2d Ising model

May 18, 2022

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import numba
from numba import njit
from scipy.ndimage import convolve, generate_binary_structure

N =64
init_random = np.random.random((N,N))
lattice_n = np.zeros((N, N))
lattice_n[init_random>=0.3] = 1
lattice_n[init_random<0.3] = -1

plt.imshow(lattice_n)
plt.title('initial configuration')
```

Metropolis algorithm for the two-dimensional Ising model

1.choose a random 64*64 spin configuration {S}^1

2.flip random one site's spin, and get a new configuration {s}_bar

3.calculate the alpha = w({S}^1)/w({S}_bar) = exp(-beta* delta(E)), where delta(E) is the energy diffence of two configuration

4.choose a random number gamma between [0,1], and compare gamma with alpha, if appha > gamma, {S}^2 = {S}_bar, otherwise {S}^2 = {S}^1

5.iterate the following procedure for 10000 times

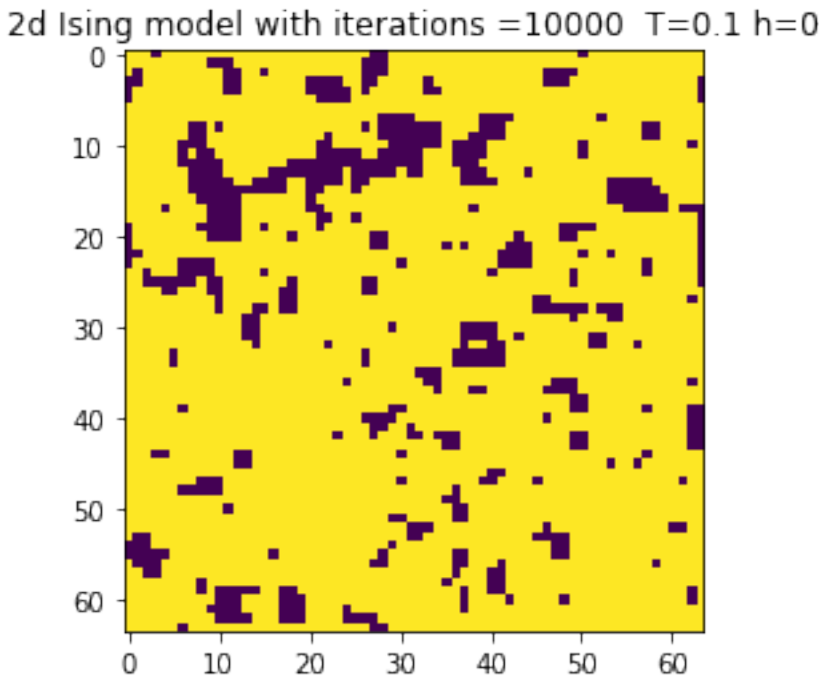
6.calculate average magnetic field , m = sum of (si)/L/N/N

[2]:
```
def metropolis(spin_arr, times, T , h):
    BJ=1/T
    N = len(spin_arr)
    spin_arr = spin_arr.copy()
    net_spins = np.zeros(times-1)
    for t in range(0,times-1):
        # 2. pick random point on array and flip spin
        x = np.random.randint(0,N)
        y = np.random.randint(0,N)
        spin_i = spin_arr[x,y] #initial spin
```

```python
        spin_f = spin_i*-1 #proposed spin flip

        # compute change in energy
        E_i = 0
        E_f = 0
        E_i += -spin_i*spin_arr[(x-1+N)%N,y]
        E_f += -spin_f*spin_arr[(x-1+N)%N,y]
        E_i += -spin_i*spin_arr[(x+1+N)%N,y]
        E_f += -spin_f*spin_arr[(x+1+N)%N,y]
        E_i += -spin_i*spin_arr[x,(y-1+N)%N]
        E_f += -spin_f*spin_arr[x,(y-1+N)%N]
        E_i += -spin_i*spin_arr[x,(y+1+N)%N]
        E_f += -spin_f*spin_arr[x,(y+1+N)%N]
        E_i += -spin_i*h
        E_f += -spin_f*h

        # 3 / 4. change state with designated probabilities
        dE = E_f-E_i
        if dE<=0:
            spin_arr[x,y]=spin_f
        elif (dE>0)*(np.random.random() < np.exp(-BJ*dE)):
            spin_arr[x,y]=spin_f

        net_spins[t] = spin_arr.sum()

    return net_spins, spin_arr
```

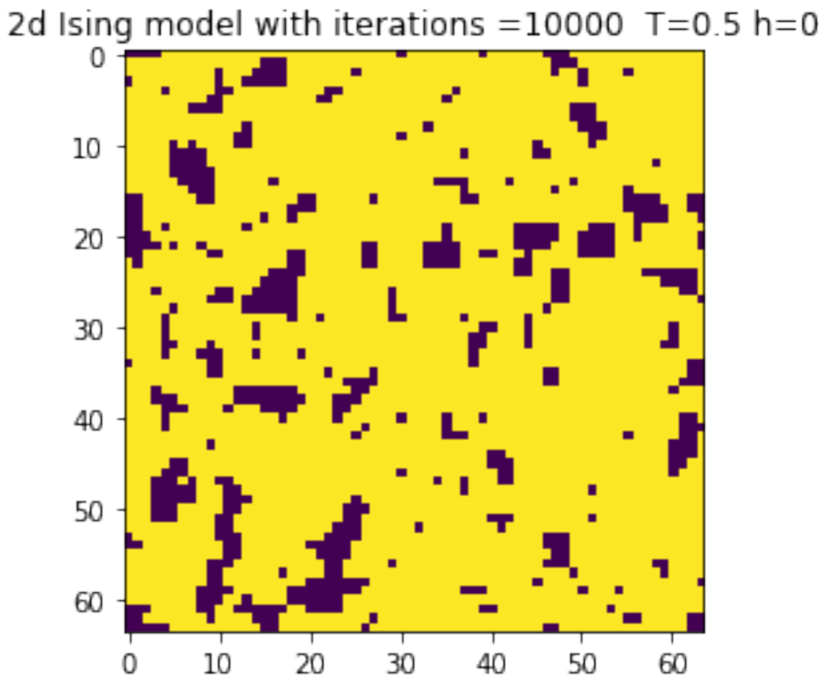
[3]:
```python
times = 10000
h=0
T = [0.1,0.5,2.0,10.0]

spins0 , spin_arr0 = metropolis(lattice_n, times, T[0],h)
plt.imshow(spin_arr0)
plt.title('2d Ising model with iterations ='+str(times)+'  T='+str(T[0])+'␣
 ↪h='+str(h))
spins0.sum()/times/N/N
```

[3]: 0.570125244140625



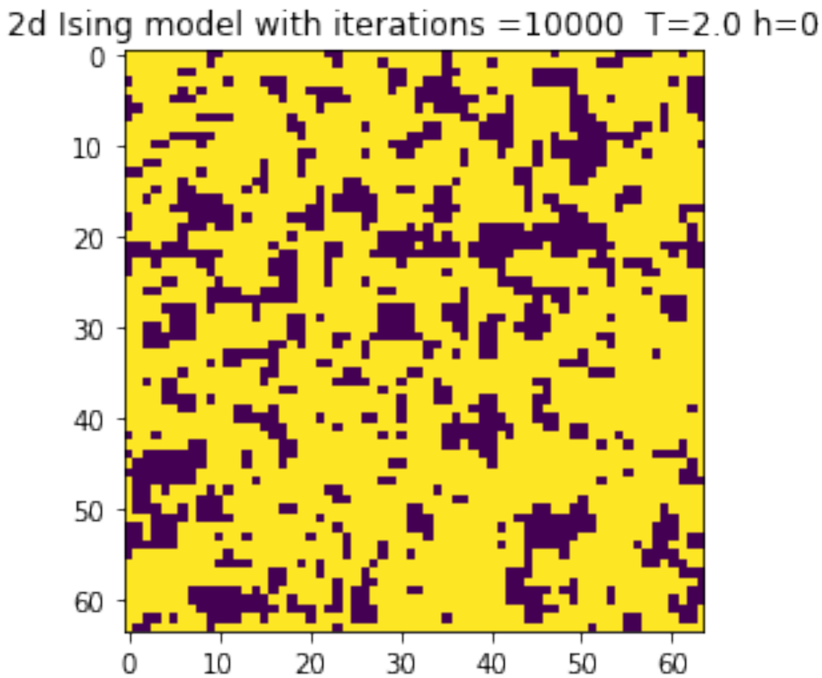
[4]:
```
spins1 , spin_arr1 = metropolis(lattice_n, times, T[1],0)
plt.imshow(spin_arr1)
plt.title('2d Ising model with iterations ='+str(times)+'  T='+str(T[1])+'␣
 ↪h='+str(h))
```

[4]:
```
Text(0.5, 1.0, '2d Ising model with iterations =10000  T=0.5 h=0')
```



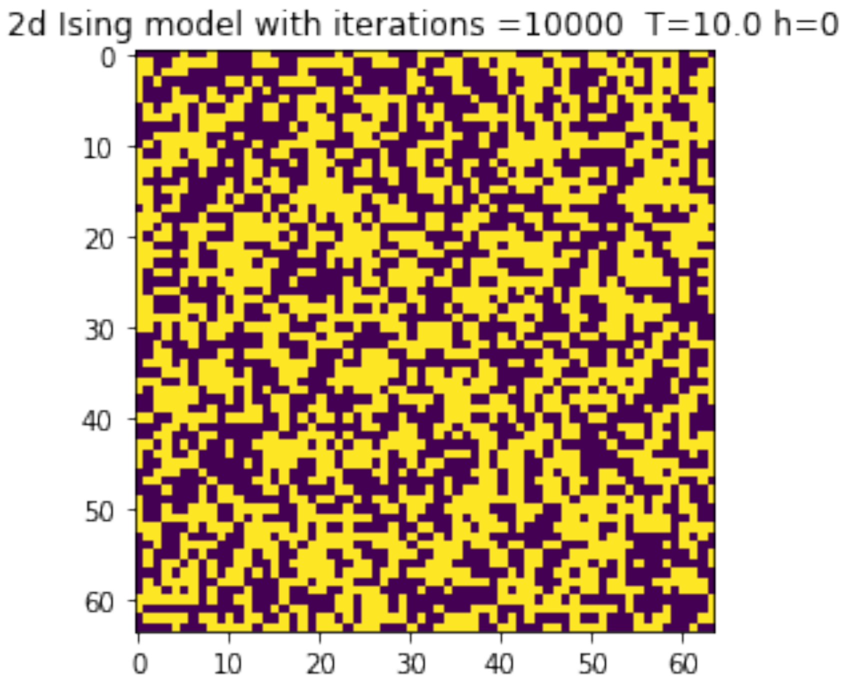
[5]:
```
spins2 , spin_arr2 = metropolis(lattice_n, times, T[2],0)
plt.imshow(spin_arr2)
plt.title('2d Ising model with iterations ='+str(times)+'  T='+str(T[2])+'␣
 ↪h='+str(h))
```

[5]:
```
Text(0.5, 1.0, '2d Ising model with iterations =10000  T=2.0 h=0')
```



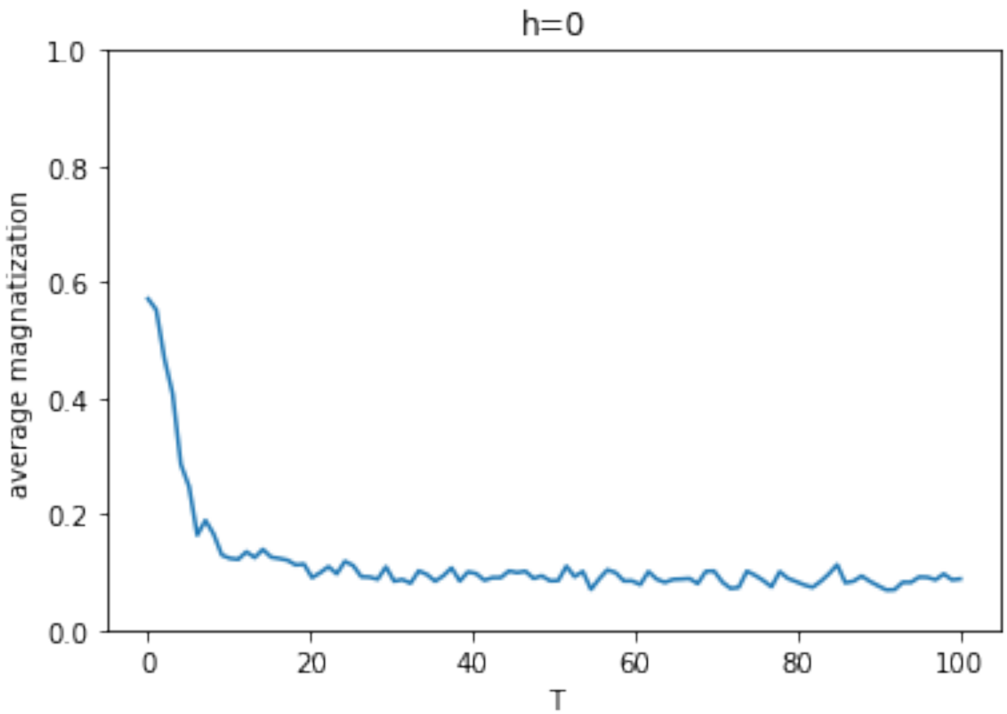

```
[6]: spins3 , spin_arr3 = metropolis(lattice_n, times, T[3],0)
     plt.imshow(spin_arr3)
     plt.title('2d Ising model with iterations ='+str(times)+'  T='+str(T[3])+'␣
      ↪h='+str(h))
     spins3.sum()/times/N/N
```

```
[6]: 0.139030517578125
```



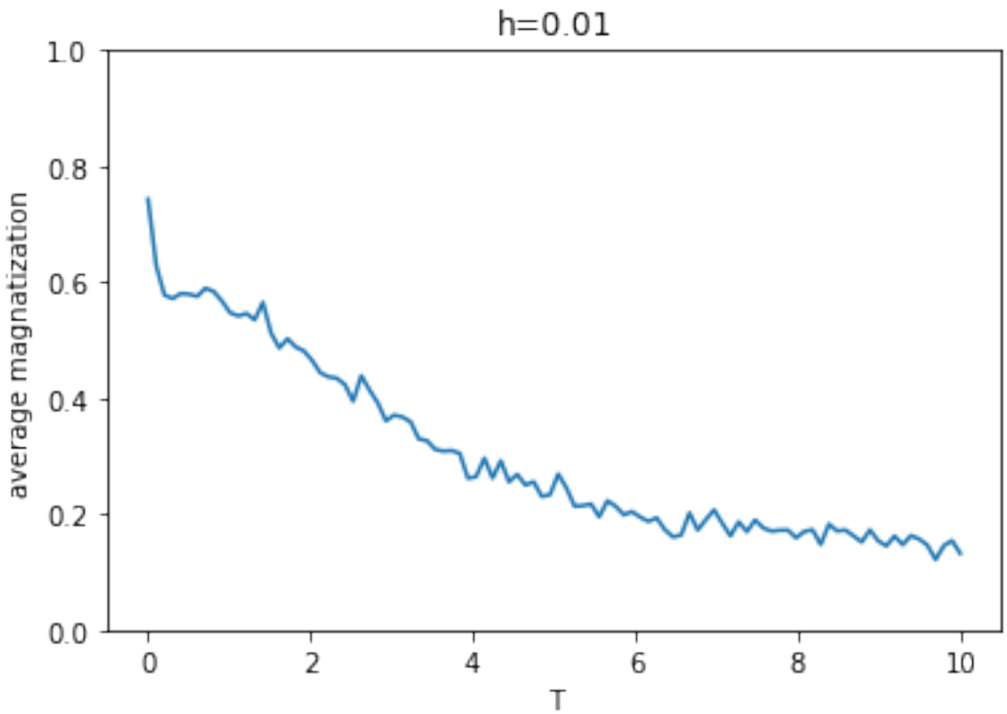
[7]:
```
steps = 100
T_list = np.linspace(0.0001,100,steps)
h=0
average_mag = np.zeros(steps)
i=0
for T in T_list:
    spins_h, arrays_h =metropolis(lattice_n, times, T,h)
    average_mag[i] = spins_h.sum()/times/N/N
    i+=1
plt.ylim([0,1])
plt.plot(T_list,average_mag)
plt.xlabel('T')
plt.ylabel('average magnatization')
plt.title('h='+str(h))
```

[7]:
```
Text(0.5, 1.0, 'h=0')
```



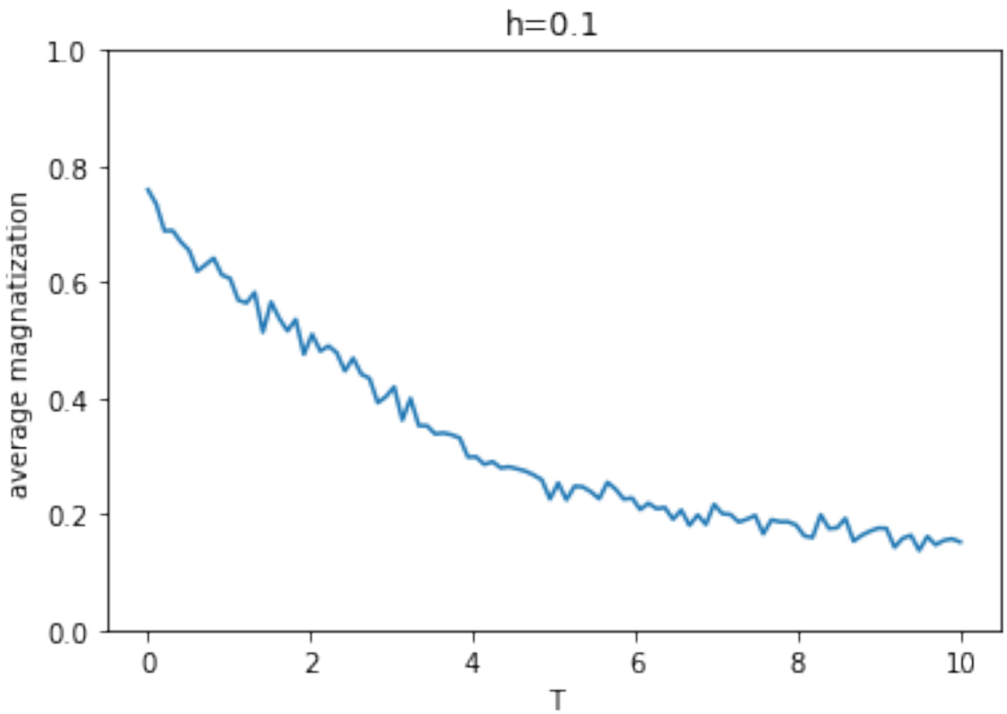
[8]:
```
steps = 100
T_list = np.linspace(0.0001,10,steps)
h=0.01
average_mag = np.zeros(steps)
i=0
for T in T_list:
    spins_h, arrays_h =metropolis(lattice_n, times, T,h)
    average_mag[i] = spins_h.sum()/times/N/N
    i+=1
plt.plot(T_list,average_mag)
plt.xlabel('T')
plt.ylabel('average magnatization')
plt.title('h='+str(h))
plt.ylim([0,1])
```

[8]: (0.0, 1.0)



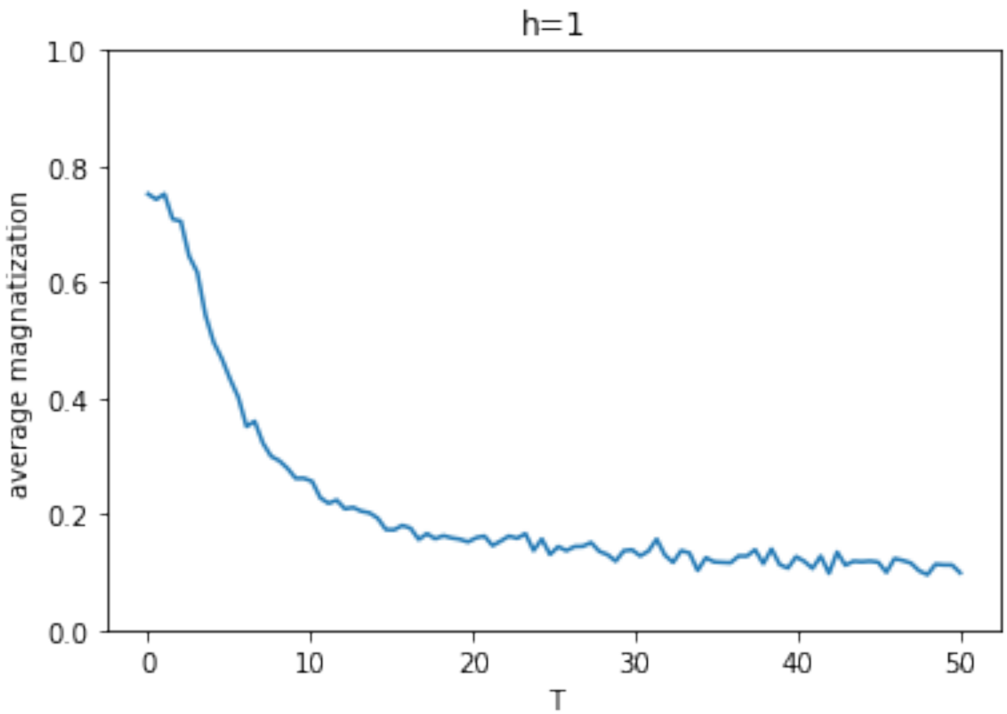
[9]:
```
steps = 100
T_list = np.linspace(0.0001,10,steps)
h=0.1
average_mag = np.zeros(steps)
i=0
for T in T_list:
    spins_h, arrays_h =metropolis(lattice_n, times, T,h)
    average_mag[i] = spins_h.sum()/times/N/N
    i+=1
plt.plot(T_list,average_mag)
plt.xlabel('T')
plt.ylabel('average magnatization')
plt.title('h='+str(h))
plt.ylim([0,1])
```

[9]: (0.0, 1.0)



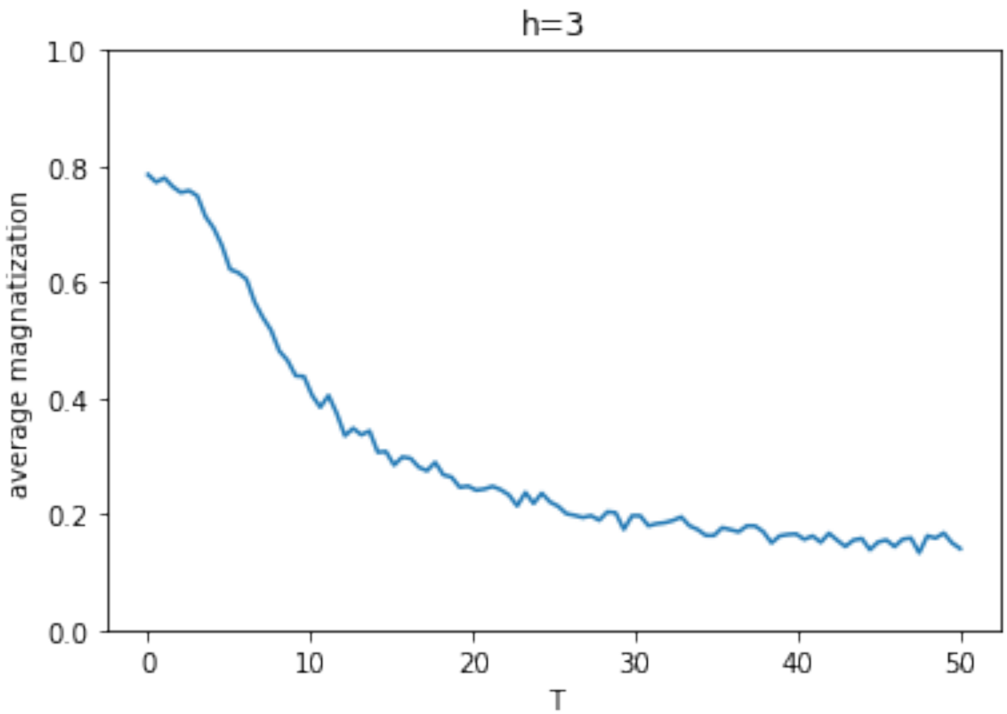

```
[10]: steps = 100
T_list = np.linspace(0.0001,50,steps)
h=1
average_mag = np.zeros(steps)
i=0
for T in T_list:
    spins_h, arrays_h =metropolis(lattice_n, times, T,h)
    average_mag[i] = spins_h.sum()/times/N/N
    i+=1
plt.plot(T_list,average_mag)
plt.xlabel('T')
plt.ylabel('average magnatization')
plt.title('h='+str(h))
plt.ylim([0,1])
```

```
[10]: (0.0, 1.0)
```



[11]:
```
steps = 100
T_list = np.linspace(0.0001,50,steps)
h=3
average_mag = np.zeros(steps)
i=0
for T in T_list:
    spins_h, arrays_h =metropolis(lattice_n, times, T,h)
    average_mag[i] = spins_h.sum()/times/N/N
    i+=1
plt.plot(T_list,average_mag)
plt.xlabel('T')
plt.ylabel('average magnatization')
plt.title('h='+str(h))
plt.ylim([0,1])
```

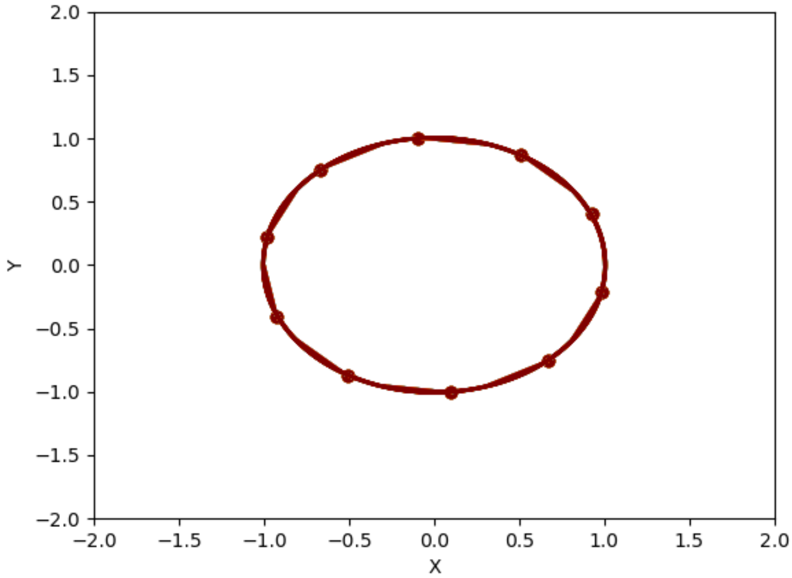
[11]: (0.0, 1.0)


h=3
1.0
0.8
0.6
0.4
0.2
0.0
average magnatization
0
10
20
30
40
50
T

```
[ ]:
```

Problem 2 (The code is NbodyF.py)

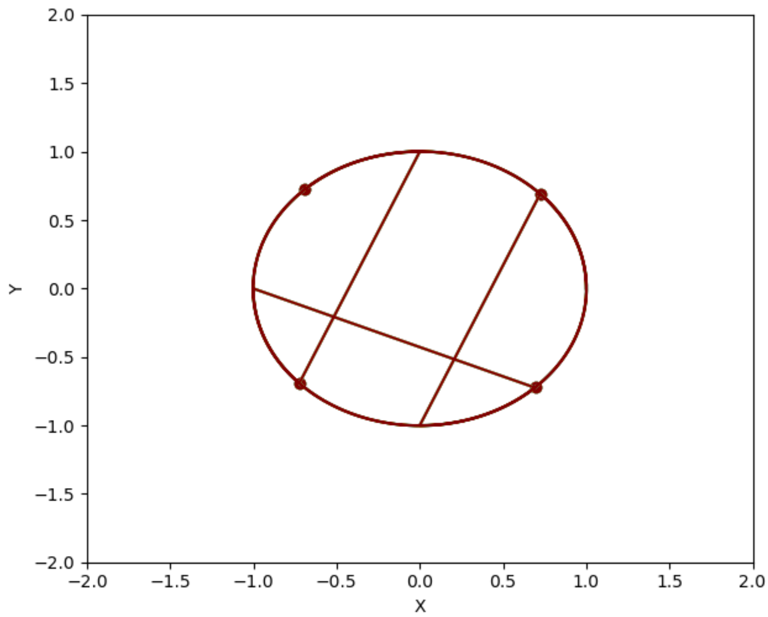
A) For N = 10, dt (time_step) = 0.01, and using Euler method. The following is a snapshot of the animation: (G = 1, m = 1, R = 1)



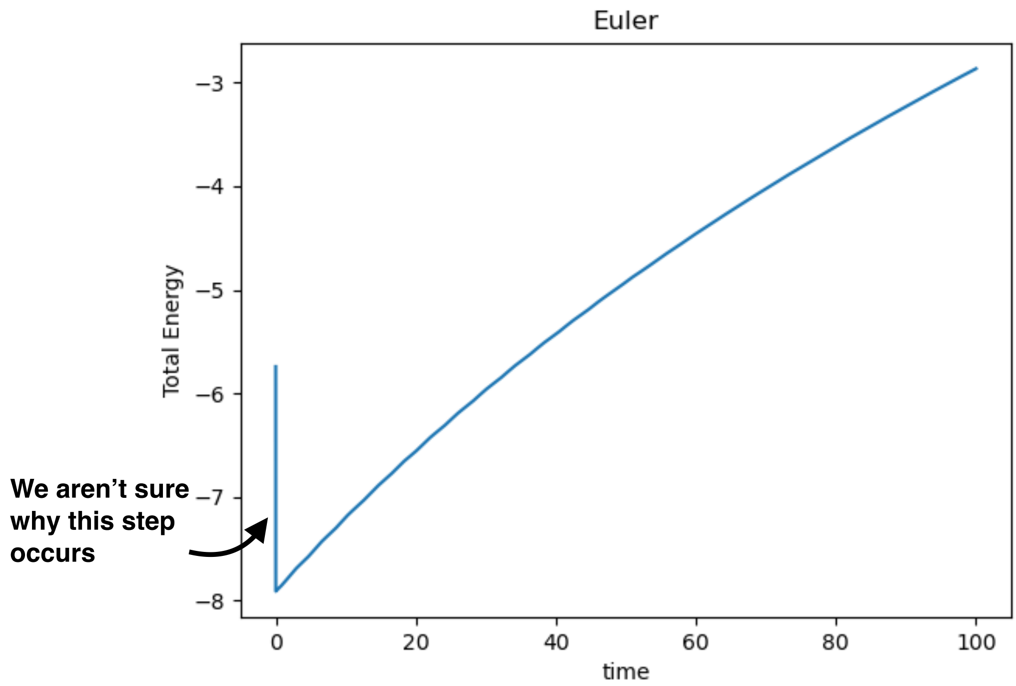
If we zoom to the path, we see some minor deviations of the path from the initial path which depends on updater rules and the time steps. The figure below is for

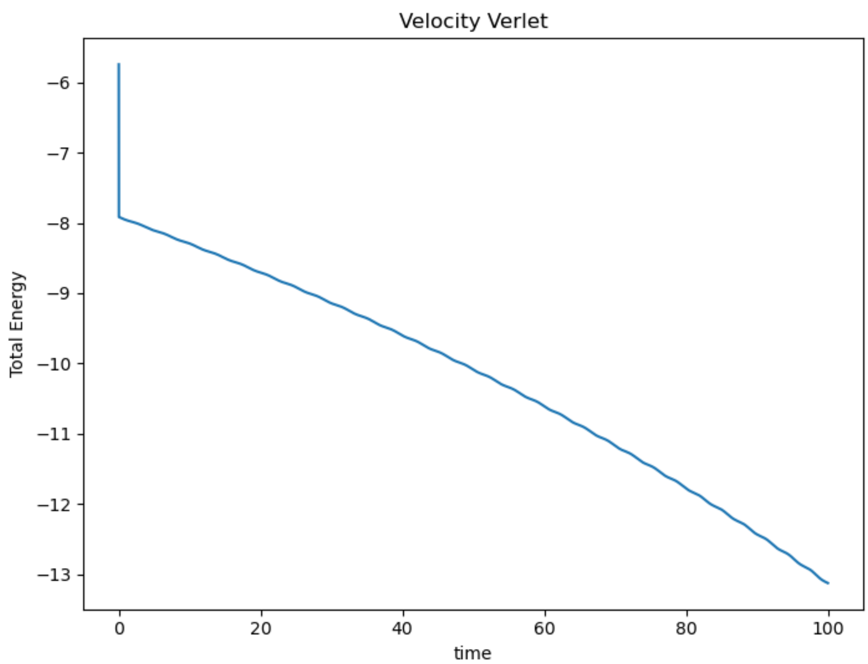
dt = 0.001.

(Ignore the lines in the circle, this is just a weird animation artifact)

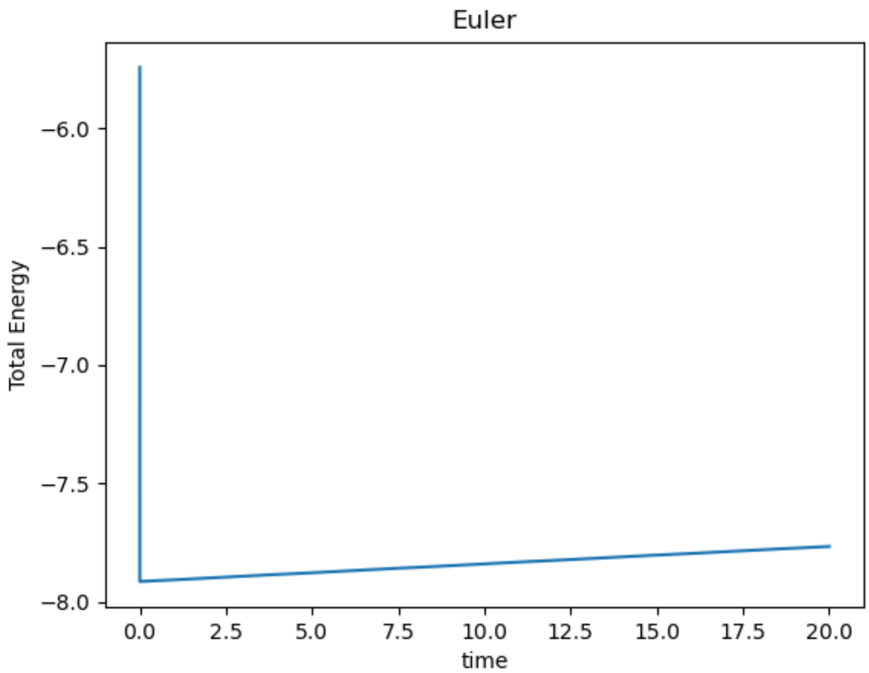


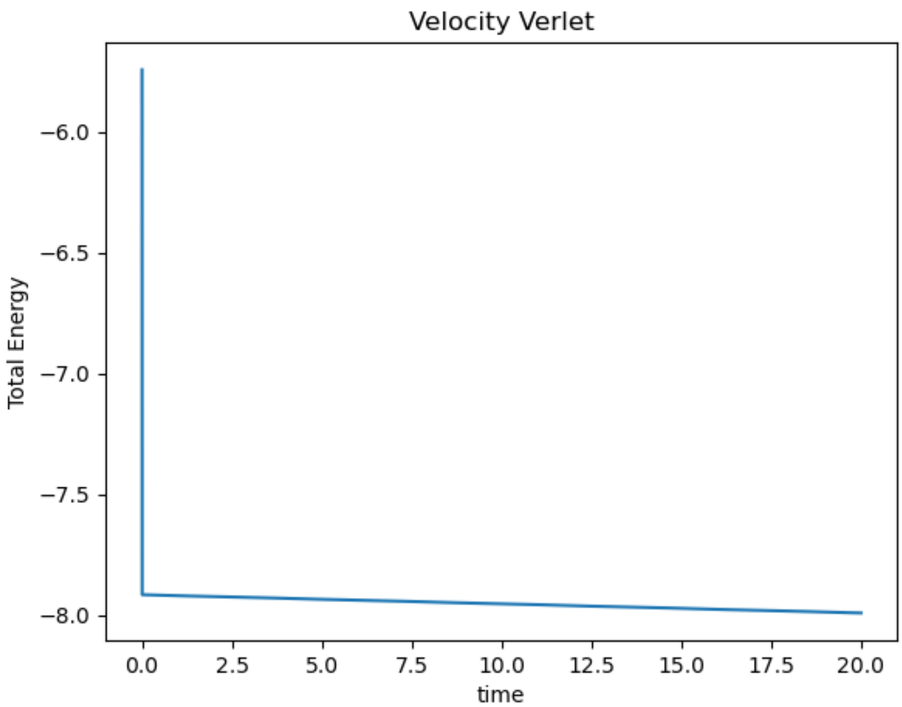
Next we studied variations in total energy of the system with time, and see the following variations with two different update rule.
Both are for dt = 0.01. For Euler the total energy seems to increase with time, and for velocity verlet it increases.



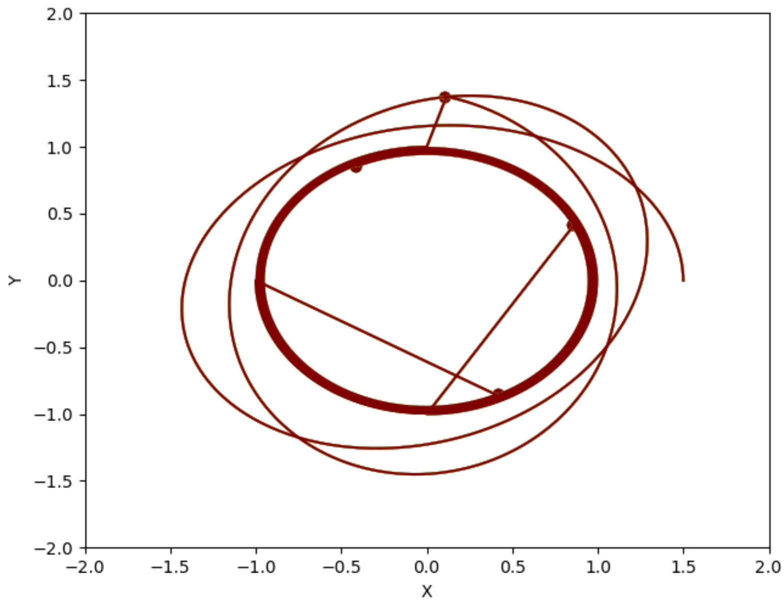


For dt = 0.001, the variation improves. Proving that these methods become more accurate if we decrease time steps.
To compare both of these methods ideally one should look at the slopes of decrease or increase and see which one is better.

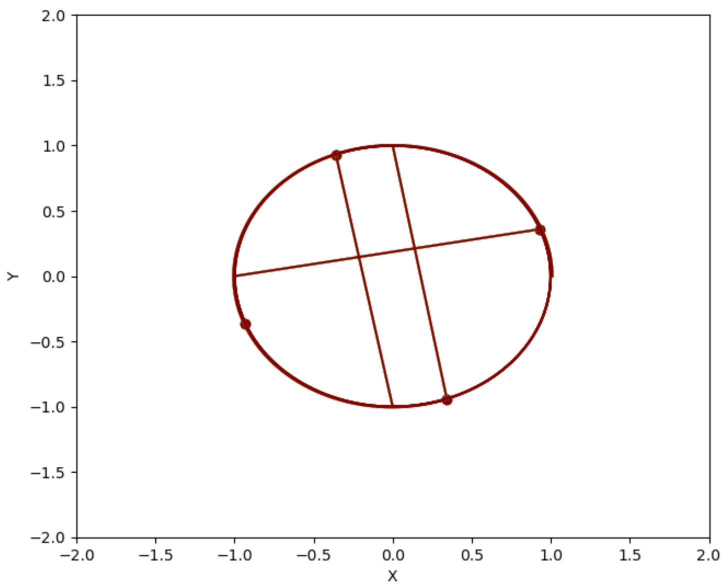




B) For the first case, dt = 0.01, and we displaced body 1, by 0.5 along the x-axis, see the snapshot:



If we displace the body 1 by even small amount say 0.01, then they form almost the same orbit (dt = 0.001 here)



So the orbits look stable.