Report of HW8

Muhammad Jamshidi
98100718

1 2D Ising Model

Code: `Code\Ising.py`
In this exercise we want to simulate the 2D Ising model using Metropolis's algorithm. To implement a Monte Carlo simulation of Ising model we need to execute the following steps:

1. *Start with a random initial arrangement, then you'll have a 2D array of +1 and -1. Choose one of these spins and flip it (just change its sign).*

2. *If step 1 leads to reduction of the energy of the system, then the change will be accepted definitely.*

3. *If step 1 leads to increase or constancy of the energy of the system. then the change will be accepted with the probability $e^{(-\beta\Delta E)}$; where ΔE is the change of the energy of the system.*

For Python implementation of the model, I defined the class `Ising_2d_model`. It needs to initialize an object made from this class by giving values of lattice length, beta and number of sampling. It is necessary to mention that we used reduced units and canonical ensemble in implementing the Ising simulations.
Before gathering data like values of energy, magnetization and spatial correlation length between spins, it is necessary to relax the system. This means the system should be evolved until it reaches the equilibrium state. For relaxing the system we defined method `relax_lattice` which uses Metropolis idea to relax the model and it executes the algorithm for `num_of_samples=100` times (here the number 100 is only a reasonable default value and the user can change it if necessary). In each Metropolis step the energy of the system will be recorded in a list and if the correlation of these energy values (`j=int(0.1*num_of_samples)`) is less than e^{-2}, then we say the system has reached to equilibrium state. If this criteria is not satisfied then we consider the system has not reached its equilibrium yet and we need to evolve it more, for this purpose we continue the loop for another `num_of_samples` steps and check again.
Finally when the system has reached its equilibrium state, we need to investigate the list of energy values to find using what value of iterating step, the correlation of energies is minimized. This job is done by the function `find_corr_length`. This function will return the best step for calculating the correlation and we'll use for final calculating and data gathering.
I executed the all instructions explained above for values of of lattice lengths and beta factors came in the below.

1. `length_s = np.array([100, 136, 184, 250])`

2. `beta_s = np.concatenate((np.linspace(0.2, 0.41, 10, endpoint=False, dtype=np.float16`