values, and the simulation lengths affect the calculated values.

2. Run a long simulation at the critical temperature $T_c = 2.269$ and make a plot of the magnetization and energy for each iteration step. You should be able to observe that the total magnetizations can change back and forth from 1 to -1. Run the same simulation, but at $T = 1.0$ and $T = 3.5$ and compare.

3. Run your simulation at different temperatures and make a plot of the heat capacity, magnetization and magnetic susceptibility versus the temperature. Do you observe a divergences around $T = 2.269$? You might have to run the same simulations more than once since there can be a substantial statistical variation in the values you obtain for C_v, M and χ.