# 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.

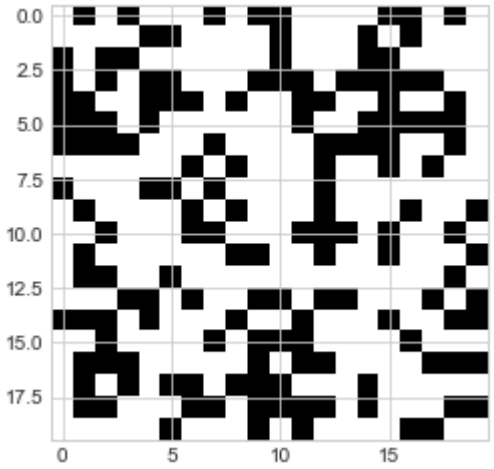
**Required parts of the project:**

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]:

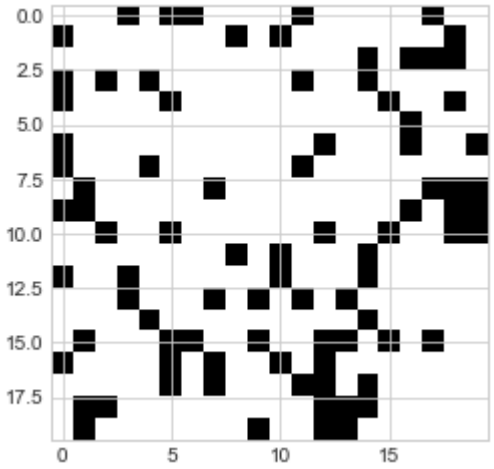
```
##############################################################################
######################### Important "String" constants #######################
str_spin_average = "Average spin M(T)"
str_temp         = "T = {T}"
######################### Important "String" constants #######################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

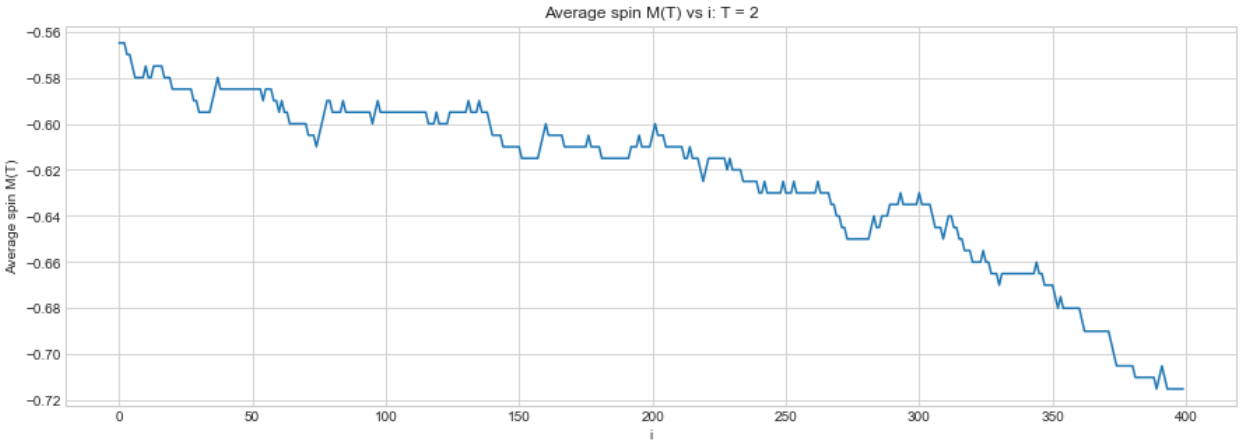
In [24]:

```
##############################################################################
######################### 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()
```

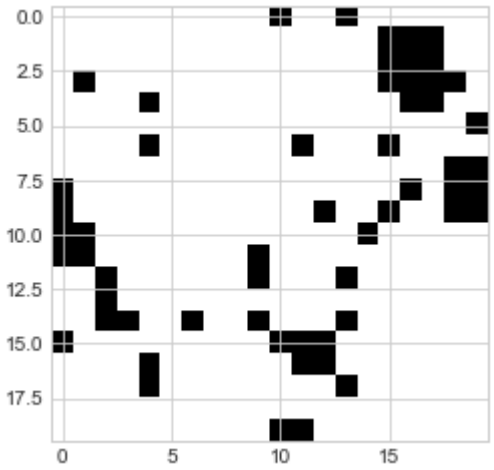
```
    E_total_new = 0

    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>



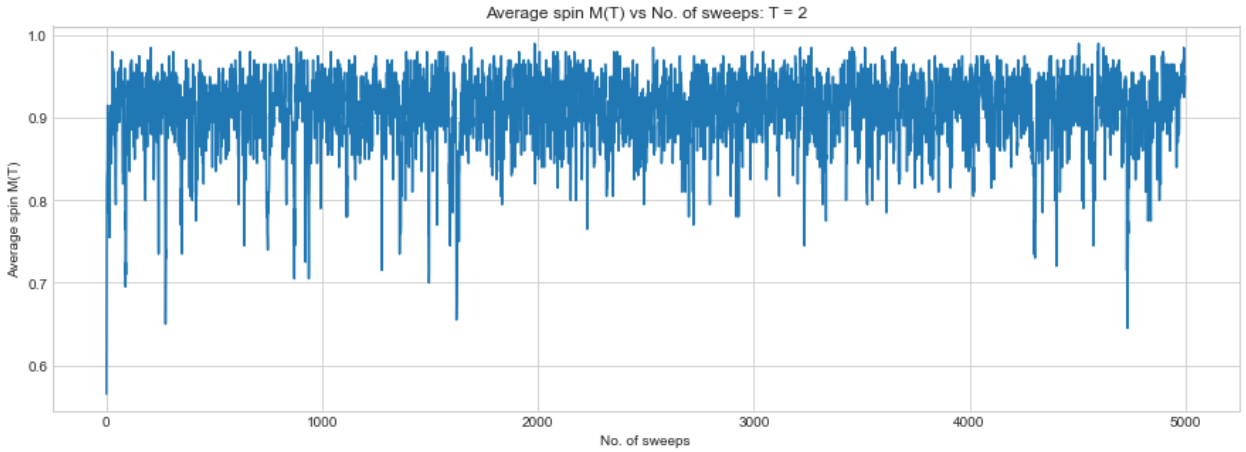
In [27]:

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

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

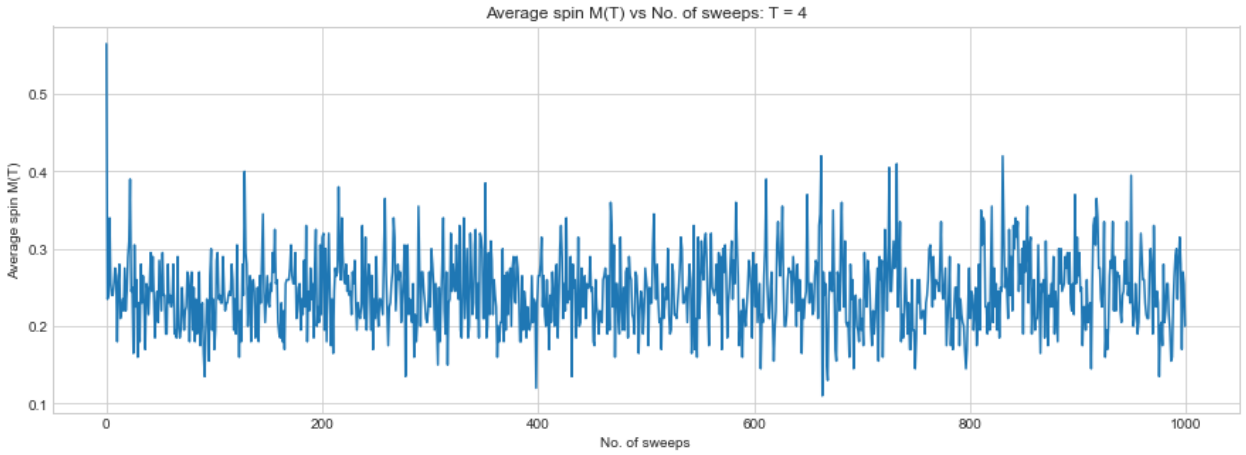
In [29]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 5000,
                                  T     = 2 )
```

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



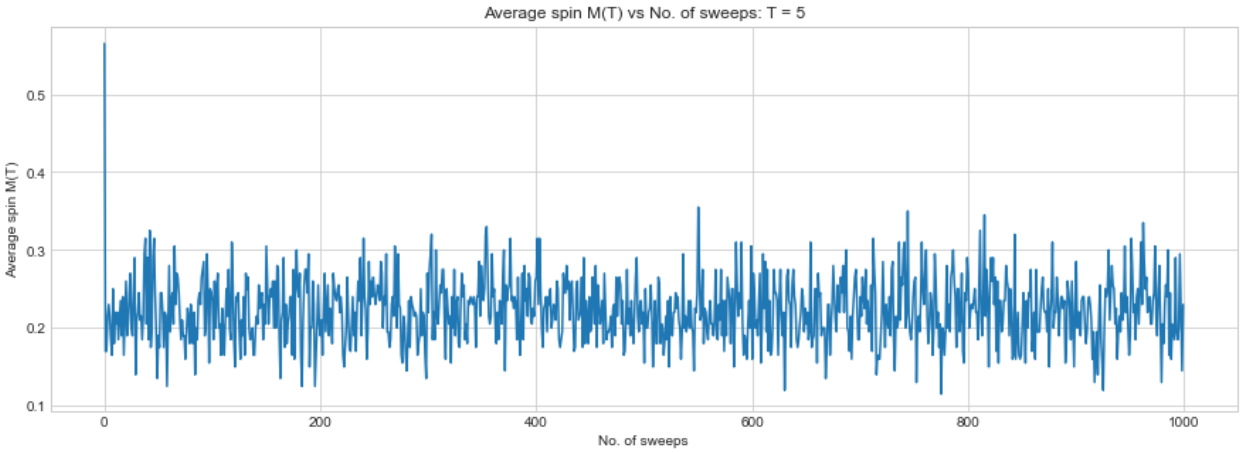

In [31]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 1000,
                                  T     = 4 )
```

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



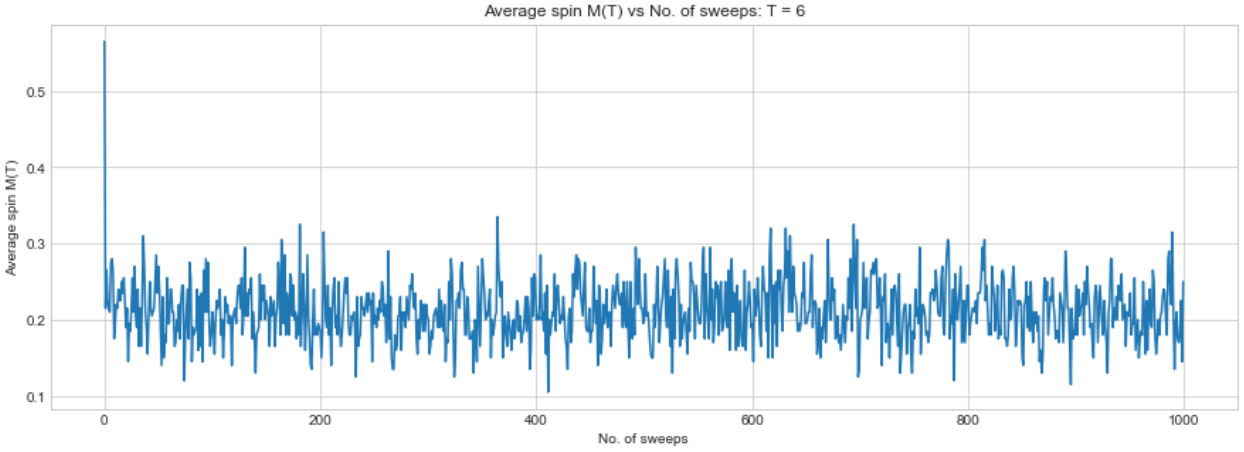
In [32]:
```
spins_new, E_total_new = sweep_N( spins = spins_old,
                                  n     = 1000,
                                  T     = 5 )
```

```
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]:

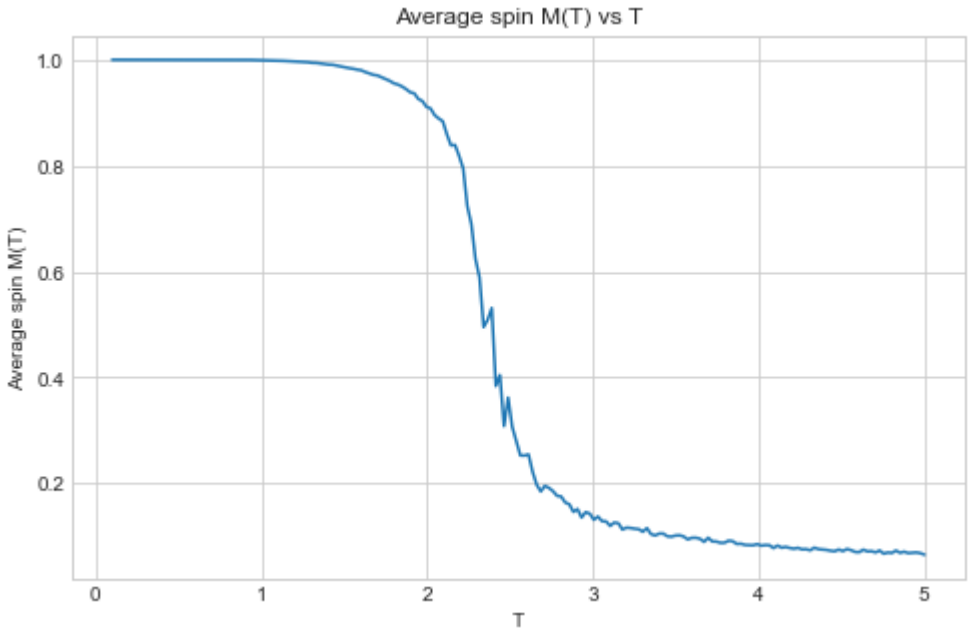
```
#################################################################################
######################### plot_results ##########################################
def plot_results(x, y, figsize = (8, 5) ):

    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 [ ]: