

ising-MC

October 27, 2026

1 2D Ising model Monte Carlo

MC code to simulate Ising model in 2D

$$H = -J \sum_{\langle ij \rangle} s_i s_j - h \sum_i s_i$$

This model, for $J > 0$ has a Curie temperature of

$$T_c = \frac{2J}{\ln(\sqrt{2} + 1)} \approx 2.269J$$

```
[1]: import numpy as np
rng = np.random.default_rng()
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [6, 6]
from numba import jit
from tqdm.auto import trange

[2]: # simple ferromagnetic coupling
param = {'J': 1.0, 'h': 0.0}

[3]: # set up cell and PBC
Nx, Ny = 256, 256
N = Nx*Ny

d, z = 2, 4 # dimensionality and coordination number
site = np.zeros((N, d), dtype=int)
neigh = np.zeros((N, z), dtype=int)

n = 0
for i in range(Nx):
    for j in range(Ny):
        site[n] = i, j
        n += 1

for n in range(N):
    i, j = site[n]
    neigh[n, 0] = ((i+1)%Nx)*Ny + j
```

```
neigh[n, 1] = ((i-1)%Nx)*Ny + j
neigh[n, 2] = i*Ny + (j+1)%Ny
neigh[n, 3] = i*Ny + (j-1)%Ny
```

```
[4]: # neighbor list check
```

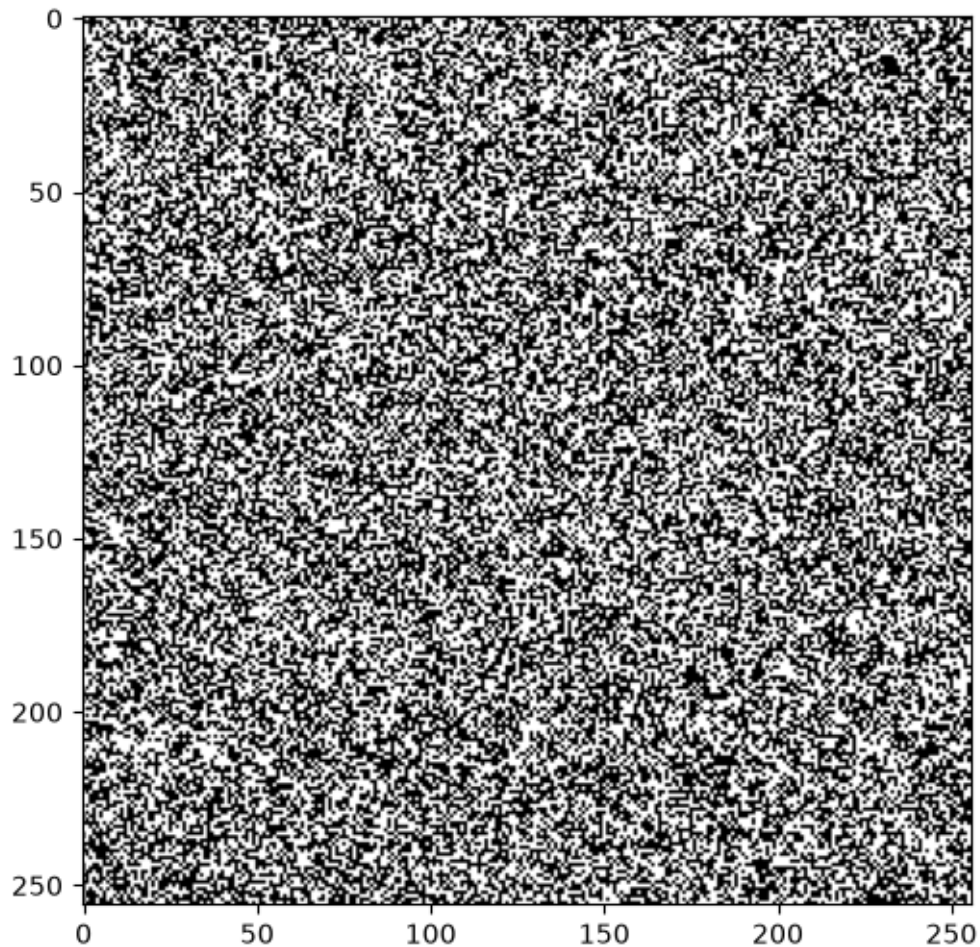
```
if (False):
    for n in range(N):
        print(f'{site[n]}:')
        for m in neigh[n]:
            print(f' {site[m]}')
```

```
[5]: # initialize our spins on a square lattice
```

```
spin = rng.choice((-1, 1), size=N)
```

```
[6]: # visualize
```

```
plt.imshow(spin.reshape(Nx, Ny), cmap="grey");
```



```
[7]: @jit(nopython=True)
def ene(spin, neigh, J=param['J'], h=param['h']):
    """Computes the energy for a given configuration `spin` using `neigh` as a
    ↪neighbor
    list, with `J` and `h` as our parameters"""
    N = neigh.shape[0]
    z = neigh.shape[1]
    E_J = 0
    for n in range(N):
        neigh_list = neigh[n]
        s = spin[n]
        for i in range(z):
            if s==spin[neigh_list[i]]:
                E_J += 1
            else:
                E_J += -1
    return -0.5*J*E_J - h*sum(spin)
```

```
[8]: ene(spin, neigh)
```

```
[8]: 172.0
```

```
[9]: @jit(nopython=True)
def MC(spin, kT, urand, neigh, J=param['J'], h=param['h']):
    """Run MC for Ising spins where we attempt to flip each spin in order.
    Returns acceptance ratio and (relative) energy, and overwrites spin in_
    ↪place."""
    N = neigh.shape[0]
    z = neigh.shape[1]

    accept = 0
    E = 0
    for i in range(len(urand)):
        u = urand[i]
        n = i%N # spin to attempt to flip
        s = spin[n]
        dE = 0
        for j in range(z):
            m = neigh[n, j]
            if s == spin[m]:
                dE += -2
            else:
                dE += 2
        dE = -dE*J + 2*h*s
        if u < np.exp(-dE/kT):
            spin[n] = -s
            E += dE
```

```
        accept += 1
    return accept/len(urand), E
```

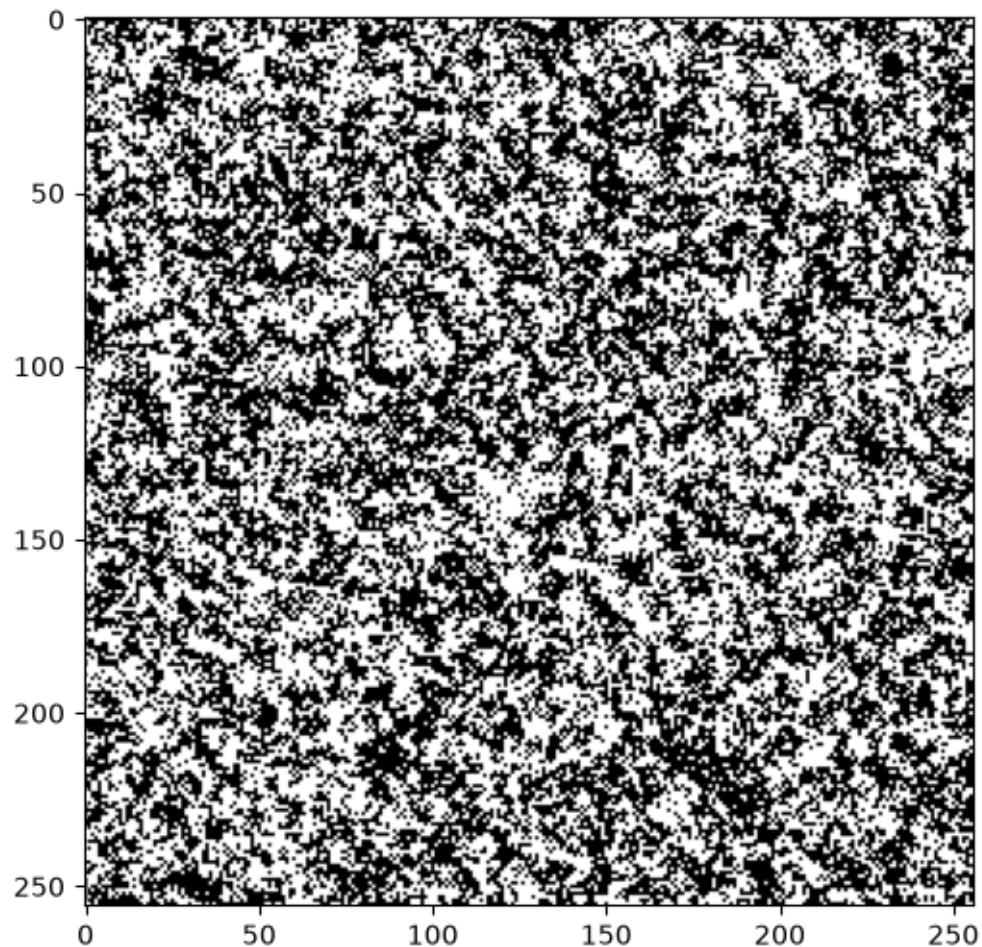
```
[10]: N_MC = N

      kT = 2.5
      E = ene(spin, neigh)
      urand = rng.uniform(size=(N_MC))
      accept, dE = MC(spin, kT, urand, neigh)
```

```
[11]: accept, dE, np.average(spin)
```

```
[11]: (0.60272216796875, -31428.0, np.float64(0.01348876953125))
```

```
[12]: # visualize
      plt.imshow(spin.reshape(Nx, Ny), cmap="grey");
```



```
[13]: ene(spin, neigh)
```

```
[13]: -31256.0
```

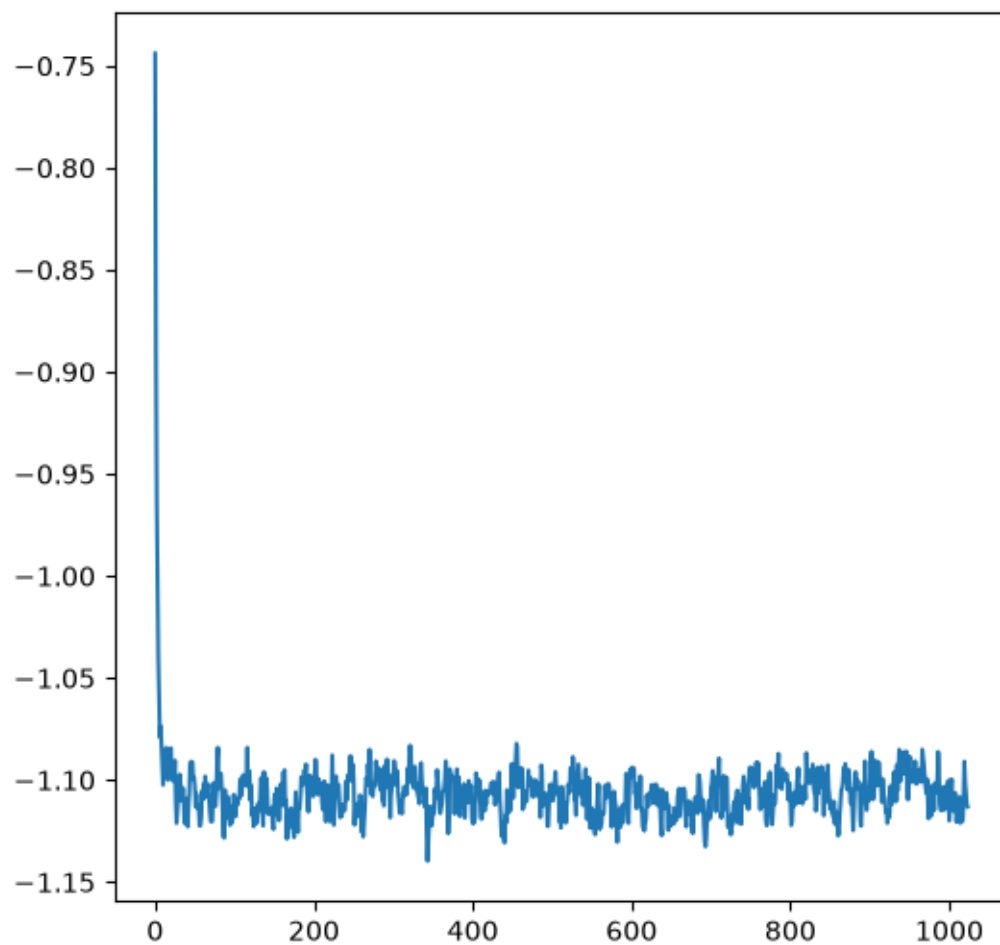
```
[14]: N_sweeps = 1024
      E_list, accept_list, m_list = [], [], []
      E = ene(spin, neigh)
      for _ in trange(N_sweeps):
          accept, dE = MC(spin, kT, rng.uniform(size=N), neigh)
          E += dE
          E_list.append(E/N)
          accept_list.append(accept)
          m_list.append(np.average(spin))
      E_list = np.array(E_list)
      accept_list = np.array(accept_list)
      m_list = np.array(m_list)
```

```
0%|          | 0/1024 [00:00<?, ?it/s]
```

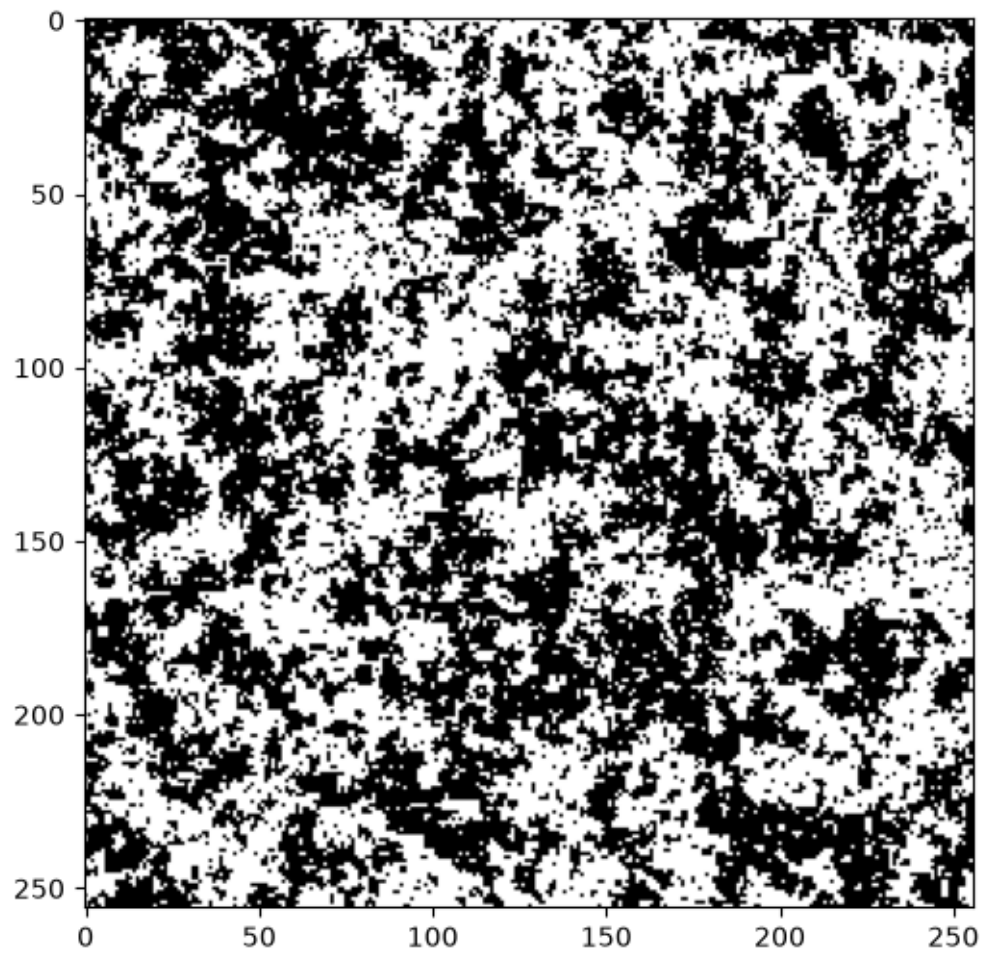
```
[15]: np.average(accept_list)
```

```
[15]: np.float64(0.32313935458660126)
```

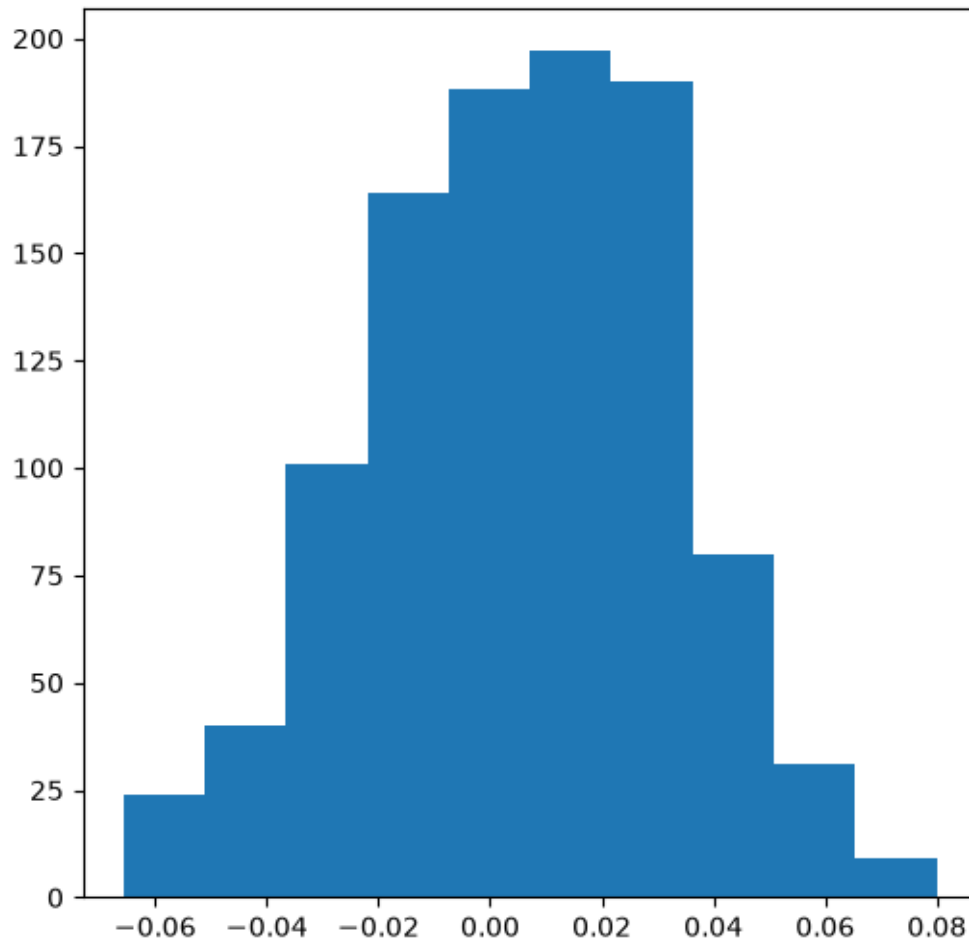
```
[16]: plt.plot(E_list);
```



```
[17]: plt.imshow(spin.reshape(Nx, Ny), cmap='grey');
```



```
[18]: plt.hist(m_list);
```



```
[19]: kT = 2.0

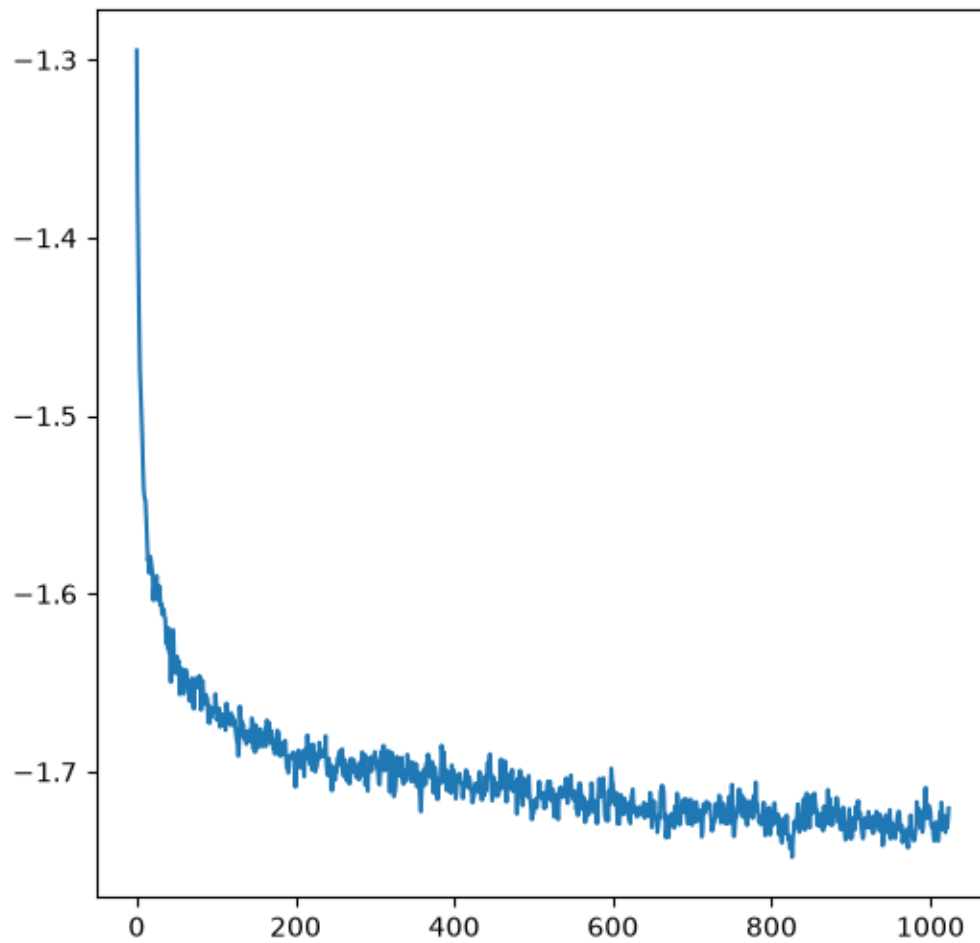
N_sweeps = 1024
E_list, accept_list, m_list = [], [], []
E = ene(spin, neigh)
for _ in trange(N_sweeps):
    accept, dE = MC(spin, kT, rng.uniform(size=N), neigh)
    E += dE
    E_list.append(E/N)
    accept_list.append(accept)
    m_list.append(np.average(spin))
E_list = np.array(E_list)
accept_list = np.array(accept_list)
m_list = np.array(m_list)
```

```
0%|          | 0/1024 [00:00<?, ?it/s]
```

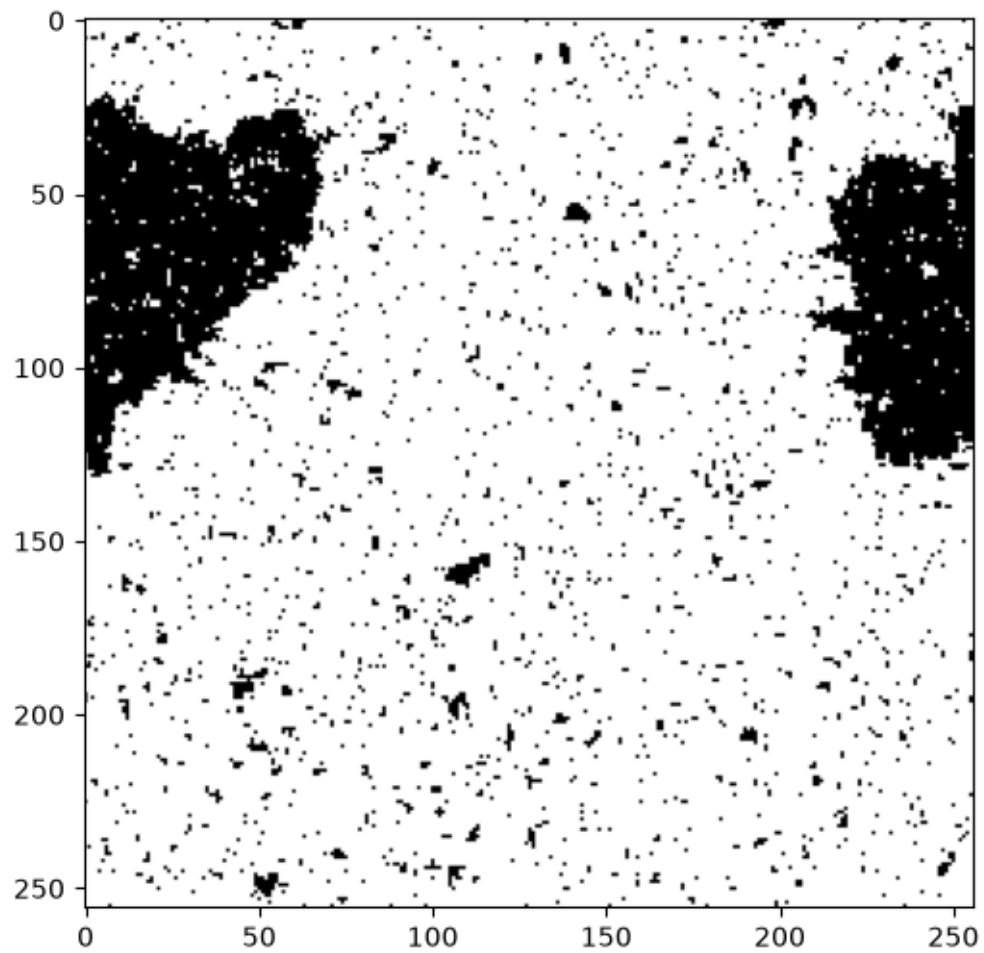
```
[20]: np.average(accept_list)
```

```
[20]: np.float64(0.0938706248998642)
```

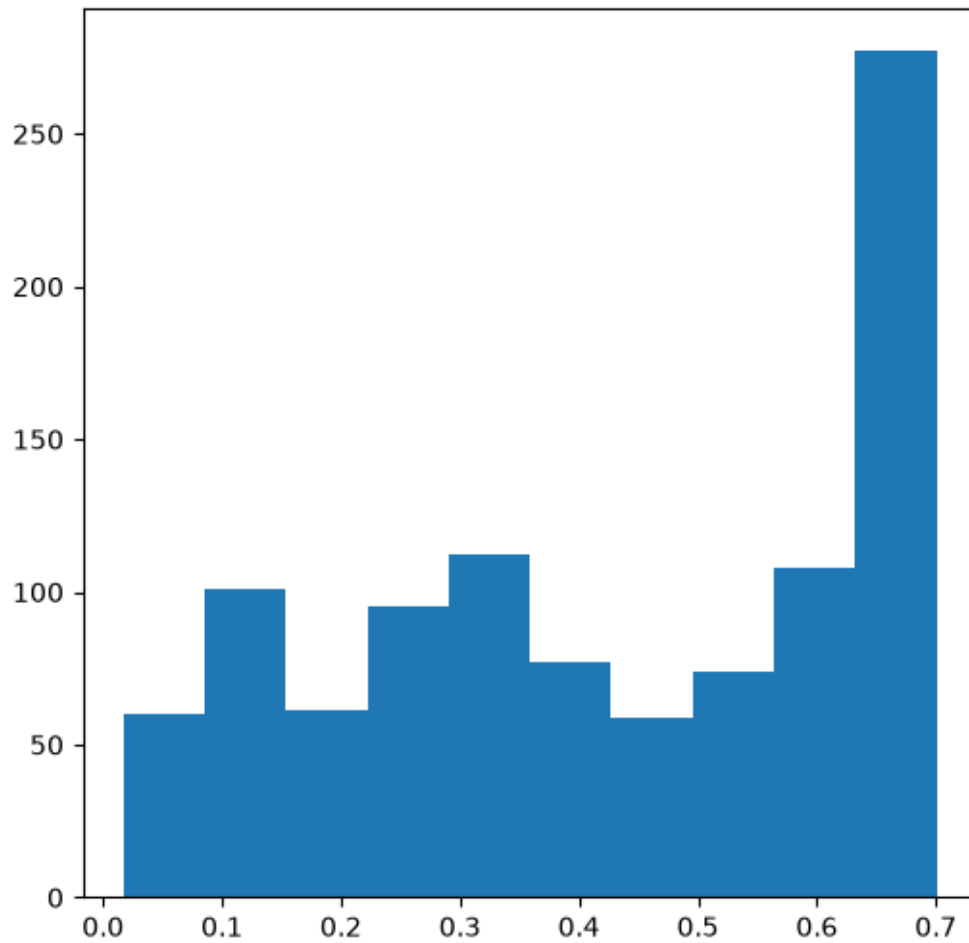
```
[21]: plt.plot(E_list);
```



```
[22]: plt.imshow(spin.reshape(Nx, Ny), cmap='grey');
```



```
[23]: plt.hist(m_list);
```



```
[24]: kT = 4.0

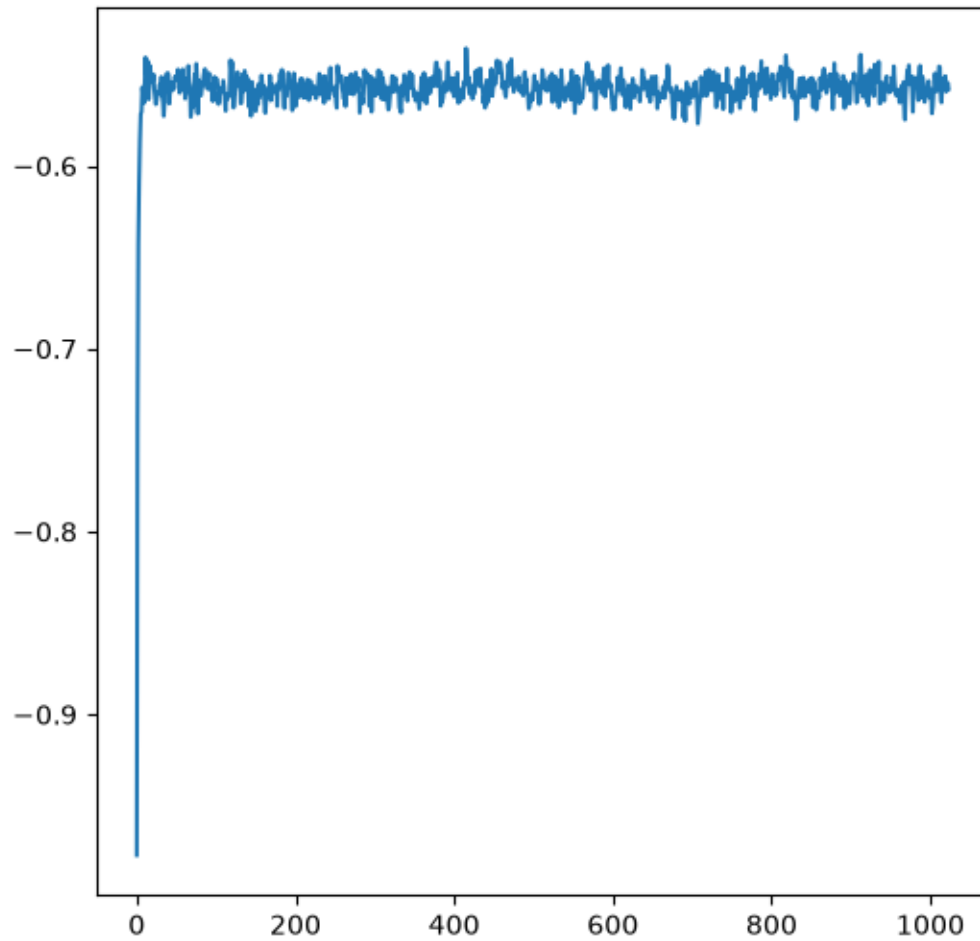
N_sweeps = 1024
E_list, accept_list, m_list = [], [], []
E = ene(spin, neigh)
for _ in trange(N_sweeps):
    accept, dE = MC(spin, kT, rng.uniform(size=N), neigh)
    E += dE
    E_list.append(E/N)
    accept_list.append(accept)
    m_list.append(np.average(spin))
E_list = np.array(E_list)
accept_list = np.array(accept_list)
m_list = np.array(m_list)
```

```
0%|          | 0/1024 [00:00<?, ?it/s]
```

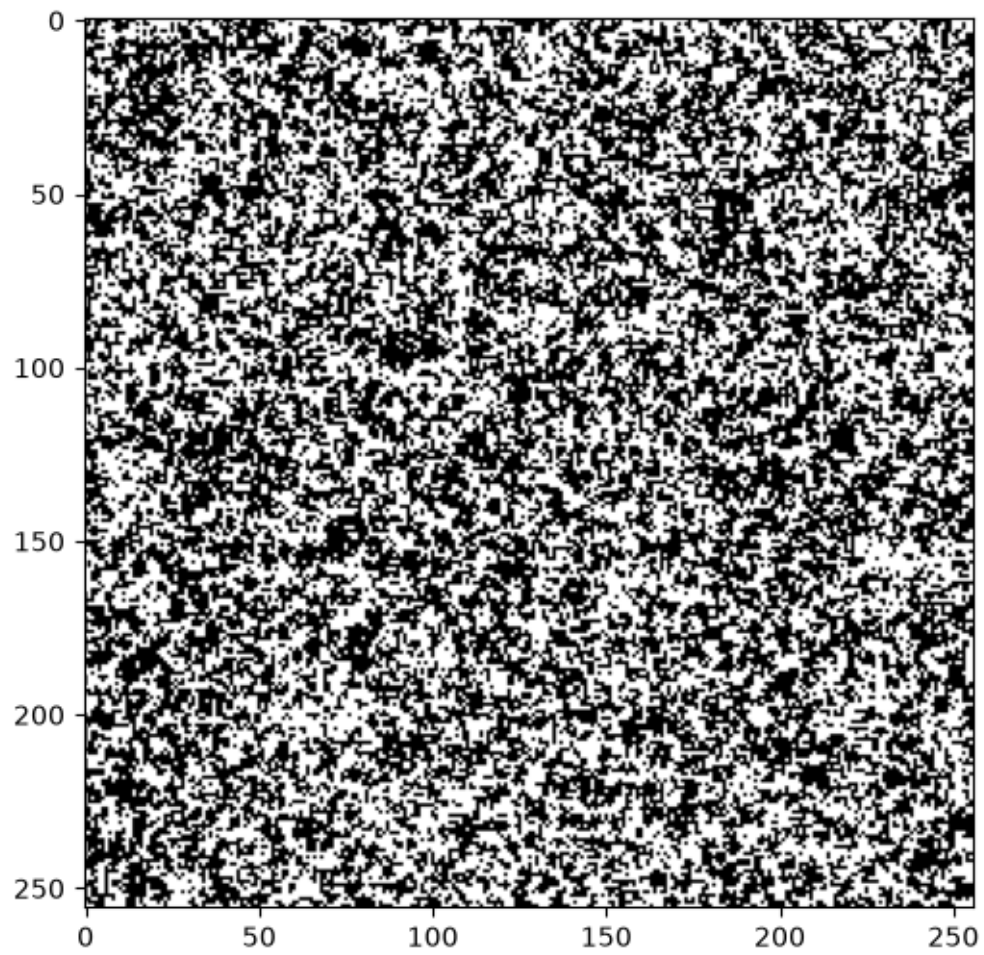
```
[25]: np.average(accept_list)
```

```
[25]: np.float64(0.6091135591268539)
```

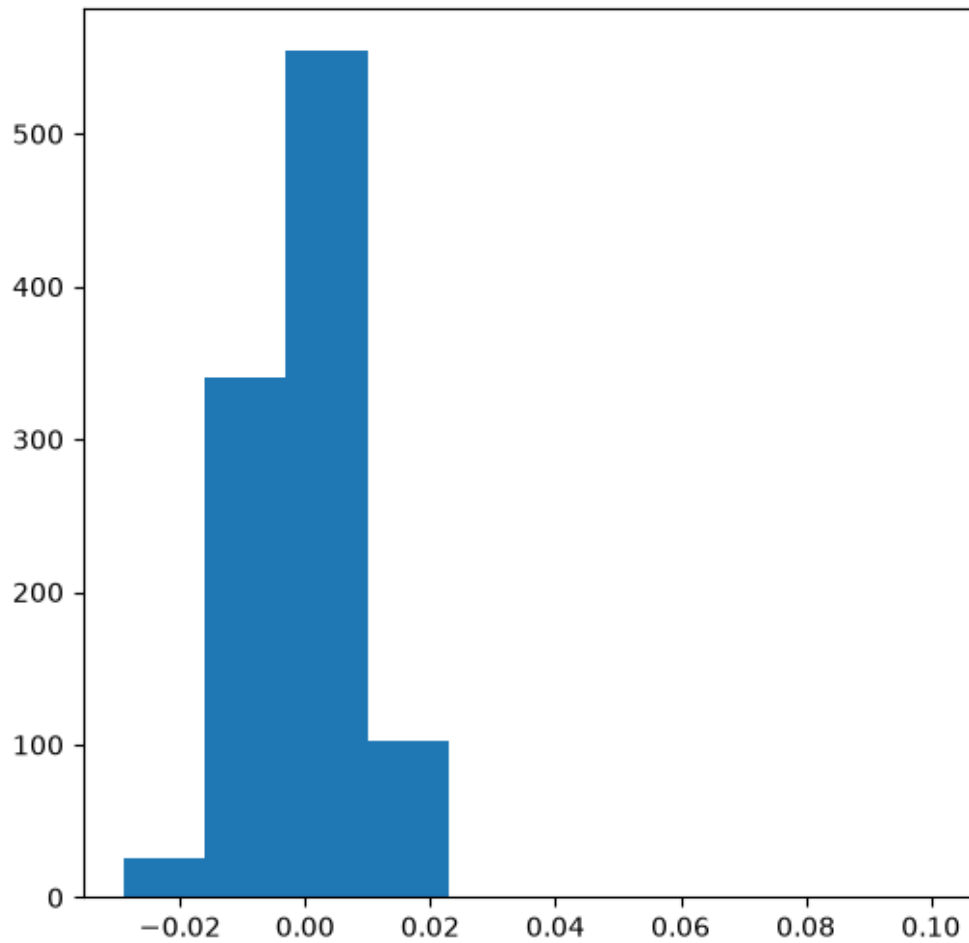
```
[26]: plt.plot(E_list);
```



```
[27]: plt.imshow(spin.reshape(Nx, Ny), cmap='grey');
```



```
[28]: plt.hist(m_list);
```



1.0.1 Magnetization with temperature

```
[29]: kT_list = np.linspace(3.0, 2.0, num=11)
N_sweeps = 4096
m_T = []
for kT in kT_list:
    m_list = []
    for _ in trange(N_sweeps):
        # just capture magnetization
        MC(spin, kT, rng.uniform(size=N), neigh)
        m_list.append(np.average(spin))
    m_T.append(np.array(m_list))
```

```
0%|          | 0/4096 [00:00<?, ?it/s]
```

```
0%|          | 0/4096 [00:00<?, ?it/s]
```

```
0%|          | 0/4096 [00:00<?, ?it/s]
```

```

0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]
0%|          | 0/4096 [00:00<?, ?it/s]

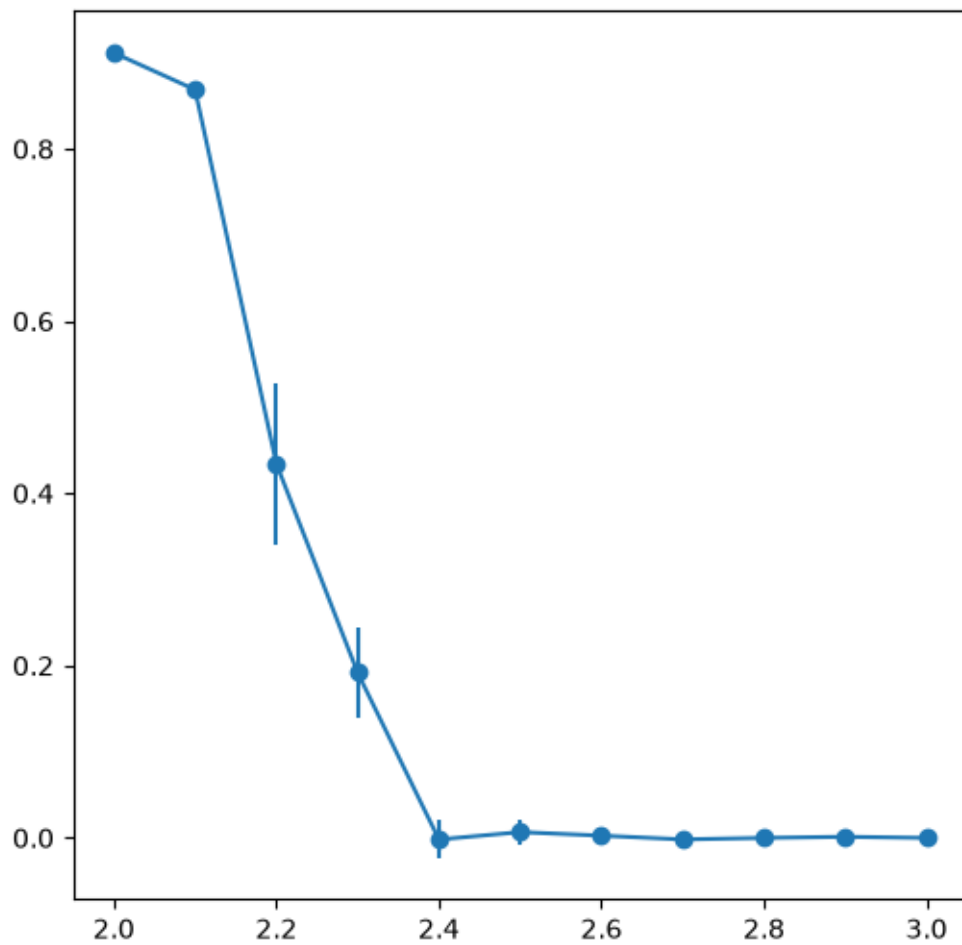
```

```
[30]: ave_m_T = np.array([np.average(m) for m in m_T])
      err_m_T = np.array([np.std(m)/np.sqrt(N_sweeps) for m in m_T])
```

```
[31]: err_m_T
```

```
[31]: array([1.96512675e-04, 2.30393591e-04, 2.53028311e-04, 2.94991018e-04,
            3.41567210e-04, 4.61144693e-04, 7.55503852e-04, 1.76955808e-03,
            3.10976042e-03, 8.85221161e-05, 5.33746274e-05])
```

```
[32]: plt.errorbar(kT_list, ave_m_T, 30*err_m_T, fmt='o-');
```

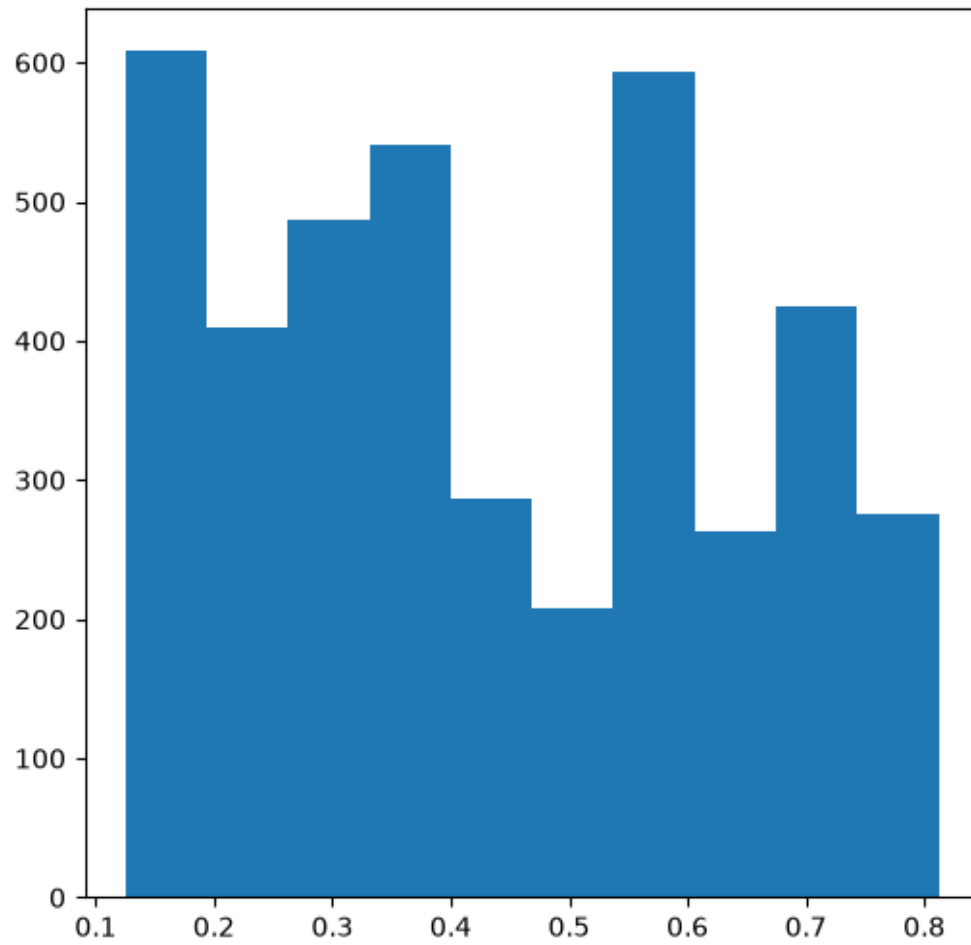


```
[33]: np.argwhere(kT_list==2.2)
```

```
[33]: array([[8]])
```

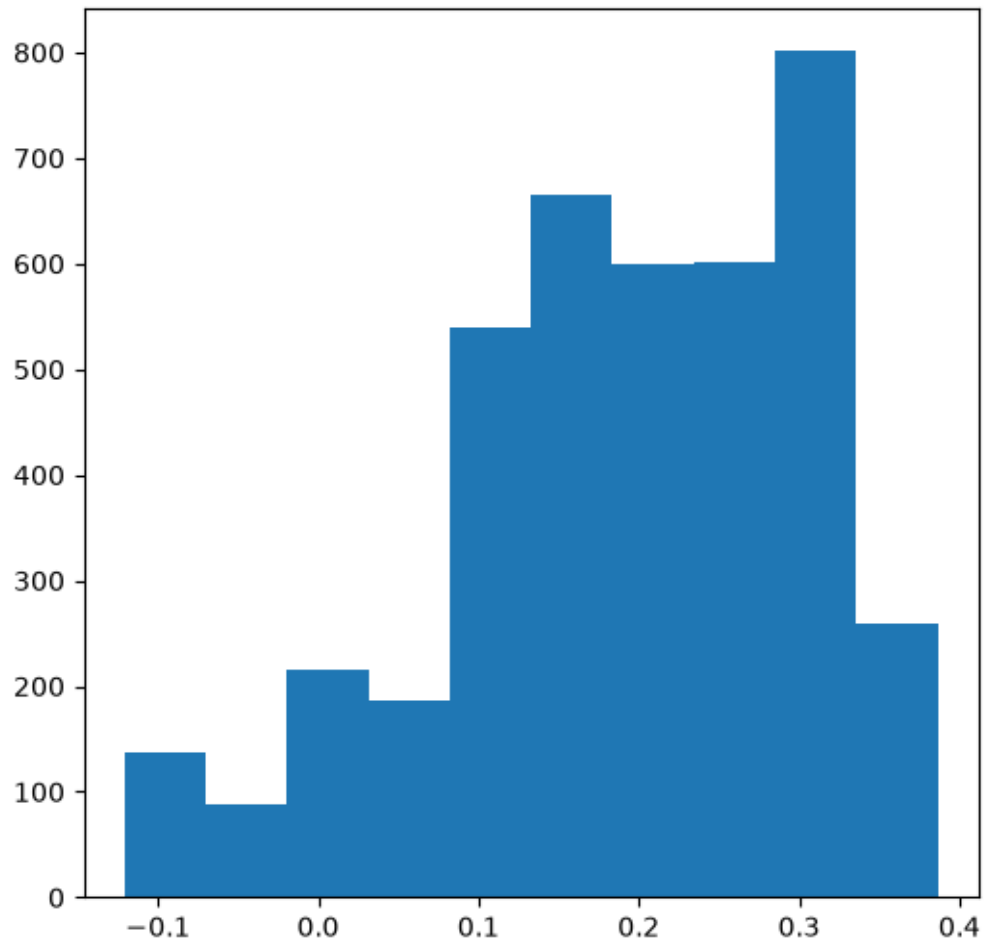
```
[34]: plt.hist(m_T[8]);  
      kT_list[8]
```

```
[34]: np.float64(2.2)
```



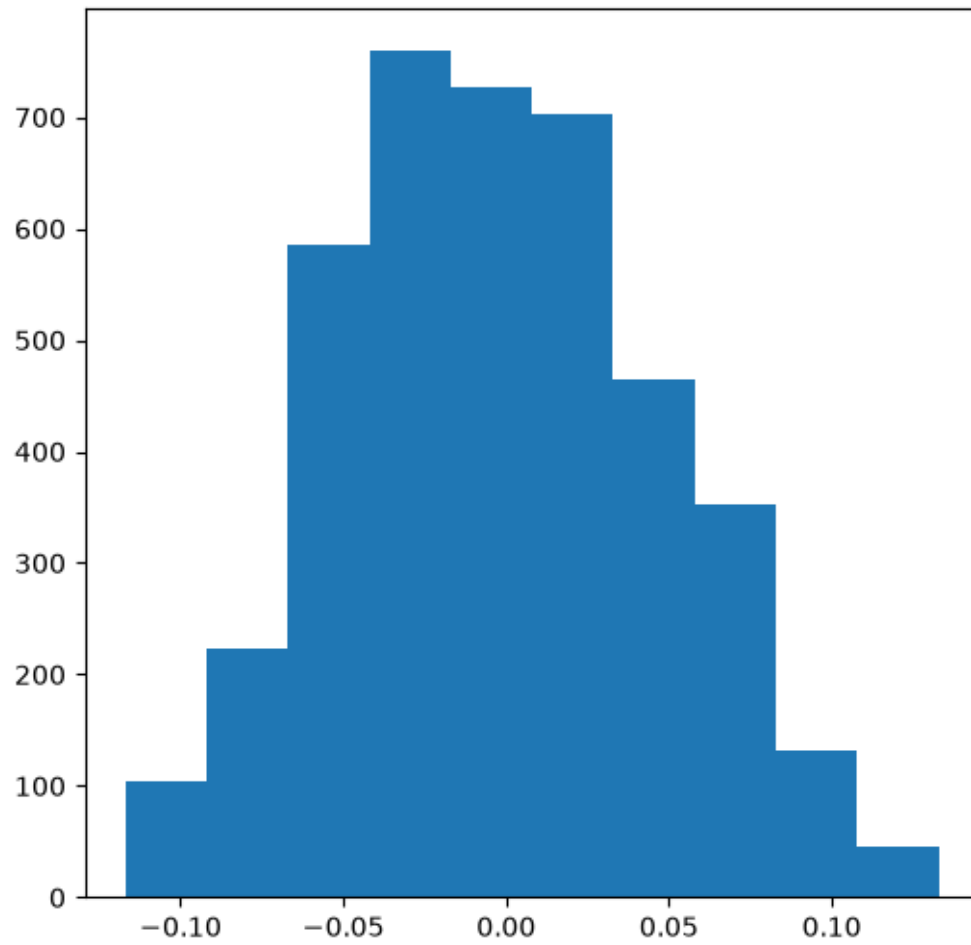
```
[35]: plt.hist(m_T[7]);  
      kT_list[7]
```

```
[35]: np.float64(2.3)
```



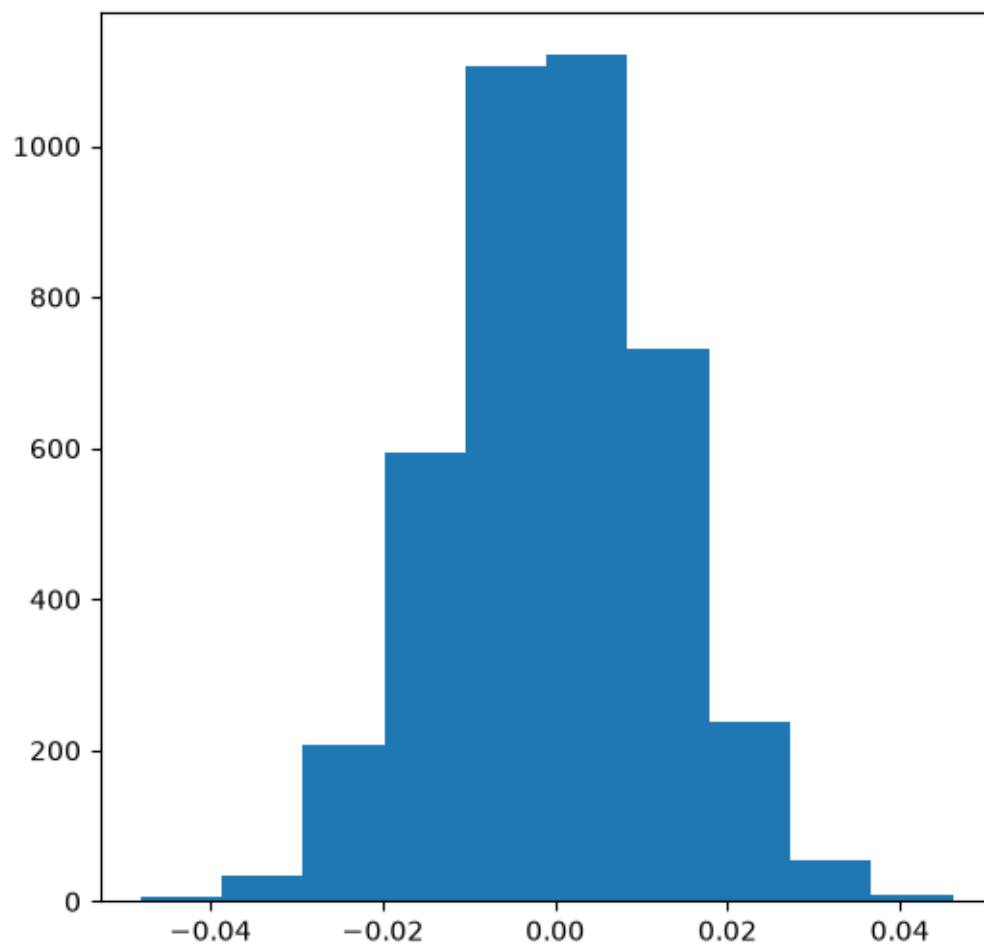
```
[36]: plt.hist(m_T[6]);  
      kT_list[6]
```

```
[36]: np.float64(2.4)
```



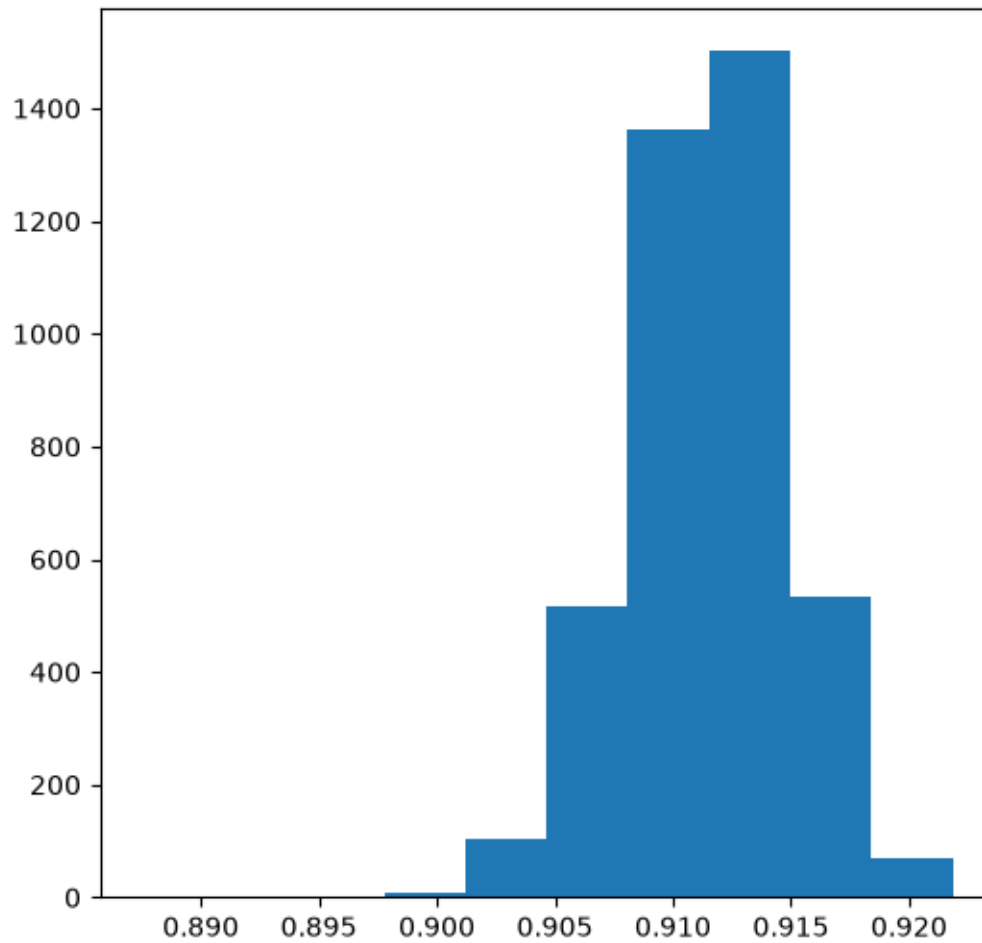
```
[37]: plt.hist(m_T[0]);  
      kT_list[0]
```

```
[37]: np.float64(3.0)
```



```
[38]: plt.hist(m_T[-1]);  
      kT_list[-1]
```

```
[38]: np.float64(2.0)
```



1.0.2 Long run near T_c

```
[47]: kT = 2.3

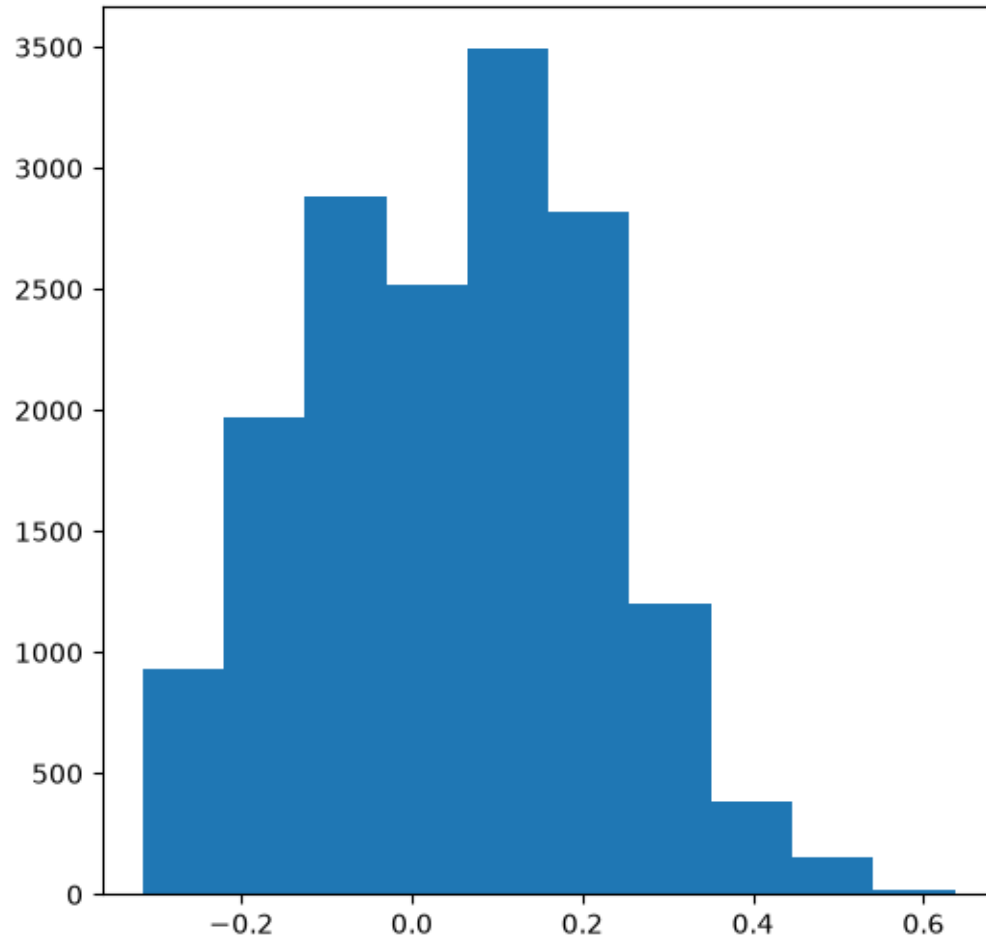
N_sweeps = 16384
E_list, accept_list, m_list = [], [], []
E = ene(spin, neigh)
for _ in trange(N_sweeps):
    accept, dE = MC(spin, kT, rng.uniform(size=N), neigh)
    E += dE
    E_list.append(E/N)
    accept_list.append(accept)
    m_list.append(np.average(spin))
E_list = np.array(E_list)
accept_list = np.array(accept_list)
m_list = np.array(m_list)
```

0%| | 0/16384 [00:00<?, ?it/s]

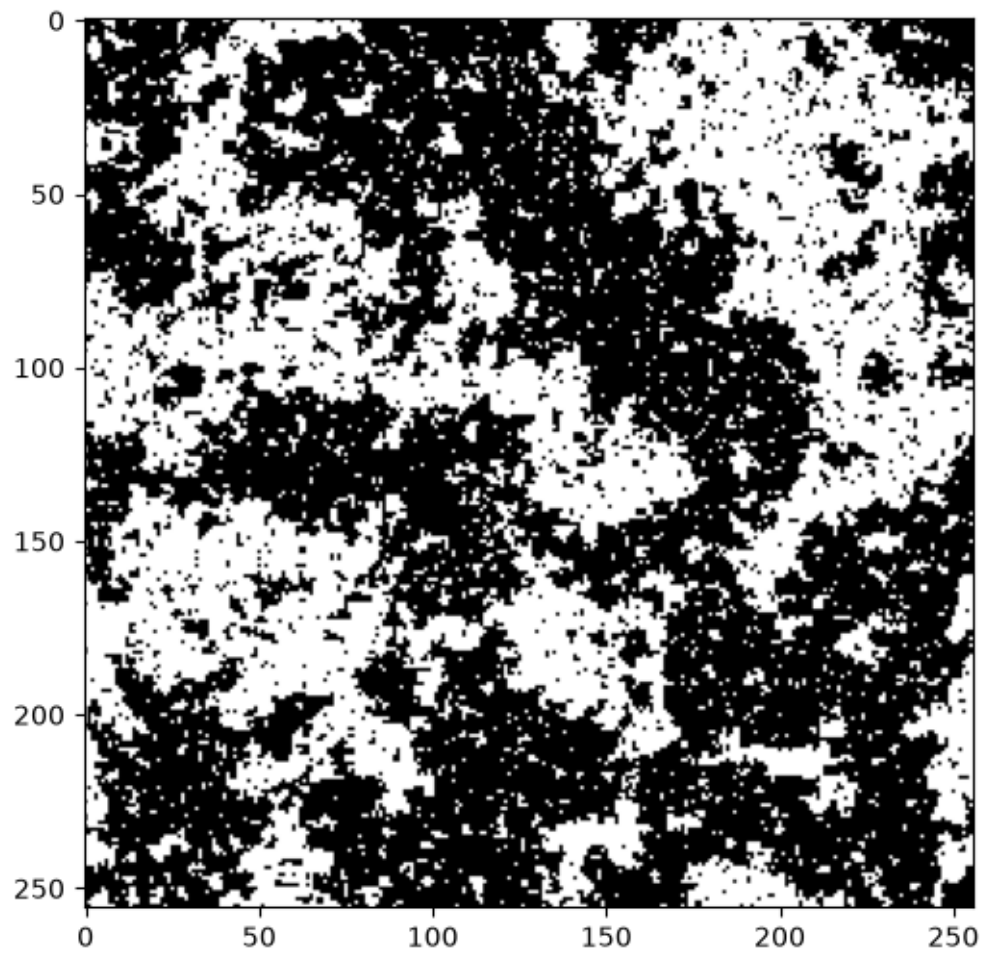
```
[48]: np.average(accept_list)
```

```
[48]: np.float64(0.22661168966442347)
```

```
[49]: plt.hist(m_list);
```



```
[50]: plt.imshow(spin.reshape(Nx, Ny), cmap='grey');
```



[]: