Results

June 1, 2023

1 Results generation

Once the functioning of the Ising Model was clear (by implementing it on the file `Ising.ipynb`) we made this file where the model is abstracted as a *class* where we can easily interact with the model by its methods and properties.

1.1 Libraries used

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
```

1.2 Model implementation with OOP paradigm

[2]:
```python
class Ising:
    def __init__(self, width, height, prob=(0.5,0.5)):
        self.height = height
        self.width = width
        self.init_lattice(prob)

    def init_lattice(self, prob=(0.5,0.5)):
        self.lattice = np.random.choice([1, -1], size=(self.height, self.
 ↪width), p=prob)
        self.measure_energy()
        self.measure_magnet()

    def measure_energy(self):
        k = np.array([[0, 1, 0],
                      [1, 0, 1],
                      [0, 1, 0]])
        self.energy = -(self.lattice * convolve(self.lattice, k, mode="wrap")).
 ↪sum()

    def measure_magnet(self):
        self.magnet = self.lattice.sum()

    def update_lattice(self, temp):
```

```
        x = np.random.randint(0, self.width)
        y = np.random.randint(0, self.height)
        s = self.lattice[y,x]
        e0 = -s*(self.lattice[y-1,x] \
                + self.lattice[(y+1)%self.height,x] \
                + self.lattice[y,x-1] \
                + self.lattice[y,(x+1)%self.width])
        if e0 > 0:
            self.lattice[y,x] = -s
            self.energy += -2*e0
            self.magnet += -2*s
        elif np.random.rand() < np.exp(2*e0/temp):
            self.lattice[y,x] = -s
            self.energy += -2*e0
            self.magnet += -2*s

    def montecarlo(self, times, temp):
        t = 0
        while t < times:
            self.update_lattice(temp)
            t += 1

    def summary(self):
        print(f"--- SUMMARY ---\nm = {self.magnet}\nE = {self.energy}")
```

1.3 Results generation

At first we will instantiate our model making a 200×200 lattice

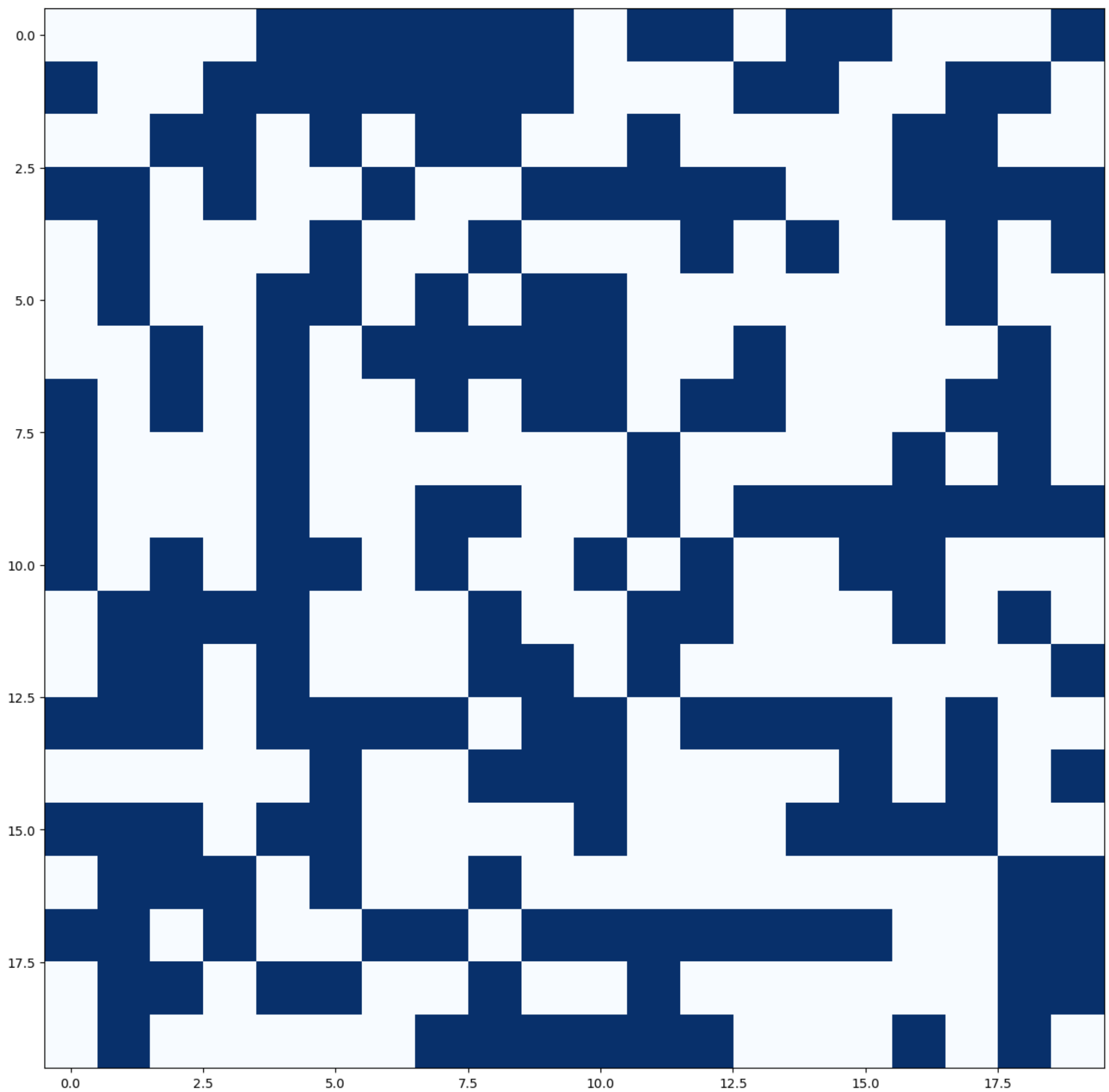
[3]:
```
sim1 = Ising(20, 20)
```

1.3.1 Plotting states

Initial state We can see a plot of the initial state

[4]:
```
fig, ax = plt.subplots(figsize=(15, 15))
ax.imshow(sim1.lattice, cmap='Blues')
```

[4]:
```
<matplotlib.image.AxesImage at 0x26f6678b6a0>
```



with energy and magnetization

[5]:
```
sim1.summary()
```

```
--- SUMMARY ---
m = -24
E = -24
```

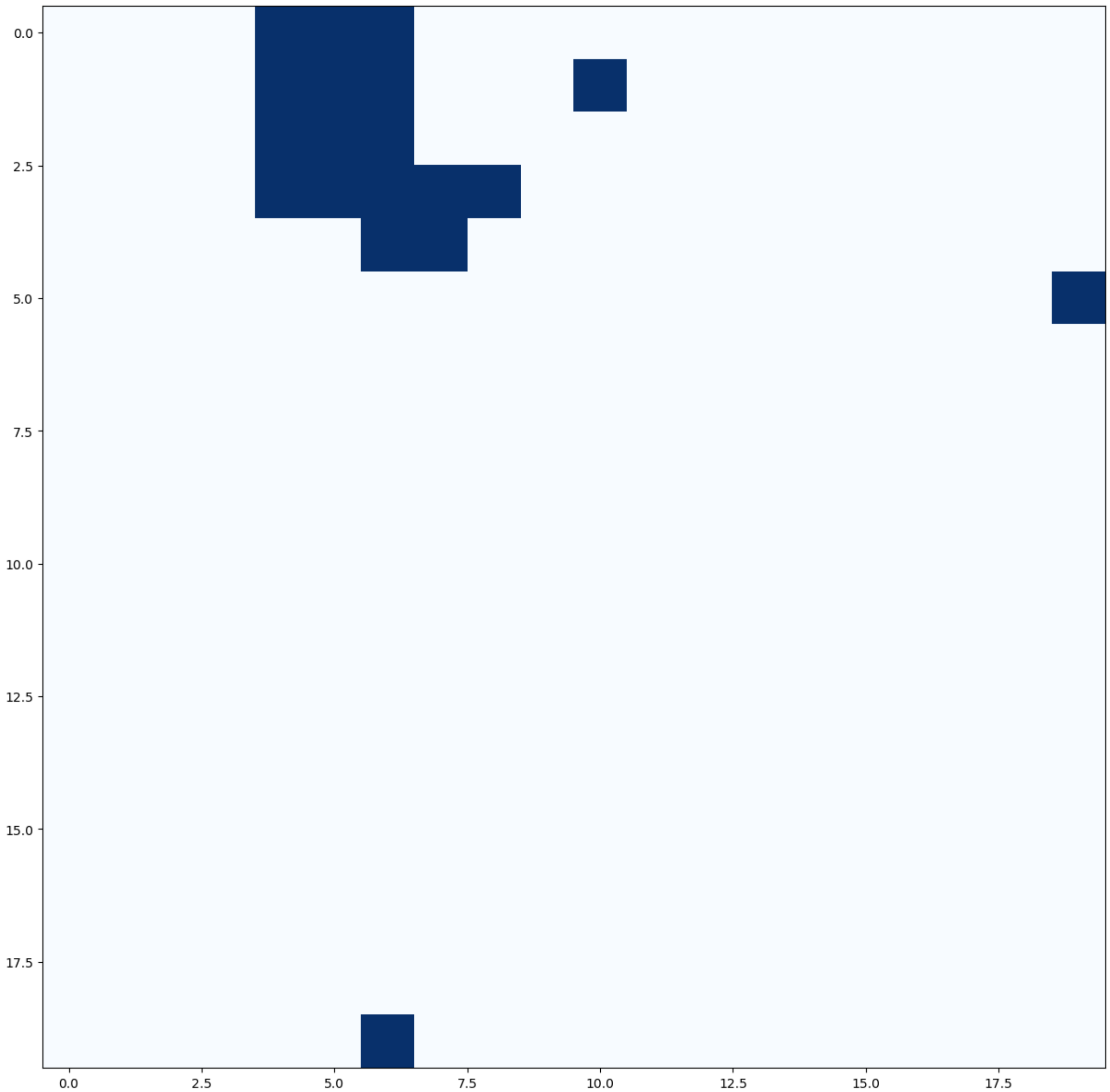
Updating the system

[6]:
```
sim1.montecarlo(100_000, 1.5)
```

Final state We can see a plot of the final state

[7]:
```
fig, ax = plt.subplots(figsize=(15, 15))
ax.imshow(sim1.lattice, cmap='Blues')
```

[7]: <matplotlib.image.AxesImage at 0x26f683d7d00>



with energy and magnetization

[8]:
```
sim1.summary()
```

```
--- SUMMARY ---
m = -362
E = -752
```

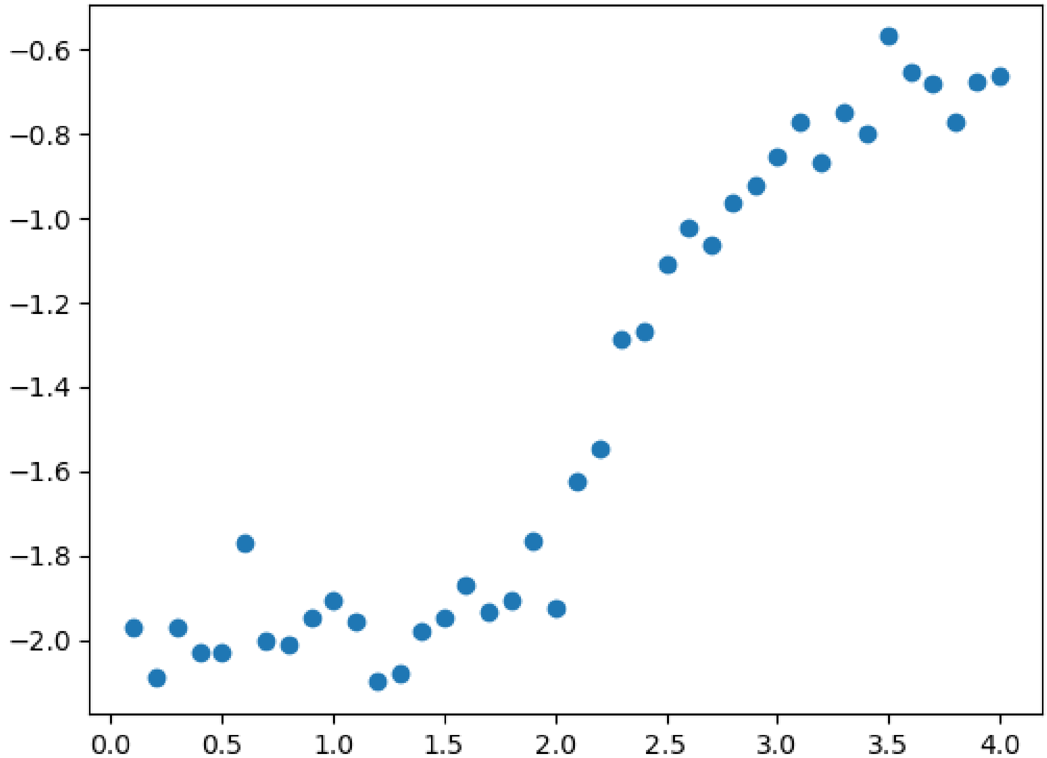
1.3.2 Expected values vs temperature

[9]:
```python
temperatures = [i/10 for i in range(1, 41, 1)]

eq_reps = 2_000
mc_reps = 2_000
```

[10]:
```python
plot_energy = np.zeros(len(temperatures))
plot_magnet = np.zeros(len(temperatures))
plot_heat = np.zeros(len(temperatures))
plot_chi = np.zeros(len(temperatures))

for T,i in zip(temperatures,range(len(temperatures))):
    sim1.init_lattice()

    for _ in range(eq_reps):
        sim1.montecarlo(20*20, T)

    list_energy = np.zeros(mc_reps)
    list_magnet = np.zeros(mc_reps)
    for j in range(mc_reps):
        sim1.montecarlo(20*20, T)
        list_energy[j] = sim1.energy
        list_magnet[j] = sim1.magnet

    plot_energy[i] = list_energy.mean()/(sim1.width*sim1.height)
    plot_heat[i] = (list_energy**2).mean()/(sim1.width*sim1.height) - list_energy.mean()**2/(sim1.width*sim1.height)
    plot_magnet[i] = list_magnet.mean()/(sim1.width*sim1.height)
    plot_chi[i] = (list_magnet**2).mean()/(sim1.width*sim1.height) - list_magnet.mean()**2/(sim1.width*sim1.height)
```

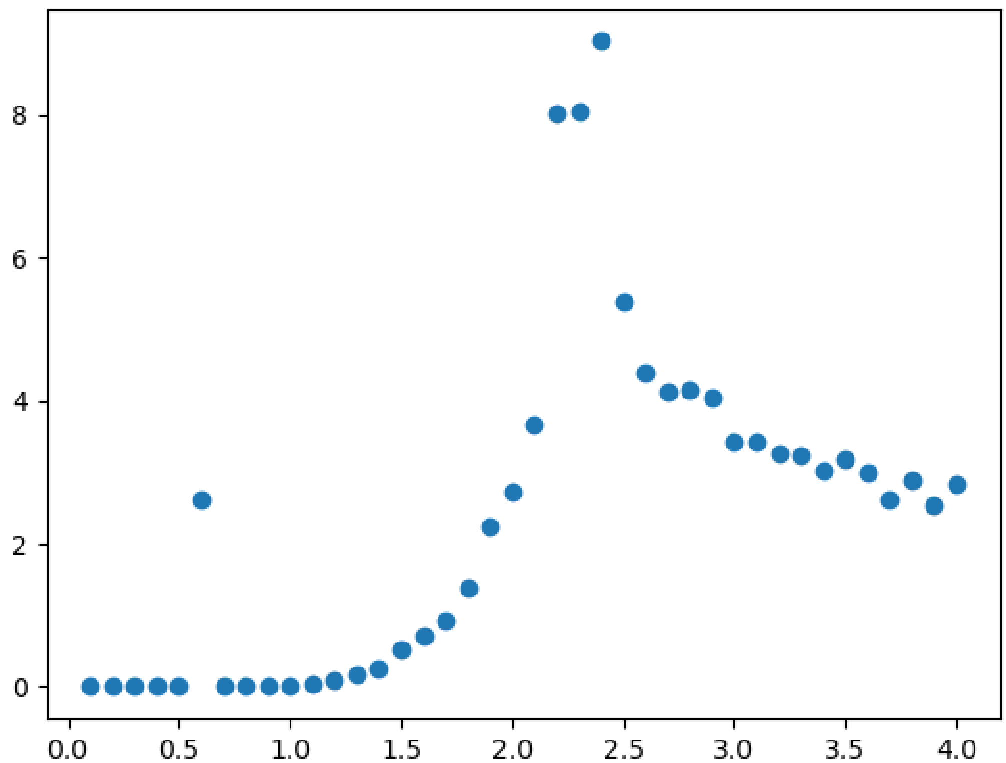
[11]:
```python
fig, ax = plt.subplots()
ax.scatter(temperatures, plot_energy)
```

[11]:
```
<matplotlib.collections.PathCollection at 0x26f66914670>
```



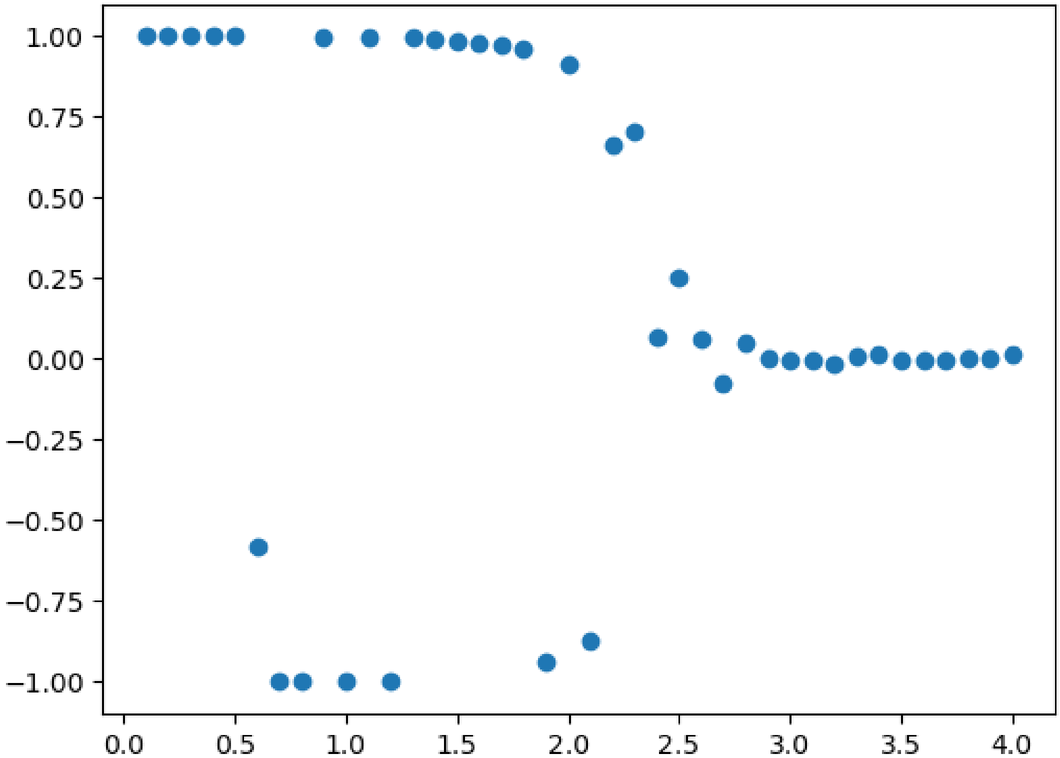
[12]:
```
fig, ax = plt.subplots()
ax.scatter(temperatures, plot_heat)
```

[12]: <matplotlib.collections.PathCollection at 0x26f66975c30>



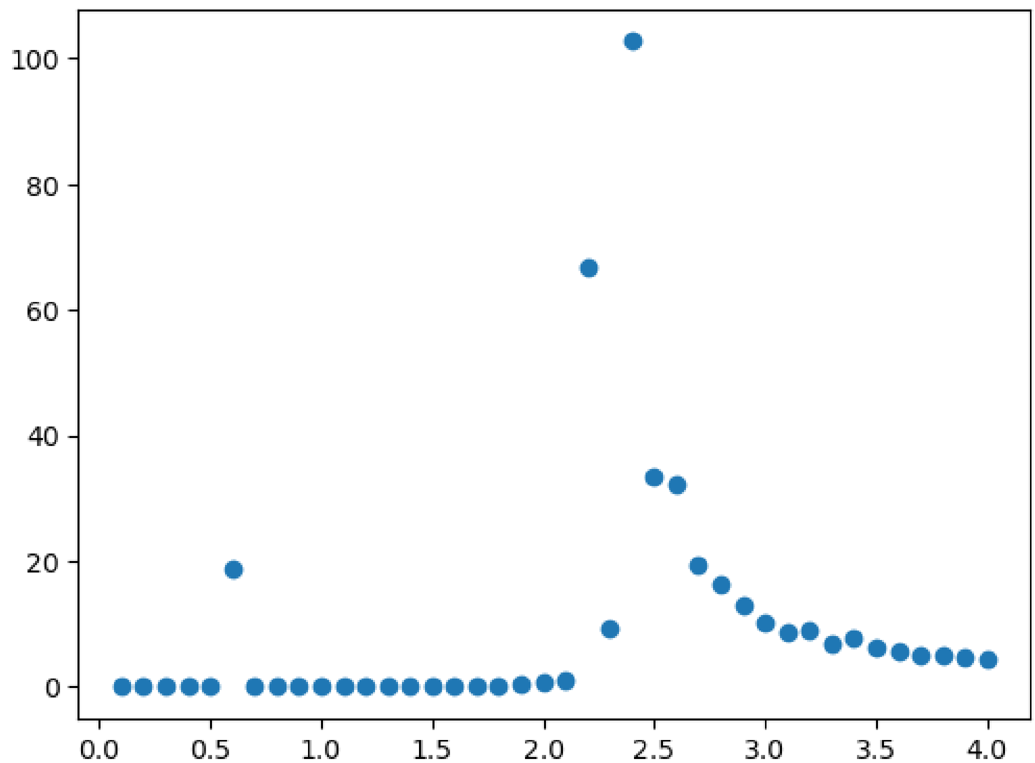
[13]:
```
fig, ax = plt.subplots()
ax.scatter(temperatures, plot_magnet)
```

[13]: <matplotlib.collections.PathCollection at 0x26f66fbf5e0>



[14]:
```
fig, ax = plt.subplots()
ax.scatter(temperatures, plot_chi)
```

[14]: <matplotlib.collections.PathCollection at 0x26f67047fa0>


100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

```
[ ]:
```