Report for Worksheet 6: Finite-Size Scaling and the Ising Model

Markus Baur and David Beyer

February 4, 2020

Contents

1 Speeding up the Simulation

In this exercise, the skeleton of a Monte Carlo Ising simulation has to be completed. Because the simulation stores the configuration of the quadratic Ising lattice in a vector, we first have to implement a mapping scheme from the two indices i and j to the linear index of the vector, this can be done in the following way:

```
int get_linear_index(int i, int j) {
    int index = i * m_l + j;
    return index;
};
```

Here, m_l is the number of spins in one direction. Since the indices i and j run from 0 to $m_l - 1$, the linear index takes all values between 0 and $m_l^2 - 1$. Because the mapping is also linear, it then must be bijective.

In order to initialize the Monte Carlo simulation, all spins σ_{ij} have to be assigned a random value $\sigma_{ij} \in \{-1, +1\}$, this is achieved by the following for-loop:

```
for (auto &i : m_data) {
      i = pow(-1, random_int(2));
      assert((i == 1) or (i == -1));
};
```

The random number generator random_int(n) is already defined in the given code and produces a random integer in the set $\{0, n-1\}$ according to a uniform distribution. The assert statement makes sure that every spin can only take values ± 1.

In order to calculate the magnetization of a given spin configuration from scratch, we implemented the following function:

```
void recalculate_magnetization() {
    m_M = std::accumulate(m_data.begin(), m_data.end(), 0);
    assert((m_M >= -m_l * m_l) and (m_M <= m_l * m_l));
};
```

This function simply sums all entries of the vector which contains the spin configuration (i.e. it sums the values of all spins). The assert statement checks that the absolute value of the magnetization does not exceed m_l^2.

We also implemented a function which calculates the total energy of a given spin configuration from scratch:

```
  void recalculate_energy() {
    m_E = 0;
    for (int i = 0; i < m_l; i = i +1 ) {
        for (int j = 0; j < m_l; j = j + 1) {
            m_E += -get(i, j) * (get(i-1, j) +\
            get(i+1, j) + get(i, j-1) + get(i, j+1));
        }
    }
    m_E /= 2;
};
```

To calculate the energy, we have to loop over all indices i and j to calculate the interaction energy of all pairs of next neighbours.

In order to make the code faster, we also changed the function set, which allows us to set to value of a given spin σ_{ij}:

```
void set(int i, int j, int v) {
    assert((v == 1) or (v == -1));
    int _i = mod(i, m_l);
    int _j = mod(j, m_l);
    assert(_i >= 0 and _i < m_l);
    assert(_j >= 0 and _j < m_l);
    if (m_data[get_linear_index(_i, _j)] == -1*v)
    {
        m_M += 2*v;
        m_E += 2.0 * get(i, j) * (get(i-1, j) + get(i+1, j) +\
        get(i, j-1) + get(i, j+1));
    }
    m_data[get_linear_index(_i, _j)] = v;
};
```

Using an if-loop, the magnetization and energy get updated if a spin gets changed.

Finally, we implemented a function which allows us to sample the configuration space of the Ising model according to the Metropolis algorithm. The function try_flip decides if a given spin σ_{ij} is flipped using the Metropolis criterion:

```
bool try_flip(int i, int j) {
    double energy_difference = -2.0 * get(i, j) * (get(i-1, j) +\
    get(i+1, j) + get(i, j-1) + get(i, j+1));
    if (random_double() <= std::min(1.0,\
    exp(m_beta * energy_difference)))
    {
        set(i, j, -1 * get(i, j));
        return true;
    }
    else
```

```
    {
        return false;
    }
};
```

If the spin gets flipped, the energy and magnetization are automatically updated as well. The function returns a Boolean variable whose value depends on the acceptance of the proposed new state.

In order to perform a Metropolis-Monte-Carlo step for a random spin, we implemented the function try_random_flip(), which simply executes the function try_flip(), which was described above, for two random integers in the set $\{0, m_l - 1\}$ and returns a Boolean variable depending on the acceptance of the proposed step.

```
bool try_random_flip() {
    return try_flip(random_int(m_l), random_int(m_l));
};
```

Furthermore, we also implemented a function try_many_random_flips(n) which simply performs n Metropolis-Monte-Carlo steps for randomly chosen spins, this is achieved by a simple for-loop:

```
void try_many_random_flips(int n) {
    for (int i = 0; i < n; i = i + 1) {
        try_random_flip();
    }
}
```

Using the provided test script, we could verify that the code functions correctly.

2 Determining Equilibrium Values and Errors

The code for this section can be found in the file ex3.py.

In order to obtain equilibrium values of the observables (magnetization and energy), we ran simulations for $L = 16, 64$ and different temperatures. To determine the errors of our simulations, we used the code provided in the script ising.py. Furthermore, we also compare our measurement to the exact summation for $L = 4$ and the analytical result for the magnetization in an infinite system. For all simulations in this section, we first ran 100000000 initial Monte Carlo steps which were not recorded, then we ran another 100000000 steps of which we stored every 1000th.

The simulation results are shown in Figure 1 – Figure 4. By comparing the different curves for the magnetization, we can easily see, that they approximate the analytical curve for an infinite system better for larger L, as L increases, the curves become steeper near the critical temperature. However we will never get the nonanalytical part at the critical temperature for a finite system. Similary, we observe that the slope of the energy as a function of temperature increases as L becomes larger. In this case also however, the function is still analytic for a finite system.

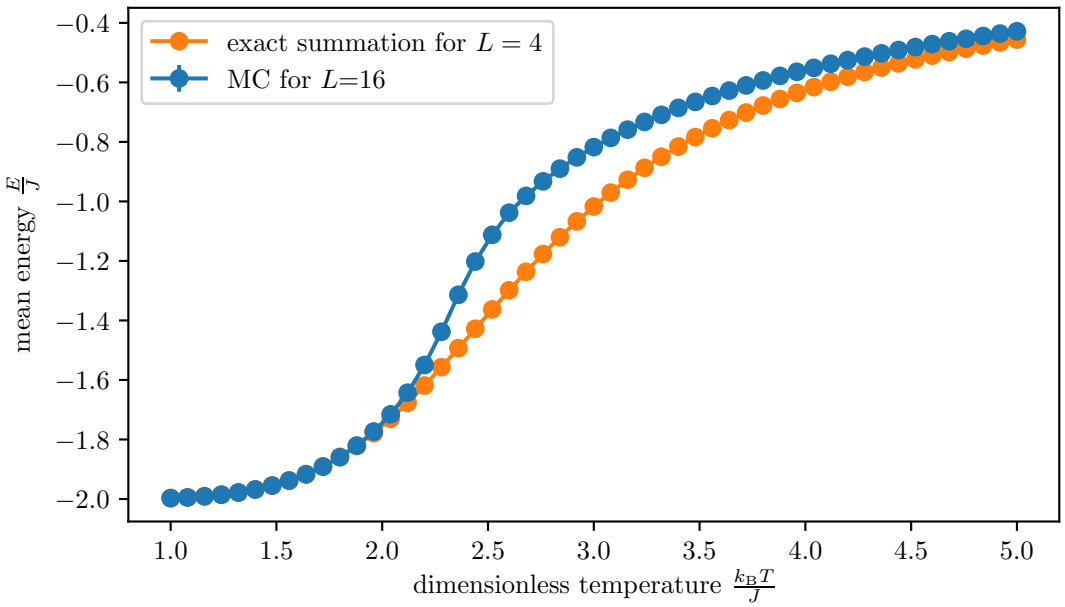

Figure 1: Plot of the mean energy per spin as a function of temperature. The plot shows the exact result for a system with $L = 4$ and a Monte-Carlo simulation for $L = 16$ (with errorbars).

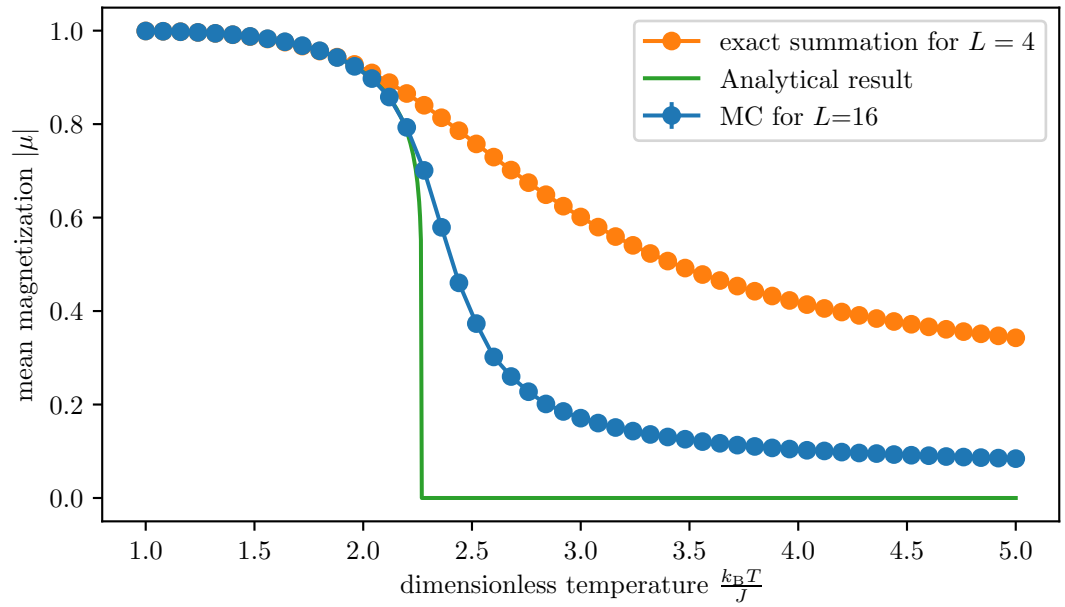

Figure 2: Plot of the mean absolute magnetization as a function of temperature. The plot shows the exact result for a system with $L = 4$ and a Monte-Carlo simulation for $L = 16$ (with errorbars). Furthermore, the analytical result for an infinte system is shown.

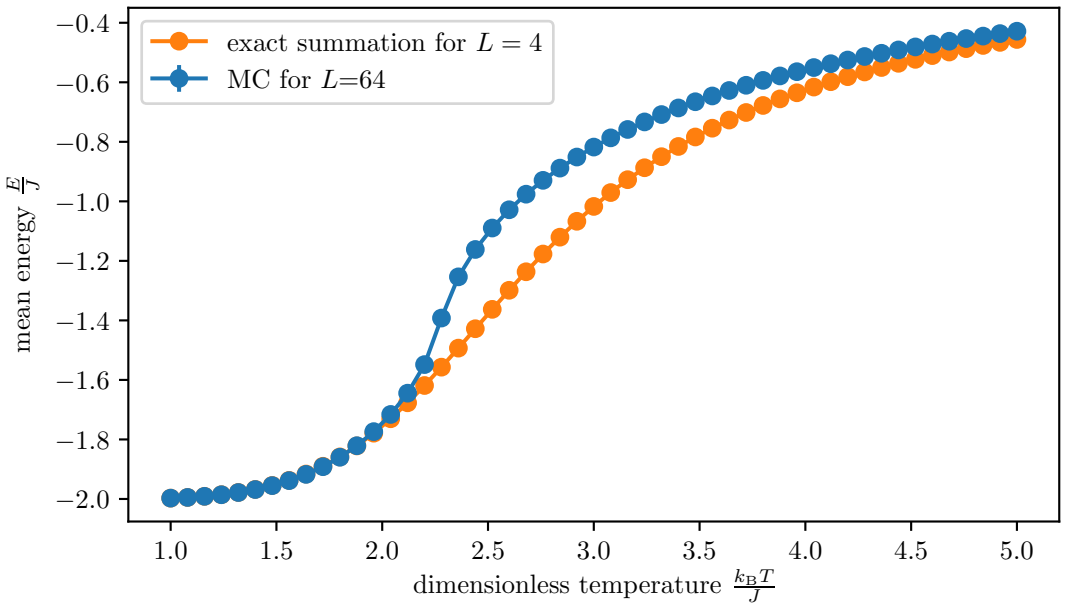

Figure 3: Plot of the mean energy per spin as a function of temperature. The plot shows the exact result for a system with $L = 4$ and a Monte-Carlo simulation for $L = 16$ (with errorbars).

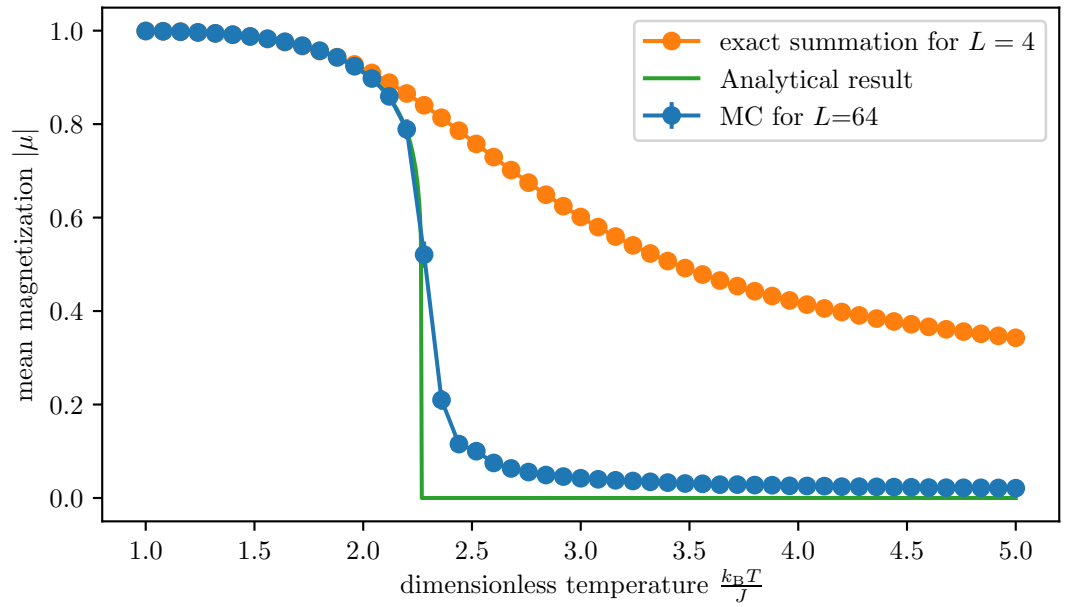

Figure 4: Plot of the mean absolute magnetization as a function of temperature. The plot shows the exact result for a system with $L = 4$ and a Monte-Carlo simulation for $L = 16$ (with errorbars). Furthermore, the analytical result for an infinte system is shown.

3 Finite Size Scaling

The code for the following tasks can be found in the files ex4_1.py, ex4_2.py, ex4_3.py and ex4_3_analyze.py.

3.1 Determining T_c

In order to calculate the Binder parameter for a given Monte-Carlo series, we implemented the following function:

```
def binder_parameter(mu):
    ret = 1 - np.average(mu**4)/(3 * np.average(mu**2)**2)
    return ret
```

To perform a Monte-Carlo simulation for different system sizes and temperatures, we used this nested for-loop:

```
for L in Ls:
    U_temp = []
    for T in tqdm(Ts):
        I = cising.IsingModel(1.0/T, L)
        Ms = []
        I.try_many_random_flips(N_initial)
        for i in range(N_MC):
            I.try_random_flip()
            Ms.append(I.magnetization())
        U_temp.append(binder_parameter(np.array(Ms)))

    U[L] = U_temp
```

U is simply a dictionary in which we store $U(T)$ for the different system sizes. In order to reach equilibrium, we first perform N_initial Monte-Carlo steps which we do not record. Then, we perform another N_MC Monte-Carlo steps which we use to calculate the Binder parameter for a given temperature.

Once we have obtained $U(T)$ for different system sizes L, we have to find the points where the different curves intersect. We solved this problem in the following way:

```
for i,j in itertools.combinations(Ls, 2):
    f1 = interpolate.interp1d(Ts, np.array(U[i])-np.array(U[j]),\\
    kind='quadratic')
    T_intersection.append(optimize.bisect(f1,2.0, 2.4))
```

This for-loop runs over all possible combinations of two different system sizes (the order does not matter). Inside the loop, an interpolation of the differnece of two of the curves is performed using scipy.interpolate.interpi1d. The intersection of these two curves can then be easily found by finding the root of this interpolated function, this is achieved using scipy.optimize.bisect over the given interval $[2.0, 2.4]$. Once the intersection has been determined, it is added to a list which is averaged at the end of the program, this gives an approximation of the critical temperature.

To determine the Binder parameter and critical temperature, we ran simulations with N_initial = 100000000 and N_MC = 100000000 steps. The value we determined for the critical temperature was

$$T_{\mathrm{c,Binder}} \approx 2.265 \tag{1}$$

which is quite close to the exact value of

$$T_{\mathrm{c}} = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.2692. \tag{2}$$

Figure 5 shows a plot of the Binder parameter as a function of the temperature T.

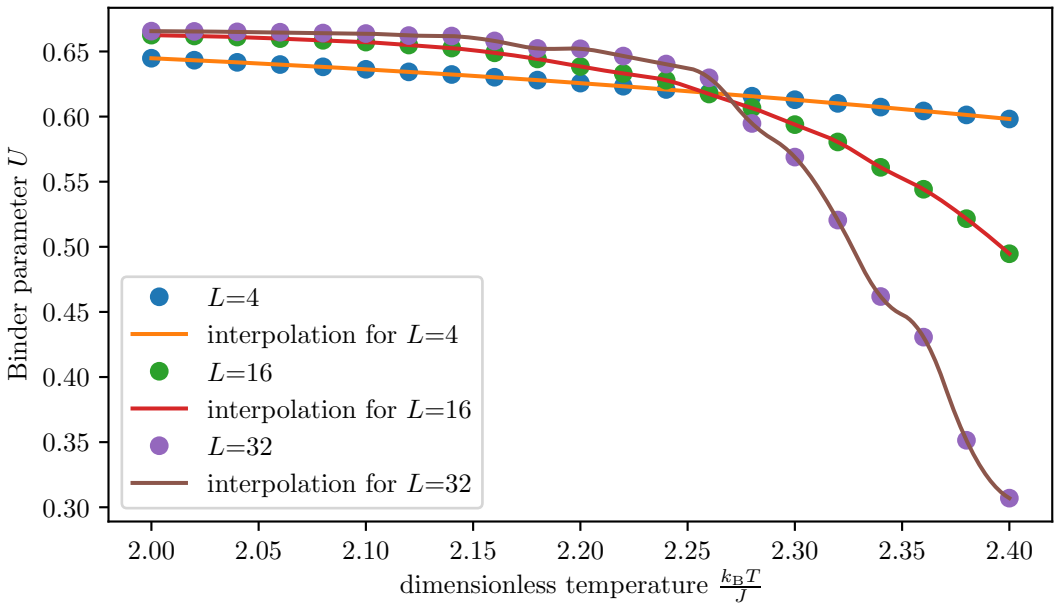

Figure 5: Plot of the Binder parameter U as a function of T for different system sizes. The different curves intersect approximately at the critical temperature T_c.

3.2 Estimating β_m

We now perform simulations for different system sizes L at the critical temperature T_c. Because we have an approximate value of the critical temperature (determined in the preceding section) as well as the exact value, we might as well compare the results for both of them. In order to determine critical exponent β_m, we make use of the scaling behaviour of the magnetization near the critical temperature. From the lecture it is known that we have

$$m \sim L^{-\beta_m/\nu} \tilde{f}\left(tL^{1/\nu}\right). \tag{3}$$

At the critical temperature ($t = 0$), the scaling function $\tilde{f}$ is constant, so we have

$$m \sim L^{-\beta_m/\nu}. \tag{4}$$

This means that we can determine the exponent β_m/ν by taking the logarithm on both sides and fitting a linear function. Since we know that $\nu = -1$ for the 2D Ising model, we can then determine the value of the critical exponent β_m.

We performed Monte Carlo simulations at both the exact critical temperature as well as at the numerically determined approximate critical temperature for the system sizes $L = 8, 16, 32, 64, 128$. The results are plotted in Figure 6 and Figure 7. For the simulation at the exact critical temperature T_c we get

$$\beta_m \approx -0.1520. \tag{5}$$

For the simulation at the numerically determined approximate critical temperature $T_{c,\text{Binder}}$ we get

$$\beta_m \approx -0.0840. \tag{6}$$

We can see that the value of β_m depends quite strongly on the value of the critical temperature T_c. As a comparison, the exact (analytical) result is

$$\beta_{m,\text{exact}} = \frac{1}{8} = -0.125. \tag{7}$$

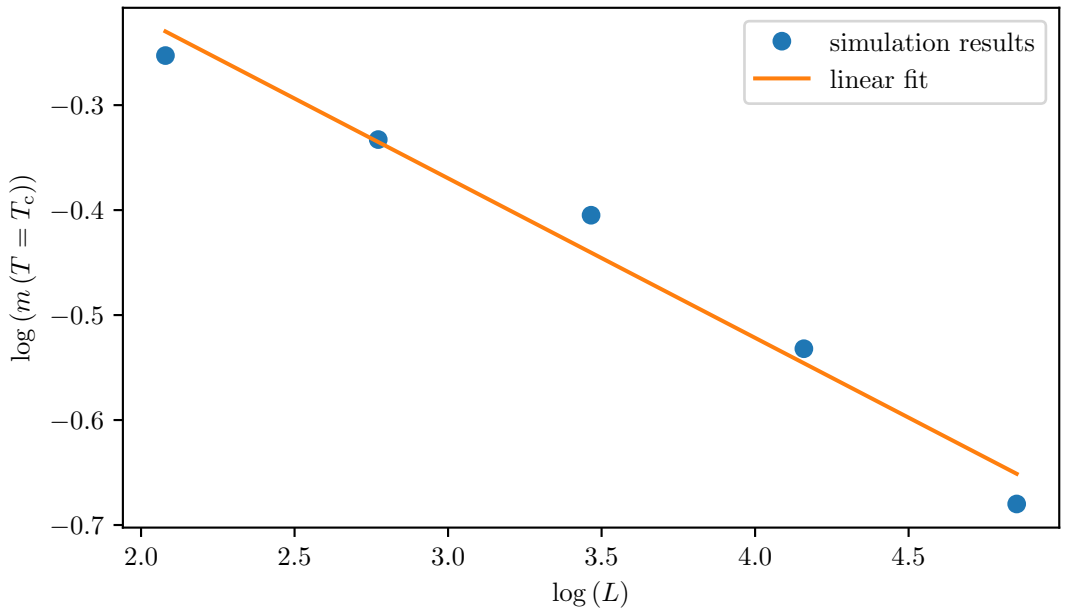

Figure 6: Double logarithmic plot of the magnetization as a function of the system size L at the exact critical temperature $T_\mathrm{c} = \frac{2}{\ln\left(1+\sqrt{2}\right)}$ and linear fit.

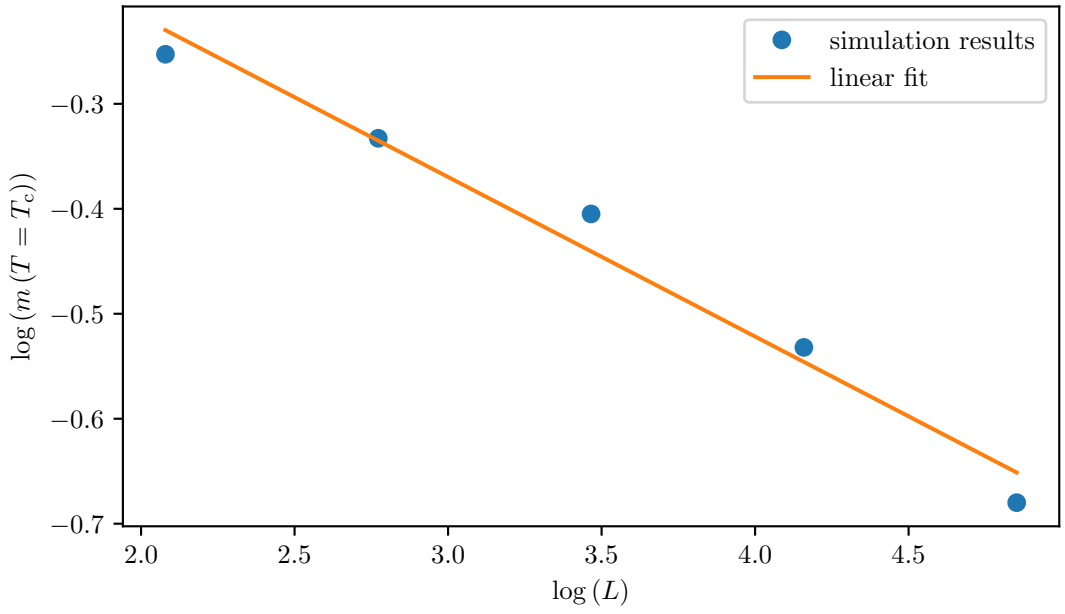

Figure 7: Double logarithmic plot of the magnetization as a function of the system size L at the approximate critical temperature (numerically determined) $TT_{\mathrm{c,Binder}} \approx 2.265$ and linear fit.

3.3 The Master Curve

The plots Figure 8 – Figure 13 show the master curve for different values of the exponent a. We can see that the plot for the estimated value of $\beta_m = -0.0840 = -a$ looks quite well around the critical temperature. The plots for $\beta_m = -0.125 = -a$ and $\beta_m = -0.152 = -a$ fit even better.

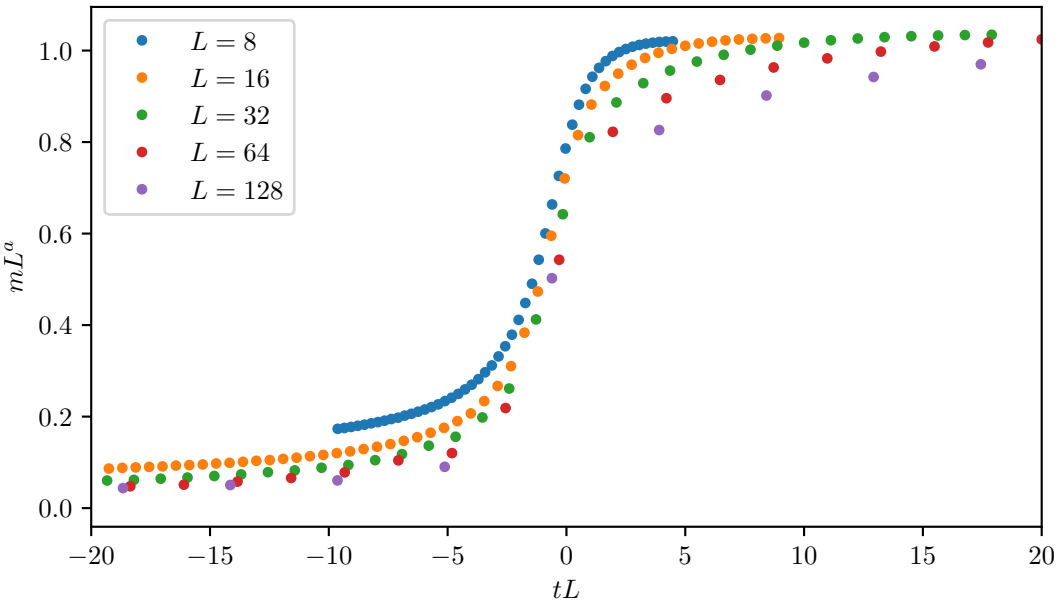

Figure 8: Plot of the master curve for $a = 0.01$.

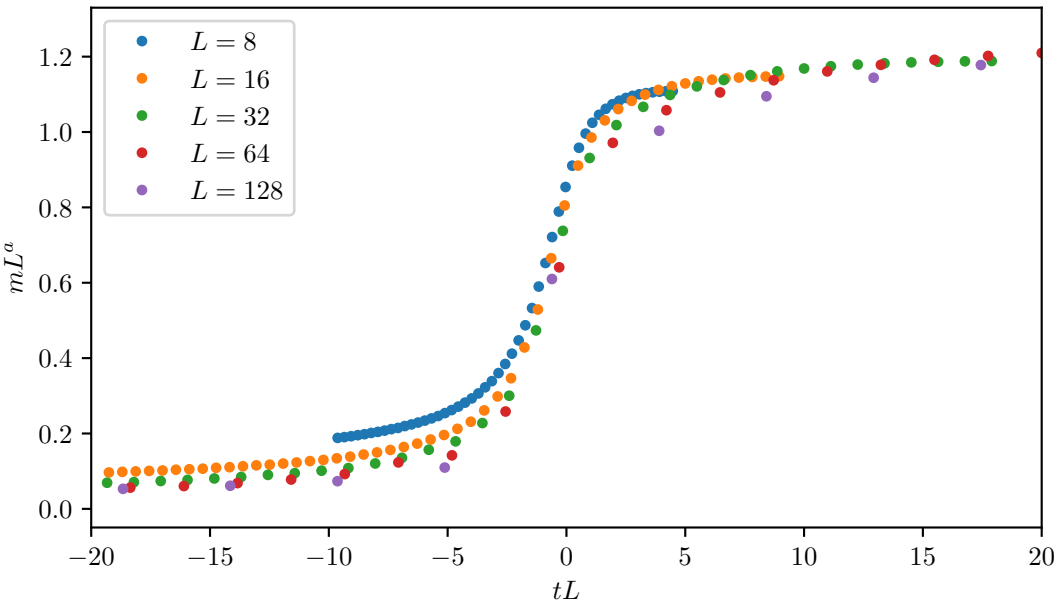

Figure 9: Plot of the master curve for $a = 0.05$.

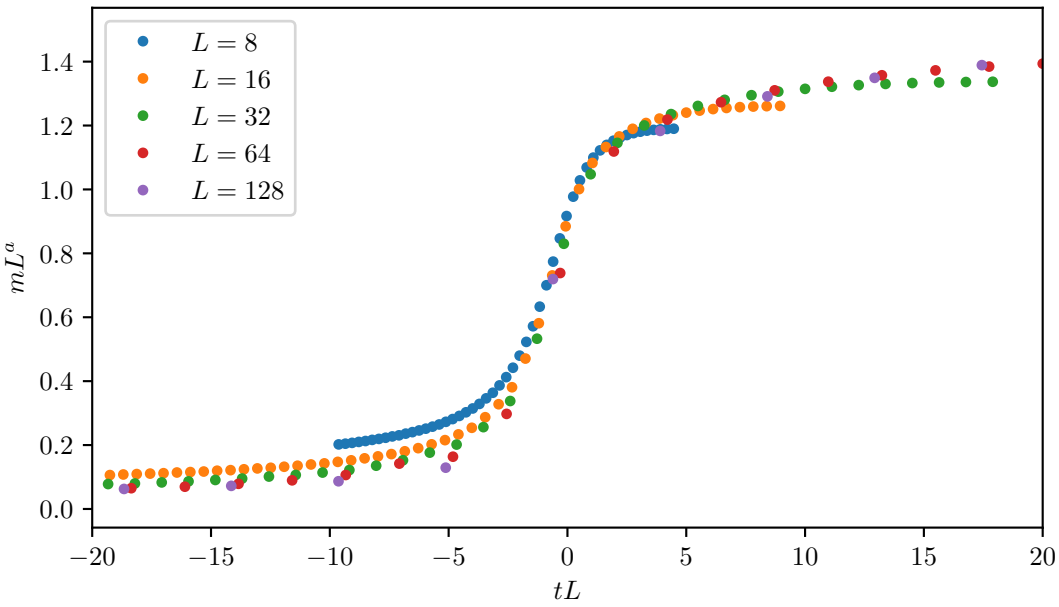

Figure 10: Plot of the master curve for $a = 0.0840$.

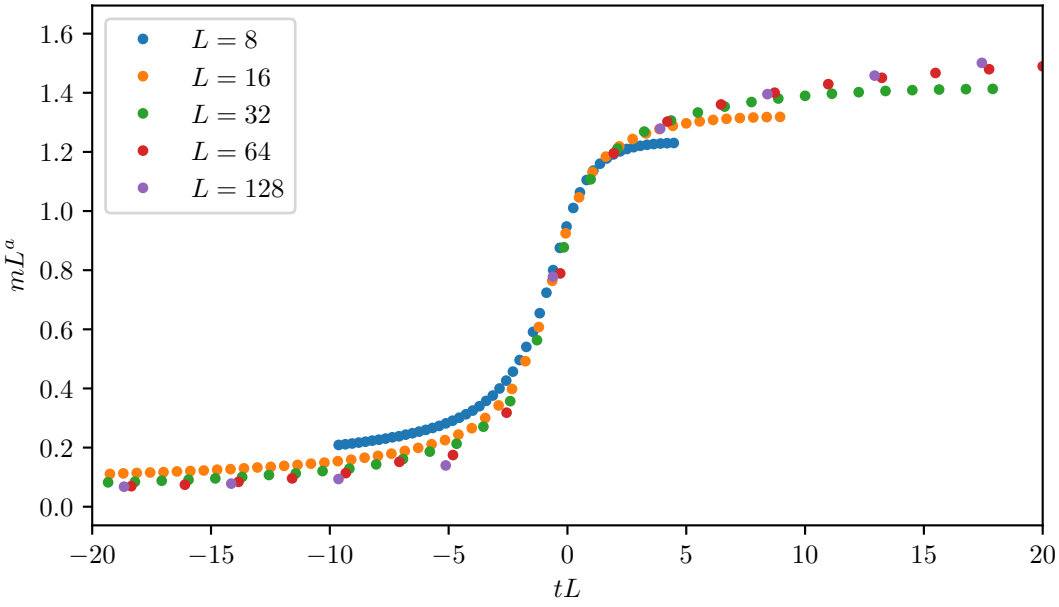

Figure 11: Plot of the master curve for $a = 0.1$.

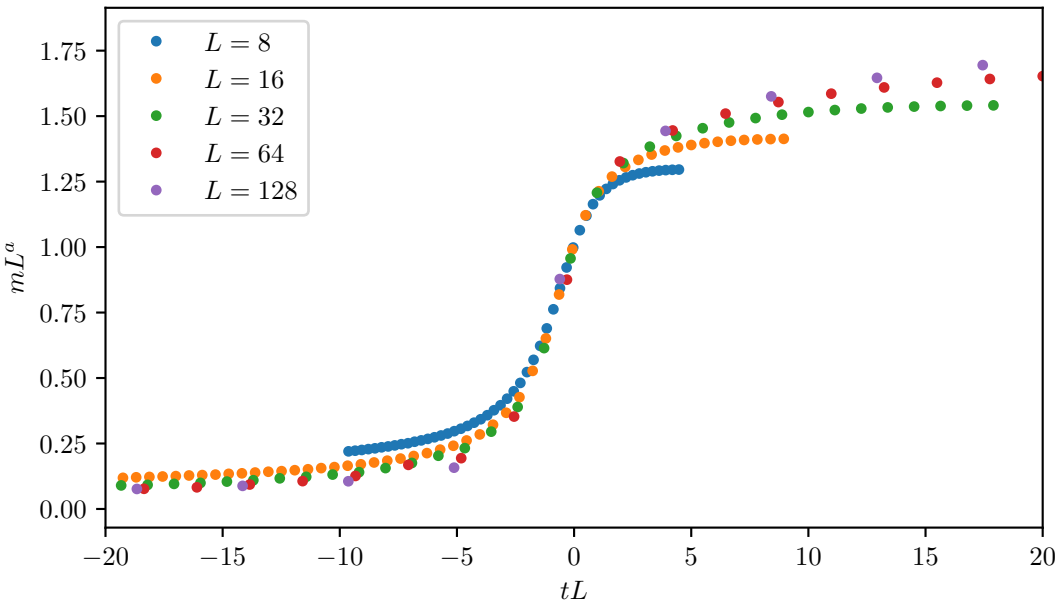

Figure 12: Plot of the master curve for $a = 0.125$.

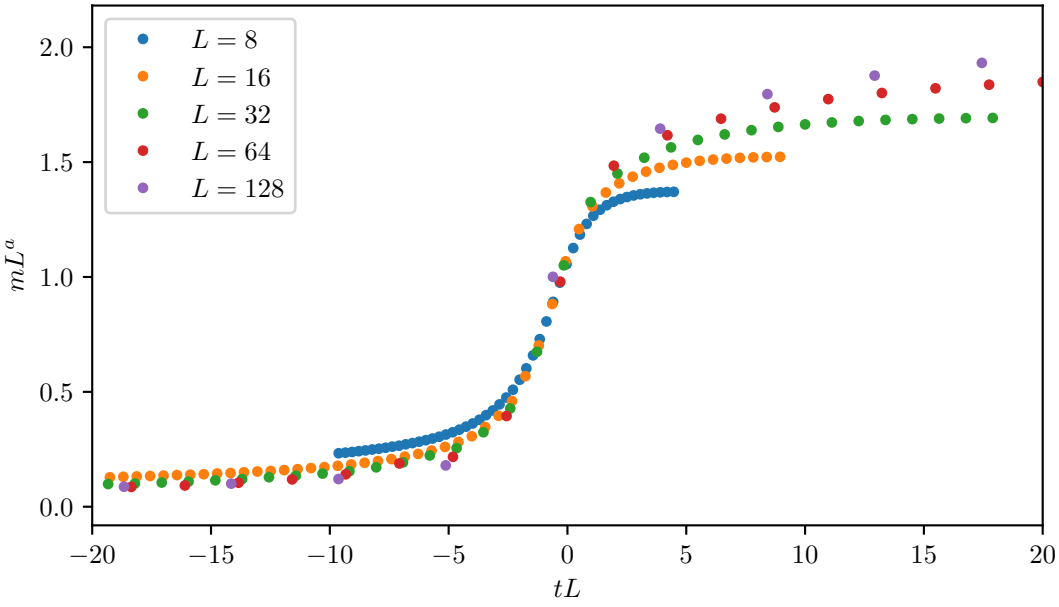

Figure 13: Plot of the master curve for $a = 0.152$.

4 Calculating the Magnetic Susceptibility from Fluctuating Magnetization

The Hamiltonian for the Ising model in an external magnetic field B is given by

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - B \sum_i \sigma_i. \tag{8}$$

The units have been chosen in such a way that the spins are dimensionless. The first sum runs over all pairs of neighbouring spins while the second sum runs over all spins. In the canonical ensemble, the mean magnetization m at the inverse temperature β is given by

$$m = \left\langle \frac{1}{L^2} \sum_i \sigma_i \right\rangle = \sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z \cdot L^2} \sum_i \sigma_i, \tag{9}$$

where the sum $\sum_{\{\sigma_i\}}$ runs over all possible spin configurations and Z is the canonical partition function, given by

$$Z = \sum_{\{\sigma_i\}} e^{-\beta H(\{\sigma_i\})}. \tag{10}$$

We can now calculate the derivative of m with respect to B:

$$\begin{aligned}
\chi = \left(\frac{\partial m}{\partial B}\right)_{B=0} &= \left(-\beta \sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z \cdot L^2} \sum_i \sigma_i \cdot \partial_B H\left(\{\sigma_i\}\right) - \sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z^2 \cdot L^2} \sum_i \sigma_i \cdot \partial_B Z \right)_{B=0} \\
&= \left(\beta \sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z \cdot L^2} \left(\sum_i \sigma_i \right)^2 - \beta \sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z^2 \cdot L^2} \sum_i \sigma_i \cdot \sum_{\{\sigma_j\}} e^{-\beta H(\{\sigma_i\})} \sum_j \sigma_j \right)_{B=0} \\
&= \beta L^2 \cdot \left(\sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z \cdot L^4} \left(\sum_i \sigma_i \right)^2 - \left(\sum_{\{\sigma_i\}} \frac{e^{-\beta H(\{\sigma_i\})}}{Z \cdot L^2} \sum_i \sigma_i \right)^2 \right)_{B=0} \\
&= \frac{L^2}{k_\mathrm{B} T} \cdot \left(\left\langle \left(\frac{1}{L^2} \sum_i \sigma_i \right)^2 \right\rangle - \left\langle \frac{1}{L^2} \sum_i \sigma_i \right\rangle^2 \right)_{B=0}
\end{aligned} \tag{11}$$