Problem 1

November 26, 2023

1 1D Ising model

Write a program that implements the Metropolis algorithm for single spin flips in a 1D Ising model of N spins producing a new spin configuration X_{k+1} from the present spin configuration X_k. Use the canonical ensemble for a heat bath of temperature T. Set up periodic boundary conditions (effectively bending the chain into circle, such that the first and the last spin are adjacent to each other). Choose units such that $J = 1$. The thermal energy $k_B T$ is given in units of J.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
seeds = np.load("seeds_2000x100.npy")
```

[2]:
```
#generating 100 lists 2000 seeds
#seeds = []
#for i in range(100):
#    seeds_i = np.random.randint(0, 100000, size=2000)
#    seeds.append(seeds_i)
#seeds = np.array(seeds)
#np.save("seeds_2000x100", seeds)
```

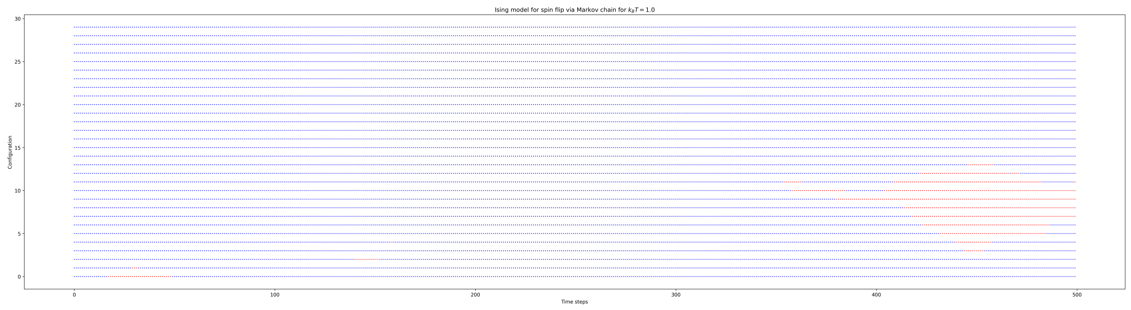
a) Set $N = 30$, and $k_B T = 1$ and simulate $L = 500$ individual trial spin flips. Begin from a "cold" initial state, where all spins are pointing to the same direction. Set the external magnetic field to $H = 0$. Visualise how the spin configuration changes with the number of trial spin flips (i.e. with time). It might be advantageous for the visualisation to show the configuration only after every 5th trial or so. What can you observe? (4 points)

[7]:
```
# canonical ensemble

def energy(config, H_external=0):
    J = 1
    H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +␣
 ↪config[0]*config[-1]
    H_ext = H_external * np.sum(config)
    return -(J*H+H_ext)

def metropolis_algorithm(initial_state, num_trials, beta=1, H_external=0,␣
 ↪list_seeds=None):
    sequence_states = [initial_state]
    for n in range(num_trials):
        sd = list_seeds[n]
        next_sequence = metropolis_step(sequence_states[-1], beta, H_external,␣
 ↪seed = sd)
        sequence_states.append(next_sequence)
    return np.array(sequence_states), np.arange(num_trials+1) #sequences and␣
 ↪corresponding time steps

def metropolis_step(current_state, beta, H_external, seed = None):
    np.random.seed(seed)
    #propose_prob = 1/len(list(current_state))
    spin_flip_index = np.random.randint(0,len(list(current_state)))
    trial_configuration = list(current_state)
    trial_configuration[spin_flip_index] *= -1
    energy_diff = energy(trial_configuration, H_external) -␣
 ↪energy(current_state, H_external)
    trial_acceptance_prob = np.min([np.exp(-beta * energy_diff),1])
    #r_propose = np.random.rand()
    r_accept = np.random.rand()
    if r_accept < trial_acceptance_prob:
        new_state = trial_configuration
    else:
        new_state = current_state
    return np.array(new_state)
```

[8]:
```
N = 30
n_trials = 500
X_0 = np.ones(N)
states, t = metropolis_algorithm(X_0, n_trials, list_seeds=seeds[0])
```

```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```

[12]:
```
def plot_time_evolution_1d_ising(states, beta=1):
    fig, ax = plt.subplots(figsize=(40, 10), dpi=300)
    for t in range(n_trials):
        scatter = ax.scatter(t*np.ones(N), np.arange(N), marker= "o", s= 0.5, c␣
 ↪= ["blue" if s==1 else "red"  for s in states[t]])
    plt.title(f"Ising model for spin flip via Markov chain for $k_B T={1/
 ↪beta}$")
    ax.set_ylabel("Configuration")
    ax.set_xlabel("Time steps")
    #legend = ax.legend(*scatter.legend_elements(), title="Classes") #Todo: Fix␣
  ↪legend
    #ax.add_artist(legend)
    #plt.show()

plot_time_evolution_1d_ising(states)
plt.savefig(f"Ising_spinflip.png", dpi = 400)
```

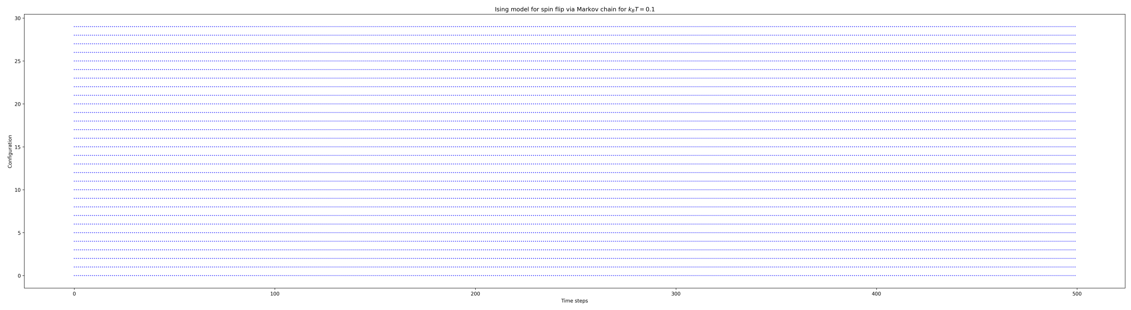


Observation: Small domains of spins with $s_i = -1$, but mostly the spins stay the same. This is due to the low temperature, making spin flips unlikely. Once a spin flip happens, the probability for new spin flips neaerby is increased. Therefore, often more than one spin flips in the same region.

Repeat the same simulation for three different thermal energies $k_BT = 0.1, 1$ and 10 and keep track of the energy E at each trial spin flip. For a given configuration $X = s_1, ..., s_N$, it is given by $E = -J\sum_{i=1}^{N} s_i s_{i+1}$ with $s_{N+1} = s_1$ due to the periodic boundary condition. Note that we still assume $H = 0$ here, such that the terms involving the interaction with the external field are omitted here. First run the simulation for at least $L = 1000$ trial spin flips. Plot the time evolution of the energy for each temperature. Your result will be strongly fluctuating. Improve it by repeating each simulation $M = 100$ times, each time starting with a different random seed. Then plot the time evolution of the mean energy $\langle E\rangle$ and its Monte Carlo error estimate $\sqrt{(\langle E^2\rangle - \langle E\rangle^2)/M}$. Discuss your result. When is equilibrium reached? What can you observe after the system has reached equilibrium? (4 points

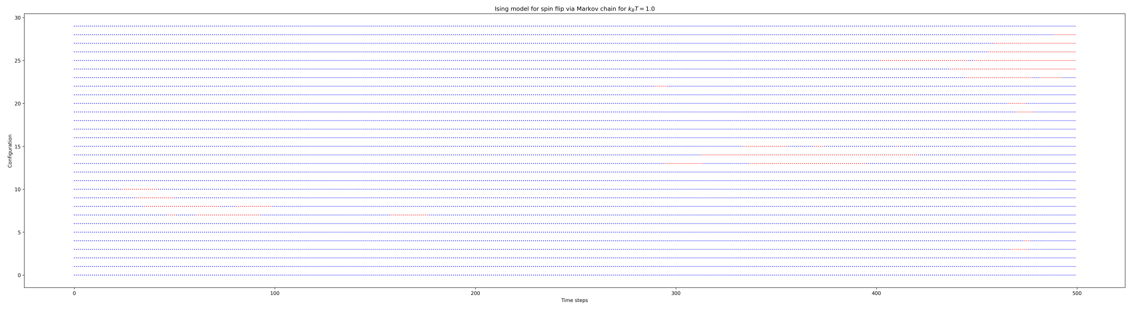
[13]:
```
#Getting beta from k_B*T:
list_beta = [10,1,0.1]

states_10, t10 = metropolis_algorithm(X_0, 1000, beta= list_beta[0],␣
  ↪list_seeds=seeds[42])
list_energy_10 = np.array([energy(state) for state in states_10])
plot_time_evolution_1d_ising(states_10, list_beta[0])
plt.savefig(f"Ising_spinflip_beta10.png", dpi = 400)
```

```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```



```
[14]: states_1, t1 = metropolis_algorithm(X_0, 1000, beta= list_beta[1],␣
       ↪list_seeds=seeds[42])
      plot_time_evolution_1d_ising(states_1, list_beta[1])
      list_energy_1 = np.array([energy(state) for state in states_1])
      plt.savefig(f"Ising_spinflip_beta1.png", dpi = 400)
```

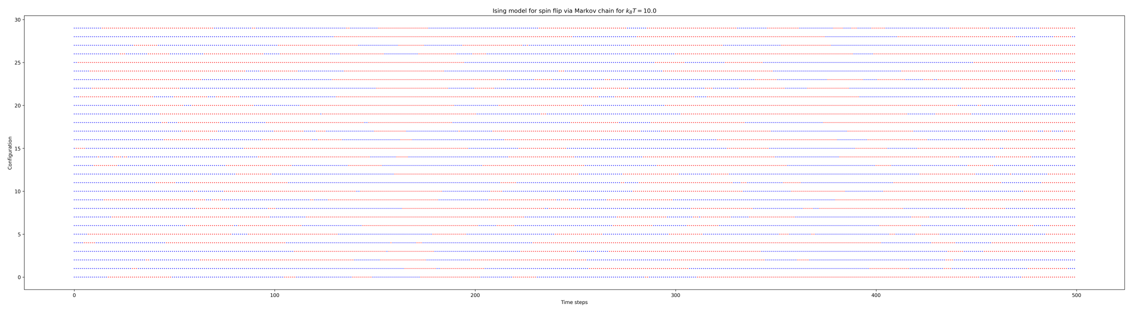
```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```



```
[15]: states_2, t2 = metropolis_algorithm(X_0, 1000, beta= list_beta[2],␣
       ↪list_seeds=seeds[0])
      plot_time_evolution_1d_ising(states_2, list_beta[2])
      list_energy_2 = np.array([energy(state) for state in states_2])
      plt.savefig(f"Ising_spinflip_beta2.png", dpi = 300) #beta = 0.1, kbT = 10
```

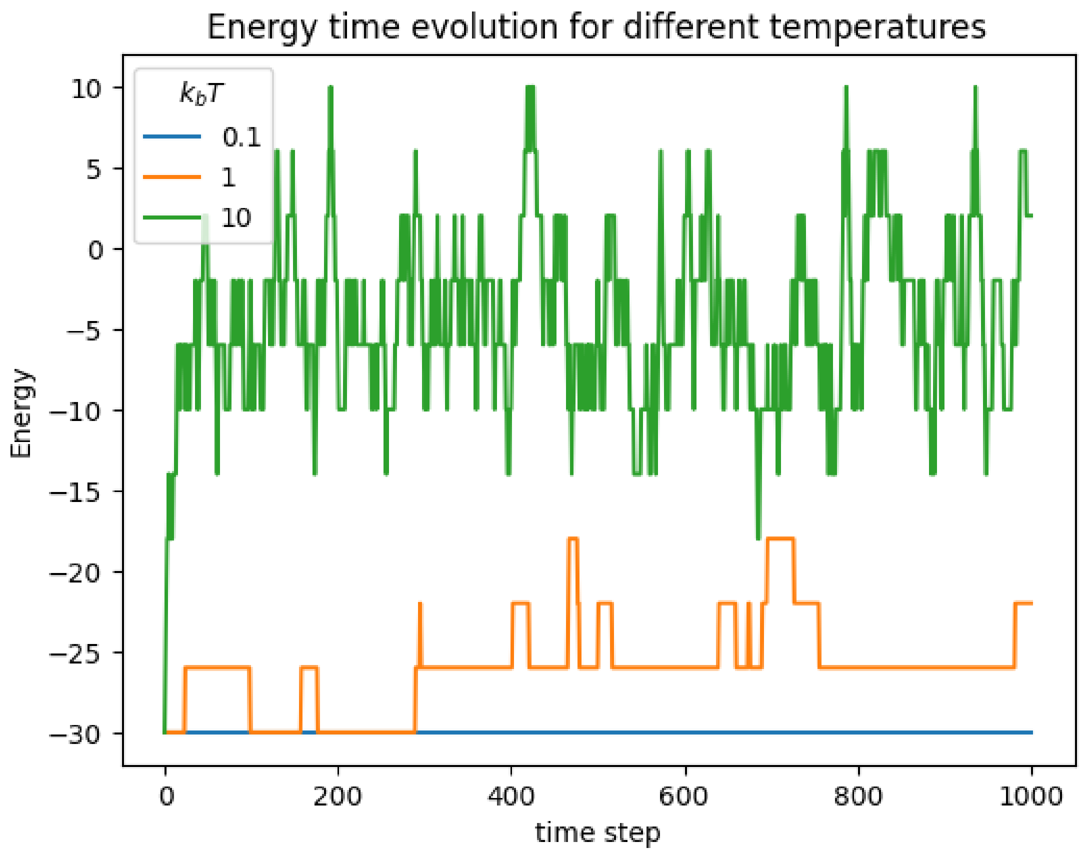
```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
```

```
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```



```
[18]: plt.plot(t10, list_energy_10, label="0.1")
      plt.plot(t1, list_energy_1, label="1")
      plt.plot(t2, list_energy_2, label="10")
      plt.legend(title="$k_{B} T$")
      plt.title("Energy time evolution for different temperatures")
      plt.xlabel("time step")
      plt.ylabel("Energy")
```

```
[18]: Text(0, 0.5, 'Energy')
```

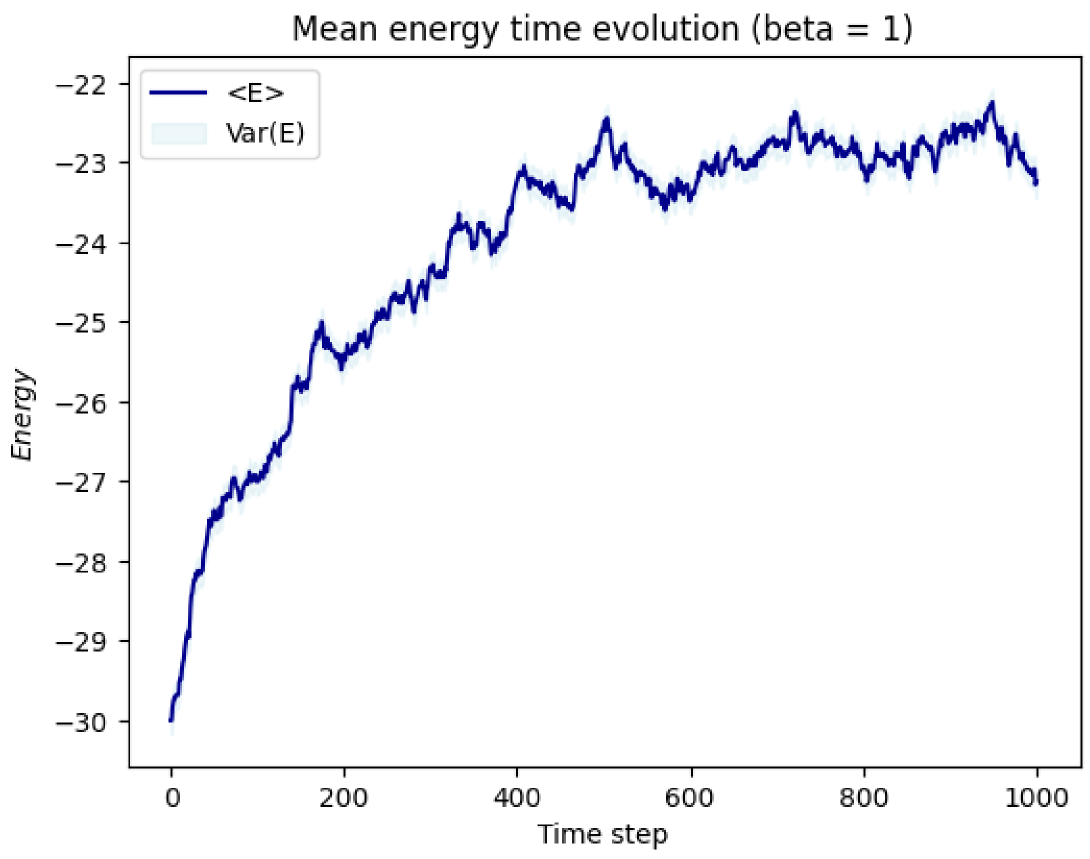


[25]:
```python
M = 100
list_energies = []
#list_errors = []

def mc_error(list_energy):
    E_mean = np.mean(list_energy)
    E_mean_squared = np.mean([E**2 for E in list_energy])
    return np.sqrt((E_mean_squared-E_mean**2)/M)

for m in range(M):
    states_m, _ = metropolis_algorithm(X_0, 1000, 1, list_seeds=seeds[m])
    energies_m = [energy(state) for state in states_m]
    errors_m = [mc_error(state) for state in states_m]
    list_energies.append(energies_m)
    #list_errors.append(errors_m)

list_energies = np.array(list_energies)
#list_errors = np.array(list_errors)
avg_energies = np.mean(list_energies, axis=0)
```

```
list_mc_errors = mc_error(avg_energies)
plt.plot(np.arange(1001), avg_energies, color="darkblue", label= "<E>")
plt.fill_between(np.arange(1001), avg_energies - list_mc_errors, avg_energies +␣
 ↪list_mc_errors,
                 color='lightblue', alpha=0.2, label="Var(E)")
plt.title("Mean energy time evolution (beta = 1)")
plt.xlabel("Time step")
plt.ylabel("$Energy$")
plt.legend()
```

```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```

[25]:
```
<matplotlib.legend.Legend at 0x29bafd26890>
```



We observe that, after some time, equilibrium will be reached ($< E >\approx const.$). After reaching equilibrium, there will be only random fluctuations which cancel out on average.

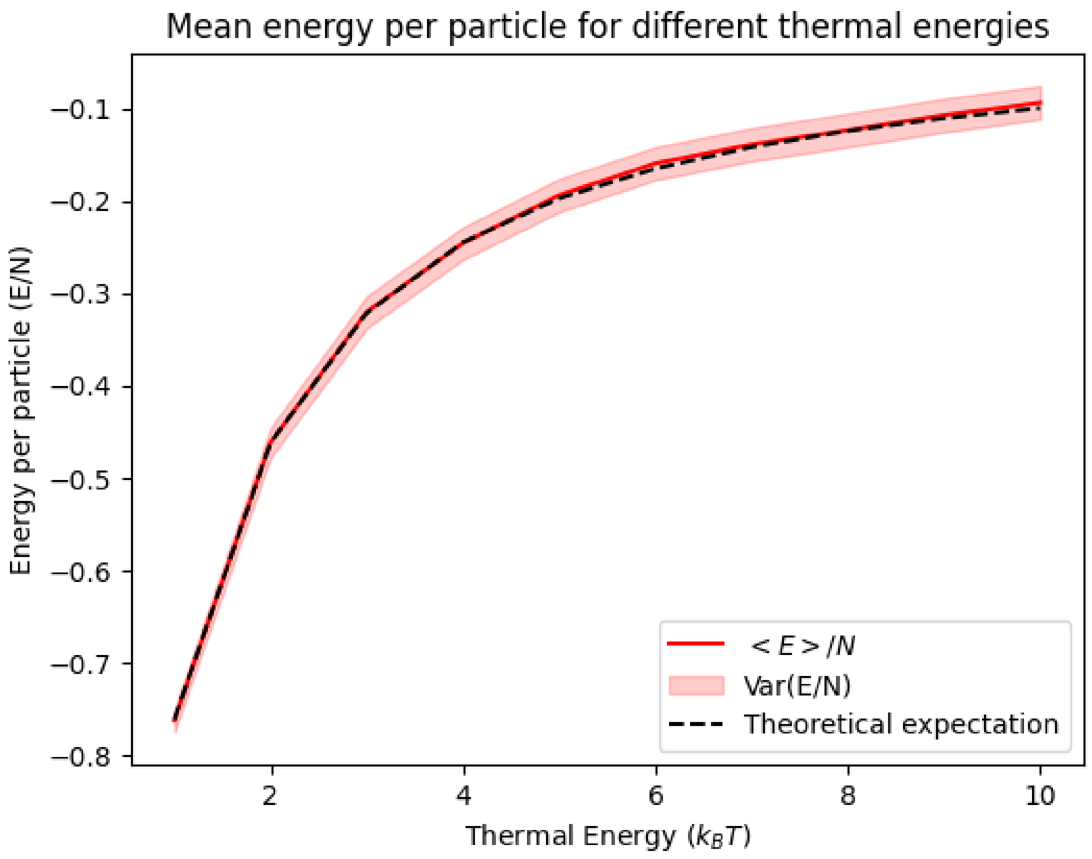
Make a plot of the mean energy per particle $\frac{1}{N}\langle E\rangle_t$ (averaged over simulation time t, e.g. over $L = 1000$ trial spin flips) versus the thermal energy k_BT after equilibrium has been reached (i.e. ignoring the first $L \sim 1000$ trial spin flips). Choose $k_BT = 1...10$ and average over $M = 100$ independent simulations to obtain a smooth result. Compare to the analytical result for the thermodynamic limit and discuss your results. (4 points)

[41]:
```
M = 100

def specific_heat(list_energy, beta):
    E_mean = np.mean(list_energy)
    E_squared_mean = np.mean([E**2 for E in list_energy])
    k_B = 1.380649 * 10**(-23)
    T = 1/(beta*k_B)
    return (E_squared_mean - E_mean**2)/(k_B*T**2)

def energy_theoretical(beta, J=1):
    return -J*np.tanh(J*beta)

list_beta = 1/np.arange(1, 11)

energies = {}
energy_errors = {}
cv = {}

for beta in list_beta:
    list_energies = []
    for m in range(M):
        states_m, _ = metropolis_algorithm(X_0, 2000, beta, list_seeds=seeds[m])
        states_m = states_m[-1000:,:] #Keep only the equilibrium states
        energies_m = [energy(state) for state in states_m]
        errors_m = [mc_error(state) for state in states_m]
        list_energies.append(energies_m)
    list_energies = np.array(list_energies)
    E_mean = np.mean(list_energies/N)
    list_cv = specific_heat(list_energies, beta)
    energies[beta] = E_mean
    energy_errors[beta] = mc_error(list_energies)/N
    cv[beta] = list_cv
```

```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```

[42]:
```
list_energies_theor = np.array([energy_theoretical(b) for b in list_beta])
plt.plot(np.arange(1,11), list(energies.values()), "red", label = "$<E>/N$")
plt.fill_between(np.arange(1,11), np.array(list(energies.values()))-np.
 ↪array(list(energy_errors.values())), np.array(list(energies.values()))+np.
 ↪array(list(energy_errors.values())), color='red', alpha=0.2, label="Var(E/
 ↪N)")
plt.plot(np.arange(1,11), list_energies_theor, "k--", label="Theoretical␣
 ↪expectation")
plt.title("Mean energy per particle for different thermal energies")
plt.xlabel("Thermal Energy ($k_{B}T$)")
plt.ylabel("Energy per particle (E/N)")
plt.legend()
```

[42]:
```
<matplotlib.legend.Legend at 0x29bc20f5f90>
```

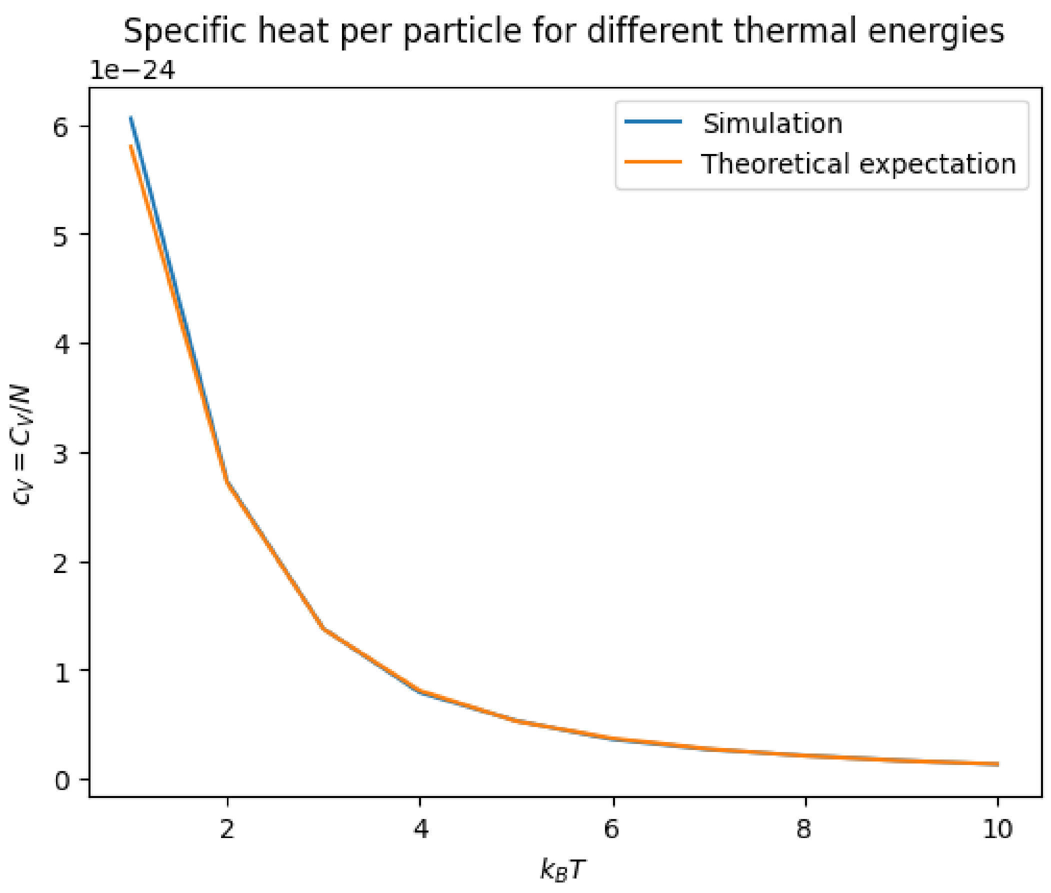


Discussion: The energy per particle grows with increasing thermal energy. For very small $k_B T$ the mean energy p.p. approaches -1. This corresponds to most spins being at $s_i = -1$. For higher temperatures, the equilibrium state is characterized by fluctuating spins, while the collective cancellation of spins results in a net energy approaching zero. The simulation results demonstrate a high degree of concordance with the theoretical expectations.

Make a plot of the specific heat per particle at constant volume $c_V = \frac{1}{N} C_V$ versus the thermal energy $k_B T$ after equilibrium has been reached. Use the same parameters as in c). Compare to the analytical result for the thermodynamic limit and discuss your results.

```
[47]: def cv_theoretical(beta, J=1):
          k_B = 1.380649 * 10**(-23)
          enum = k_B*(beta*J)**2
          denom = np.cosh(beta*J)**2
          return enum/denom

      plt.plot(1/np.array(list(cv.keys())), np.array(list(cv.values()))/N,␣
       ↪label="Simulation")
      plt.plot(1/np.array(list(cv.keys())), cv_theoretical(np.array(list(cv.
       ↪keys()))), label="Theoretical expectation")
      plt.title("Specific heat per particle for different thermal energies")
      plt.xlabel("$k_{B}T$")
      plt.ylabel("$c_{V} = C_{V}/N$")
      plt.legend()
```

```
[47]: <matplotlib.legend.Legend at 0x29bc22fe710>
```



The simulation results for c_V fit well to the analytical solution. For low temperatures, the capacity of the ising model to absorb energy from the environment is very large, as the system is in the lowest energy state. For high temperatures, a single spin flip will not result in a drastic change of the specific heat per particle, as it changes the total energy only slightly.

Finally, make a plot of the magnetisation per particle $m = \frac{1}{N}\langle M\rangle$ versus the thermal energy k_BT after equilibrium has been reached. Use the same parameters as in c), but repeat your measurement for different external magnetic fields $H = 0, 0.1, 1$ and 10. For a given configuration X, the magnetisation M is given by $M = \sum_{i=1}^{N} s_i$. Compare to the analytical result for the thermodynamic limit and discuss your results.

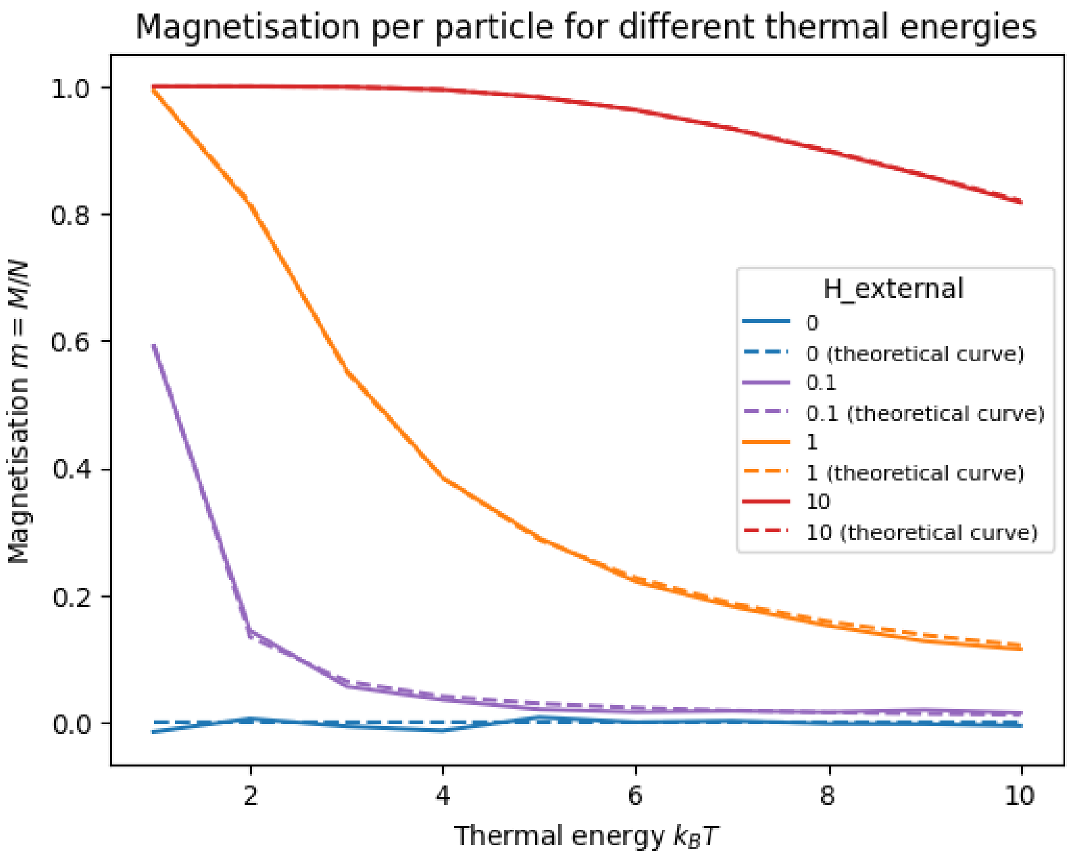
[60]:
```
list_H = [0,0.1,1,10]
dict_M ={}

def magnetisation(config):
    return np.sum(config)

def magnetisation_theor(beta, H, J=1):
    enum = np.exp(J*beta)*np.sinh(H*beta)
    denom = np.exp(-2*J*beta)+np.exp(2*J*beta)*np.sinh(H*beta)**2
    return enum/np.sqrt(denom)

for H in list_H:
    list_M_H = []
    for beta in list_beta:
        list_M = []
        for m in range(M):
            states_m, _ = metropolis_algorithm(X_0, 2000, beta, H, seeds[m])
            states_m = states_m[-1000:,:] #Keep only the equilibrium states
            errors_m = [mc_error(state) for state in states_m]
            mag_m = [magnetisation(state) for state in states_m]
            list_M.append(mag_m)
        M_mean = np.mean(list_M)
        list_M_H.append(M_mean)
    dict_M[H] = list_M_H #contains all average magnetisations for all beta
```

```
C:\Users\corin\AppData\Local\Temp\ipykernel_15012\1611310802.py:5:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  H = np.sum(config[i]*config[i+1] for i in range(len(config)-1)) +
config[0]*config[-1]
```

[64]:
```
dict_colors = {0: "tab:blue", 0.1: "tab:purple", 1: "tab:orange", 10: "tab:red"}
for h in dict_M.keys():
    plt.plot(1/list_beta, np.array(dict_M[h])/N, color= dict_colors[h], label =␣
  ↪h)
```

```
    plt.plot(1/list_beta, [magnetisation_theor(b, h) for b in list_beta],␣
↪color= dict_colors[h], linestyle= "dashed", label = f"{h} (theoretical␣
↪curve)")
plt.title("Magnetisation per particle for different thermal energies")
plt.xlabel("Thermal energy $k_{B}T$")
plt.ylabel("Magnetisation $m = M/N$")
plt.legend(title = "H_external", prop={'size': 8})
```

```
[64]: <matplotlib.legend.Legend at 0x29bb60620d0>
```



Again, the simulation results closely align with the theoretical expectations. We observe that for high external magnetic field, the magnetisation does not approach zero, as in the case for $H_{external} = 0$. This is due to an extra term in the Hamiltonian of the Ising model, that penalizes spin flips antiparallel to the magnetic field.