5CCP211C — Introduction to Numerical Modelling
Physics Department — King's College London

Metropolis Monte Carlo

Zhi Jun Cheung; Muhammad Haider

January 17, 2024

Date performed ..... October 16, 2018
Group 1
Pair ..... Pair#2

1 Background and Theory

Metropolis Monte Carlo The Metropolis Monte Carlo (MMC) algorithm is used to generate samples constricted to a given probability function. This is done in situations where direct sampling may be difficult to achieve such as in high dimension distributions. Below is an outline of how the Monte Carlo method may be used for integration:

Consider the following integral:

$$I = \int_a^b f(x)\, dx \tag{1}$$

Now we introduce a second function $\rho(x)$ which is the probability function of $f(x)$, the integral now becomes:

$$I = \int_a^b \frac{f(x)}{\rho(x)} \rho(x)\, dx \tag{2}$$

This implies that we can picking N values of x_i from the probability distrubution, the integral can now be approximated by:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{\rho(x_i)} \tag{3}$$

This will give the mean of $\frac{f}{\rho}$ in the limit that the number of samples, $N \to \infty$. In the case that $\rho(x)$ is continuous on the interval $[a, b]$, then:

$$\rho(x) = \frac{1}{b-a}, \quad a < x < b \tag{4}$$

$\therefore$

$$I \approx \frac{a-b}{N} \sum_{i=1}^{N} f(x_i) \tag{5}$$

Using this simple Monte Carlo technique to combat problems of low dimensions can prove costly. However, even in high dimension cases due to the complexity of calculating ensemble averages, using this numerical technique can be impossible. Within this technique the probability distribution is such that it is uniform

within the interval $[a, b]$, if another probability distribution such as a Gaussian is used to constrict the randomly generated points, then priority will be given to points closer to high density regions within the distribution and less to the outliers in the low density regions. Using this method is more efficient as it requires less points to achieve a greater accuracy.

The Metropolis method applied to Monte Carlo is one such way to achieve this. This involves generating Markhov chain of states in such a way that limits the resulting distribution further. The Markhov chain starts off by generating an arbitrary point constricted within the Gaussian distribution. By limiting values to a Gaussian, points generated at high denisity regions are more likely to be visited next- making the sequence of samples into a random walk.The distribution of consequent values are dependent on the outcomes that proceeded it. By limiting the distribution further, an acceptance ratio is calculated which is used to determine if the outcome is to be accepted or rejected. The Markhov chain is subject to some conditions in order for it to operate. One such condition is called detailed balance. This states that the probability of transitioning from a given state i to another state j us equal to the probability of transitioning from j to i:

$$\mathbf{P}\left(x_i\right)\pi_{ij} = \mathbf{P}\left(x_j\right)\pi_{ji} \tag{6}$$

Now if :

$$\pi_{ij} = \alpha_{ij}\boldsymbol{P}_{ij} \tag{7}$$

Where $\mathbf{P}_{ij}$ is the probability of a move from the i state to the j and α is the probability ratio.
Then it follows that:

$$\frac{\boldsymbol{P}_{ij}}{\boldsymbol{P}_{ji}} = \frac{\rho_j}{\rho_i} = \frac{e^{-\beta\Delta E(j)}}{e^{-\beta\Delta E(i)}} \tag{8}$$

The fraction of the probabilities of the transition from $i \to j$ is equal to the fraction of limiting distributions applied at steps i and j and the fraction of the Gaussian distribution at states i and j ($\beta = 1/K_bT$). This gives:

$$\frac{\boldsymbol{P}_{ij}}{\boldsymbol{P}_{ji}} = e^{-\beta\Delta E_{ij}} \tag{9}$$

By doing so, we now have a system by which the generated points can be accepted or rejected:

$$\boldsymbol{P}_{ij} = \begin{cases} e^{-\beta\Delta E_{ij}} & \Delta E_{ij} > 0 \\ 1 & \Delta E_{ij} < 0 \end{cases} \tag{10}$$

Structure of MMC:

We start from some initial generated configuration restricted to the Gaussian distribution, i. The system is now perturbed to generate a new configuration j. The energy for both of these configurations is calculated and compared. If the initial energy is higher than the consequent ($\Delta E_{ij} < 0$) then the change is unconditionally accepted (with $\mathbf{P}_{ij}$ as 1). If however the opposite is true ($\Delta E_{ij} > 0$) then a random number, x, is generated from the a uniform distribution in the interval $[0, 1]$ and if $x \leq \mathbf{P}_{ij}$ then the change is accepted and j is taken as the new initial condition.

Ising Model The Ising model is a mathematical model of ferromagnetism in statistical mechanics. In a two dimensional square-lattice model, discrete variables representing magnetic dipole moments of atomic spins are arranged in a square array; in this case of dimensions 30 by 30.

The energy per site is the sum of all energies of all the spins divided by the squared length of side of the lattice. This is the same as the magnetization per site. This is equal to the average energy at each point in the 2D lattice.

The computational model can be checked for equilibrium by increasing the number of Monte Carlo steps. This will ensure that the model is in a state of equilibrium if the model is capable of reaching that state. This is possible at temperatures lower than the critical temperature. At equilibrium, the energy is at its lowest possible value. Therefore, the lowest possible energy at each site is the lowest value of the Hamiltonian. The equation of the Hamiltonian for the Ising model is given below,

$$\hat{H} = -J \sum_{i,j=1}^{N} s_i s_j \quad . \tag{11}$$

where J is a constant magnetic field, which is taken to be equal to positive 1, and s_i and s_j are the spins of the dipole moment of the atom and its neighbour, which can be +1 or -1.

For a positive constant J, the Hamiltonian is a minimum when the product $s_i s_j$ is equal to 1.

2 Overview of Algorithms

Plan and description The basic structure of the code consists of two separate .py scripts, where one is the main script that executes functions and the other is a script consisting of class methods and modules that are imported. The importance of the second script is to generate a class object that represents the configuration of the lattice. Within the class, there are several methods that are used to refer to properties of the object or modify the object. The key

2.1 *Ising obj*

The importance of this part of the code is in the formation of the fundamental elements which are used in the Ising Model. First, the spin of the electrons as well as their coordinates are generated. The spin is restricted to the integers -1 and 1 whilst the xy coordinates are restricted to the dimensions of the lattice which can later be specified. Both quantities are generated by using the random function. The consequent values are stored in a dictionary ($(x, y)M$) which can be recalled later. Next the generated values are fed into the Hamiltonian. Before the energy can be calculated, the configuration which will be used needs to be defined. Since the energy of any given initial configuration depends on the nearest four neighbours of the electron or more specifically their spin. A function which reads and stores the spin of neighbouring electrons is written. The function first reads the spin of the vertical neighbours (up, down) and then the horizontal (left, right). The values along with the position of the spins is then stored in a list. Once the spin for all coordinates have been found and stored, the Hamiltonian is utilised. The energy of each configuration (initial central electron and four neighbours) is calculated.

Next the Metropolis flipper function is used. This is the part of the code where a trail move is made, temporarily generating a new set of configurations on the existing lattice. The Hamiltonian is again used to find the energy states in this configuration. The change in energy is found, if the energy has decreased then the new configuration is accepted unconditionally. If the energy has increased, then the acceptance ratio is employed and a random number $0 \rightarrow 1$ is compared with the probability of obtaining that state, if the probability is smaller than this number then the configuration is accepted, otherwise it is ignored and a new configuration is made which is subject to the same conditions. Lastly, this whole code is iterated for the given lattice dimension.

2.2 Ising2D

The previous code is first imported as a module. The dimensions of the square lattice is specified along with the number of MMC (Metropolis Monte Carlo) steps.

The main parts of the code includes the collection of data and values during the execution of the script. The location at which the values are collected or summed depend on the desired variable. The total average energy of each configuration after each flip is found by accumulating the energy of the configuration over each loop then dividing by the number of Monte Carlo steps. A pseudo code example of how our code collects data points is given below.

```
plotlist1 = []
plotlist2 = []
temp = [1, 1.5, 2...]
for x in temp:
```

```
lattice = initialized_class_object
for i in range(mc_step):
    lat.metropolis_flipper()
    element = somevariable
    plotlist1.append(element)
element2 = somevariable2
plotlist2.append(element2)
```

The total energy of the lattice is calculated by running the code for every configuration and taking the sum of the Hamiltonian output.

The specific heat capacity can be found by using Equation 12,

$$c = \frac{(< E^2 > - < E >^2)}{k_B T^2 N} \quad . \tag{12}$$

where the <> symbols are the average per Monte Carlo step, T the temperature and N the number of sites, which is equal to the squared number of sides. The numerator of Equation 12 is also equal to the variance.

The key variables that are obtained during the loop are $< E^2 >$ and $< E >^2$. This is done by initializing two variables, the total energy and the total squared energy as 0. During each Monte Carlo step, the energy and squared energy of the configuration is added to these variables. At the end of the iteration, the two total values are stored in the variables.

The average of these variables are obtained by dividing by the number of Monte Carlo steps, and the specific heat capacity for the specific temperature can be found.

3 Results and Discussion of Data

A plot for the specific heat capacity against the temperature is shown in Figure 1.

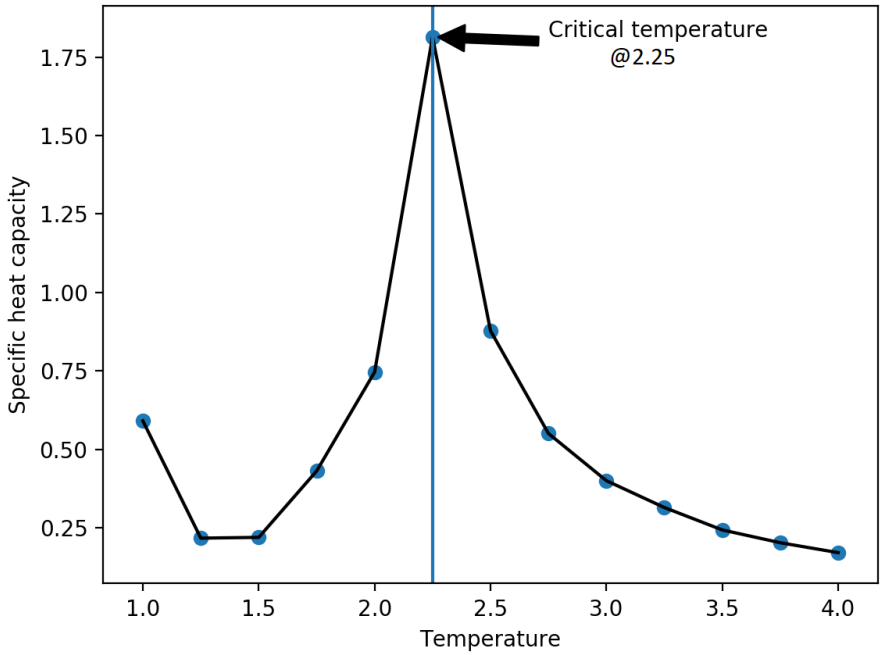

Figure 1: Plot of specific heat capacity against temperature for 20000 Monte Carlo steps

The temperature is given as the reduced temperature, where the Boltzmann constant k_B has been factored in as well. The units are therefore J/k_B. A maxima can be observed at a temperature of value $T_c = 2.25$ in Figure 1. This is the reduced critical temperature determined by the code. The analytical solution for the 2D Ising model is $T_c = 2.269$ as the number of Monte Carlo steps approaches infinity. This suggests that the solution obtained from the code is relatively accurate.

However, a limitation of the current code is that it runs in discrete temperature increments of 0.25. Although lowering this increment is possible, the execution of the script will be difficult and inefficient due to the time and computing power needed.

Energy per Site The number of steps required for the configuration of the lattice reaches equilibrium at a specific temperature can be estimated by plotting graphs. An example is given below for the same temperatures and lattice size for several Monte Carlo steps.

Figure 2 shows that at approximately the 22nd Monte Carlo step, the energy per site has reached a minimum. This is known to be a minimum since the minimum possible value of the energy is -2 (J).

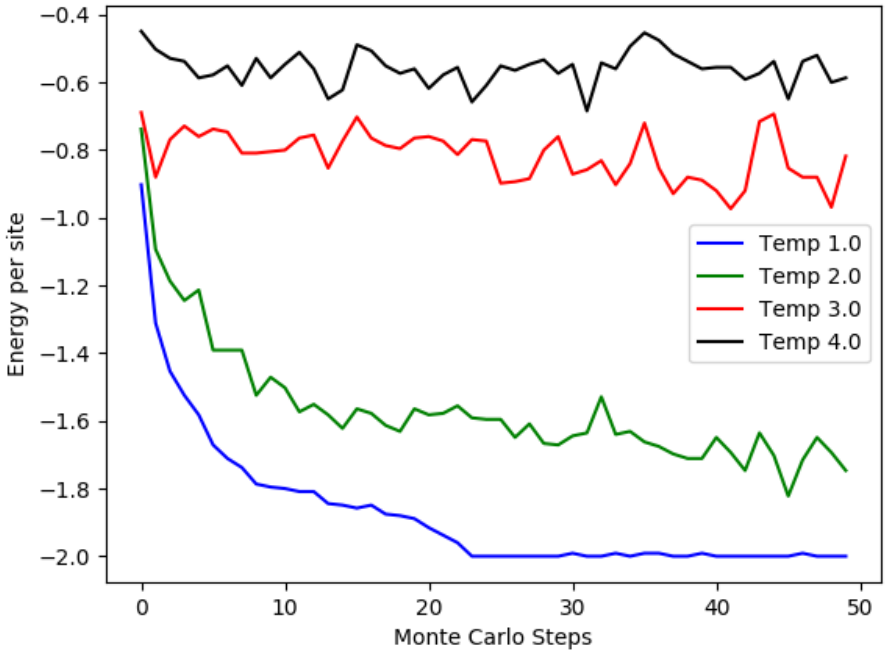

Figure 2: Energy per site against Monte Carlo steps for 50 steps

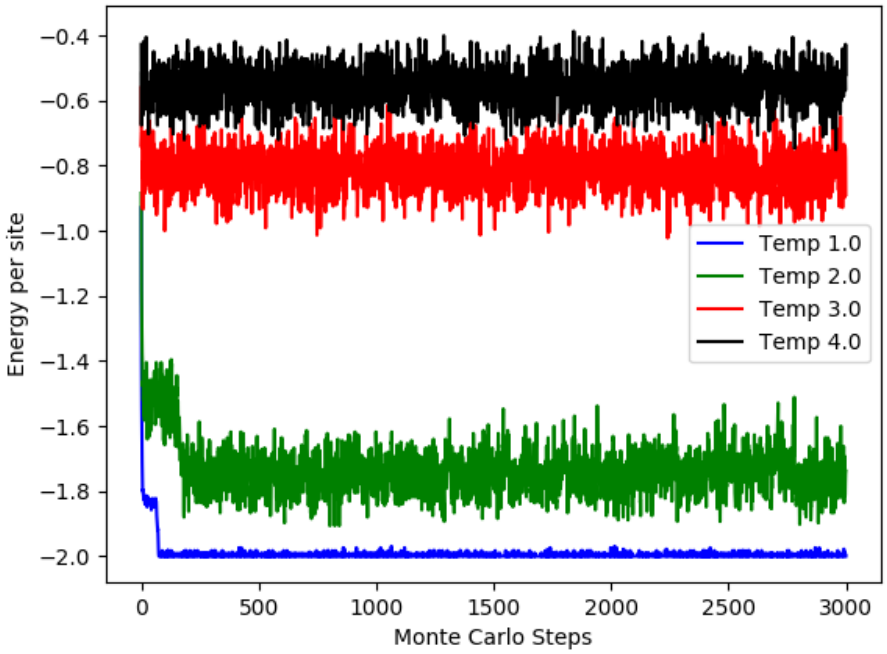

Figure 3: Energy per site against Monte Carlo steps for 3000 steps

Figure 3 shows the same as Figure 2 but at a larger amount of steps. It is expected that for a temperature equal to 2, it would reach equilibrium due to being lower than the critical temperature. However, this does not happen within 3000 Monte Carlo steps. It can be assumed that the number of Monte Carlo steps taken is not high enough and at sufficiently high amounts of steps, the energy per site will reach equilibrium as well. For a temperature of 3 and 4, the energy per site is expected to oscillate regardless of the number of steps due to the temperature being higher than the critical temperature.

4 Conclusion

The code successfully models the Ising model while yielding correct and accurate values. However, the main limitation of the code is its efficiency. This is mainly due to the flipper function which checks for acceptance. The function is executed for each Monte Carlo step, which is executed for each temperature.

5 Constructive Feedback

An extra session/lecture about the Physics part of the theory and material to be covered during the labs would be useful; since it is difficult to create a code when the theory is not fully understood/clear.