```python
# Exercise 2.2 (d).
import numpy as np
import matplotlib.pyplot as plt

# Initialize lattice
lattice_size = 200
lattice = np.sign(np.random.rand(lattice_size,lattice_size) - 0.5)
new_lattice = lattice.copy()
ten_percent = int(lattice_size*lattice_size/10)

# Constants
J = 1
iterations = 1000
H_values = np.linspace(-0.05,0.05,50)
temperatures = np.array([1,2.269,5])
magnetization_array = np.zeros([len(H_values),len(temperatures)])

for temp_i in range(len(temperatures)):

    T = temperatures[temp_i]
    beta = 1/T
    energy_step = 0

    for H_i in H_values:

        H = H_i
        m = 0
        lattice_copy = lattice.copy()

        # MC loop
        for time_step in range(iterations):

            # Update randomly 10% of the cells
            for update in range(ten_percent):

                i = np.random.randint(lattice_size)
                j = np.random.randint(lattice_size)

                M = 0

                # Due to boundaries
                if i > 0:
                    M += lattice_copy[i-1,j]
                if i < lattice_size-1:
                    M += lattice_copy[i+1,j]
                if j > 0:
                    M += lattice_copy[i,j-1]
                if j < lattice_size-1:
                    M += lattice_copy[i,j+1]

                E_plus = -H-J*M
                E_minus = H+J*M

                prob_plus = np.exp(-beta*E_plus) / (np.exp(-beta*E_plus) + np.exp(-beta*E_minus))
                rnd = np.random.rand()

                if rnd < prob_plus:
                    new_lattice[i,j] = 1
```

```python
                else:
                    new_lattice[i,j] = -1

            lattice_copy = new_lattice.copy()

        # Computing magnetization per unit volume to measure the state of the magnetic property
        m = lattice_copy.sum() / np.power(lattice_size, 2)
        magnetization_array[energy_step,temp_i] = m
        energy_step += 1

        print(H_i)

    print(temp_i)

coef_0 = np.polyfit(H_values,magnetization_array[:,0],1)
coef_1 = np.polyfit(H_values,magnetization_array[:,1],1)
coef_2 = np.polyfit(H_values,magnetization_array[:,2],1)

poly1d_0 = np.poly1d(coef_0)
poly1d_1 = np.poly1d(coef_1)
poly1d_2 = np.poly1d(coef_2)

x = np.array([0,0,0])
for i in range(len(temperatures)):
    for j in range(len(H_values)):
        x[i] += magnetization_array[j,i] / H_values[j]
x = x / len(H_values)
print(f'x = {x}')

plt.figure() # T = 1
plt.plot(H_values, magnetization_array[:,0], 'g', H_values, poly1d_0(H_values), '--k')
plt.xlabel('Magnetic field, H')
plt.ylabel('Magnetization, m')
plt.legend(['Magnetic susceptibility','Linear regression'])
plt.savefig('22dt1.png', bbox_inches='tight')

plt.figure() # T = 2.269
plt.plot(H_values, magnetization_array[:,1], 'r', H_values, poly1d_1(H_values), '--k')
plt.xlabel('Magnetic field, H')
plt.ylabel('Magnetization, m')
plt.legend(['Magnetic susceptibility','Linear regression'])
plt.savefig('22dtc.png', bbox_inches='tight')

plt.figure()  # T = 5
plt.plot(H_values, magnetization_array[:,2], 'b', H_values, poly1d_2(H_values), '--k')
plt.xlabel('Magnetic field, H')
plt.ylabel('Magnetization, m')
plt.legend(['Magnetic susceptibility','Linear regression'])
plt.savefig('22dt5.png', bbox_inches='tight')

# plt.legend(['T = 1', 'T = 2.269', 'T = 4'])
# plt.xlabel('Magnetic field, H')
# plt.ylabel('Magnetization, m')

plt.show()
```