Homework 1, Monte Carlo Simulation

Erik Norlin, 19970807-9299

November 13, 2022

2.1a

The time to reach the equilibrium state with $E = 2k_bT$ was approximately 40000 time steps most of the runs. After this time the position probability barely fluctuates. The transition frequency shows that the particle make any transitions.

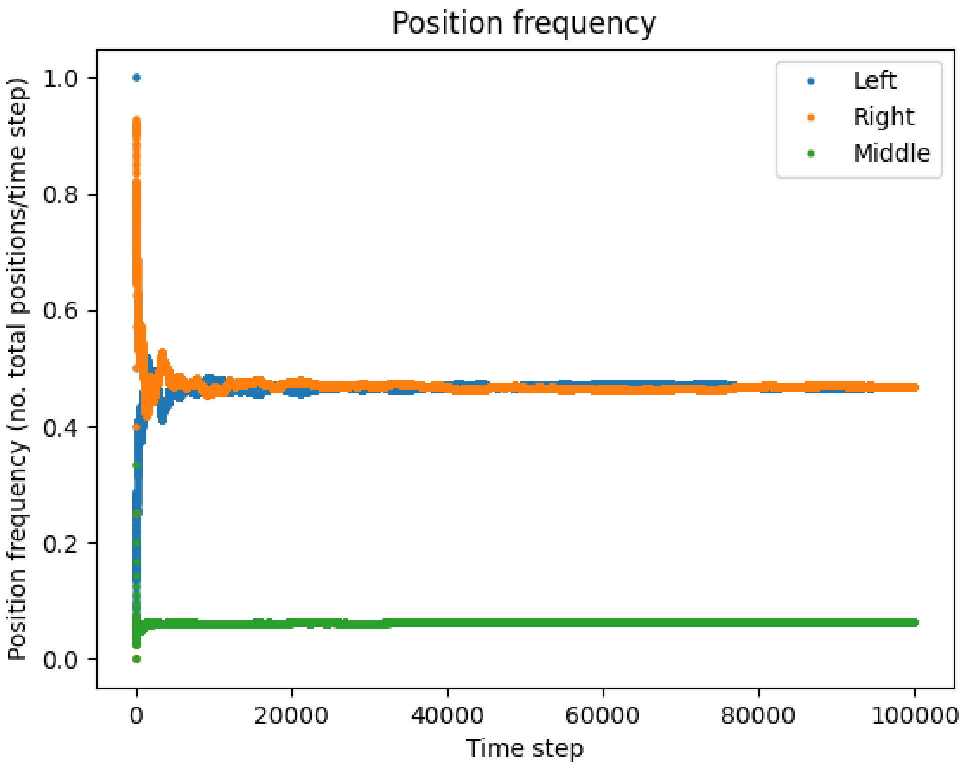

Figure 1: Position frequency (position probability) with $E = 2k_bT$.

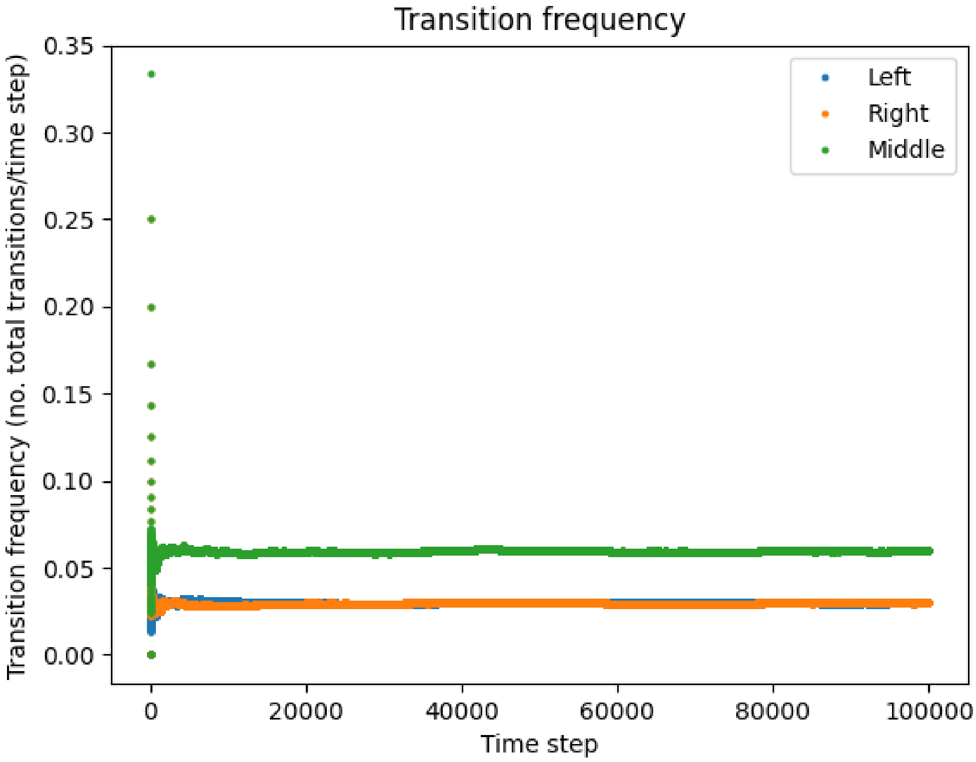

Figure 2: Transition frequency with $E = 2k_bT$.

Table 1: Probability distribution of positions and transition frequency for $E = 2k_bT$.

	Left	*Middle*	*Right*
Positions	0.46774	0.06329	0.46897
Transitions	0.02992	0.05975	0.02983

2.1b

The results show that the lower the temperature is in relation to the energy, the particle tend to transition to another state with lower probability. In contrast, for higher temperature, the particle tend to transition with higher probability. This can be seen through out the figure series. The theory supports this as well. For very high temperatures $e^{-E/k_bT} \to 1$, and for very low temperatures $e^{-E/k_bT} \to 0$. This means that the probability for transitioning to another state increases as the temperature increases.

The time to reach equilibrium varied for the different temperatures. The time to reach equilibrium for extremely low temperature was instantaneous because of the very high probability to never change state. For very high temperature, the probabilities of changing state at any state becomes almost equivalent. With approximately equal probabilities the equilibrium is reached almost instantly. In between very high and very low temperatures it takes longer to reach equilibrium, the total time can vary from run to run and depends on the temperature. For some temperatures it can take up to 60,000 iterations to reach equilibrium.

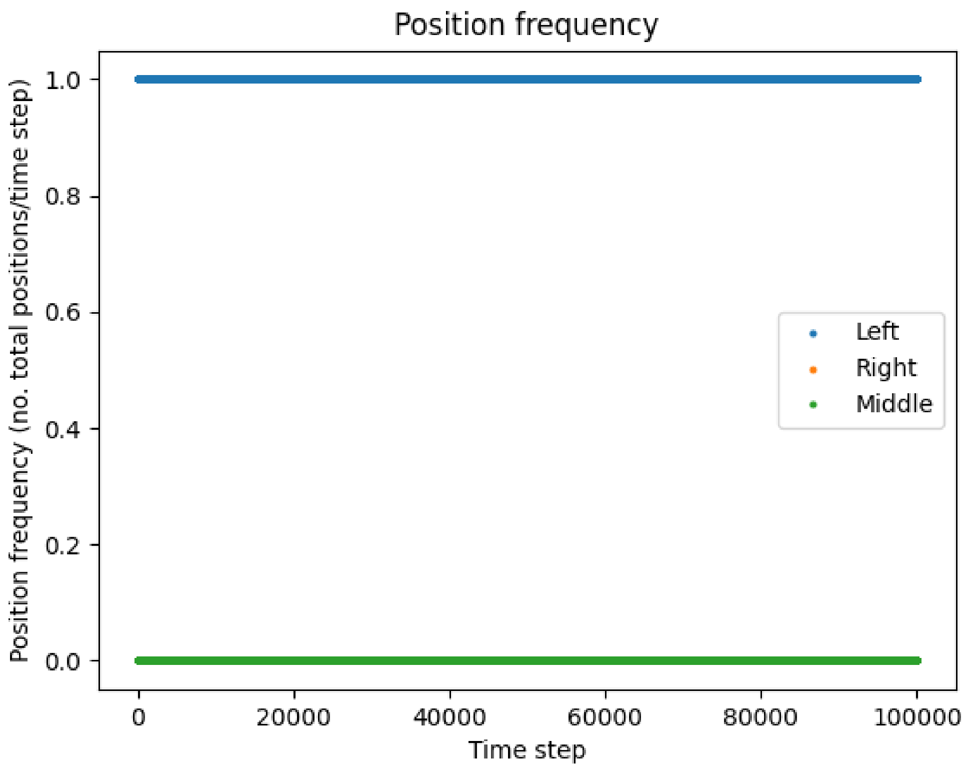

Figure 3: Position frequency (position probability) with $E = 10, T = 0.1$.

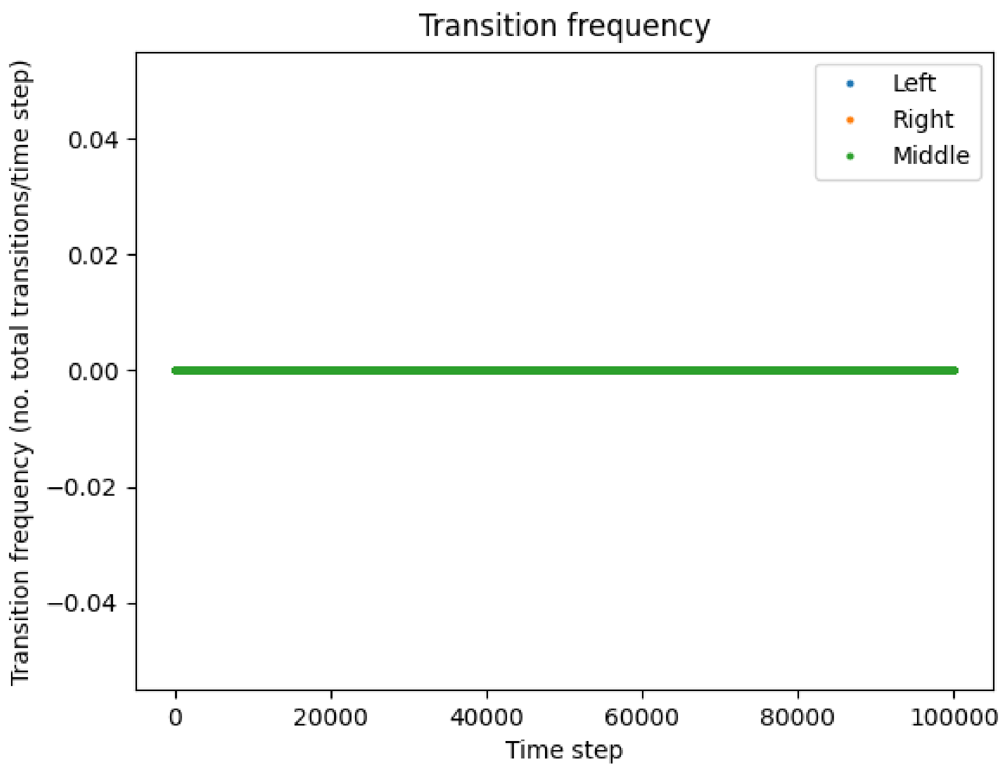

Figure 4: Transition frequency with $E = 10, T = 0.1$.

Table 2: Probability distribution of positions and transition frequency for $E = 10$, $T = 0.1$.

	$Left$	$Middle$	$Right$
Positions	1	0	0
Transitions	0	0	0

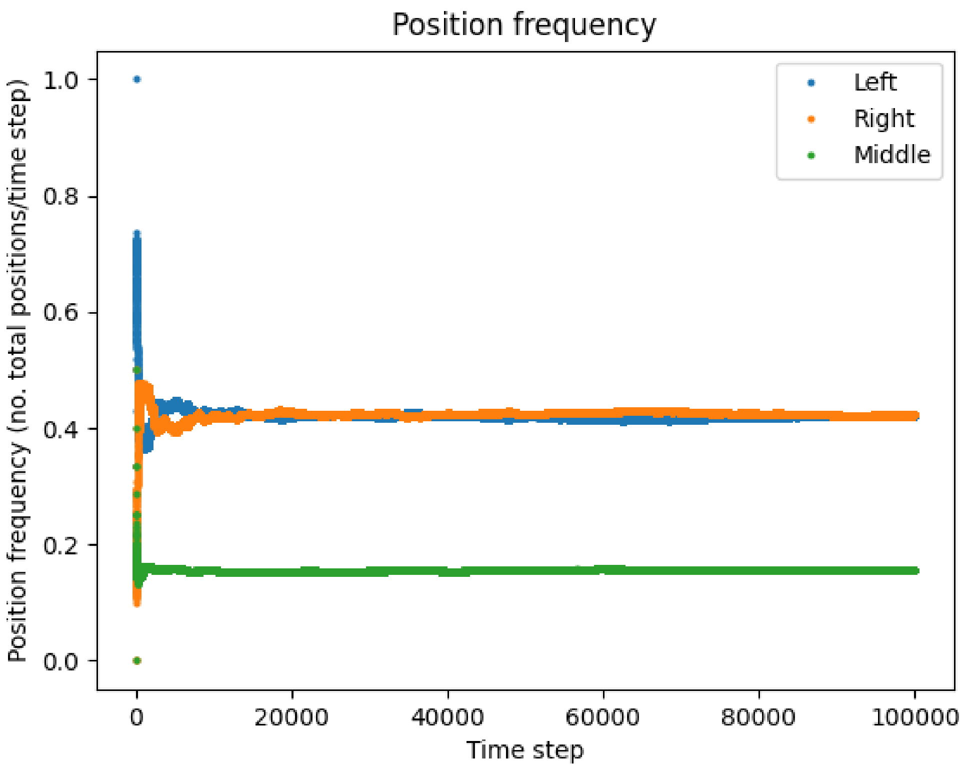

Figure 5: Position frequency (position probability) with $E = 10, T = 10$.

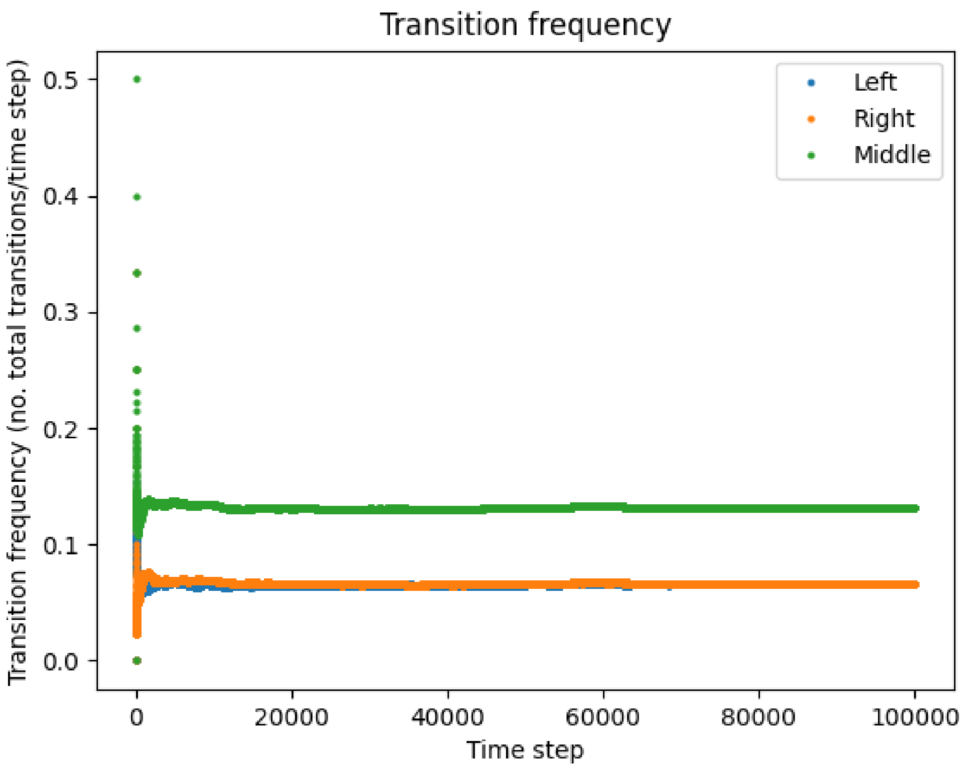

Figure 6: Transition frequency with $E = 10, T = 10$.

Table 3: Probability distribution of positions and transition frequency for $E = 10$, $T = 10$.

	Left	*Middle*	*Right*
Positions	0.42121	0.15641	0.42238
Transitions	0.06553	0.13177	0.06624

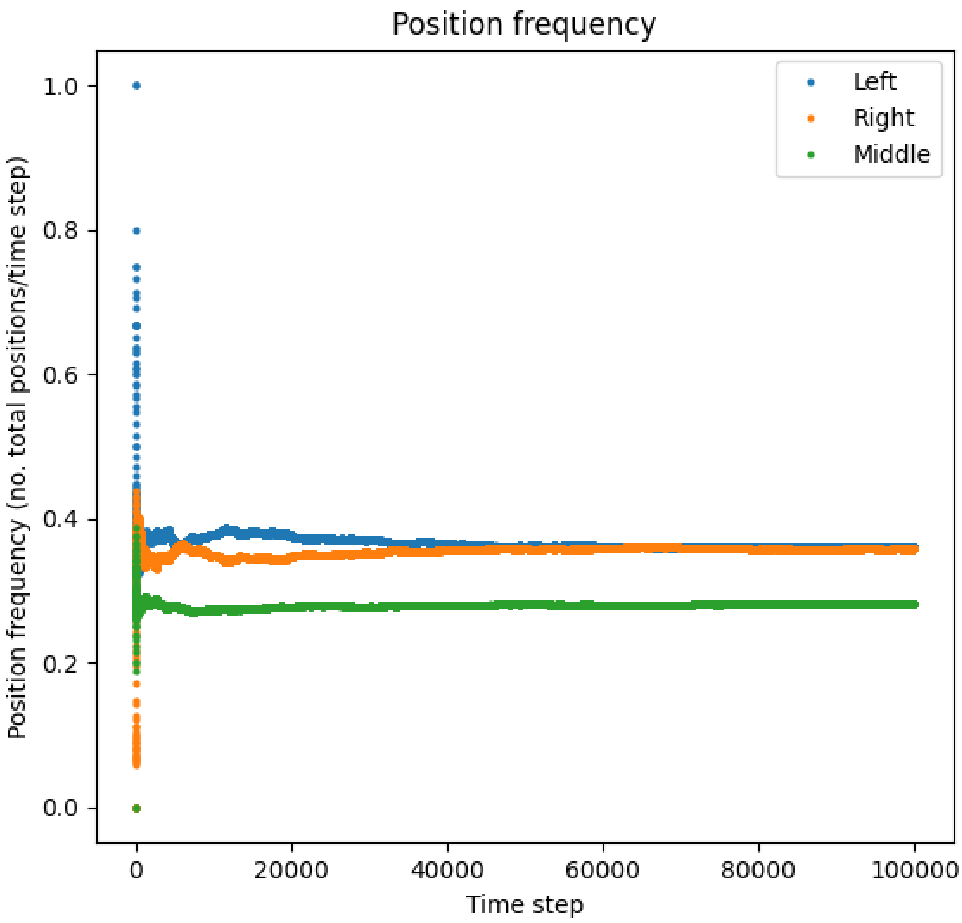

Figure 7: Position frequency (position probability) with $E = 10, T = 40$.

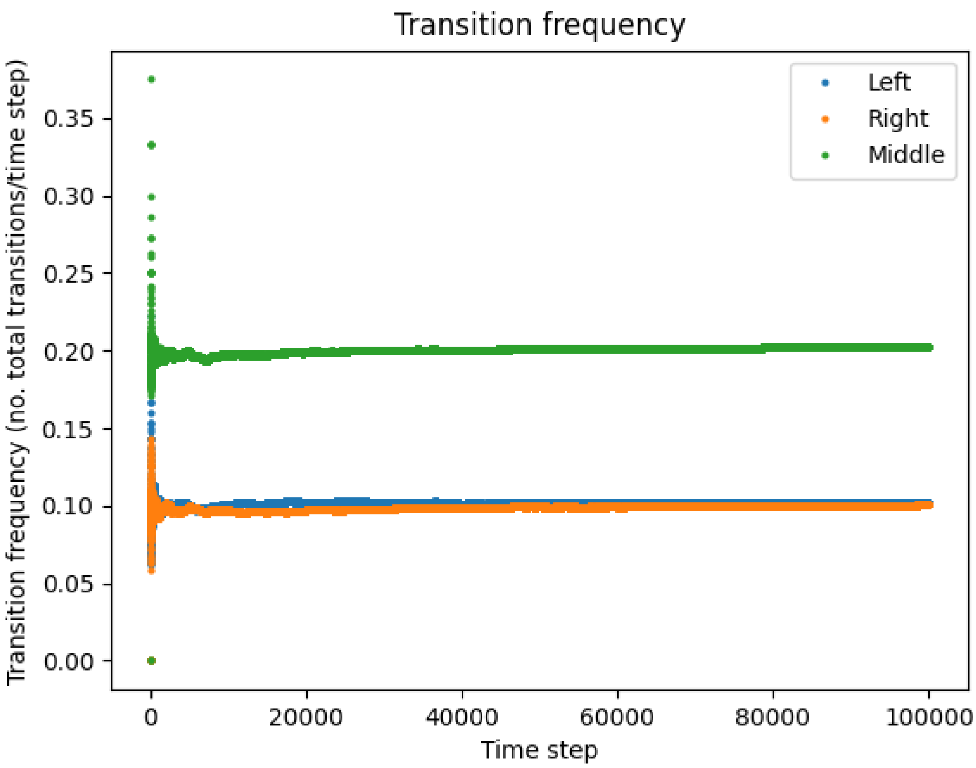

Figure 8: Transition frequency with $E = 10, T = 40$.

Table 4: Probability distribution of positions and transition frequency for $E = 40$, $T = 40$.

	Left	*Middle*	*Right*
Positions	0.3605	0.28202	0.35748
Transitions	0.10219	0.20264	0.10045

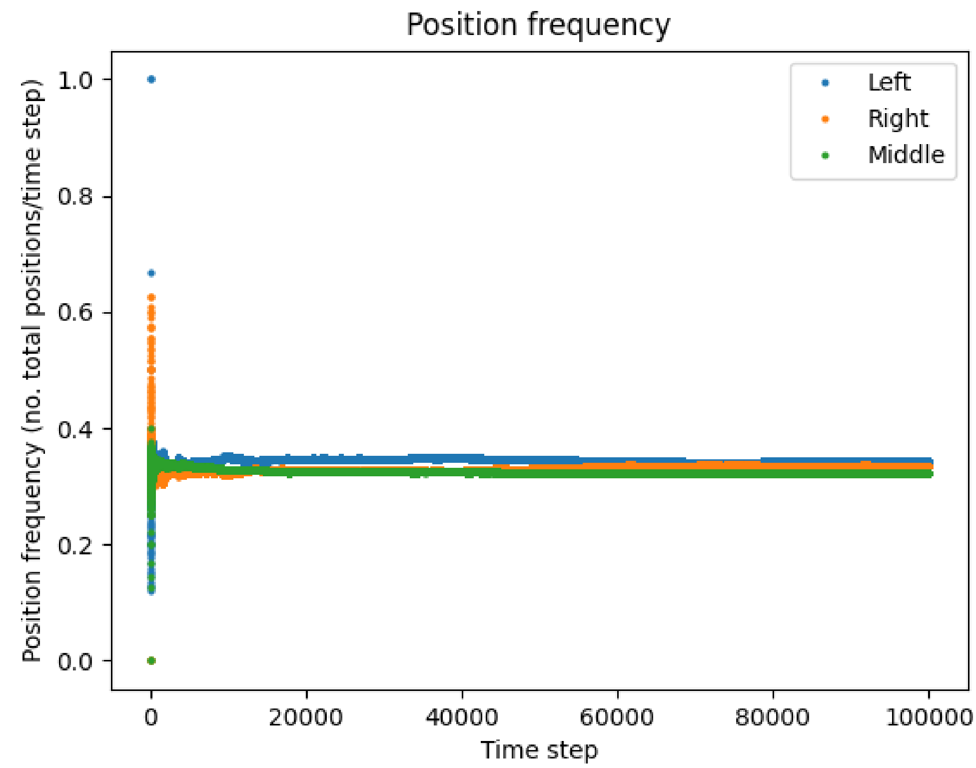

Figure 9: Position frequency (position probability) with $E = 10, T = 200$.

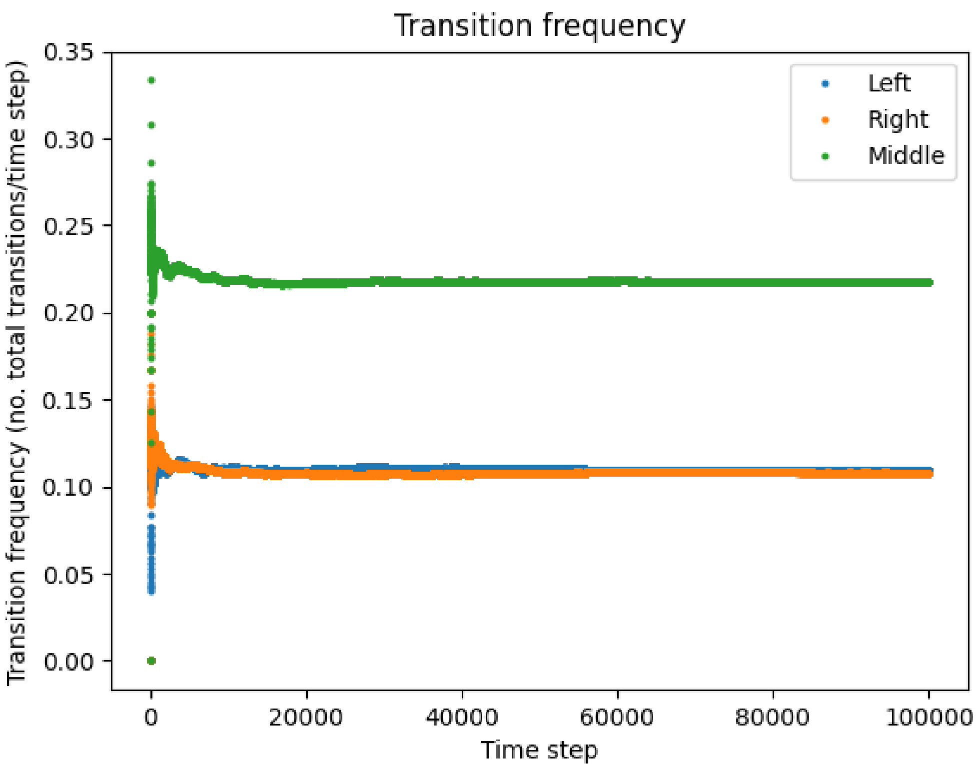

Figure 10: Transition frequency with $E = 10, T = 200$.

Table 5: Probability distribution of positions and transition frequency for $E = 10$, $T = 200$.

	Left	*Middle*	*Right*
Positions	0.34365	0.32168	0.33467
Transitions	0.10969	0.21727	0.10758

2.2b and c

We can see in the following figure that clusters of the same spins form at a subcritical temperature, that clusters don't form at a supercritical temperature and also that weaker defined clusters form at the critical temperature. Clusters of the same spins form in subcritical temperatures because the probability that a spin to be in a certain direction (+1 or -1) is larger than the opposite direction. Since the direction of every spin is probabilistically determined by the directions of the neighbouring spins, over time this results in clustering of the same spins. The probability of a spin being in a certain direction in supercritical temperatures become more random as the temperature increases. Instead of formed clusters, the direction of the spins results in a more stochastic nature, as one can see in the last row in the following figure. In metals, this is referred to as a *paramagnetic* property if the metal is subjected to a magnetic field without formation of clustering spins. This means that the metal react weakly to magnetic fields. The paramagnetic property can be observed in the spins in the simulation at a supercritical temperature T = 6 when the model is subjected to a magnetic field of H = 1. We can see that no clusters of the same spins are formed. However, the magnetic field still has an impact and shows a slight domination of yellow (+1) spins but there are still no clustering of the spins.

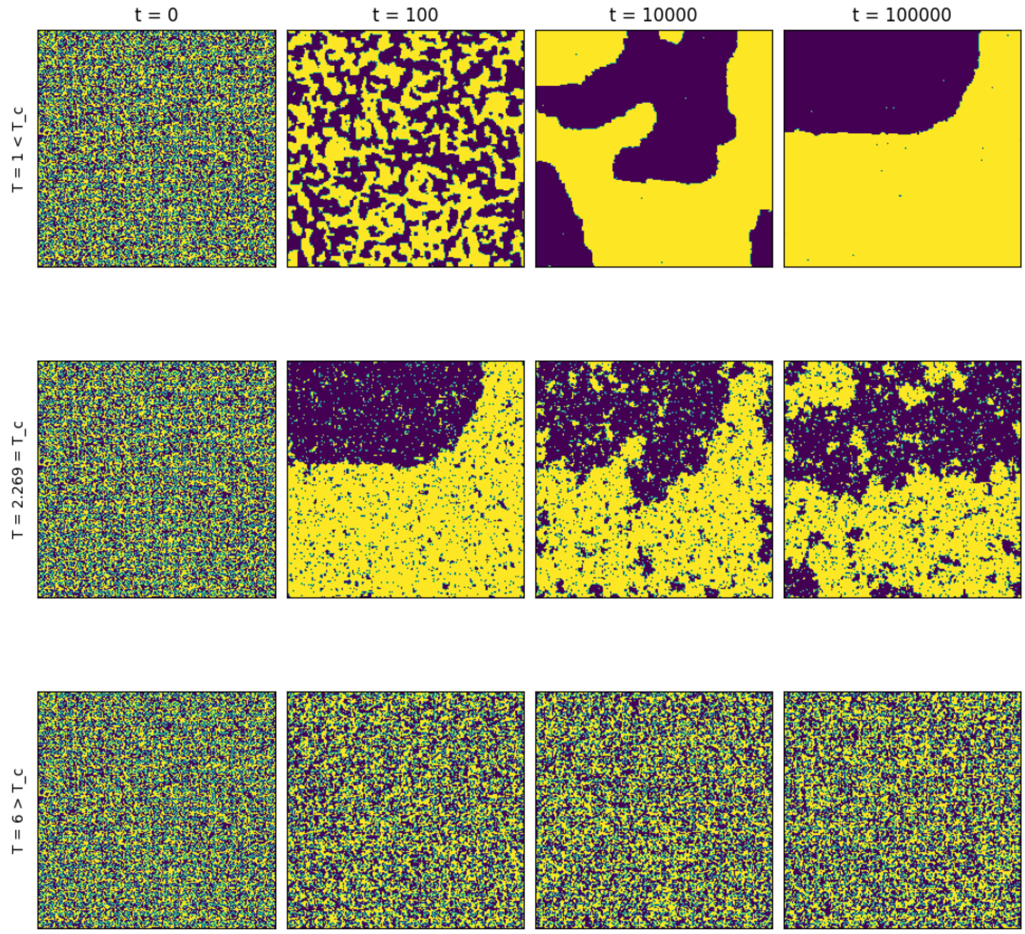

Figure 11: Spins after 100,000 iterations. Top row T = 1, middle row T = 2.269 and bottom row T = 6.

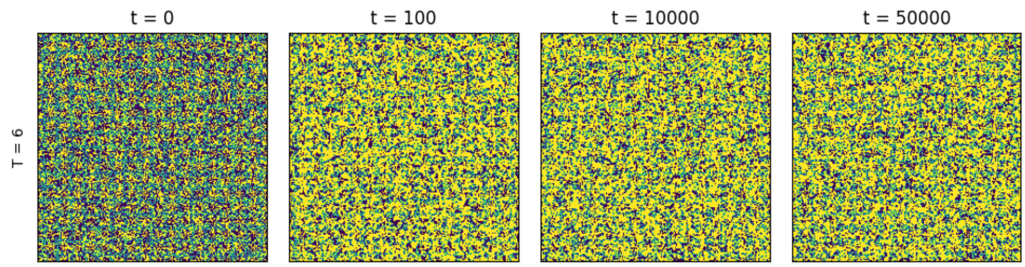

Figure 12: Paramagnetism T = 6, H = 1, 50,000 iterations.

2.2d

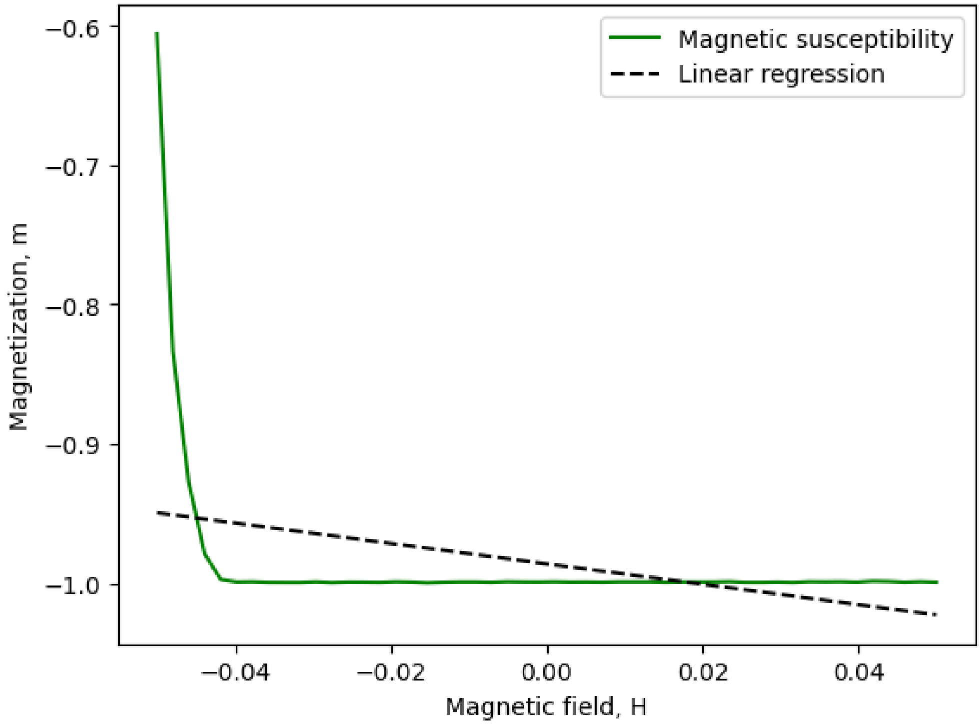

Figure 13: Spins after 1000 iterations. For T = 1, X = -0.66.

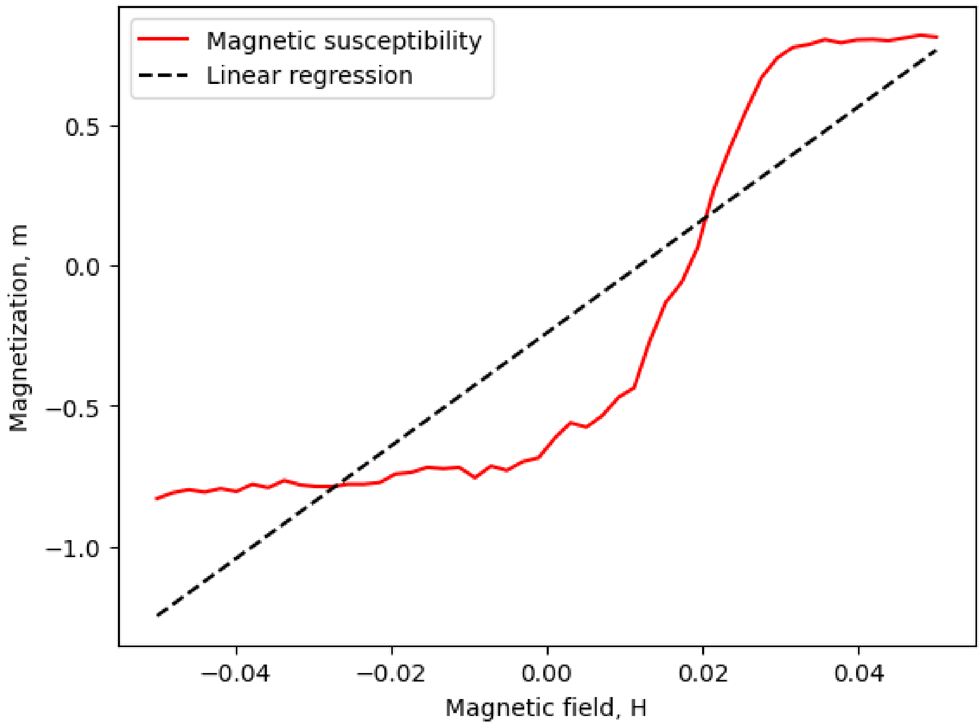

Figure 14: Spins after 1000 iterations. For T = 2.269, X = 20.2.

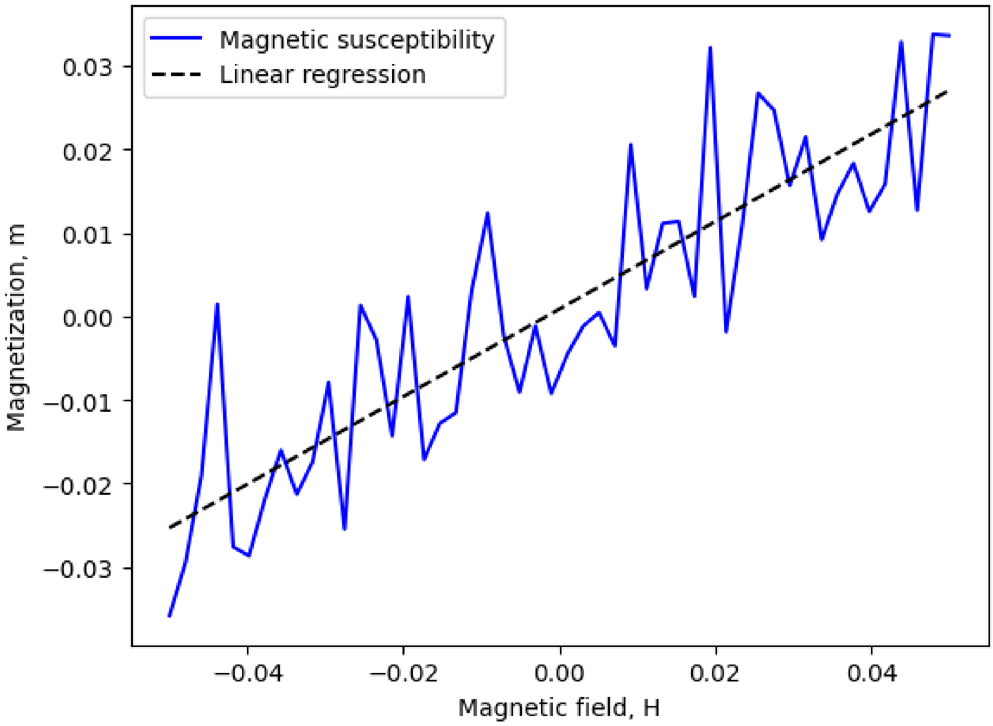

Figure 15: Spins after 1000 iterations. For T = 5, X = 0.12.

```python
# Exercise 2.1 (a) and (b).

import numpy as np
import matplotlib.pyplot as plt

left_index = 0
middle_index = 1
right_index = 2

# Exercise (a), E = 2*k*T
prob_left = 1 / (2 + np.exp(-2))
prob_right = prob_left
prob_middle = np.exp(-2) / (2 + np.exp(-2))

# Exercise (b), E != 2*k*T
E = 10
k = 1
T = 200
prob_left = 1 / (2 + np.exp(-E/(k*T)))
prob_right = prob_left
prob_middle = np.exp(-E/(k*T)) / (2 + np.exp(-E/(k*T)))

no_steps = np.power(10,5) - 1
iterations = np.linspace(1,no_steps+1,no_steps+1)

position = "left"

position_counter = np.array([1,0,0]) # Left, middle, right
transition_counter = np.array([0,0,0]) # Left, middle, right

position_index = 0
transition_index = 1

left_stat = np.zeros((no_steps+1, 2))
right_stat = np.zeros((no_steps+1, 2))
middle_stat = np.zeros((no_steps+1, 2))
left_stat[0, position_index] = 1

for i in range(no_steps):

    # MC simulating new position for the unit
    rand = np.random.uniform()
    if rand < prob_left:
        new_position = "left"
    elif rand < (prob_left + prob_right):
        new_position = "right"
    elif rand < (prob_middle + prob_left + prob_right):
        new_position = "middle"

    # Assigning the unit to the new position if the conditon allows
    if new_position == "left":
        if position == "middle" or position == "left":
            if position == "middle":
                transition_counter[left_index] += 1
            position = new_position
            position_counter[left_index] += 1
        else:
            position = "right"
            position_counter[right_index] += 1
```

```python

    elif new_position == "right":
        if position == "middle" or position == "right":
            if position == "middle":
                transition_counter[right_index] += 1
            position = new_position
            position_counter[right_index] += 1
        else:
            position = "left"
            position_counter[left_index] += 1

    elif new_position == "middle":
        if position != "middle":
            transition_counter[middle_index] += 1
        position = "middle"
        position_counter[middle_index] += 1

    # Stats for position frequency
    no_left = position_counter[left_index]
    left_stat[i + 1, position_index] = no_left / (i + 2)

    no_right = position_counter[right_index]
    right_stat[i + 1, position_index] = no_right / (i + 2)

    no_middle = position_counter[middle_index]
    middle_stat[i + 1, position_index] = no_middle / (i + 2)

    # Stats for transition frequency
    no_left = transition_counter[left_index]
    left_stat[i + 1, transition_index] = no_left / (i + 2)

    no_right = transition_counter[right_index]
    right_stat[i + 1, transition_index] = no_right / (i + 2)

    no_middle = transition_counter[middle_index]
    middle_stat[i + 1, transition_index] = no_middle / (i + 2)


position_distribution = position_counter / (no_steps + 1)
transition_distribution = transition_counter / (no_steps + 1)

# Output distributions
print(f'Probability distribution of positions: {position_distribution}')
print(f'Probability distribution of transition: {transition_distribution}')

fig, axs = plt.subplots(1,2)

# Plotting position frequency
axs[0].plot(iterations, left_stat[:,position_index], 'o', markersize=2)
axs[0].plot(iterations, right_stat[:,position_index], 'o', markersize=2)
axs[0].plot(iterations, middle_stat[:,position_index], 'o', markersize=2)
axs[0].set_title('Position frequency')
axs[0].set_xlabel('Time step')
axs[0].set_ylabel('Position frequency (no. positions/time step)')
axs[0].legend(['Left', 'Right', 'Middle'])
axs[0].set_box_aspect(1)

axs[1].plot(iterations, left_stat[:,transition_index], 'o', markersize=2)
axs[1].plot(iterations, right_stat[:,transition_index], 'o', markersize=2)
axs[1].plot(iterations, middle_stat[:,transition_index], 'o', markersize=2)
```

```
axs[1].set_title('Transition frequency')
axs[1].set_xlabel('Time step')
axs[1].set_ylabel('Transition frequency (no. transitions/time step)')
axs[1].legend(['Left', 'Right', 'Middle'])
axs[1].set_box_aspect(1)

fig.suptitle('2.1a: E = 2kT', fontsize=16)
plt.show()
```

```python
# Exercise 2.2 (a) and (b).
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
H = 0
iterations = 100000

fig, axs = plt.subplots(3, 4, figsize=(12,12))
time_0 = lattice.copy()
temperatures = np.array([1, 2.269, 6])

# Performing MC over 3 different temperatures
for temp in range(len(temperatures)):

    T = temperatures[temp]
    beta = 1/T
    lattice = time_0.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[i-1,j]
            if i < lattice_size-1:
                M += lattice[i+1,j]
            if j > 0:
                M += lattice[i,j-1]
            if j < lattice_size-1:
                M += lattice[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
            else:
                new_lattice[i,j] = -1

```

```python
        lattice = new_lattice.copy()

        # Snapshots of certain time steps
        if time_step == 100-1:
            time_1 = lattice.copy()
        elif time_step == 10000-1:
            time_2 = lattice.copy()
        elif time_step == 100000-1:
            time_3 = lattice.copy()

    # Plotting
    axs[temp,0].imshow(time_0)
    axs[temp,0].yaxis.set_ticks([])
    axs[temp,0].xaxis.set_ticks([])

    axs[temp,1].imshow(time_1)
    axs[temp,1].yaxis.set_ticks([])
    axs[temp,1].xaxis.set_ticks([])

    axs[temp,2].imshow(time_2)
    axs[temp,2].yaxis.set_ticks([])
    axs[temp,2].xaxis.set_ticks([])

    axs[temp,3].imshow(time_3)
    axs[temp,3].yaxis.set_ticks([])
    axs[temp,3].xaxis.set_ticks([])

# Plotting
axs[0,0].set_ylabel('T = 1 < T_c')
axs[1,0].set_ylabel('T = 2.269 = T_c')
axs[2,0].set_ylabel('T = 6 > T_c')

axs[0,0].set_title('t = 0')
axs[0,1].set_title('t = 100')
axs[0,2].set_title('t = 10000')
axs[0,3].set_title('t = 100000')

plt.subplots_adjust(wspace=0.05, hspace=0.05)
plt.savefig('22ab.png', bbox_inches='tight')
plt.show()
```

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

# 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
T = 6
H = 1
beta = 1/T
iterations = 50000
magnetization_array = np.zeros((1,iterations))
energies = np.array([0.1, 0.2, 0.3, 0.4])

# Animation
fig1, ax = plt.subplots()
ims = []
im = ax.imshow(lattice.copy())
ims.append([im])

# Plotting
fig2, axs = plt.subplots(1, 4, figsize=(12,12))
time_0 = lattice.copy()

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[i-1,j]
        if i < lattice_size-1:
            M += lattice[i+1,j]
        if j > 0:
            M += lattice[i,j-1]
        if j < lattice_size-1:
            M += lattice[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
```

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

    lattice = new_lattice.copy()

     # Snapshots of certain time steps
    if time_step == 100-1:
        time_1 = lattice.copy()
    elif time_step == 10000-1:
        time_2 = lattice.copy()
    elif time_step == 50000-1:
        time_3 = lattice.copy()

    # Computing magnetization per unit volume to measure the state of the magnetic
property
    m = 0
    for i in range(lattice_size):
        for j in range(lattice_size):
            m += (lattice[i,j] / np.power(lattice_size, 2))

    magnetization_array[0,time_step] = m

    # Images for animation
    # im = ax.imshow(lattice.copy(), animated=True)
    # ims.append([im])

    if time_step % 100 == 0:
        print(time_step)

# Plotting
axs[0].imshow(time_0)
axs[0].yaxis.set_ticks([])
axs[0].xaxis.set_ticks([])

axs[1].imshow(time_1)
axs[1].yaxis.set_ticks([])
axs[1].xaxis.set_ticks([])

axs[2].imshow(time_2)
axs[2].yaxis.set_ticks([])
axs[2].xaxis.set_ticks([])

axs[3].imshow(time_3)
axs[3].yaxis.set_ticks([])
axs[3].xaxis.set_ticks([])

axs[0].set_ylabel('T = 6')
axs[0].set_title('t = 0')
axs[1].set_title('t = 100')
axs[2].set_title('t = 10000')
axs[3].set_title('t = 50000')

plt.subplots_adjust(wspace=0.1, hspace=0.05)
plt.savefig('22c.png', bbox_inches='tight')

# ani = animation.ArtistAnimation(fig1, ims, interval=5, blit=True)
# writergif = animation.PillowWriter(fps=30)
# ani.save("22c.gif", writer=writergif)

plt.axis('off')
```

```
plt.show()
```

```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
```

```
                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()
```