Studying the Kosterlitz-Thouless Transition of the two-dimensional XY model using Monte Carlo methods

Lennart Voorgang

Bachelorarbeit in Physik
angefertigt am
Helmholtz-Instituts für Strahlen- und Kernphysik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

August 2025

I hereby declare that this thesis was formulated by me and that no sources or tools other than those cited were used.

Bonn, ...

Date Signature

1. Assessor: Prof. Dr. Stefan Krieg
2. Assessor: Prof. Dr. Thomas Luu

Contents

1. **Introduction** **1**

2. **Theory** **3**
  - 2.1. XY Model 3
    - 2.1.1. Definition 3
    - 2.1.2. Observables 3
    - 2.1.3. Kosterlitz–Thouless Transition 4
  - 2.2. Numerical Methods 5
    - 2.2.1. Metropolis Algorithm 5
    - 2.2.2. Autocorrelation 6
    - 2.2.3. Thermalization and Blocking 6
    - 2.2.4. Critical Slowing Down 6
    - 2.2.5. Wolff Cluster Algorithm 7
    - 2.2.6. Bootstrapping 8

3. **Implementation** **9**
  - 3.1. Procedure 10
    - 3.1.1. Temperature Scanning 10
    - 3.1.2. Distributed Computing 11
    - 3.1.3. Task-based scheduler 12
  - 3.2. Optimizations 13

4. **Results** **15**
  - 4.1. Cluster Size 15
  - 4.2. Energy 16
    - 4.2.1. Observable 16
    - 4.2.2. Specific Heat 18
    - 4.2.3. Helicity Modulus 21
  - 4.3. Magnetization 23
    - 4.3.1. Observable 23
    - 4.3.2. Magnetic Susceptibility 23
    - 4.3.3. Estimating T_C using the Magnetic Susceptibility χ 26
  - 4.4. Vortex Unbinding 28

5. **Conclusion** **29**

A. **Source Code** **31**

B. TOML Config Schema **33**

C. Database Schema **35**

D. Additional Observables **37**
  D.1. Energy Squared per Spin 37
  D.2. Magnetization Squared per Spin 40

E. Residuals **45**

List of Figures **I**

Bibliography **III**

1. Introduction

In physics, there are a variety of problems for which it is either very hard or impossible to find analytical solutions. These problems often involve many particle systems and/or integrals of high dimensionality. With ever-increasing computing power available to researchers, it is becoming increasingly feasible to study such systems using Monte Carlo methods.

The most well-known application of Monte Carlo methods in physics is the Ising model, a mathematical model commonly used to study phenomena such as phase transitions, critical temperatures, and universality. This model was first introduced and solved in one dimension by Ising (1925) and later in two dimensions by Onsager (1944). It describes the total magnetization of a lattice as the superposition of all spin sites $\sigma_i = \pm 1$. It features a second-order phase transition at a critical temperature T_C, where the system transitions from an ordered low-temperature state to an unordered high-temperature state.

One can now generalize the problem from a discrete $\mathbb{Z}_2$ symmetry to a continuous $U(2)$ symmetry. First introduced by Matsubara and Matsuda (1956) as a model for liquid helium, it is now known as the XY model and it describes the spins as two-dimensional vectors on the unit circle. Unlike the Ising model, the XY model does not exhibit a second-order phase transition. However, as shown by Kosterlitz and Thouless (1973) (2016 Nobel Prize in Physics), it does feature something we now call a KT phase transition, where below a critical temperature T_C metastable states can exist. These metastable states are closely related to vortex-antivortex pairs on the lattice, which annihilate as $T_C \to 0$.

These kinds of systems are often simulated using the Metropolis-Hastings algorithm, which is prone to severe critical slowing-down effects. Cluster update algorithms introduced by Swendsen and Wang (1987) and Wolff (1989) reduce critical slowing-down effects and yield more accurate numerical results.

In this bachelor's thesis, we will employ Monte Carlo methods to study the two-dimensional XY model. The primary objective is to develop a program that simulates the XY model in a distributed manner, observes critical slowing-down effects when using the Metropolis-Hastings algorithm, and utilizes the Wolff cluster algorithm to mitigate them. The second goal is to observe the physical phenomena of the system, such as the KT phase transition, estimate the critical temperature T_C, and identify the annihilation of vortex-antivortex pairs.

2. Theory

2.1. XY Model

The XY model is the generalization of the Ising model where the spatial inversion symmetry $\sigma_i \to -\sigma_i$ ($\mathbb{Z}_2$) is replaced by a continuous $U(2)$ symmetry.

2.1.1. Definition

The XY model describes the spins σ_i as two-dimensional vectors on the unit circle

$$\sigma_i = \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} \tag{2.1}$$

parametrized by the angle $\theta_i \in [0, 2\pi)$. The Hamiltonian gives the total energy of the system as

$$H = -J \sum_{\langle i,j \rangle} s_i \cdot s_j = -J \sum_{\langle i,j \rangle} \cos(\Delta\theta) \tag{2.2}$$

where $\langle i, j \rangle$ are neighbouring spins, $\Delta\theta = \theta_i - \theta_j$ is the angle between two neighbouring spins, and $J > 0$ is the interaction strength for the ferromagnetic case. The partition function for such a system is given by

$$Z = \sum \exp -\beta H = \sum \exp\left(\beta J \sum_{\langle i,j \rangle} \cos(\Delta\theta)\right) \tag{2.3}$$

with $\beta = (k_B T)^{-1}$ so that $[\beta] = \frac{1}{\mathrm{J}}$ and k_B being the Boltzmann constant. The coupling constant J is set to $J = 1\,\mathrm{J}$ so that βJ is dimensionless and $[H] = \mathrm{J}$.

2.1.2. Observables

For our studies, we will be using a square two-dimensional lattice with side length L, nearest neighbour approximation, and periodic boundary conditions, which, topologically, forms a taurus. As the observables of the XY model scale with the number of lattice sites, it is generally more insightful to study the observables per spin. This way, the finite-size scaling of the observables can be observed better.

Energy per spin The total energy of the system is given by its Hamiltonian and thus

$$E = -\frac{1}{L^2} \sum_{\langle i,j \rangle} \cos(\Delta\theta). \tag{2.4}$$

Specific heat per spin In addition to the energy per spin E, one can also record E^2 and derive the specific heat capacity of the system

$$C_V = \frac{\langle E^2 \rangle - \langle E \rangle^2}{T^2}. \tag{2.5}$$

Magnetization per spin The total absolute magnetization per spin for a system with side length L is given by

$$|M|^2 = \frac{1}{L^2} \left((\sum \cos\theta_i)^2 + (\sum \sin\theta_i)^2 \right). \tag{2.6}$$

Magnetic susceptibility Similar to the specific heat, the magnetic susceptibility of the system can be derived as

$$\chi = \frac{\langle M^2 \rangle - \langle M \rangle^2}{T}. \tag{2.7}$$

Helicity modulus per spin According to Teitel and Jayaprakash (1983), one can also derive the *spin wave stiffness* or *helicity modulus* Υ, which “is a measure of the phase correlations of the system” (Teitel and Jayaprakash (1983)). It can be shown that the *helicity modulus* is given by

$$\Upsilon = -\frac{1}{2}\langle E \rangle - \frac{J}{k_B T L^2} \left\langle \left(\sum_{\langle i,j \rangle} \sin(\Delta\theta) \vec{r_{ij}} \cdot \vec{e} \right)^2 \right\rangle \tag{2.8}$$

(Teitel and Jayaprakash, 1983, eq. 3.2) where, in addition to the existing definitions, $\vec{r}_{i,j}$ is the vector from site i to site j and $\vec{e}$ is an arbitrary unit vector. One may choose $\vec{e}$ such that it aligns along the rows of the lattice. Thus, only neighbours within a row contribute as $\vec{r_{ij}} \cdot \vec{e} = 0$ for neighbours outside the row.

2.1.3. Kosterlitz–Thouless Transition

Unlike the Ising model, the XY model does not feature a second-order phase transition of the usual kind, “as the ground state is unstable against low-energy spin-wave excitations” (Kosterlitz and Thouless, 1973, p. 1190). It was shown by Kosterlitz and Thouless (1973) that, instead, a transition now referred to as the KT phase transition occurs, where metastable states emerge below a critical temperature, T_C.

Vortices

These metastable states "correspond to vortices which are closely bound in pairs" (Kosterlitz and Thouless, 1973, p. 1190). The vortex-antivortex pairs come closer together as the temperature decreases, until they pair-annihilate each other.

2.2. Numerical Methods

As stated in the introduction in Chapter 1, the goal is to simulate the two-dimensional XY model using the Metropolis-Hastings and the Wolff-Cluster algorithms. In the following sections, we will present the algorithms and concepts needed for understanding and implementing the simulation.

2.2.1. Metropolis Algorithm

The fundamental problem we need to solve is that we want to generate configurations of our system with probability $P(\sigma) = \exp(-\beta H[\sigma])/Z$. The problem is that the partition sum Z is not easy to calculate. As shown by Metropolis et al. (1953), it is sufficient to calculate the ratio of probabilities instead

$$\frac{P(y)}{P(x)} = \exp(-\beta(H[y] - H[x])) = \exp(-\beta \Delta H). \tag{2.9}$$

This procedure generates a Markov chain of dependent configurations. The overall procedure for the lattice is as follows:

1. Calculate E, M, and Υ for the initial lattice configuration.
2. Perform a lattice sweep by iterating over all lattice sites. For every site i do:
    a) Propose a new angle $\theta_i \in [0, 2\pi)$ from a uniform distribution.
    b) Calculate $\Delta H = \Delta E$, ΔM, and $\Delta \Upsilon$ for the proposed new state.
    c) Accept or reject state with probability $P = \min(1, \exp(-\beta \Delta H))$.
3. If the new state is accepted, update the observables E, M, and Υ by adding ΔE, ΔM, and $\Delta \Upsilon$, respectively, and add them to the result set.
4. Repeat from 2. for a total of N sweeps.

2.2.2. Autocorrelation

The configurations obtained from that procedure are dependent on the configurations preceding them. To measure this correlation, one can introduce the autocorrelation

$$C(\Delta t) = \langle (O(t) - \mu_O)(O(t + \Delta t) - \mu_O) \rangle \tag{2.10}$$

(Berkowitz (2024, eq. (41))) and, to get a size-invariant measure, the normalized autocorrelation

$$\Gamma(\Delta t) = \frac{C(\Delta t)}{C(0)} \tag{2.11}$$

(Berkowitz (2024, eq. (43))) with Δt being the step between samples, O an observable, and μ_O the mean of that observable. This measure decays exponentially, with the timescale τ being the point where samples are no longer correlated

$$\tau = \frac{1}{2} + \sum_{t=1}^{T} \Gamma(t) \tag{2.12}$$

(Berkowitz (2024, eq. (46))) where we cut off the summation when it first crosses zero. In the implementation, we use a Discrete Fourier Transform to accelerate this step.

2.2.3. Thermalization and Blocking

To obtain independent and identically distributed (iid) samples, we discard the first $\lceil 3\tau \rceil$ values in a thermalization step. We then perform a blocking step, where we divide the remaining samples into chunks of size $\lceil \tau \rceil$ and calculate the mean over each chunk. All further processing will use these blocked samples, which are now iid.

2.2.4. Critical Slowing Down

Near the critical temperature, the autocorrelation drastically increases, which is due to critical slowing down effects and the dynamical exponent z, which is defined as

$$\tau \propto \xi^z \tag{2.13}$$

(Berkowitz (2024, eq. (6))) where ξ is the correlation length, which is a measure of how aligned spins are at a distance.

While the correlation length ξ is inherent to our physical system, the dynamical exponent is a property of the algorithm (e.g. Metropolis-Hastings) used to generate our configurations. It is therefore necessary to use other algorithms if we want to lower the dynamical exponent and thus reduce the critical slowing-down effects.

2.2.5. Wolff Cluster Algorithm

As discussed in Section 2.2.4, the Metropolis-Hastings algorithm is prone to severe critical slowing-down effects. In a desire to mitigate such effects, Swendsen and Wang (1987) introduced a multi-cluster Monte Carlo algorithm for the Potts spin model, "giving a highly efficient method of simulation for large systems near criticality" (Swendsen and Wang (1987)).

Adjacent to the multi-cluster Swendson and Wang algorithm, the Wolff algorithm exhibits similar improvements in the critical exponent, while utilizing only a single cluster, which makes implementations more straightforward.

The key insight is that the Ising spin-flip operation $\sigma_i \rightarrow -\sigma_i$ needs to be generalized to the $U(2)$ symmetry of the XY model. Wolff (1989) defines the spin-flip as the reflection along the hyperplane orthogonal to an arbitrarily chosen unit vector $\vec{r}$

$$R(\vec{r})\sigma_i = \sigma_i - 2(\sigma_i \cdot \vec{r})\vec{r} \tag{2.14}$$

which is an idempotent operation $R(\vec{r})^2 = 1$ (Wolff, 1989, eq. 3).

The overall procedure for the Wolff cluster algorithm was adapted from Wolff (1989, p. 361) with the addition of item 2. The step was added, as the probability of a site joining the lattice is lower for unordered states that emerge at high temperatures. Consequently, the average cluster size tends towards 1 in those states. This results in a very uneven distribution of potential updates (*marked* sites in the algorithm procedure). At low temperatures, many sites get visited and potentially updated, while at high temperatures, as few as five (1 starting site + 4 neighbours) sites get marked. To mitigate this, we propose item 2 as an additional step that ensures a sufficient number of potential updates are made, even at high temperatures.

1. Calculate E, M, and Υ for the initial lattice configuration.
2. Perform the following until we have marked L^2 lattice sites in total:
    a) Choose a random two-dimensional unit vector $\vec{r}$ as the reflection vector.
    b) Select a random lattice site x as the first element of the cluster.
    c) Flip the spin at the initial lattice site $\sigma_x \rightarrow R(\vec{r})\sigma_x$ and mark site x.
    d) Visit all direct unmarked neighbours y of x and add them to the cluster with probability:

    $$P(\sigma_x, \sigma_y) = 1 - \exp(\min[0, 2\beta(\vec{r} \cdot \vec{\sigma_x})(\vec{r} \cdot \vec{\sigma_y})]) \tag{2.15}$$

    and calculate ΔE, ΔM, and $\Delta \Upsilon$ for the proposed new state.
    e) If site y is accepted into the cluster, its spin is flipped $\sigma_y \rightarrow R(\vec{r})\sigma_y$.

    f) Site y becomes the new x, and we continue from item 2d until there are no more sites to add to the cluster.

3. Update the observables E, M, and Υ by adding ΔE, ΔM, and $\Delta\Upsilon$, respectively, and add them to the result set.
4. Continue from item 2 until the exit condition is fulfilled.

2.2.6. Bootstrapping

Since both algorithms scale with the number of lattice sites, it is computationally impractical to run them for long timescales. Given that we have an observable whose iid samples follow a Gaussian distribution and we already have "enough" iid samples that cover a sufficient portion of the possible configuration space, we can then use bootstrapping to generate additional samples (Berkowitz (2024)).

1. Collect B intermediate means by repeating the following:
    a) Take A random samples from the blocked samples obtained in section 2.2.3 with replacement.
    b) Calculate the mean of those samples and add the result to the set of intermediate means.
2. Calculate the sample standard deviation of those B intermediate means as the uncertainty of our observable.

The estimate for the observable is still calculated as the mean of the iid samples.

3. Implementation

To simulate the two-dimensional XY model with periodic boundary conditions (Section 2.1), we implemented the numerical methods of Section 2.2 in a distributed, multi-threaded *C++26* program. A brief guide on how to obtain and compile the sources can be found in Chapter A. For the following sections, these two definitions will be helpful:

1. The term *core* refers to physical cores when a compute node has SMT disabled and to threads when SMT is enabled.

2. The term *configuration* will refer to any combination of algorithm, lattice size L, and temperature T.

Having worked on the XY model during the *physics760 - Computational Physics* class, one of the shortcomings of the implementation back then was that every compute node simulated exactly one lattice size L from start to finish before saving the final results and moving on to the next lattice size. This procedure is suboptimal for various reasons:

1. The program divided the temperature interval into N steps. When the compute node has more cores than the number of steps, the left-over cores are idle.

2. Only the final bootstrapped estimates were saved, which becomes a problem when one wants to increase the number of...

   a) ...sweeps after the simulation had finished. Since the final lattice spins were not saved, the program could not continue the simulation, and the entire simulation had to be rerun from start to finish

   b) ...bootstrap samples after the simulation had finished. Since the iid observables were not saved, the program could not redo the bootstrap procedure, and the entire simulation had to be rerun from scratch.

3. When the program or the compute node crashed, only the finished lattice sizes were saved. All lattice sizes that had not yet finished had to start over once the node came back online.

4. The results were saved on the disk in an SQLite database, which requires that all compute nodes have access to the same network share and that this network share is adequately fast.

3.1. Procedure

We propose and implement the following procedure to mitigate the shortcomings mentioned before. The various subsystems of the simulation are described in Sections 3.1.1 to 3.1.3.

The high-level procedure is as follows:

1. The database schema (Chapter C) is initialized by the compute nodes in Section 3.1.2.
2. The compute nodes read the configuration file from the current working directory.
3. The initial temperature interval is divided into steps according to item 1 of Section 2.2.5. The resulting configurations — defined by the algorithm (Metropolis or Wolff), the lattice size L, and the temperature T — are inserted into the database if they do not already exist.
4. The program checks whether any existing configuration is incomplete and, if so, executes the task pipeline defined in Section 3.1.3.
5. One of the compute nodes simulates the annihilation of the vortex/antivortex pair.

3.1.1. Temperature Scanning

The program splits the temperature interval $T \in (0.0, 3.0]$ into N steps and simulates each of those temperatures. As observed by others (Olsson and Minnhagen, 1991), there should be an area near the critical temperature where the magnetic susceptibility peaks.

We therefore chose an iterative approach where the simulation *zooms* in on the area near the critical temperature. After having picked the maximum number of iterations $I_{\text{max}} > 0$, the procedure is as follows:

1. Divide the temperature interval $T \in (0.0, 3.0]$ into 64 discrete steps[1].
2. Simulate the system at the given temperatures. An iteration counter $I_{\text{c}} = 0$ is incremented and, if $I_{\text{c}} >= I_{\text{max}}$, the procedure exits.
3. Find the temperature T_{max} where the magnetic susceptibility is maximal.
4. Pick a new temperature interval $T \in (T_{\text{max}} - 3\Delta T, T_{\text{max}} + 3\Delta T)$ where ΔT is the step size from the last iteration[2]. The factor of 3 was chosen to ensure that the next iteration simulates a sufficiently large range around T_{max}, so that the actual peak is definitely in that range.
5. Go to item 2.

---

[1] While the number of steps can be arbitrary, it is sensible to pick a number that evenly divides the total number of cores across all nodes.

[2] As an example: Dividing $T \in (0.0, 3.0]$ into 64 steps yields $\Delta T = 0.046\,875$

One downside of this procedure is that one must be careful to simulate an adequate number of sweeps at every temperature value. As the procedure *zooms* in on the area near the critical temperature, which is also the point where critical slowing down is most severe, the integrated autocorrelation time increases, meaning there are fewer iid samples. Fewer samples result in larger statistical errors for the magnetic susceptibility, so the procedure might inadvertently choose the wrong T_{max} value for the next iteration.

3.1.2. Distributed Computing

The temperature scanning of Section 3.1.1 divides every lattice size L into N steps and I iterations. As the number of configurations far exceeds the core count of a single computer for any realistic choice of L, N, and I, the computational efforts should be scaled out across multiple compute nodes. The implementation addresses the shortcomings mentioned at the beginning of Chapter 3 and requires two kinds of nodes.

Data Node

The *data* node is responsible for orchestrating the compute nodes and storing the intermediate and final results. The orchestration is realized using a *PostgreSQL*[3] database and does not require any additional software. Having a central *data* node fixes shortcoming item 4.

The schema for the *PostgreSQL* database is presented in Chapter C. Foreign keys link all relevant database entities, and unique constraints have been added to ensure that every configuration is simulated exactly once.

Compute Node

One or more *compute* nodes are required to run the simulation. These nodes should have modern, high-frequency, multi-core CPUs with *AVX2* support (section 3.2) as our simulation primarily utilizes the CPU.

The compute nodes only need internal network access to the *data* node to access the *PostgreSQL* instance. To automatically install all required dependencies and compile the program, a *CloudInit*[4] configuration file is available in the repository[5]. The script also registers the program as a *systemd*[6] service, which provides resilience by restarting the service when it exits with an error code[7].

---

[3] `https://www.postgresql.org`

[4] `https://cloudinit.readthedocs.io/en/latest/index.html`

[5] `https://github.com/lennartvrg/physik690-Bachelorarbeit/blob/main/cloud-config.yaml`

[6] `https://systemd.io/`

[7] `https://github.com/lennartvrg/physik690-Bachelorarbeit/blob/main/bachelor.service`

3.1.3. Task-based scheduler

As each *compute* node has multiple cores, the workload can be distributed across them. These so-called worker threads work on tasks until there are no more tasks available.

1. In the *simulation* task, the worker thread simulates a configuration and writes the observables E, E^2, M, M^2, and Υ back to the database.
2. In the *bootstrap* task, the worker threads retrieve the intermediate results for each configuration that was simulated beforehand. The observables are bootstrapped, and the final results are written to the database.
3. In the *derivative* task, the workers calculate the observables (C_V, χ) derived from the bootstrapped results.

These steps are executed in stages, so initially, all worker threads focus on *simulation* tasks until none remain. As soon as all the compute nodes' worker threads have finished with one type of task, they can proceed to the following kind of task.

As the pipeline finishes, the *compute* nodes wait for each other in a synchronization step. From here, they go back to item 2 of the temperature scanning procedure. If the exit condition $I_C \geq I_{\max}$ is fulfilled, the *compute* nodes exits.

Chunking

The *simulation* task uses chunking to alleviate some of the problems discussed earlier in the chapter. The total number of sweeps is defined here as the product of the number of sweeps per chunk and the number of chunks N_{Chunks}, which can be set in the config file. The worker threads work on a given configuration, one chunk at a time, and write back the intermediate results and the current spins to the database. This way, another worker can continue working on a configuration when the original worker crashes.

Scheduling

A task scheduler was implemented, which delegates the workload to the worker threads. The number of worker threads equals the number of CPU cores available to the program. A dedicated scheduling thread is started, which connects to the database. The task distribution works as follows:

1. The worker threads register themselves as available.
2. The scheduling thread searches for any available worker threads.
    a) If a worker is available, the next task is fetched and delegated to the available worker.
    b) If no worker is available, the scheduling thread sleeps for one second.

3. The scheduling tasks check if the worker threads have published any results and write them to the database.

4. Go to item 2 unless all workers are waiting and no new tasks are available.

3.2. Optimizations

Apart from distributing the workload across multiple threads and nodes, some optimizations can be made to improve performance on a single thread.

SIMD Instructions As already introduced in the definitions of the Metropolis (section 2.2.1) and Wolff (section 2.2.5) algorithms, it is computationally more sensible to calculate the delta energy/magnetization/helicity modulus that a proposed spin update would entail, instead of recalculating the observable for the entire lattice. In our two-dimensional lattice model with nearest neighbour approximation, this means that only the four direct neighbours of a spin σ_i contribute to the calculation.

As many modern-day CPUs have the AVX2[8] instruction set enabled, which allows us to perform standard mathematical operations on vectors of four 64-bit scalars simultaneously, we can use this to improve the performance of our implementation. *AVX2* in itself does not provide intrinsics for the operations $(\sin, \cos)$ that are needed. Therefore, we will also use the *Intel® Short Vector Math Library* (SVML[9]), which provides methods that generate an optimized sequence of intrinsics for sin and cos.

To optimize data access for our SIMD instructions, the internal `std::vector<double_t>` storing the spins is aligned on a 32-byte boundary.

Compiler Flags The program uses the `-march=native` compiler flag and should therefore be compiled on a system with the same or very similar CPU as the compute nodes. Additionally, the `-ffast-math` flag was used, which sacrifices ISO compliance for execution speed. Although no systematic testing was conducted, the performance improvement on the development machine was $\approx 20\,\%$ better, with no observable loss in numerical results.

---

[8] `https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet\-volume-1-of-2/009/intel-advanced-vector-extensions-2-intel-avx2/`

[9] `https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-8/intrinsics-for-short-vector-math-library-ops.html`

Intel Compiler We also wanted to try the *Intel®* oneAPI DPC++/C++ compiler as the manufacturer claims it offers superiority over *gcc* in HPC workloads. No systematic testing was conducted, but on the development machine (AMD Ryzen 1700) a 6 % reduction of the total wall-time was observed. Tests on a second machine (*Intel® i7-10710U*) yielded a 14 % improvement over *gcc 15*.

4. Results

The following results were obtained from a run on a cluster with eight nodes, each with 16 cores. The simulation was configured to run for lattice sizes $L \in \{4, 8, 16, \ldots, 378\}^1$. The initial setup is a cold state where $\sigma_i = 0$ for all sites i, 200 000 bootstrap resamples and a scanning depth of two. Each chunk performed 50 000 sweeps, with the number of chunks depending on the algorithm.

1. For the Metropolis algorithm, 32 chunks were used, resulting in a total of 1 600 000 sweeps.
2. For the Wolff algorithm, six chunks were used, resulting in a total of 300 000 sweeps.

These numbers were chosen as initial, smaller-scale runs during development suggested that the Wolff algorithm suffers significantly less from the critical slowing down effect, resulting in more accurate results even with fewer sweeps. The simulation ran for five days on the aforementioned cluster.

The following sections will discuss the results and derive estimates for the critical temperature T_C. Note that for the discussion of the observables, the focus will be on a subset of lattice sizes $L \in \{4, 32, 128, 352\}$ so as not to clutter the plots more than necessary. The plots for all observables at all lattice sizes may be found in the repository (see chapter A).

4.1. Cluster Size

Some of the observations we will make in the following sections will only be explainable with the average cluster size. This observable is exclusive to the Wolff algorithm and is plotted in Figure 4.1. The observable is dependent on the lattice size, as the maximum number of sites a cluster may have is at maximum equal to the total number of lattice sites.

Taking into account the acceptance probability $P(\sigma_x, \sigma_y)$ (eq. (2.15)) of the Wolff algorithm, it also makes sense that at low temperatures almost the entire lattice is part of the cluster. As the temperature increases, the average cluster size experiences a sharp dropoff, which is more pronounced for larger lattice sizes. For high temperatures, the average cluster size tends towards 1 for all latices sizes. An essential aspect is that

---

[1] Full list: $\{4, 8, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240, 256, 272, 288, 304, 320, 336, 352, 378\}$

the dropoff happens after the point where we expect the critical temperature T_C to be. If this were not the case, the Wolff algorithm would not yield an improvement over the Metropolis algorithm when estimating T_C.

The observation that the average cluster tends towards 0 when $T \to \infty$ also justifies the introduction of item 2 in section 2.2.5. If we did not do that, the lattice would not be given enough chances to update, which, at high temperatures, would result in a highly correlated observable. Correlated observables, in turn, would result in a larger integrated autocorrelation time τ. As we use $\lceil \tau \rceil$ to block the intermediate results, it would also reduce the accuracy and precision of our final estimates, since we have fewer iid samples.

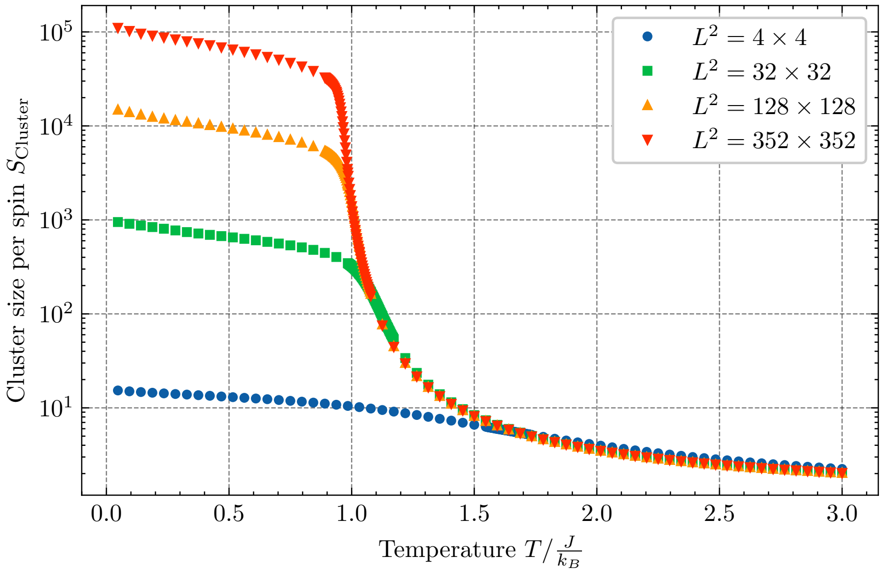

Figure 4.1.: Temperature dependence of the average cluster size per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

4.2. Energy

4.2.1. Observable

Figure 4.2 shows the temperature dependence of the energy per spin (eq. (2.4)) using the Metropolis algorithm.

We observe the expected quasi-ordered state at low temperatures, where the spins mostly align. The energy tends towards $-2\,\mathrm{J}$ for $T \to 0$, which matches the Ising model. With increasing temperature, the absolute value of the energy per spin slowly reduces. For

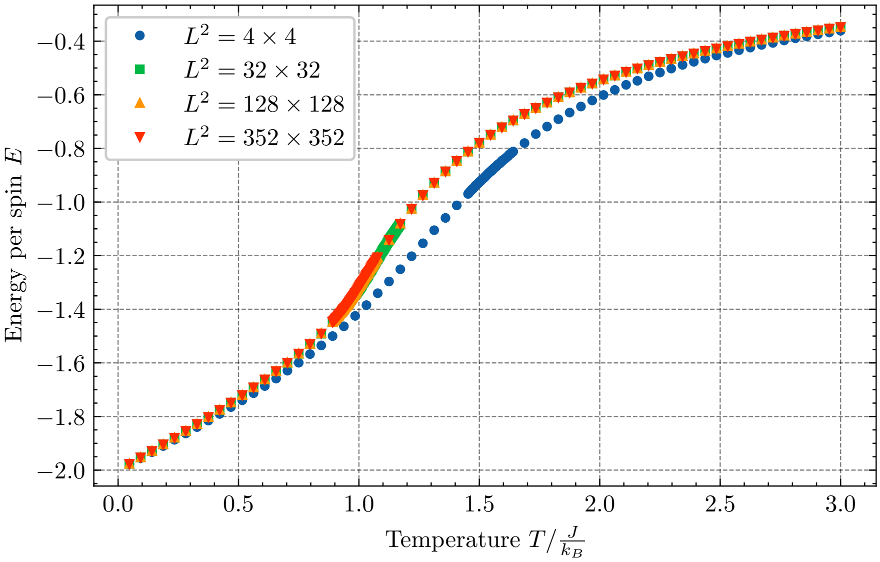

Figure 4.2.: Temperature dependence of the energy per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

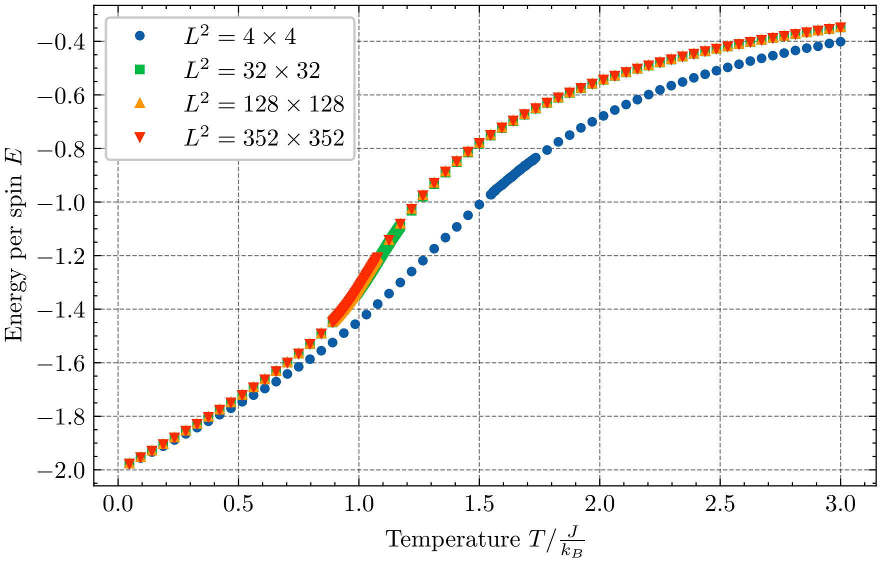

Figure 4.3.: Temperature dependence of the energy per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

$T \to \infty$ we can expect $E \to 0$ as the spins become less and less aligned, and the spin energies average out.

Comparing the form of the curve with the results of the Wolff algorithm in Figure 4.3 shows the same overall form. A slight deviation can be observed for $L = 4$, where at $T = 3\,\frac{\mathrm{J}}{\mathrm{k_B}}$, the data point for the Wolff algorithm is slightly lower than that for the Metropolis algorithm. The lower data point is probably due to the small lattice size and the fact that cluster sizes tend towards one at higher temperatures (see section 4.1). At these higher temperatures the acceptance probabilities of both algorithms is smaller and, as the Wolff algorithm always flips at least one site (the starting site of the cluster-building procedure), we obtain a slightly different curve compared to the Metropolis algorithm, as the Metropolis algorithm may flip as few as zero spins.

While the energy per spin curve is visually the same for both algorithms, the residuals and integrated autocorrelation time τ differ significantly. We plotted the autocorrelation function $\Gamma(\Delta t)$ at the lowest, highest, and the temperature where the magnetic susceptibility χ peaked[2]. The autocorrelation function was cut off at the last datapoint where $\Gamma(\Delta t) > 0$.

Comparing the Metropolis algorithm (fig. 4.4) and the Wolff algorithm (fig. 4.5) reveals that low temperatures generally exhibit autocorrelation function curves with long tails. In contrast, high temperatures feature a short autocorrelation curve. Near criticallity, the autocorrelation function approaches zero much more quickly when using the Wolff algorithm. This behaviour is expected as the Wolff algorithm is supposed to give "a highly efficient method of simulation for large systems near criticality" (Swendsen and Wang, 1987).

The advantage becomes even clearer when plotting the integrated autocorrelation time τ of the Metropolis algorithm (fig. 4.6) and the Wolff algorithm (fig. 4.7). When comparing these graphs, we can see that τ is much smaller when using the Wolff algorithm. As already discussed in Section 4.1, this results in more iid samples and a more accurate and precise measurement of our observables. We may conclude that the Wolff algorithm suffers less from critical slowing and yields superior estimates of our observables in terms of accuracy and precision.

Energy squared per Spin A thorough discussion of E^2 will be omitted as the general observations of E also hold for E^2. The plots were, for completeness' sake, enclosed in Section D.1.

4.2.2. Specific Heat

The specific heat capacity C_V can be derived from $\langle E \rangle$ and $\langle E^2 \rangle$ using Equation (2.5). Comparing C_V using the Metropolis (fig. 4.8) and Wolff (fig. 4.9) algorithms reveals that

---

[2] Note that this means that this could mean different T when comparing Metropolis and Wolff plots.

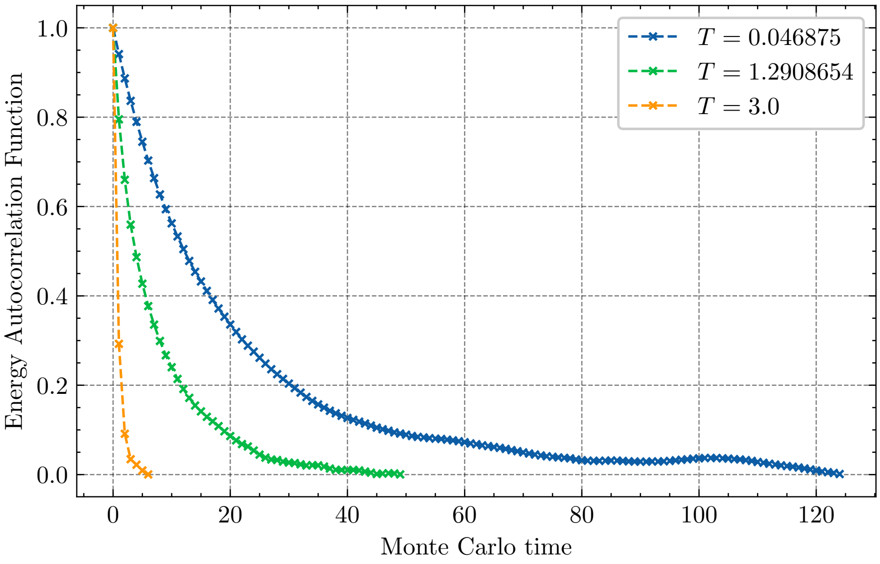

Figure 4.4.: Autocorrelation function $\Gamma(\Delta t)$ of the energy per spin using the Metropolis algorithm.

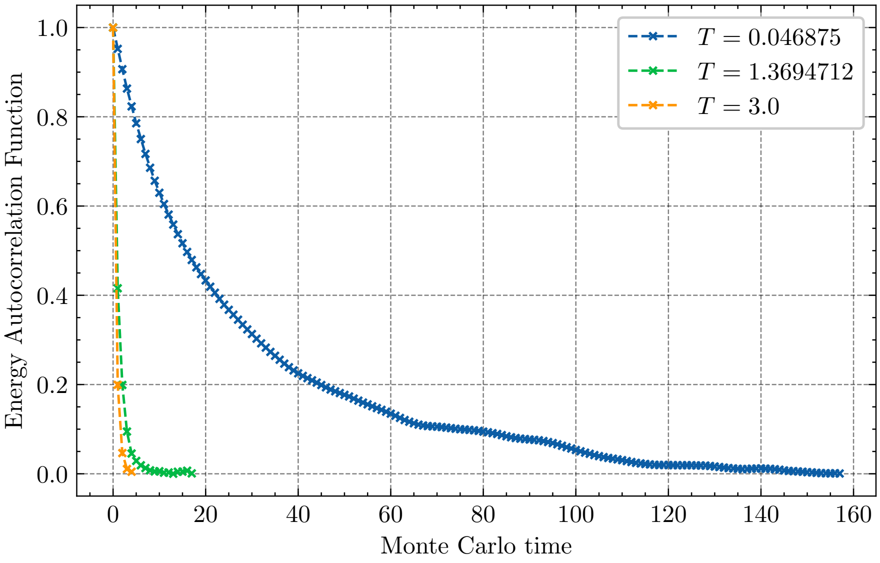

Figure 4.5.: Autocorrelation function $\Gamma(\Delta t)$ of the energy per spin using the Wolff algorithm.

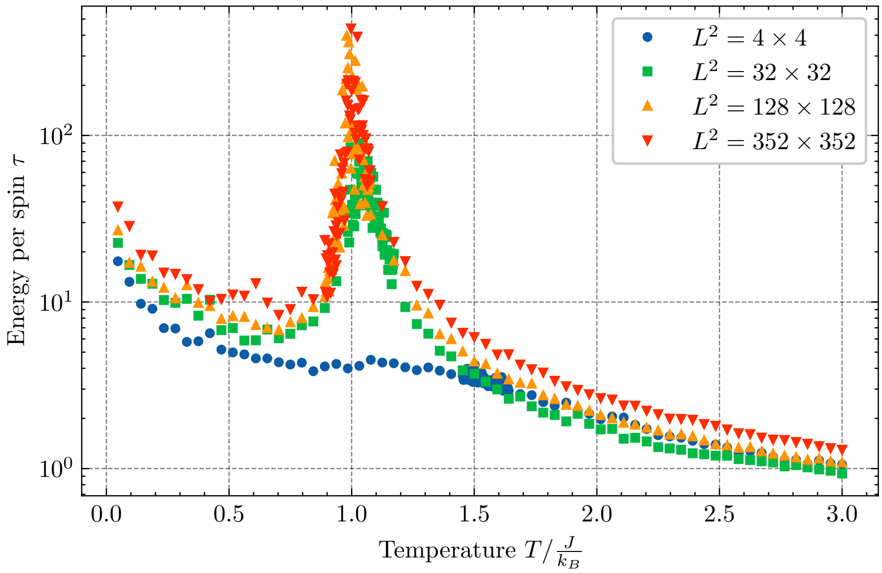

Figure 4.6.: Integrated autocorrelation time τ of the energy per spin using the Metropolis algorithm.

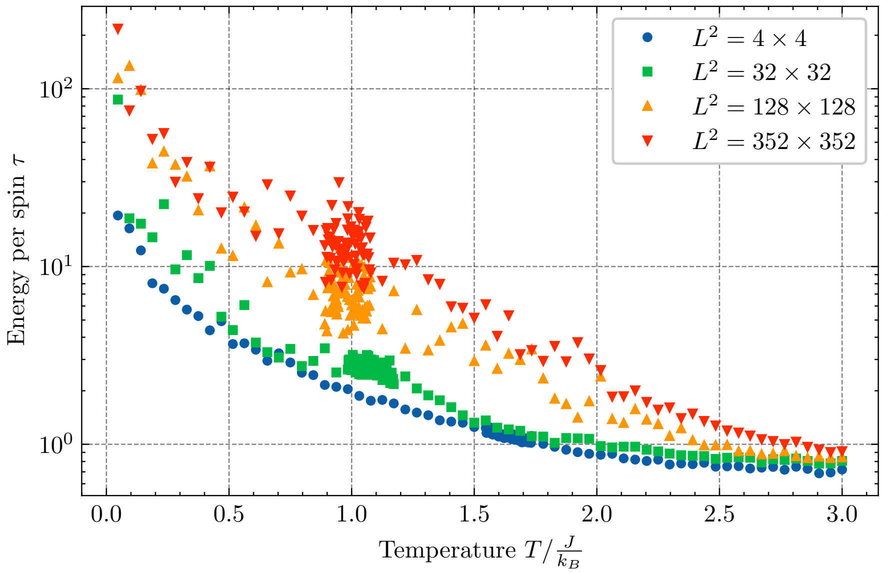

Figure 4.7.: Integrated autocorrelation time τ of the energy per spin using the Wolff algorithm.

both follow the same curve. The specific heat per spin starts small, has a small peak which moves closer to T_C with increasing lattice sizes, and then falls off for $T \to 0$.

Near criticality, the curve of C_V obtained using the Wolff algorithm is a lot smoother, which can be attributed to the reduced critical slowing-down effects experienced by the Wolff algorithm.

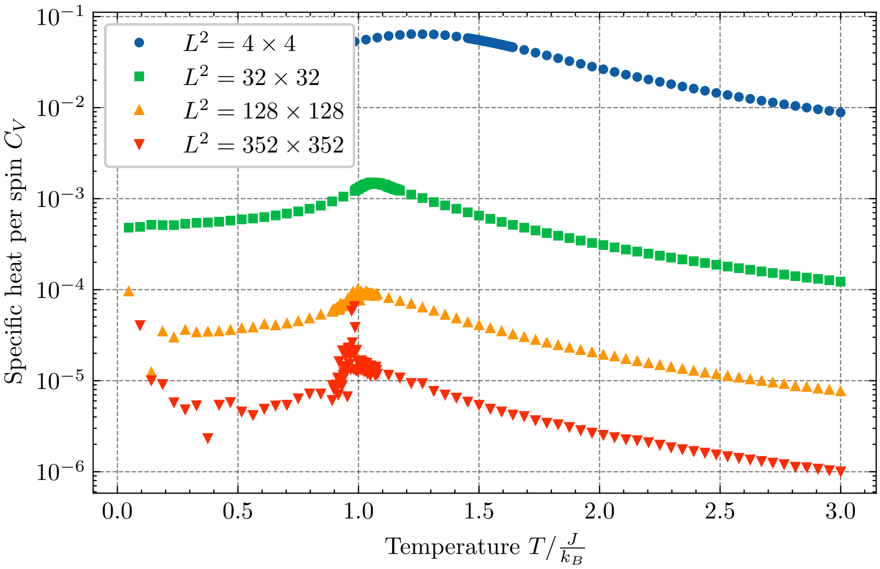

Figure 4.8.: Temperature dependence of the specific heat per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

4.2.3. Helicity Modulus

As introduced in Equation (2.8), the helicity modulus Υ "measures the increase in the amount of energy when we rotate the order parameter of a magnetically long-range-order system along a given direction by a small angle" Krüger et al. (2006). Comparing the curves of Metropolis (Figure 4.10) and Wolff (Figure 4.11) reveals no difference in overall form, although the curve obtained using the Wolff algorithm is smoother. For $T \to 0$, the XY model has a long-range order and the helicity modulus Υ tends towards 1. Above T_C, the long-range order vanishes and Υ drops towards 0.

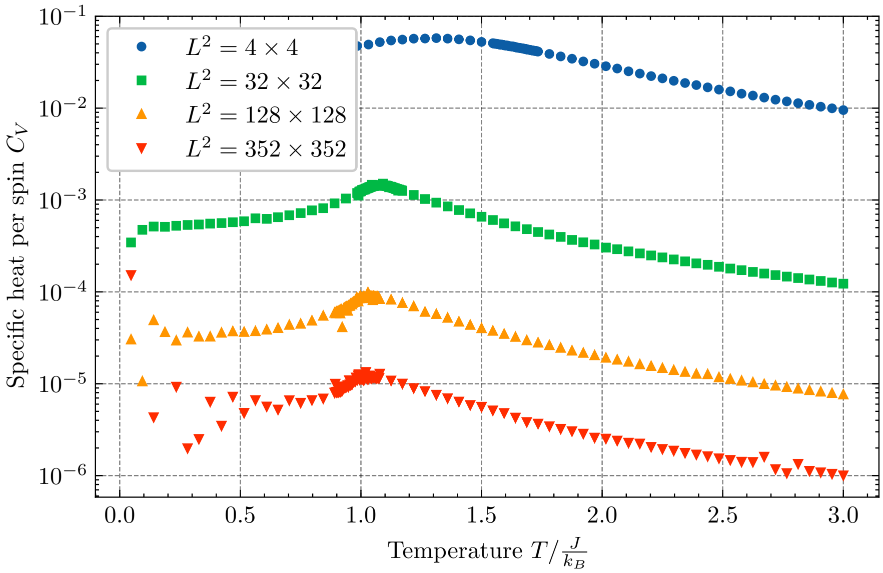

Figure 4.9.: Temperature dependence of the specific heat per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

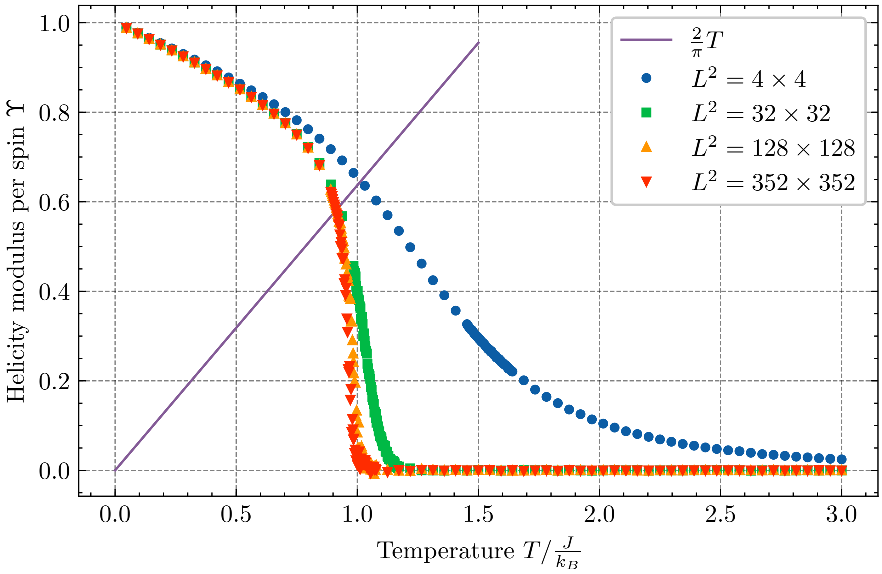

Figure 4.10.: Temperature dependence of the helicity modulus per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

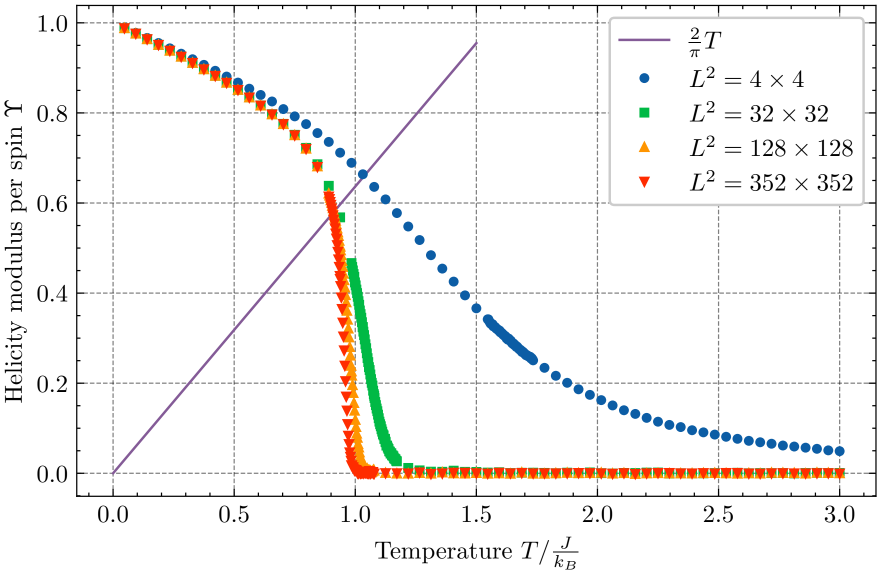

Figure 4.11.: Temperature dependence of the helicity modulus per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

4.3. Magnetization

4.3.1. Observable

Studying the total absolute magnetization per spin for the Metropolis (fig. 4.12) and Wolff (fig. 4.13) algorithms shows that, for low temperatures, the magnetization tends to 1, indicating the existence of a quasi-ordered low-temperature state. With increasing temperature, the magnetization decreases steadily at first and then rapidly until it slowly levels out in the unordered high-temperature state. One can also see finite-size effects as, for increasing lattice sizes, the form of the curve becomes more pronounced.

Magnetization Squared per Spin The plots for the absolute total magnetization per spin, its autocorrelation function, and integrated autocorrelation time may be found in Section D.2.

4.3.2. Magnetic Susceptibility

The magnetic susceptibility χ can be derived from $\langle M \rangle$ and $\langle M^2 \rangle$ according to Equation (2.7). Comparing the curve of the Metropolis (fig. 4.14) and Wolff (fig. 4.15) algorithms reveals a similar form. The magnetic susceptibility is small for low and high temperatures. A peak exists near the critical temperature, which slowly shifts to the left and becomes narrower as the lattice size increases and the peak's magnitude decreases.

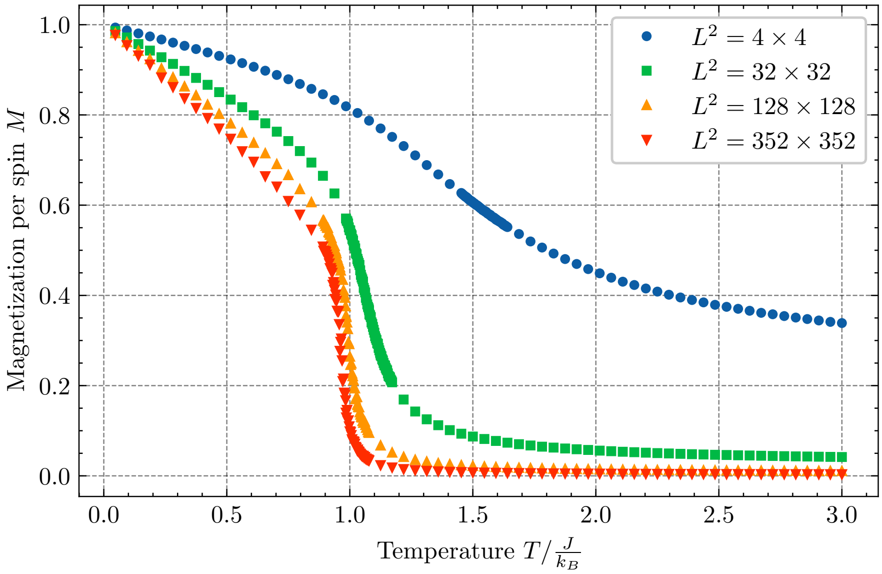

Figure 4.12.: Temperature dependence of the magnetization per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

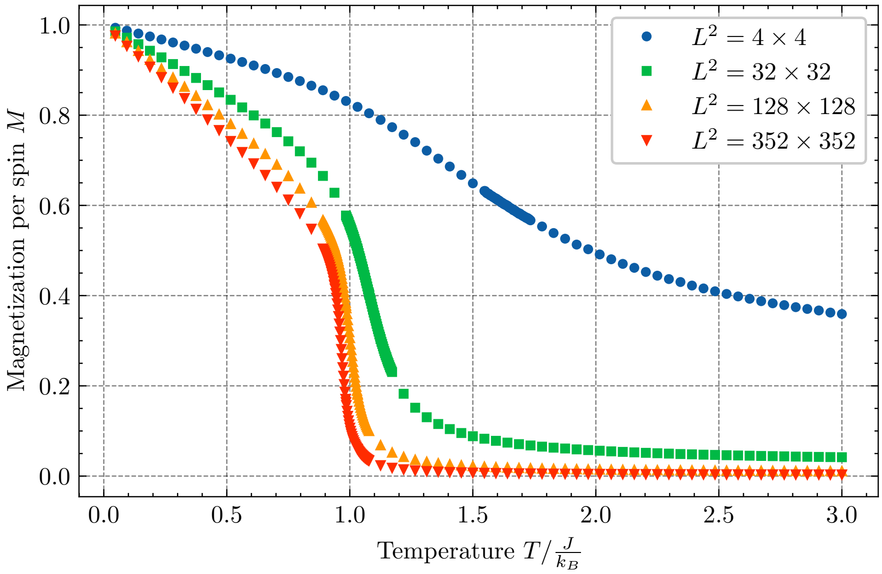

Figure 4.13.: Temperature dependence of the magnetization per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

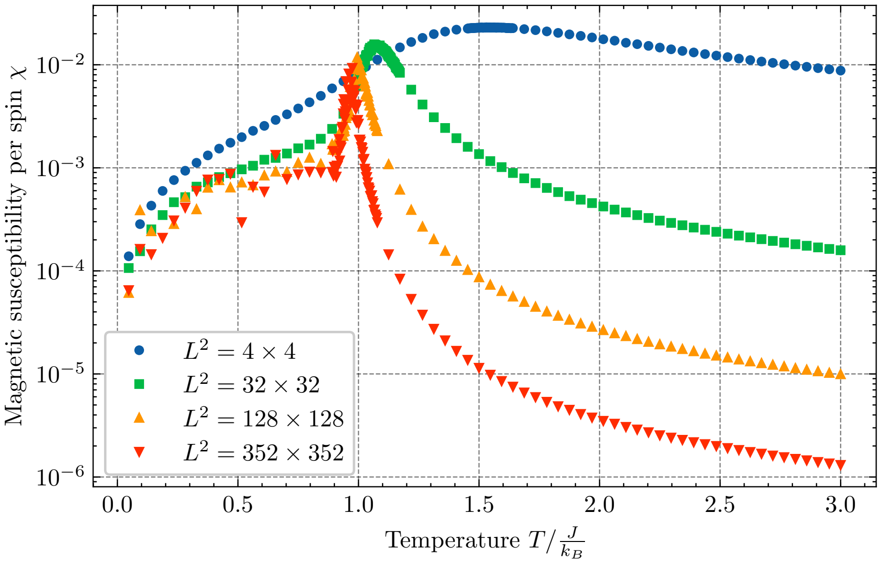

Figure 4.14.: Temperature dependence of the magnetic susceptibility per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

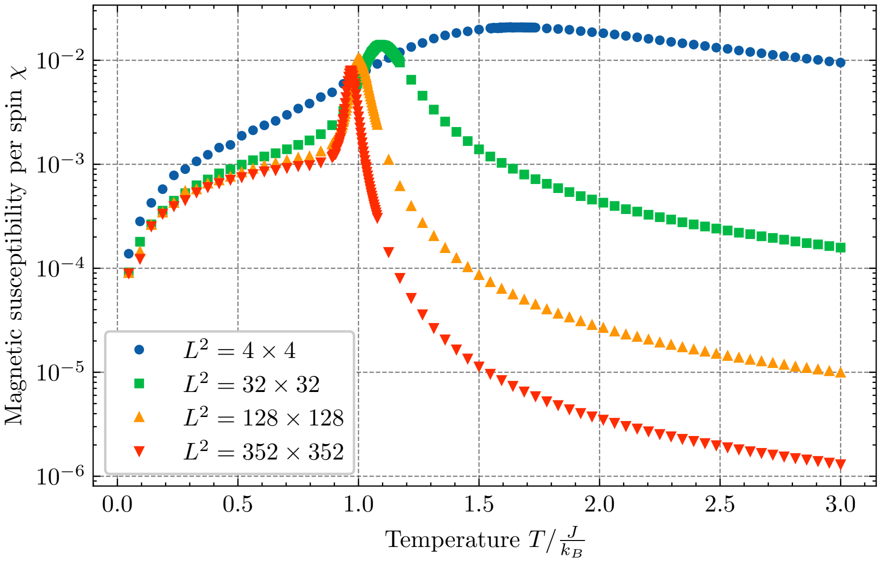

Figure 4.15.: Temperature dependence of the magnetic susceptibility per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

Near criticality, the curve of X obtained using the Wolff algorithm is a lot smoother, which can be attributed to the reduced critical slowing down effects experienced by the Wolff algorithm.

4.3.3. Estimating T_C using the Magnetic Susceptibility χ

As shown by Chung (1999), there exists a shifted temperature T^* that asymptotically approaches the critical temperature T_C for an infinite lattice

$$T^*(L) \approx T_C + \frac{\pi^2}{4c(\ln L)^2} \tag{4.1}$$

(Chung, 1999, eq. 3). The shifted temperature gives us a way to estimate the critical temperature T_C. To confirm the correlation between lattice size and shifted temperature, we now take the maximum magnetic susceptibility $\chi_{\max}$ per lattice size from Figure 4.14 and plot the temperature T where it occurs against $(\ln L)^{-2}$ in Figure 4.16.

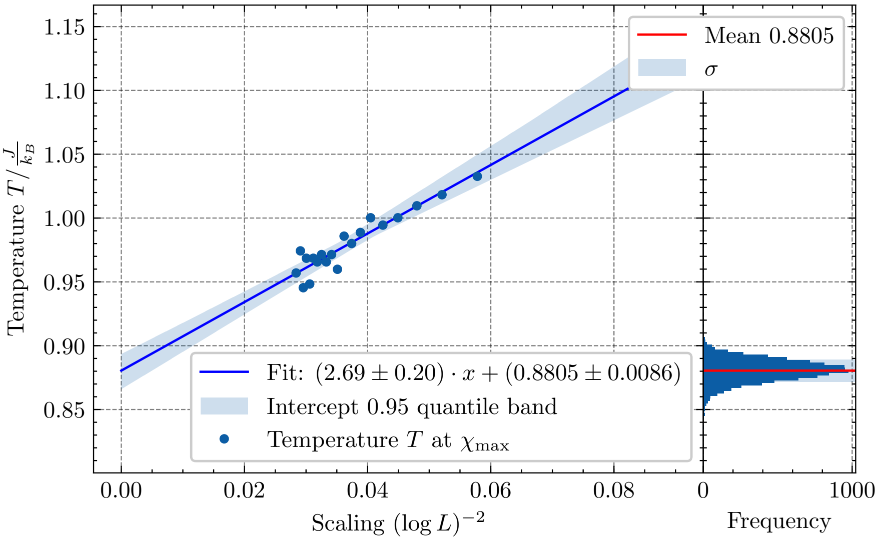

Figure 4.16.: The temperature at which χ is maximal against $(\ln L)^{-2}$ yields a linear correlation and confirms Equation (4.1) for the Metropolis algorithm. The data was fitted using least-squares linear regression model and the errors were estimated using a bootstrap approach.

We fitted an ordinary linear regression model using a least-squares method against our measurements, which confirms the correlation of lattice size and shifted temperature. The intercept of the fit gives the estimate for the critical temperature. To estimate our uncertainties, we used a bootstrap approach by resampling 10 000 times from our measurements and redoing our linear regression model each time. As seen on the right

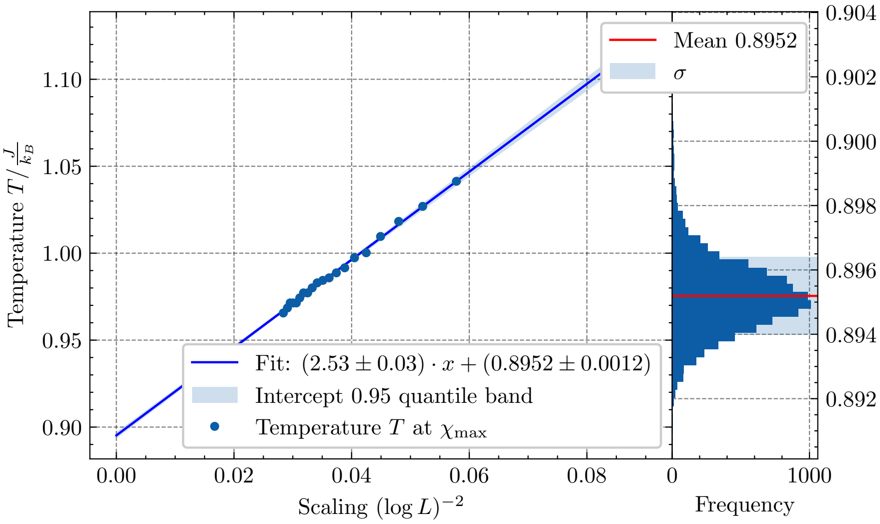

Figure 4.17.: The temperature at which χ is maximal against $(\ln L)^{-2}$ yields a linear correlation and confirms Equation (4.1) for the Wolff algorithm. The data was fitted using least-squares linear regression model and the errors were estimated using a bootstrap approach.

side in Figure 4.16, the distribution of intercepts b follows a Gaussian distribution and therefore the central limit theorem. The standard deviation σ of our bootstrapped intercepts was taken as the error of our estimate of the critical temperature

$$T_{C,\text{Metropolis}} = 0.8807(66)\,\frac{\text{J}}{\text{k}_\text{B}}. \tag{4.2}$$

Doing the same for the Wolff algorithm (fig. 4.17) for our $\chi_{\max}$ obtained from Figure 4.15 yields

$$T_{C,\text{Wolff}} = 0.8952(8)\,\frac{\text{J}}{\text{k}_\text{B}}. \tag{4.3}$$

Comparing the two estimates shows that the Wolff algorithm yielded a more precise measurement. The increase in precision can be attributed to the reduced critical slowing down effects experienced by the Wolff algorithm.

We can compare our result with some literature values:

- In Hsieh et al. (2013), the authors used a GPU-based Monte Carlo approach to estimate the critical temperature to $T_C = 0.8935(1)\,\frac{\text{J}}{\text{k}_\text{B}}$.
- In Olsson (1995), the authors used a CPU-based Monte Carlo approach to estimate the critical temperature to $T_C = 0.892\,13(10)\,\frac{\text{J}}{\text{k}_\text{B}}$.
- A theoretical transfer matrix approach employed by Mattis (1984) led to a critical temperature of $T_C \approx 0.8916\,\frac{\text{J}}{\text{k}_\text{B}}$.

4.4. Vortex Unbinding

To visualize the vortex unbinding at low temperatures, we performed a single run where we first heated a 64×64 lattice to $T = 1.5$ and allowed it to thermalize for 100 000 sweeps. The temperature was then lowered back down to $T = 0.05$ in 90 steps with 20 sweeps of thermalization at each temperature step. At $T = 0.05$, the system was given 90 000 sweeps to allow the vortices to unbind. An animation of the procedure can be viewed here, with some key frames shown

Upload to some video platform

Still need to export them from the animation and put them in the appendix

Below the critical temperature T_C, bound vortex-antivortex pairs appear. As the temperature is further lowered, the pairs come closer together until they annihilate. The lattice is left in a quasi-ordered low-temperature state.

5. Conclusion

Going to write this when I am rested

A. Source Code

The program requires the installation of an *gcc*[1] or *Intel® oneAPI DPC++/C++ Compiler*[2], with support for *C++26*. The build process was tested with versions 15.1.1 and 2025.0.4, respectively. The source code for the simulation, as well as the sources for this report, can be obtained from GitHub:

`https://github.com/lennartvrg/physik690-Bachelorarbeit`

As the program makes use of submodules, it is necessary to clone with

```
git clone --recurse-submodules [...]
```

to automatically initialize all submodules.

Dependencies The program requires several third-party libraries to compile. The following package names are those used in the *Ubuntu Launchpad*[3].

```
sudo apt-get install meson libpqxx-dev libsimde-dev
↪  libboost-all-dev libfftw3-dev libtomlplusplus-dev
↪  libflatbuffers-dev libsqlite3-dev libtbb-dev
```

The build process was tested using the versions available in Ubuntu *Ubuntu 25.04* as of August 26, 2025. Build errors might occur when using older versions. In particular, for `libpqxx-dev` at least version 7.10.x is required.

Building First, create a build directory in the repository root via `mkdir buildDir`. Then, the build files can be generated either for *gcc* via

```
CC=gcc CXX=g++ meson setup --reconfigure
↪ --buildtype=release buildDir/ .
```

or for the *Intel® oneAPI DPC++/C++ Compiler* via

```
CC=icx CXX=icpx meson setup --reconfigure
↪ --buildtype=release buildDir/ .
```

---

[1] `https://gcc.gnu.org/`

[2] `https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html`

[3] `https://launchpad.net/ubuntu`

B. TOML Config Schema

```toml
[simulation]
identifier = 0 # Multiple runs can be saved in the same database and
↪   this is their identifier
bootstrap_resamples = 200000 # The number of bootstrap resamples to
↪   perform

[storage]
engine = 2 # 1 is SQLite and 2 is PostgreSQL
connection_string = "" # Either a valid PostgreSQL connection string or
↪   a path to an SQLite file

[temperature]
max = 3.0 # The maximum T for the initial temperature range (0.0, max]
steps = 64 # The number of divisions for the temperatures
max_depth = 2 # The total number of zoom steps.

[vortices]
sizes = [64] # The lattice sizes for which the vortices are simulated

[metropolis]
num_chunks = 24 # The number of chunks when using Metropolis
sweeps_per_chunk = 50000 # The number of sweeps per chunk for Metropolis
sizes = [ # The lattice sizes for Metropolis
        4, 8, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208,
        ↪   224, 240, 256, 272, 288, 304, 320, 336, 352, 368
]

[wolff]
num_chunks = 6 # The number of chunks when using Wolff
sweeps_per_chunk = 50000 # The number of sweeps per chunk for Wolff
sizes = [ # The lattice sizes for Wolff
        4, 8, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208,
        ↪   224, 240, 256, 272, 288, 304, 320, 336, 352, 368
]
```

C. Database Schema

Add database schema

D. Additional Observables

D.1. Energy Squared per Spin

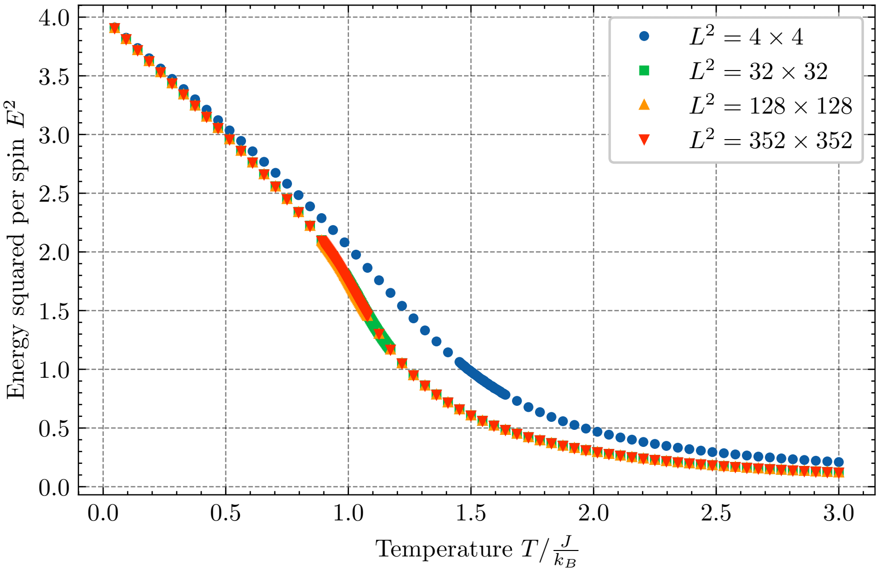

Figure D.1.: Temperature dependence of the energy squared per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

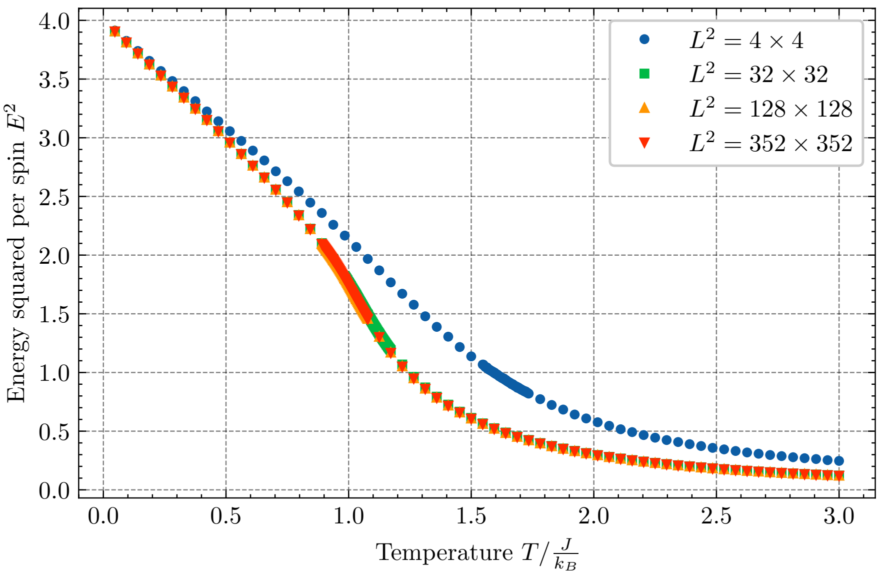

Figure D.2.: Temperature dependence of the energy squared per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

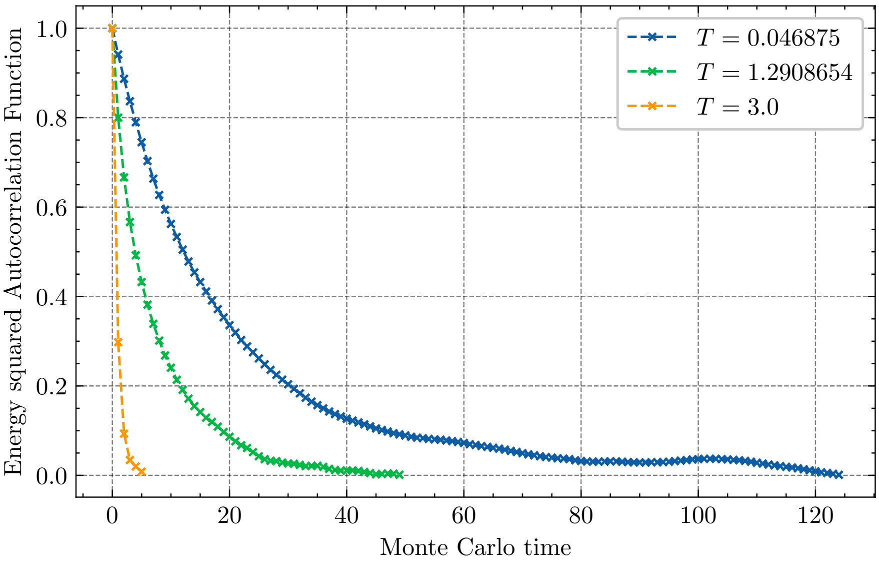

Figure D.3.: Autocorrelation function $\Gamma(\Delta t)$ of the energy squared per spin using the Metropolis algorithm.

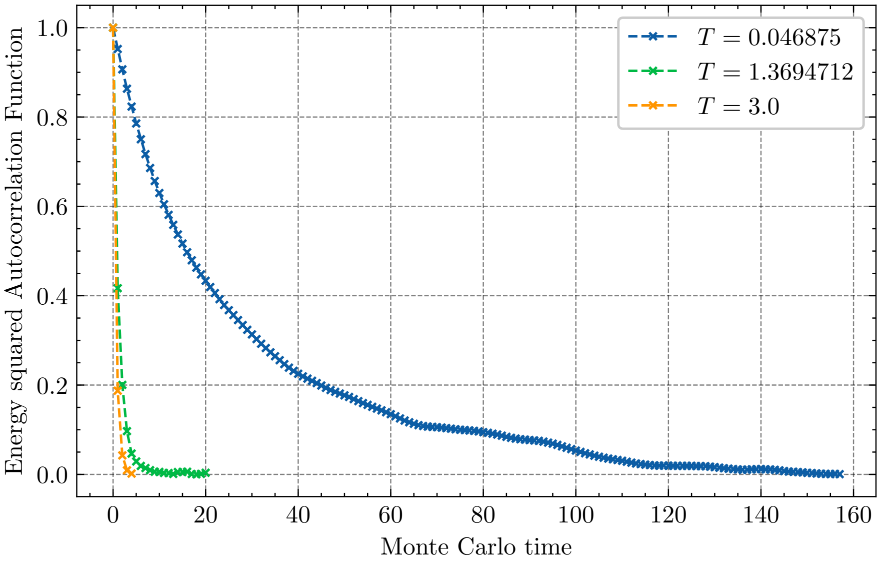

Figure D.4.: Autocorrelation function $\Gamma(\Delta t)$ of the energy squared per spin using the Wolff algorithm.

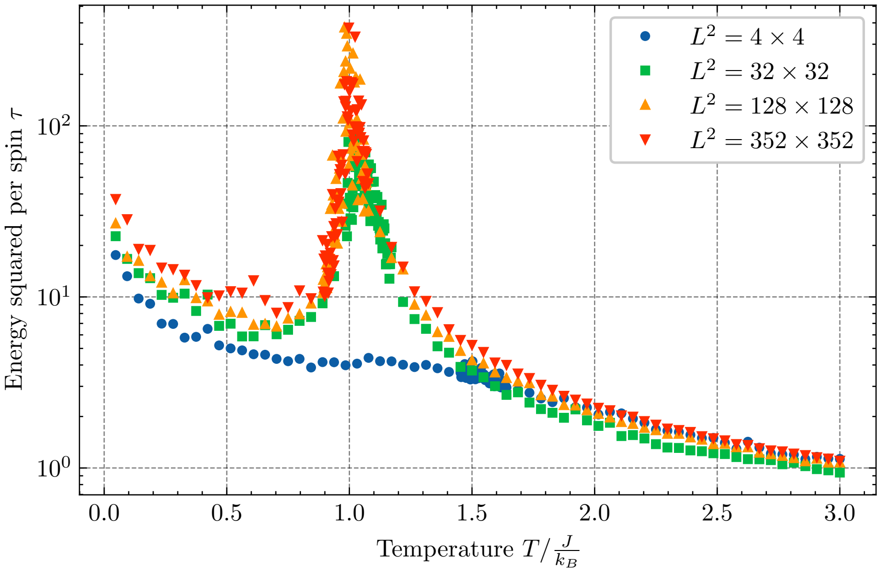

Figure D.5.: Integrated autocorrelation time τ of the energy squared per spin using the Metropolis algorithm.

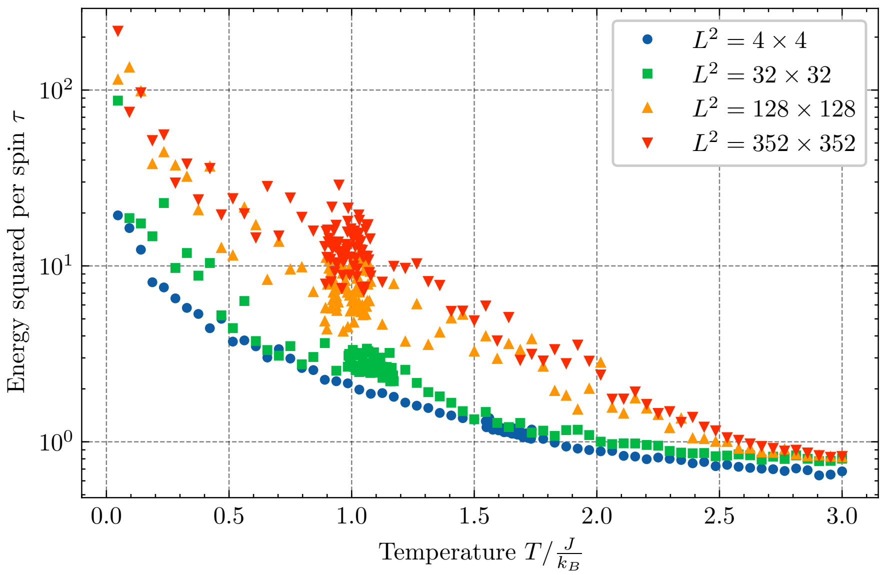

Figure D.6.: Integrated autocorrelation time τ of the energy squared per spin using the Wolff algorithm.

D.2. Magnetization Squared per Spin

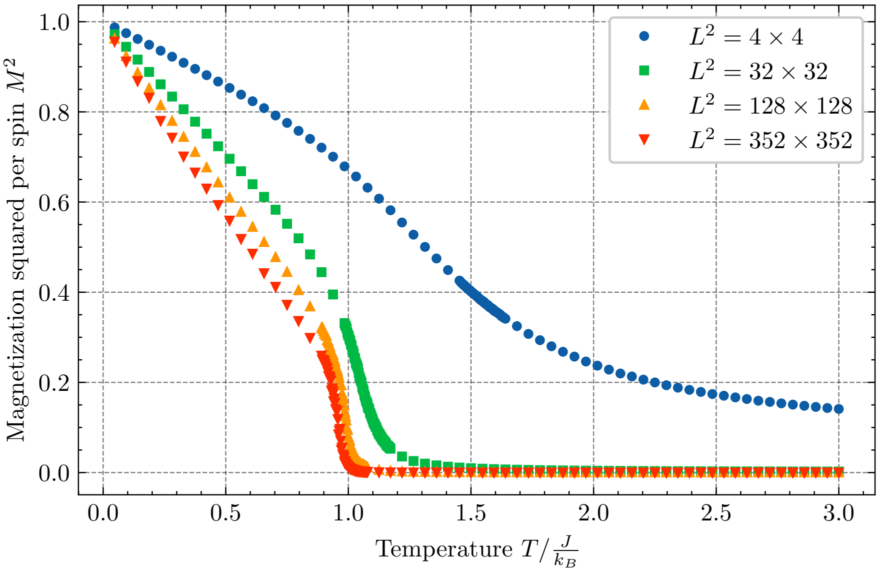

Figure D.7.: Temperature dependence of the magnetization squared per spin using the Metropolis algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

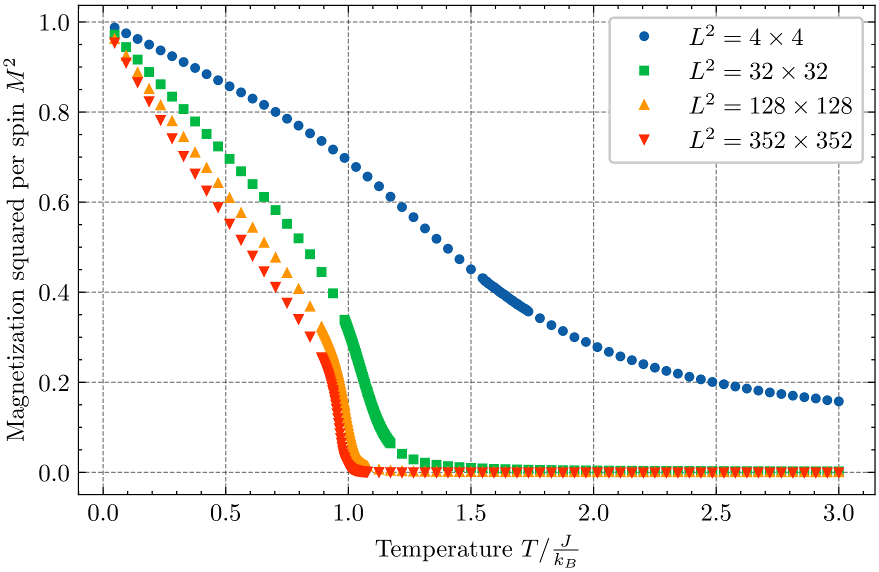

Figure D.8.: Temperature dependence of the magnetization squared per spin using the Wolff algorithm for lattice sizes $L \in \{4, 32, 128, 352\}$.

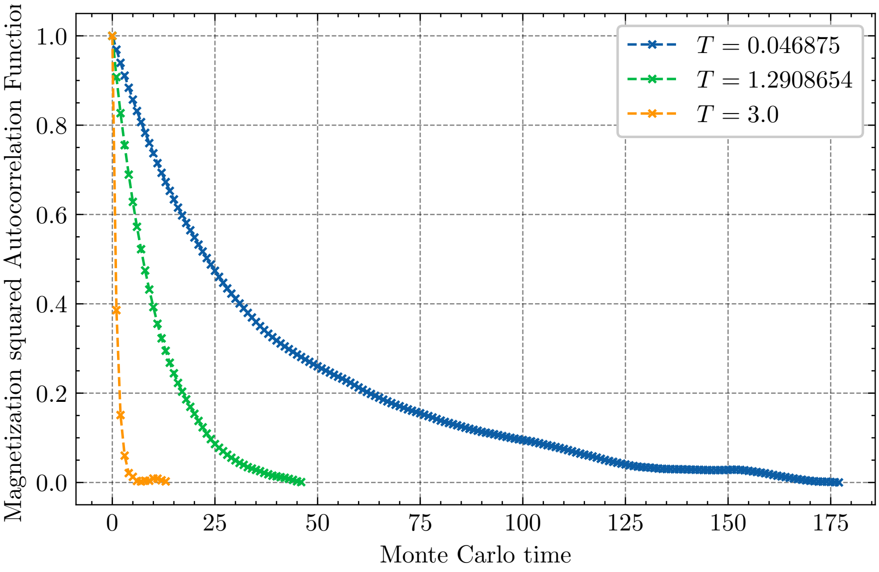

Figure D.9.: Autocorrelation function $\Gamma(\Delta t)$ of the magnetization squared per spin using the Metropolis algorithm.

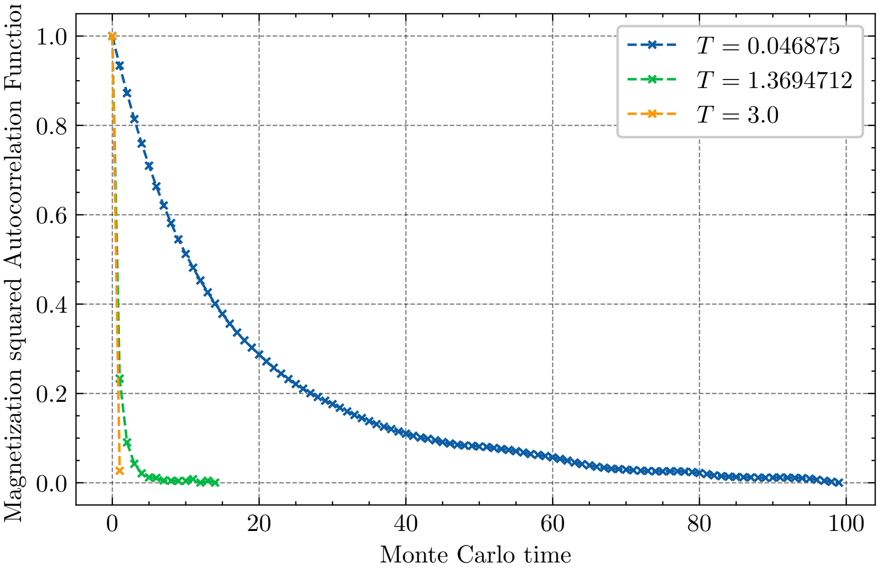

Figure D.10.: Autocorrelation function $\Gamma(\Delta t)$ of the magnetization squared per spin using the Wolff algorithm.

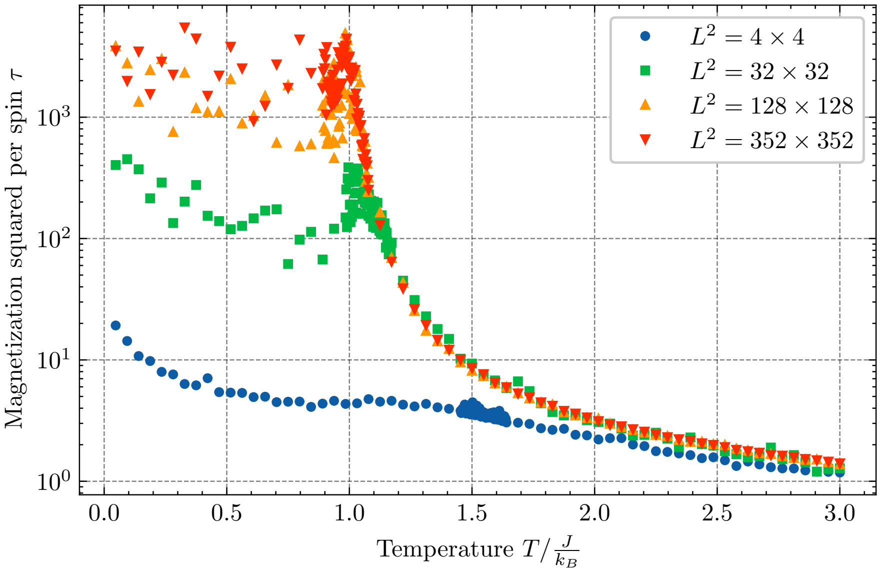

Figure D.11.: Integrated autocorrelation time τ of the magnetization squared per spin using the Metropolis algorithm.

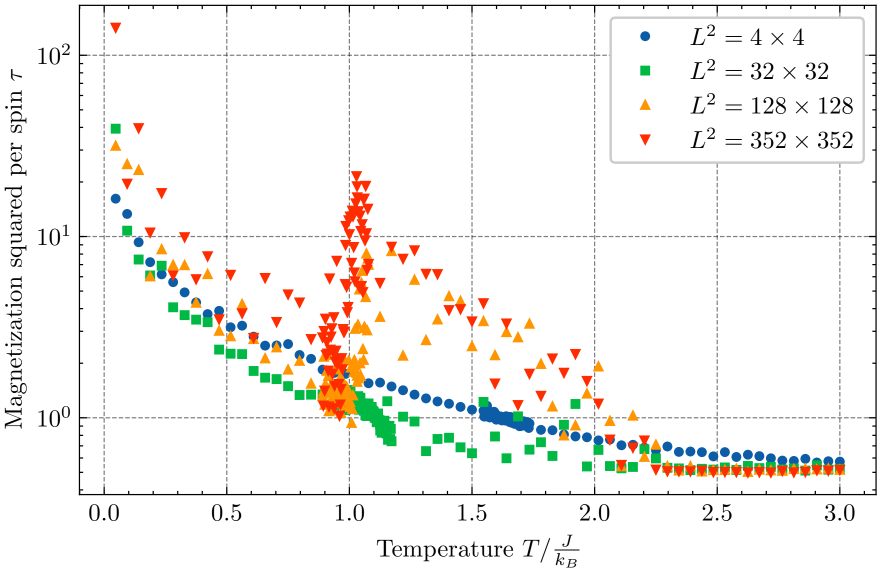

Figure D.12.: Integrated autocorrelation time τ of the magnetization squared per spin using the Wolff algorithm.

E. Residuals

Export residuals and plot them

List of Figures

4.1. Temperature dependence of the average cluster size per spin (Wolff) . . . 16
4.2. Temperature dependence of the energy per spin (Metropolis) 17
4.3. Temperature dependence of the energy per spin (Wolff) 17
4.4. Autocorrelation function $\Gamma(\Delta t)$ of the energy per spin (Metropolis) 19
4.5. Autocorrelation function $\Gamma(\Delta t)$ of the energy per spin (Wolff) 19
4.6. Integrated autocorrelation time τ of the energy per spin (Metropolis) . . . 20
4.7. Integrated autocorrelation time τ of the energy per spin (Wolff) 20
4.8. Temperature dependence of the specific heat per spin (Metropolis) 21
4.9. Temperature dependence of the specific heat per spin (Wolff) 22
4.10. Temperature dependence of the helicity modulus per spin (Metropolis) . . 22
4.11. Temperature dependence of the helicity modulus per spin (Wolff) 23
4.12. Temperature dependence of the magnetization per spin (Metropolis) . . . 24
4.13. Temperature dependence of the magnetization per spin (Wolff) 24
4.14. Temperature dependence of the magnetic susceptibility per spin (Metropolis) 25
4.15. Temperature dependence of the magnetic susceptibility per spin (Wolff) . 25
4.16. Estimating T_C using the Metropolis algorithm by plotting T where χ is maximal against $(\ln L)^{-2}$ 26
4.17. Estimating T_C using the Wolff algorithm by plotting T where χ is maximal against $(\ln L)^{-2}$ 27

D.1. Temperature dependence of the energy squared per spin (Metropolis) . . . 37
D.2. Temperature dependence of the energy squared per spin (Wolff) 38
D.3. Autocorrelation function $\Gamma(\Delta t)$ of the energy squared per spin (Metropolis) 38
D.4. Autocorrelation function $\Gamma(\Delta t)$ of the energy squared per spin (Wolff) . . 39
D.5. Integrated autocorrelation time τ of the energy squared per spin (Metropolis) 39
D.6. Integrated autocorrelation time τ of the energy squared per spin (Wolff) . 40
D.7. Temperature dependence of the magnetization squared per spin (Metropolis) 41
D.8. Temperature dependence of the magnetization squared per spin (Wolff) . 41
D.9. Autocorrelation function $\Gamma(\Delta t)$ of the magnetization squared per spin (Metropolis) 42
D.10. Autocorrelation function $\Gamma(\Delta t)$ of the magnetization squared per spin (Wolff) 42
D.11. Integrated autocorrelation time τ of the magnetization squared per spin (Metropolis) 43

D.12. Integrated autocorrelation time τ of the magnetization squared per spin (Wolff) 43

Bibliography

Evan Berkowitz. Lecture notes in physics760 - computational physics, November 2024. URL `https://ecampus.uni-bonn.de/ilias.php?ref_id=3481499&cmd=getFile&cmdClass=ilobjcloudgui&cmdNode=yh:ml&baseClass=ilRepositoryGUI&id=2990636427`.

S. G. Chung. Essential finite-size effect in the two-dimensional xy model. *Phys. Rev. B*, 60:11761–11764, Oct 1999. doi: 10.1103/PhysRevB.60.11761. URL `https://link.aps.org/doi/10.1103/PhysRevB.60.11761`.

Yun-Da Hsieh, Ying-Jer Kao, and Anders W Sandvik. Finite-size scaling method for the berezinskii–kosterlitz–thouless transition. *Journal of Statistical Mechanics: Theory and Experiment*, 2013(09):P09001, September 2013. ISSN 1742-5468. doi: 10.1088/1742-5468/2013/09/p09001. URL `http://dx.doi.org/10.1088/1742-5468/2013/09/P09001`.

Ernst Ising. Contribution to the Theory of Ferromagnetism. *Z. Phys.*, 31:253–258, 1925. doi: 10.1007/BF02980577.

J M Kosterlitz and D J Thouless. Ordering, metastability and phase transitions in two-dimensional systems. *Journal of Physics C: Solid State Physics*, 6(7):1181, apr 1973. doi: 10.1088/0022-3719/6/7/010. URL `https://dx.doi.org/10.1088/0022-3719/6/7/010`.

S. E. Krüger, R. Darradi, J. Richter, and D. J. J. Farnell. Direct calculation of the spin stiffness of the spin-$\frac{1}{2}$ heisenberg antiferromagnet on square, triangular, and cubic lattices using the coupled-cluster method. *Phys. Rev. B*, 73:094404, Mar 2006. doi: 10.1103/PhysRevB.73.094404. URL `https://link.aps.org/doi/10.1103/PhysRevB.73.094404`.

Takeo Matsubara and Hirotsugu Matsuda. A lattice model of liquid helium, i. *Progress of Theoretical Physics*, 16(6):569–582, 12 1956. ISSN 0033-068X. doi: 10.1143/PTP.16.569. URL `https://doi.org/10.1143/PTP.16.569`.

Daniel C. Mattis. Transfer matrix in plane-rotator model. *Physics Letters A*, 104(6):357–360, 1984. ISSN 0375-9601. doi: https://doi.org/10.1016/0375-9601(84)90816-8. URL `https://www.sciencedirect.com/science/article/pii/0375960184908168`.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 06 1953. ISSN 0021-9606. doi: 10.1063/1.1699114. URL `https://doi.org/10.1063/1.1699114`.

Peter Olsson. Monte carlo analysis of the two-dimensional xy model. ii. comparison with the kosterlitz renormalization-group equations. *Phys. Rev. B*, 52:4526–4535, Aug 1995. doi: 10.1103/PhysRevB.52.4526. URL `https://link.aps.org/doi/10.1103/PhysRevB.52.4526`.

Peter Olsson and Petter Minnhagen. On the helicity modulus, the critical temperature and monte carlo simulations for the two-dimensional xy-model. *Physica Scripta*, 43 (2):203, feb 1991. doi: 10.1088/0031-8949/43/2/016. URL `https://dx.doi.org/10.1088/0031-8949/43/2/016`.

Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944. doi: 10.1103/PhysRev.65.117. URL `https://link.aps.org/doi/10.1103/PhysRev.65.117`.

Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Phys. Rev. Lett.*, 58:86–88, Jan 1987. doi: 10.1103/PhysRevLett.58.86. URL `https://link.aps.org/doi/10.1103/PhysRevLett.58.86`.

S. Teitel and C. Jayaprakash. Phase transtions in frustrated two-dimensional XY models. *Phys. Rev. B*, 27:598–601, Jan 1983. doi: 10.1103/PhysRevB.27.598. URL `https://link.aps.org/doi/10.1103/PhysRevB.27.598`.

Ulli Wolff. Collective monte carlo updating for spin systems. *Phys. Rev. Lett.*, 62:361–364, Jan 1989. doi: 10.1103/PhysRevLett.62.361. URL `https://link.aps.org/doi/10.1103/PhysRevLett.62.361`.