Final Project Report PHYS 5319 MM3

2D Ising model using Monte Carlo method

Author:

Shashank Kumbhare

University of Texas at Arlington, TX

1001433909

shashank.kumbhare@mavs.uta.edu

Project Advisor:

Dr. Qiming Zhang

University of Texas at Arlington, TX

zhang@uta.edu

September 2021

Abstract

The goal of this project is to design the Ising model using Monte Carlo simulation method. This report contains the analysis of the problem, the methodology used, the python code used, the results, and discussion.

The Ising model, named after the physicists Ernst Ising and Wilhelm Lenz, is a mathematical model of ferromagnetism in statistical mechanics. The model consists of discrete variables that represent magnetic dipole moments of atomic "spins" that can be in one of two states (+1 or −1). The spins are arranged in a graph, usually a lattice (where the local structure repeats periodically in all directions), allowing each spin to interact with its neighbours. Neighbouring spins that agree have a lower energy than those that disagree; the system tends to the lowest energy, but heat disturbs this tendency, thus creating the possibility of different structural phases. The model allows the identification of phase transitions as a simplified model of reality. The two-dimensional square-lattice Ising model is one of the simplest statistical models to show a phase transition.

This project serves as an introduction to the use of Monte Carlo simulations as a useful way to evaluate the observables of a ferromagnet. Key background is given about Metropolis-Hastings's algorithm.

An Ising model is introduced and used to investigate the properties of a two-dimensional ferromagnet with respect to its magnetization and energy at varying temperatures. The observables are calculated and a phase transition at a critical temperature is also illustrated and evaluated. The results obtained from the simulation and a copy of the code used, written in Python, is enclosed.

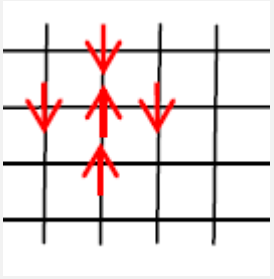
Acknowledgement

Thanks to the course instructor Dr. Qiming Zhang for his wonderful classes and his simplistic explanations. I really appreciate his guidance and help for the completion of this project.

Contents

1 Introduction

The goal of this project is to design the Ising model using Monte Carlo simulation method on a square lattice (2-dimension: L×L) at the no external magnetic field case (H=0). The lattice edge L = 20 with the periodic boundary condition. And only the nearest coupling/interaction is considered.



1.1 Project aims

The motivation for this project is the identification of phase transitions as a simplified model of reality. The aims of this project are as following:

a) Briefly describe the phases (ferromagnetic and paramagnetic) and the phase transition; Microstates and its probability; Statistical average.

b) Ising model: spin and its possible values; the energy of the system; how many microstates for N spins (N= L×L).

c) Monte Carlo method on a Ising model; Important sampling; How to calculate Eflip.

1.2 Approach

a) Python is a well-suited language for scientific programming with clear, easily readable syntax and add-on packages for many computing needs. This project uses the *NumPy* & *SciPy* libraries and the *matplotlib* plotting environment. It is free and operating system independent, making it easily transferable.

b) The important parameter to observe the phase transition of a material is the magnetization (*M*) and energy (*E*) when the material is at equilibrium with the given temperature *T* which are given by,

$$M = N < S >, \qquad \begin{matrix} M > 0 \ \ at \ \ T < T_c \\ M = 0 \ \ at \ \ T < T_c \end{matrix} \tag{1}$$

where,
T: $Temperature$
T_c: $Critical\ temperature$

In general,

$$M \propto (T - T_c)^{\beta} \tag{2}$$

where,
β: critical exponent

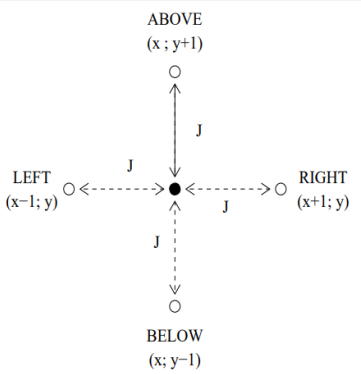
and

$$E\{s_i\} = -\sum_{<ij>}^{N} J_{ij} s_i s_j - H \sum_{i=1}^{N} s_i \tag{3}$$

c) We start with random spins at lattice points with some overall energy of the system say E. We then traverse through each point in the lattice, and we flip (change) the spins and accept the new states by Metropolis algorithm based on the condition on energy change, E_{flip}, where E_{flip} is given by,

$$E_{flip} = E_{final} - E_{initial} \tag{4}$$
$$E_{flip} = 2Js_{ij} \sum_{<nn>} s_{ij}$$

where summation is over nearest neighbours,



All the lattice points would be checked, and spins would flip with metropolis algorithm. Let's call the above a complete 'sweep'.

d) We repeat the above step c) i. e. we perform complete sweeps for several times (say 1000) to reach the equilibrium state where the energy of the system does not change anymore. Then we perform 500 more sweeps each having a net magnetization M at the end of each sweep to get the average net magnetization $\bar{M}$ for those 500 sweeps at that perticular temperature T.

e) We will get net magnetization $\bar{M}$ for a bunch of temperature in some range. Then we will plot $\bar{M}$ vs T to get the critical temperature, T_c, where the phase transition happens.

2 Methodology

A material such as a piece of metal or a glass sample has physical properties such as electrical conductivity, magnetic susceptibility, thermal conductivity, and many others. But how do you determine the properties of an object? How is it possible to calculate the properties of a material composed of billions of billions of microstates?

Strictly speaking, we need to know the energy of each microstate to calculate the properties of a material. In this project, we will focus on energy and magnetic susceptibility. These quantities are those that are interconnected and define a system well because they are fundamental physical properties although energy is the basis of all the relationships we define below. But one question remains: how to simulate the behaviour of a material to calculate its properties?

Let's look at the mathematics and physics that establish the relationships for the different properties we want to evaluate.
The probability of finding the system in a particular microstate with the energy E is given by the canonical distribution:

$$P(E) = \frac{\exp(-E/kT)}{Z} \tag{5}$$

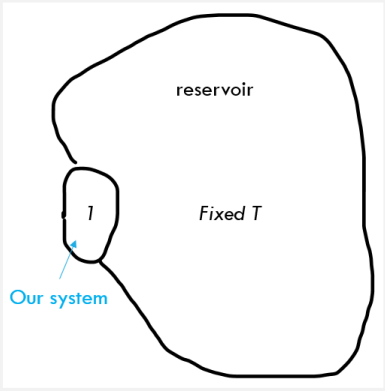
where E is the energy of a configuration of a state of the system given by equation (3).
And Z is the partition function and is given by,

$$Z = \sum_{all\ microstates} \exp(-E/kT) \tag{6}$$

2.1 The Ising model

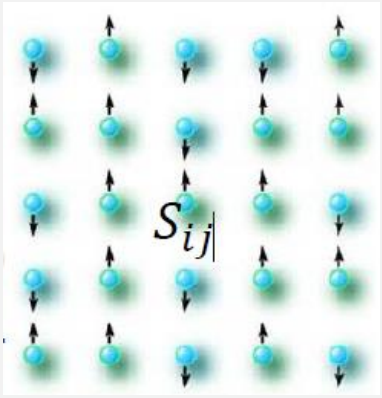
Ferromagnetism arises when a collection of atomic spins aligns such that their associated magnetic moments all point in the same direction, yielding a net magnetic moment which is macroscopic in size. The simplest theoretical description of ferromagnetism is called the Ising model.

Consider *N* atoms in the presence of a *z*-directed magnetic field of strength *H* at temperature T.



Suppose that all atoms are identical spin *-1/2* systems. It follows that either $s_i = +1$ (spin up) or $s_i = -1$(spin down), where s_i is (twice) the *z*-component of the i_{th} atomic spin.

Then the total energy of the system is written by equation (3).



Since we are using no external magnetic field case ($H = 0$), equation (3) becomes,

$$E\{s_i\} = -\sum_{<ij>}^{N} J_{ij} s_i s_j \tag{7}$$

Here, $<i\,j>$ refers to a sum over nearest neighbour pairs of atoms. Furthermore, J is called the exchange energy. Equation (7) is the essence of the Ising model.

And we are using unit $J/k_B = 1$, so our T is in J/k_B and $<E>$ is in J.

Now, we are ready to use Monte Carlo method to simulate our Ising model.

2.2 Monte Carlo method: Metropolis hasting algorithms

In statistics and statistical physics, the Metropolis–Hastings algorithm is a Markov chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution from which direct sampling is difficult. This sequence can be used to approximate the distribution or to compute an integral (e.g. an expected value).

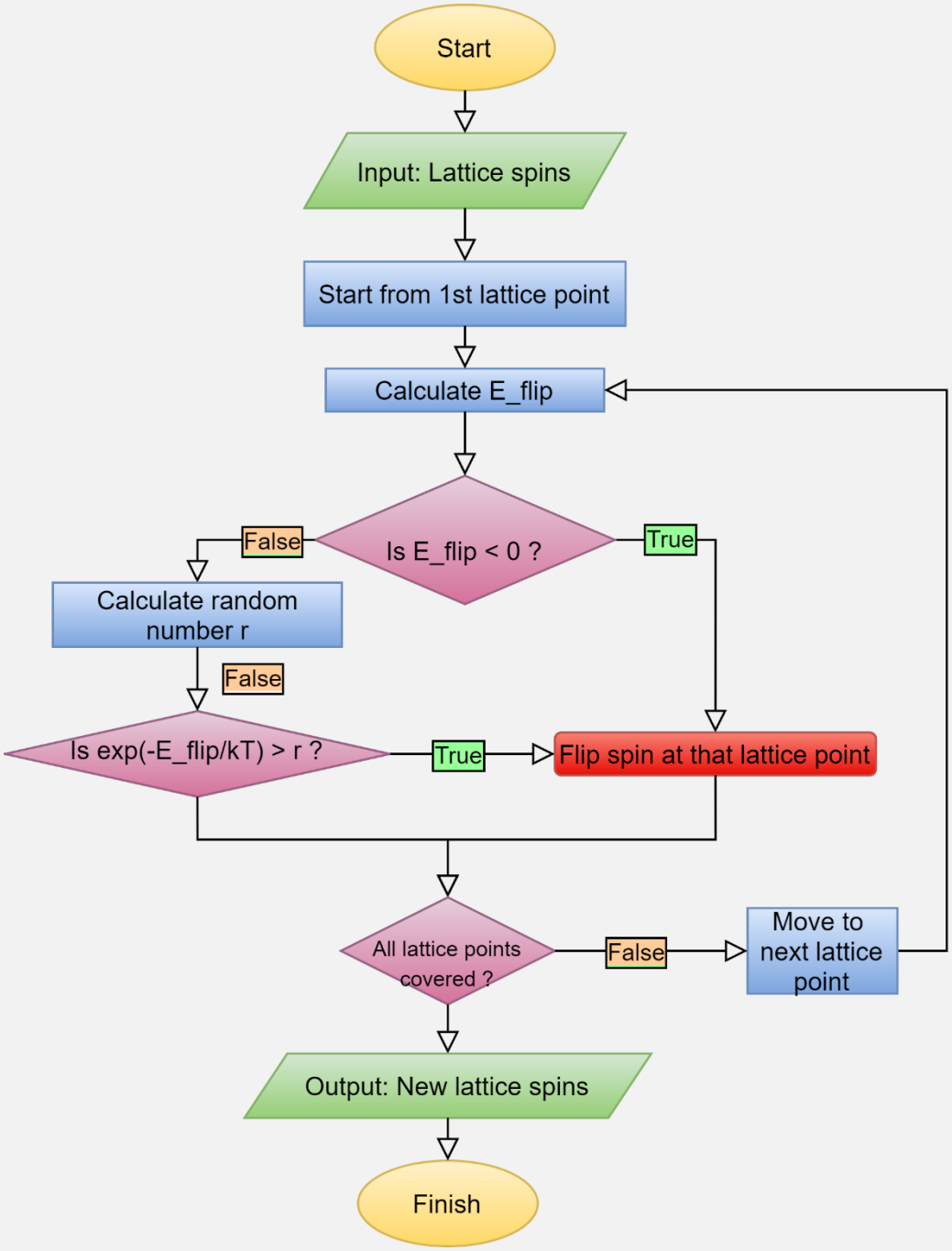
For our project, the Metropolis-algorithm will be as following,

The metropolis algorithm is given by,

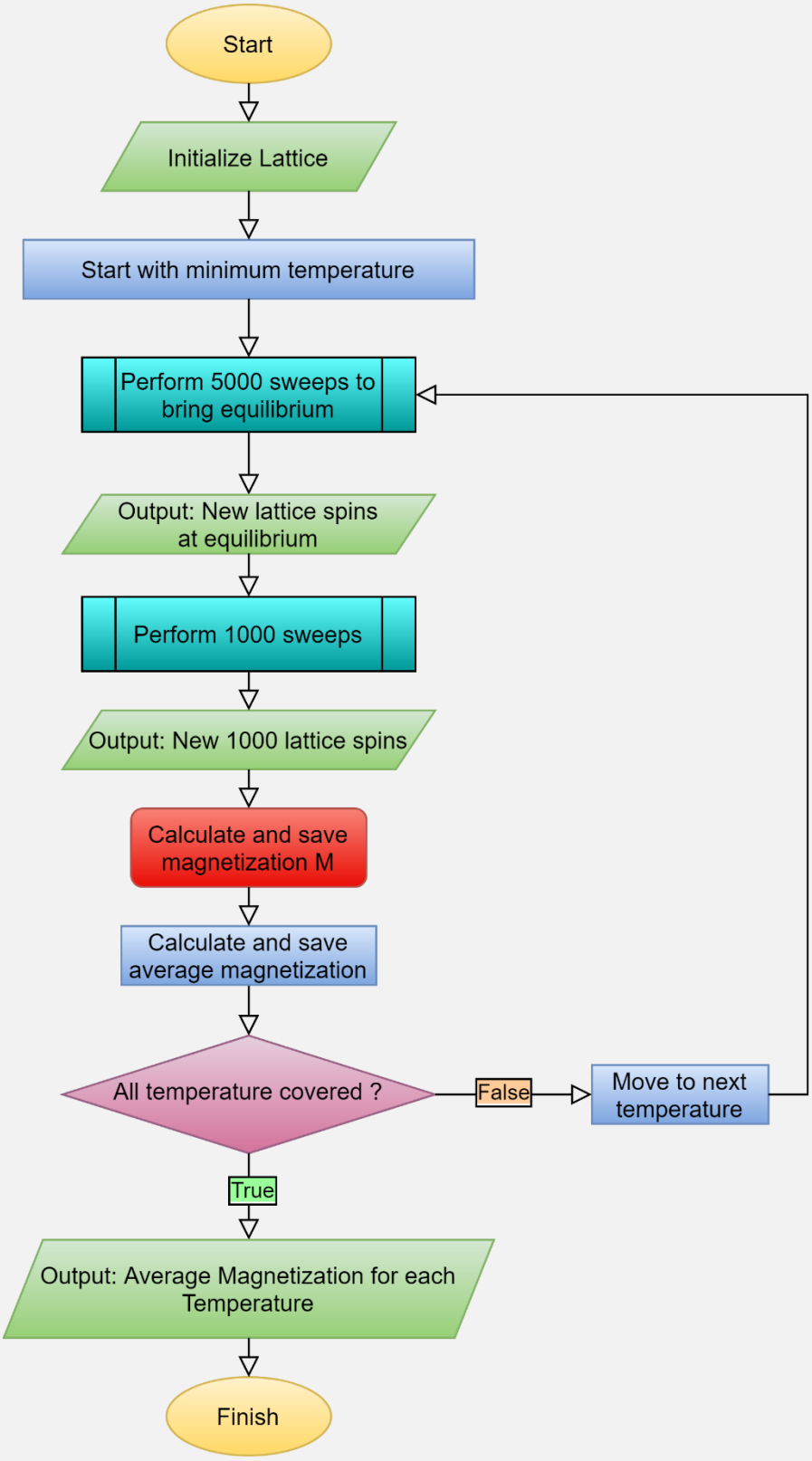
1. Flip the current lattice.
2. If $E_{flip} < 0$, accept the new state.
3. If $E_{flip} > 0$, generate a random number r and $p = e^{-E_{flip}/kT}$
4. a. if $p > r$, accept the new state.
   b. if $p < r$, reject the new state.
5. Move to next lattice point and repeat from step 1 till all the lattice points are covered.

3 Flowchart of the algorithm

3.1 Single Sweep

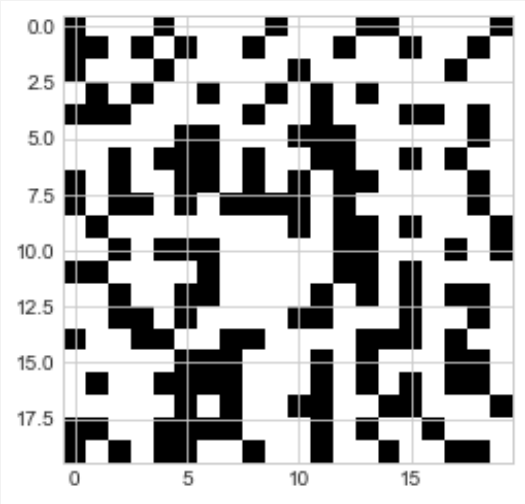


3.2 Ising Model

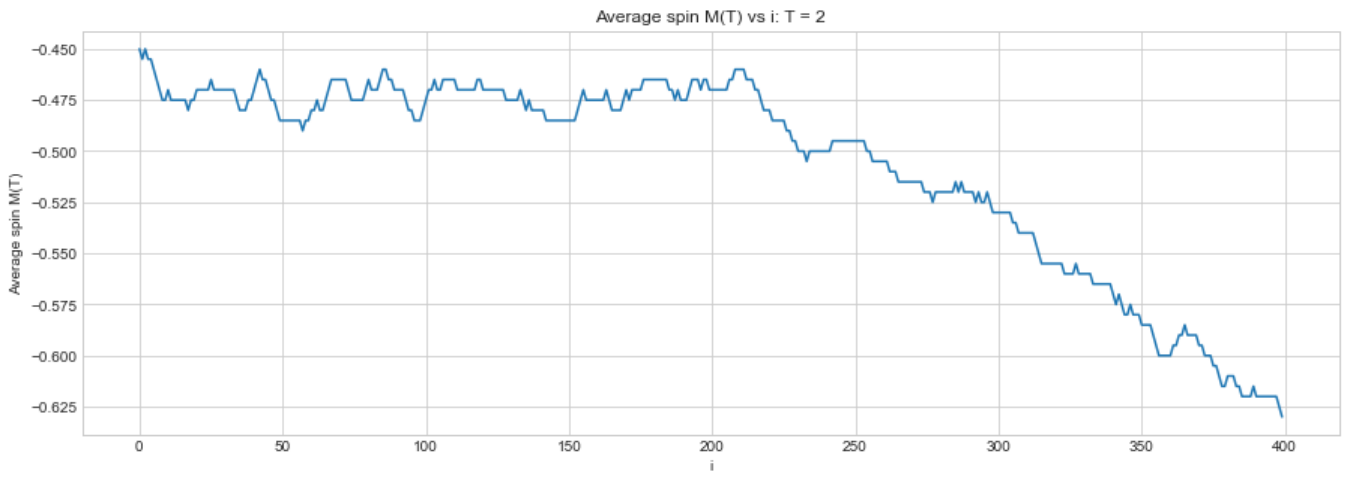


4 Results

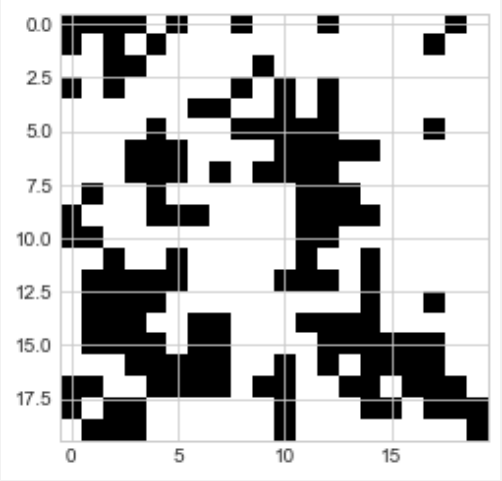
4.1 Initial spins



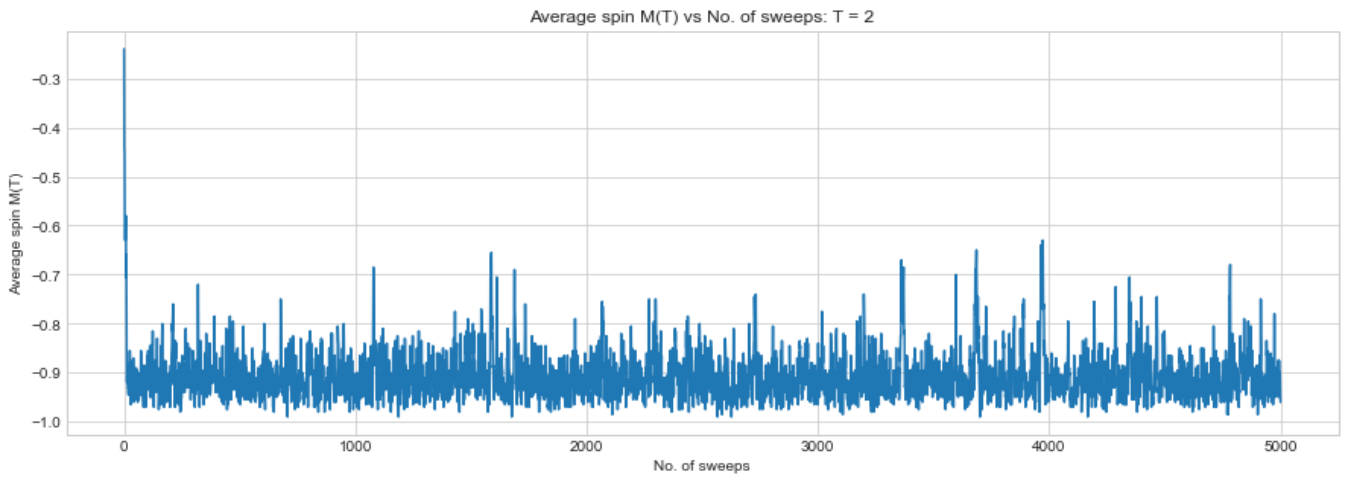
4.2 Example: Single sweep at T = 2



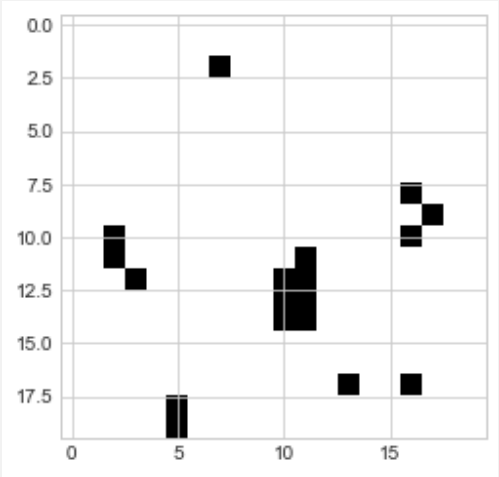
New Spins after single sweep



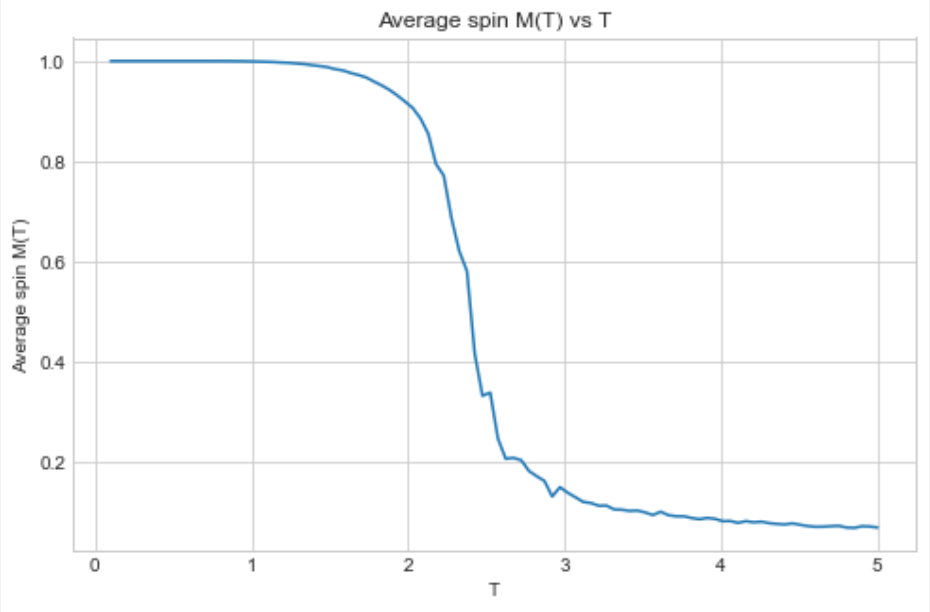
4.3 Example: 5000 sweeps at T = 2 to bring state at equilibrium



Spins at equilibrium



4.4 Average magnetization for T in range (0.1, 5)



5 Discussion

Plot of “Average magnetization $\bar{M}$ vs T” show magnetization versus temperature curves in the absence of an external magnetic field. In all cases, the Monte-Carlo simulation is iterated 1000 times, and the first 5000 iterations are discarded (to allow the system to attain thermal equilibrium). Since there is no external magnetic field, it is irrelevant whether the magnetization, M, is positive or negative. Hence, M is replaced by $|M|$ in all plots.

Note that the $\bar{M}$ versus T curves generated by the Monte-Carlo simulations look very much like those predicted by the mean field model. The major difference is the presence of a magnetization ``tail" for $T > T_c$ in the Monte-Carlo simulations: i.e., in the Monte-Carlo simulations the spontaneous magnetization does not collapse to zero once the critical temperature is exceeded--there is a small lingering magnetization for $T > T_c$.

6 Code

Please see the next page for the code used in this project.

Project 1

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
from scipy.ndimage import convolve, generate_binary_structure
```

In [ ]:
```

```

Problem

Develop a program using Monte Carlo method to simulate the Ising model on a square lattice (2-dimension: L×L) at the no external magnetic field case (H=0). Choose the lattice edge L=20. Use the periodic boundary condition. And only consider the nearest coupling/interaction.

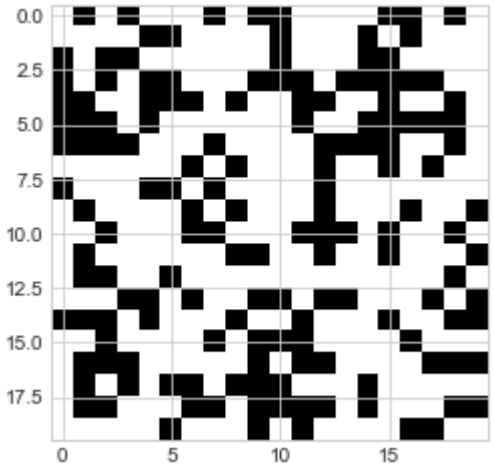
Required parts of the project:

1. Introduction
    - Briefly describe the phases (ferromagnetic and paramagnetic) and the phase transition; Microstates and its probability; Statistical average;
    - Ising model: spin and its possible values; the energy of the system; how many microstates for N spins (N= L×L).
    - Monte Carlo method on a Ising model; Important sampling; How to calculate Eflip.
2. Pseudocode of 'float chart' of your design of the Monte Carlo simulation.
3. Write your own code (in Matlab or python). Attach your code.
4. Calculate the magnetization M (= <s>) and the total energy <E> for 50 different temperature steps from T=0.0 to T=5.0 (or a range your can observe a phase transition). Plot them against T's. Discuss the physics from your results.
   Use the unit J/kB=1 (so your T is in J/kB and <E> is in J).
   Estimate the Curry temperature Tc.

Solution

In [6]:
```
##############################################################################
######################### get_rand_lattice ###################################
def get_spin_lattice( size_lattice = (20, 20),
                      prob_spin_up = 0.5,
                      plot_enabled = True ):

    """
    General info:
        This function returns a random 2D-numpy-array of spins of given size
        <size_lattice>.
    Arguments:
        size_lattice : 2D lattice size
    """

    pts_rand_2D                    = np.random.random(size_lattice)
    spins                          = np.zeros(size_lattice)
    spins[pts_rand_2D <  prob_spin_up] = +1
    spins[pts_rand_2D >= prob_spin_up] = -1
    if plot_enabled: plt.imshow(spins)

    return spins
######################### get_rand_lattice ###################################
##############################################################################
```

In [11]:
```
spins = get_spin_lattice( size_lattice = [20, 20],
                          prob_spin_up = 0.4 )
```



In [ ]:
```

```

In [12]:
```
##########################################################################################
######################### get_E_total ####################################################
def get_E_total(spins):

    """
    General info:
        This function calculates energy of the lattice spins for ising model.
    Arguments:
        spins : input of which we have to calculate energy
    """

    spins  = np.array(spins)
    n_rows = np.shape(spins)[0]
    n_cols = np.shape(spins)[1]

    # Loop over lattice points to calculate energy >>
    E = 0
    for i in range(n_rows):
        for j in range(n_cols):
            i_u = i - 1 if not(i == 0)          else n_rows - 1     # << index of row up
            i_d = i + 1 if not(i == n_rows - 1) else 0              # << index of row do
            j_l = j - 1 if not(j == 0)          else n_cols - 1     # << index of col le
            j_r = j + 1 if not(j == n_cols - 1) else 0              # << index of col ri
            E   = E - spins[i, j] * (spins[i_u, j] + spins[i_d, j] + spins[i, j_l] + spi
    return E
######################### get_E_total ####################################################
##########################################################################################
```

In [18]:
```
E_total = get_E_total(spins)
E_total
```

Out[18]: -64.0

In [ ]:
```

```

In [19]:
```
##########################################################################################
######################### get_E_flip #####################################################
def get_E_flip( spins    = [[1, -1, 1], [-1, 1, -1], [1, -1, 1]],
                loc_spin = [1, 1] ):

    """
    General info:
        This function calculates change in energy (E_flip) of a lattice system
        due to flip of one of the lattice point spin.
    Arguments:
        spins    : 2D array/list of spins
        loc_spin : location of the spin to be flipped
    """

    spins  = np.array(spins)
    n_rows = np.shape(spins)[0]
    n_cols = np.shape(spins)[1]

    i = loc_spin[0]
    j = loc_spin[1]

    # Neighbour points >>
    i_u = i - 1 if not(i == 0)          else n_rows - 1     # << index of row up
    i_d = i + 1 if not(i == n_rows - 1) else 0              # << index of row down
    j_l = j - 1 if not(j == 0)          else n_cols - 1     # << index of col left
    j_r = j + 1 if not(j == n_cols - 1) else 0              # << index of col right

    # Calculating E_flip >>
    J = 1
    E_flip = 2 * J * spins[i, j] * (spins[i_u, j] + spins[i_d, j] + spins[i, j_l] + spin

    return E_flip
######################### get_E_flip #####################################################
##########################################################################################
```

In [21]:
```
get_E_flip( spins    = spins,
            loc_spin = [18, 18] )
```

Out[21]: -4.0

In [ ]:
```
```

In [23]:

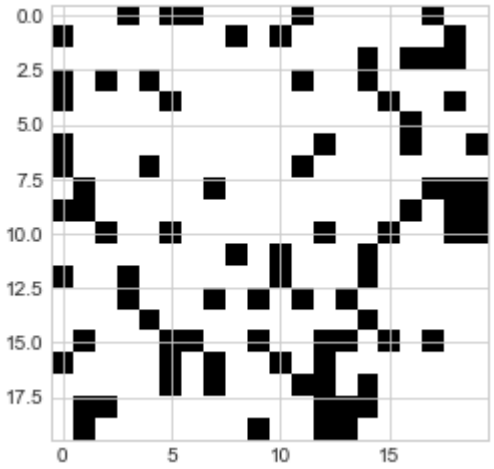
```
##############################################################################
######################### Important "String" constants #######################
str_spin_average = "Average spin M(T)"
str_temp         = "T = {T}"
######################### Important "String" constants #######################
##############################################################################
```

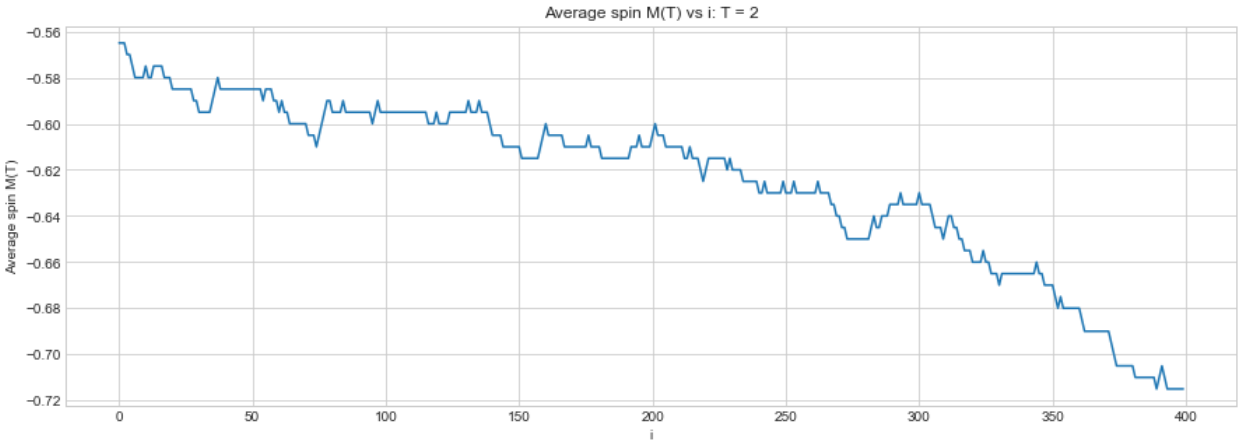
In [24]:

```
##############################################################################
######################### sweep_single #######################################
def sweep_single( spins = [[-1,  1, -1],
                           [-1, -1, -1],
                           [ 1, -1,  1]],
                  T     = 1,
                  plot_enabled = True ):

    """
    General info:
        This function performs a single sweep across all the latice points with
        the periodic boundary to bring the system in equilibrium with heat bath.
    Arguments:
        spins : 2D array/list of spins
        T     : Temperature of the system
    """

    spins = np.array(spins)

    n_rows = np.shape(spins)[0]
    n_cols = np.shape(spins)[1]

    if plot_enabled: spins_collection = [np.nan] * n_rows * n_cols

    # Loop for flipping spins over all lattice points >>
    spins_new = spins.copy()
    m = 0
    for i in range(n_rows):
        for j in range(n_cols):
            E_flip = get_E_flip( spins    = spins_new,
                                 loc_spin = [i, j] )
            if E_flip <= 0 or np.exp(-E_flip/T) > np.random.uniform():
                spins_new[i, j] = - spins_new[i, j]

            if plot_enabled: spins_collection[m] = spins_new.copy()
            m = m + 1

    # Plotting "i vs M" >>
    if plot_enabled:
        fig  = plt.figure(figsize = (15, 5))
        axes = plt.gca()
        axes.plot( range(n_rows*n_cols), np.average( np.average(spins_collection, 1), 1
        # Setting plot elements >>
        axes.set_title(f"{str_spin_average} vs i: T = {T}")
        axes.set_xlabel("i")
        axes.set_ylabel(str_spin_average)
        plt.show()
```

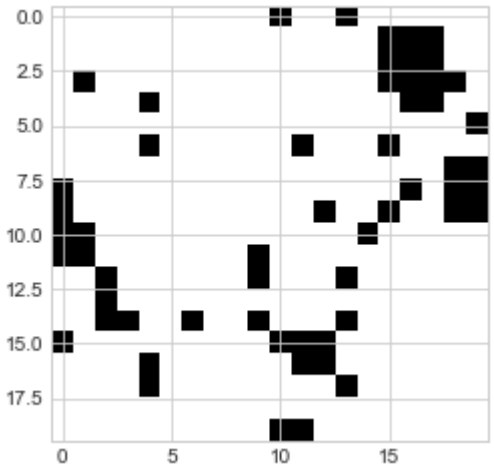
```
    E_total_new = 0

    return spins_new, E_total_new
######################### sweep_single ##########################################
#################################################################################
```

In [25]:
```
spins_old   = get_spin_lattice( size_lattice = [20, 20],
                                 prob_spin_up = 0.25 )
E_total_old = get_E_total(spins_old)

spins_new, E_total_new = sweep_single( spins = spins_old,
                                       T     = 2 )


plt.imshow(spins_new)
```





Out[25]: <matplotlib.image.AxesImage at 0x1945544cfa0>



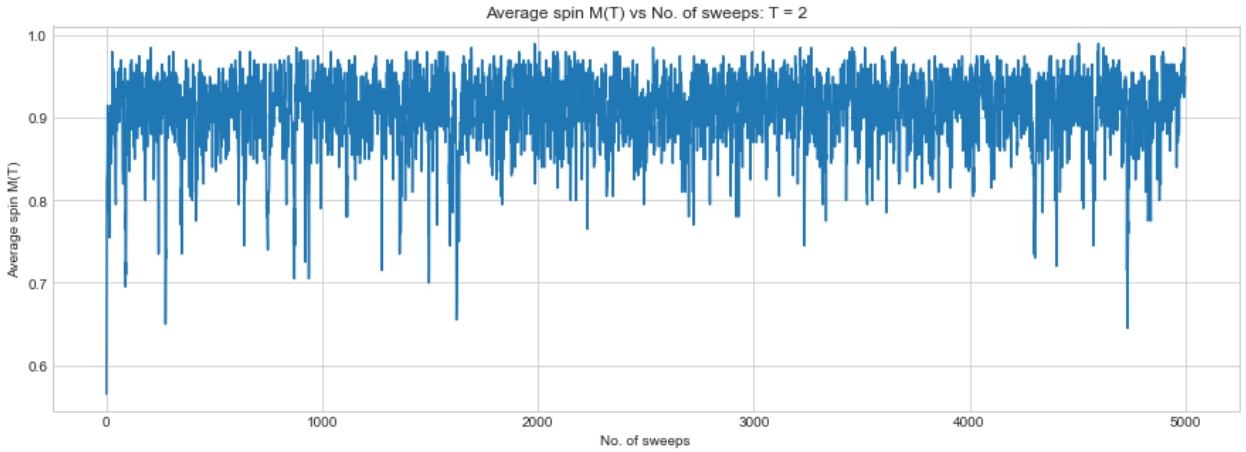
In [27]:

```
################################################################################
######################### sweep_N ##############################################
def sweep_N( spins = [[1, -1, 1], [-1, 1, -1], [1, -1, 1]],
             n     = 1000,
             T     = 1,
             pbar_enabled = True,
             plot_enabled = True ):

    """
    General info:
        This function performs 'n' no. of sweeps across all the latice points with
        the periodic boundary to bring the system in equilibrium with heat bath.
    Arguments:
        spins : 2D array/list of spins
        n     : No. of sweeps
        T     : Temperature of the system
    """

    from progressbar import ProgressBar

    spins     = np.array(spins)
    spins_new = [np.nan] * n

    # Loop for 'n' no. of sweeps >>
    spins_new[0] = spins.copy()
    pbar         = ProgressBar() if pbar_enabled else lambda x: x
    for i in pbar(range(1, n)):
        spins_new[i], E_new = sweep_single( spins = spins_new[i-1].copy(),
                                            T     = T,
                                            plot_enabled = False
                                          )
    spins_new = np.array(spins_new)

    # Plotting "i vs M" >>
    if plot_enabled:
        fig  = plt.figure(figsize = (15, 5))
        axes = plt.gca()
        axes.plot( range(n), np.average( abs(np.average(spins_new, 1)), 1 ) )
        # axes.plot( range(n), np.average( np.average(spins_new, 1), 1 ) )
                            # np.average( [ abs(np.average(spins)) for spins in spins_n
        # Setting plot elements >>
        axes.set_title(f"{str_spin_average} vs No. of sweeps: T = {T}")
        axes.set_xlabel("No. of sweeps")
        axes.set_ylabel(str_spin_average)
        plt.show()

    return spins_new, E_new
######################### sweep_N ##############################################
```

```
##############################################################################
```

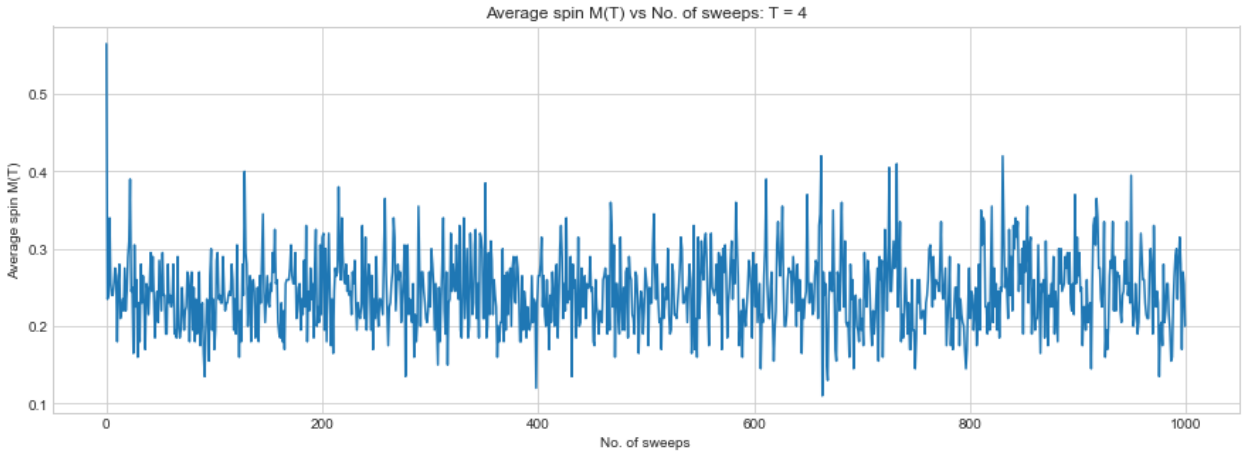
In [29]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 5000,
                                  T     = 2 )
```

```
100% |################################################################################|
```



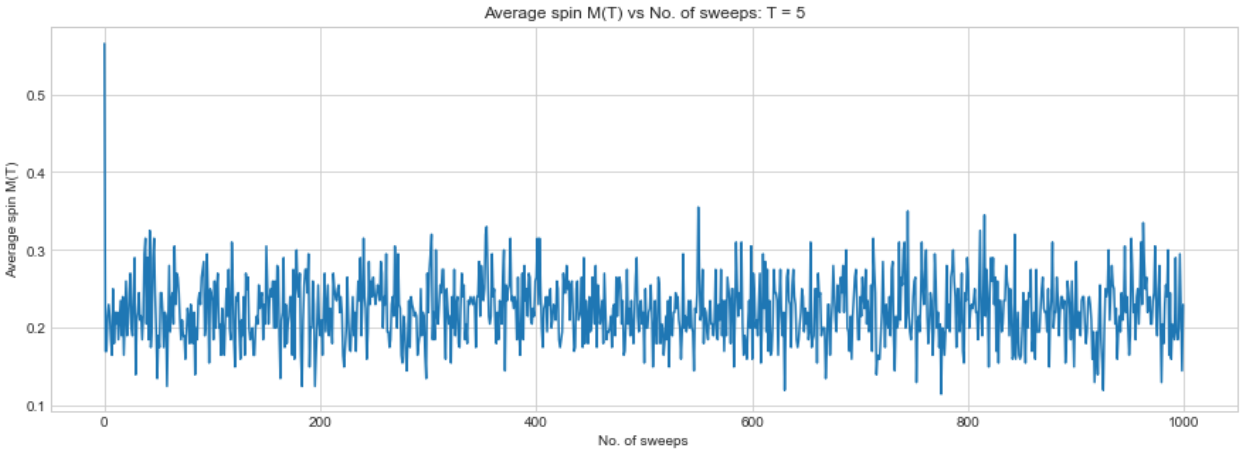
In [31]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 1000,
                                  T     = 4 )
```

```
100% |################################################################################|
```



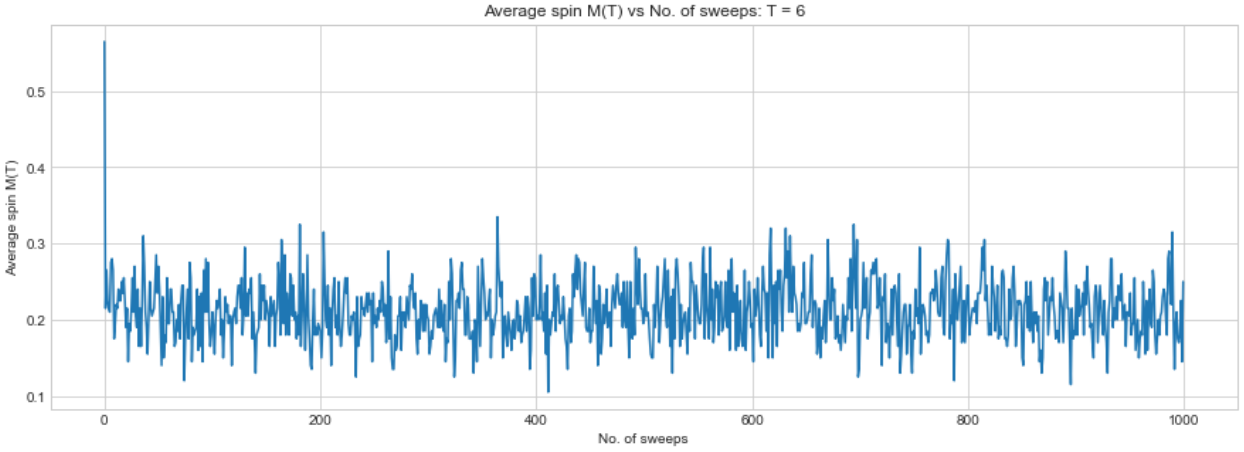
In [32]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 1000,
                                  T     = 5 )
```

```
100% |################################################################################|
```



In [33]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 1000,
                                  T     = 6 )
```

```
100% |################################################################################|
```



In [ ]:

In [35]:

```
##########################################################################################
######################### ising_model_2D #################################################
def ising_model_2D(  size_lattice   = (10, 10),
                     prob_spin_up   = 0.5,
                     period_burnout = 1000,
                     temp_range     = (0.1, 5),
                     n_temp_steps   = 50,
                     N              = 1000 ):

    """
    General info:
        #$^#$%$#%#$%#$%#$%#$%@!#@!!^%*#%^(#^#$^)
    Arguments:
        size_lattice   : 2D lattice size
        period_burnout : Burnout period (no. of sweeps) to reach equilibrium state
        temp_range     : Temperature range
        N              : No. of sweeps for the simulation
    """

    from progressbar import ProgressBar

    spins_old = get_spin_lattice(   size_lattice = size_lattice,
                                    prob_spin_up = 0.5,
                                    plot_enabled = False )

    Ts     = np.linspace(temp_range[0], temp_range[1], num = n_temp_steps)
    len_Ts = len(Ts)
    M_avg  = [np.nan] * len_Ts
    E_avg  = [np.nan] * len_Ts

    # Loop for temperatures >>
    pbar = ProgressBar()
    j    = -1
    for T in pbar(Ts):
        j = j + 1
        # Burnout runs >>
        spins_eqib, _ = sweep_N( spins = spins_old,
                                 n     = period_burnout,
                                 T     = T,
                                 pbar_enabled = False,
                                 plot_enabled = False )
        # Real N sweeps >>
        spins_new, E_total_new = sweep_N( spins = spins_eqib[-1],
                                          n     = N,
                                          T     = T,
                                          pbar_enabled = False,
                                          plot_enabled = False )
        Ms       = [ abs(np.average(spins)) for spins in spins_new ]
```

```
        # Es       = E_total_new
        Es       = [ get_E_total(spins) for spins in spins_new ]

        # M_avg[j] = np.average(Ms)
        M_avg[j] = abs(np.average(Ms))
        E_avg[j] = np.average(Es)

    return Ts, Ms, Es, M_avg, E_avg, spins_new
######################### ising_model_2D ########################################
#################################################################################
```

In [36]:

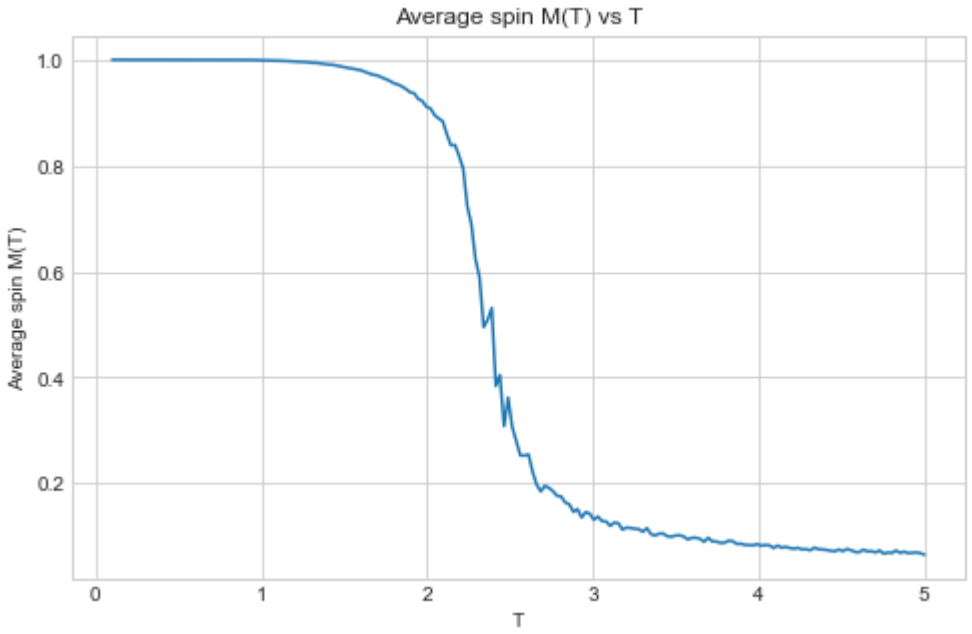
```
#################################################################################
######################### plot_results ##########################################
def plot_results(x, y, figsize = (8, 5) ):

    import matplotlib.pyplot as plt

    # Plotting Results >>
    fig  = plt.figure(figsize = figsize)
    axes = plt.gca()
    axes.plot(x, y)
    # Setting plot elements >>
    axes.set_title(f"{str_spin_average} vs T")
    axes.set_xlabel("T")
    axes.set_ylabel(str_spin_average)
    plt.show()

    return None
######################### plot_results ##########################################
#################################################################################
```

In [37]:
```
################################################################################
######################### plot_energy ##########################################
def plot_energy(x, y, figsize = (8, 5) ):

    import matplotlib.pyplot as plt

    # energies = [ get_E_total(spin) for spin in spins ]

    # Plotting Results >>
    fig  = plt.figure(figsize = figsize)
    axes = plt.gca()
    axes.plot(x, y)
    # Setting plot elements >>
    str_energy = "Energy"
    axes.set_title(f"{str_energy} vs T")
    axes.set_xlabel("T")
    axes.set_ylabel(str_energy)
    plt.show()

    return None
######################### plot_energy ##########################################
################################################################################
```

In [58]:
```
Ts, Ms, Es, M_avg, E_avg, spins_new = ising_model_2D(   size_lattice   = (20, 20),
                                                        prob_spin_up   = 0.5,
                                                        period_burnout = 5000,
                                                        temp_range     = (0.1, 5),
                                                        n_temp_steps   = 100,
                                                        N              = 1000 )
```

```
100% |################################################################################|
```

In [99]:
```
plot_results(Ts, M_avg, figsize = (8, 5) )
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: