Universität
Regensburg

Fakultät Physik

Numerical Simulation of the Critical Behaviour of the XY-Model

Bachelorarbeit

im Studiengang Computational Science
Schwerpunkt Mathematik/Physik

zur Erlangung des akademischen Grades
Bachelor of Science

Autor: Moritz Fink
MatNr. 1539016

Betreuer: Prof. Dr. Bali

Version vom: 7. April 2015

Contents

List of Figures

List of Tables

1 Introduction

Whenever physicists try to explain a phenomenon, they strive for creating a model that abstracts the problem and helps to solve or at least to understand it. One of the most famous models is the Ising model which was named after Ernst Ising who already did investigations on it in 1924. Although this model is rather a simple one, it needs a lot of effort to find out its behaviour theoretically. Through the introduction of numerical simulations due to the progress in computer technology, it has become easier to study this and other (more complex) models like the one examined in this thesis: the XY-model. This one is primarily analysed with simulations as a theoretical approach seems to be the harder way. Though the numerical approaches will not lead to an exact and analytical solution, they approximate the desired results via increasing computing power and improved methods of computation.
This bachelor thesis presents the physical and numerical fundamentals of the XY-model and shows how to set up the numerical simulation. The objective is to obtain results which describe the model's critical behaviour. At first, chapter 2 introduces technical terms concerning the physical background of the model. The next part contains numerical methods that can be helpful in any simulation on a physical model. The main part is chapter 4 which deals with the set-up of the simulation and the evaluation of its output.

2 Physical background

The motivation for the studies on the XY-model is the wide range of physical applications. A good example is the lanthanide *Gadolinium* which changes its magnetic behaviour according to its temperature:
At temperatures below a critical temperature (or Curie temperature) T_c this material behaves ferromagnetic so that all spins tend to point towards the same direction, whereas at temperatures above T_c their angles spread over the whole phase space and the solid becomes paramagnetic. This performance will also be observed in the numerical simulation of the XY-model in chapter 4.

2.1 Statistics

But before this simulation can be set up and run there are some basic statistical considerations to be made. Those deal with the systems composition, its states and the expected values of its observables.

2.1.1 Canonical ensemble

The canonical ensemble identifies a system A with contact to a larger reservoir B with temperature T (shown in figure 1). Both systems are only allowed to exchange energy, so that the systems hold a fixed number of particles N and a fixed volume V.

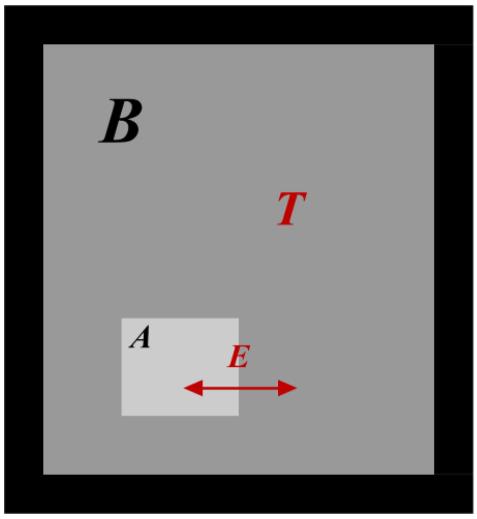

Figure 1: The canonical ensemble: system A is in contact with the larger reservoir B.

The system A can hold different energies E_n depending on its current state $\mu \in \mathcal{S}$ (where $\mathcal{S}$ is the state space). These states are distinguished by e.g. positions and momenta of the system's particles or (as used in this case) the particles' spins. The probability to find the system in state $\mu_n \in \mathcal{S}$ is given by

$$P(\mu_n) = \frac{1}{Z} e^{-\beta E_n}, \quad Z = \sum_n e^{-\beta E_n} \tag{1}$$

where $\beta = \frac{1}{k_B T}$ and Z is the canonical partition function that sums up all possible states (cf. [2]). Hence, $P(\mu_n)$ depends on the so called *Boltzmann factors* $e^{-\beta E_n}$ which in turn depend on the reservoir's temperature T.

2.1.2 Equilibrium

It has to be pointed out that the considerations mentioned above are only valid if the system has already reached thermal equilibrium. This term describes a situation in which the probability to go from state μ to state ν is the same as the other way around. This can be formalized by the *detailed balance* condition:

$$P(\mu)P(\mu \to \nu) = P(\nu)P(\nu \to \mu) \tag{2}$$

where $P(\mu)$ is the probability to find to system in state μ and $P(\mu \to \nu)$ is the probability to go from state μ to ν.
A system that resides in an arbitrary, non-equilibrated state will tend to its equilibrium and reach it after the *relaxation time* τ_{rel} which depends on the system's composition (cf. [14]).

2.1.3 Expected value, standard deviation & standard error

As the system strives for the equilibrium any variable X will as well strive for a certain value, namely the expected value $\langle X \rangle$ which can be estimated by the average over a long time (or rather many *Monte-Carlo* steps):

$$\langle X \rangle \approx \bar{X} = \sum_n X_n P_n = \frac{1}{Z} \sum_n X_n e^{-\beta E_n} \tag{3}$$

where the last step follows from (1) (cf. [15] & [16]).
Though the variable X will tend to its expected value, it may not exactly reach $\langle X \rangle$, but fluctuate around it. These fluctuations can be described by the standard deviation σ_X:

$$\sigma_X = \sqrt{\langle X^2 \rangle - \langle X \rangle^2} \tag{4}$$

where the expected values can be computed as above.
As the expected values have to be estimated in simulations with the estimator $\hat{\sigma}_X$ for a sample of size N

$$\hat{\sigma}_X = \sqrt{\frac{1}{N}\sum_{i=0}^{N} X_i^2 - \left(\frac{1}{N}\sum_{i=0}^{N} X_i\right)^2} \tag{5}$$

the standard error SE_X drops with the square root of the sample size:

$$SE_X = \frac{\hat{\sigma}_X}{\sqrt{N}} \tag{6}$$

2.2 Thermodynamics

The discussion on the XY-model presumes the introduction of some thermodynamic terms such as potentials, magnetization, susceptibility and heat capacity. Those concepts are specified in this chapter via definitions and further considerations.

2.2.1 Free energy

In thermodynamics there are a lot of potentials: internal energy, free energy, enthalpy and the Landau potential. But for the canonical ensemble the Helmholtz free energy F will be used in the following chapters. This energy can be derived[1] as:

$$F(T,V,N) = -k_B T \; lnZ(T,V) \tag{7}$$

2.2.2 Magnetization

Magnetization describes the *magnetic momentum* $\vec{\mu}$ per volume V (cf. [18]):

$$\vec{M} = \frac{1}{V}\langle\vec{\mu}\rangle \tag{8}$$

For an arbitrary state n the system's magnetization is $M_n = -\frac{\partial E_n}{\partial B}$ [2] and consequently the expected value for the magnetization can be calculated as

$$\langle M\rangle = \frac{1}{Z}\sum_n M_n e^{-\beta E_n} \tag{9}$$

[1]For a detailed derivation take a look at [3] or [11].

[2]Assuming that the magnetic field $\vec{B} = \begin{pmatrix} B_x \\ 0 \end{pmatrix}$ has only one non-zero component B_x.

2.2.3 Magnetic susceptibility

A relation between magnetization M and its behaviour towards a change of the magnetic field B is given by the magnetic susceptibility:

$$\chi := \left(\frac{\partial M}{\partial B}\right)_T \tag{10}$$

Its calculation can be done by measuring the variance of the magnetization per spin $\sigma_m^2 := \langle (m - \langle m \rangle)^2 \rangle$ (cf. [18]):

$$\begin{aligned} \chi &= \frac{1}{V}\left(\frac{\partial \langle m \rangle}{\partial B}\right)_T = \frac{1}{V}\left(\frac{\partial}{\partial B}\frac{tr\{e^{-\beta\mathcal{H}}m\}}{tr\{e^{-\beta\mathcal{H}}\}}\right)_T \\ &= \frac{\beta}{V}\left\langle (m - \langle m \rangle)^2 \right\rangle = \frac{\beta}{V}\sigma_m^2 \end{aligned} \tag{11}$$

2.2.4 Specific heat capacity

A measure for the amount of heat Q needed to change a system's temperature by ΔT is given by the heat capacity (cf. [17]):

$$C := \frac{Q}{\Delta T} = \frac{\Delta U - W}{\Delta T} \tag{12}$$

At fixed volume the *compression work* W cancels and consequently the heat capacity at constant volume is

$$C_V := \left(\frac{\partial U}{\partial T}\right)_V \tag{13}$$

An easy way for calculating C_V is to find its dependency on the energy's variance $\sigma_E^2 := \langle E^2 \rangle - \langle E \rangle^2$:

$$\begin{aligned} C_V &= \left(\frac{\partial U}{\partial T}\right)_V = \left(\frac{\partial \beta}{\partial T}\frac{\partial U}{\partial \beta}\right)_V = \frac{1}{k_B T^2}\frac{\partial}{\partial \beta}\left(\frac{1}{Z}\frac{\partial Z}{\partial \beta}\right) \\ &= \frac{1}{k_B T^2}\left(\frac{1}{Z}\frac{\partial^2 Z}{\partial^2 \beta} - \frac{1}{Z^2}\left(\frac{\partial Z}{\partial \beta}\right)^2\right) \\ &= \frac{1}{k_B T^2}\left(\langle E^2 \rangle - \langle E \rangle^2\right) = k_B \beta^2 \sigma_E^2 \end{aligned} \tag{14}$$

using (1) to get the last line.

3 Basic principles of simulations

The behaviour of a system with the properties discussed in chapter 2 can often only be determined by a numerical simulation, as analytical approaches may not be possible. E.g. the XY-model has an infinite number of possible states. That fact makes it hard to handle and demands several techniques for simulating the system.
But how can such a system's observables be calculated at all, as the expected value claims to contain all of these states (see (1))? The answer lies in only considering important states that are specified in 3.1. The next step is generating a huge amount of those states, selecting the uncorrelated ones (see 3.3) and performing calculations with the states obtained. The principles for executing these steps are described in this chapter.

3.1 Importance sampling

Importance sampling is motivated by the simple fact, that in a sufficiently complex system there are too many states to be simulated by a computer in reasonable time. But when it comes to predicting any observable $\mathcal{O}$, one needs to sum up all possible states (as shown in (3)). As there might be a system with almost an infinite number of states, this is obviously not possible. The solution lies in taking only states with high probability. In physical systems these might be e.g. states with low energy. The idea of importance sampling is now to generate states according to their probability.
Assume that there is a function $f(x)$ where x is generated according to the *nominal distribution* $p(x)$. The expected value of $f(x)$ can be rewritten by inserting an *importance distribution* $q(x)$ (cf. [13]):

$$\langle f(x)\rangle_p = \frac{1}{N}\sum_{i=1}^{N} f(x_i)p(x_i) = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)p(x_i)}{q(x_i)}q(x_i) = \left\langle \frac{f(x)p(x)}{q(x)}\right\rangle_q \tag{15}$$

where x is now generated according to $q(x)$.
In the next step, this procedure is applied to a sample of states $\mathcal{S}_N \subset \mathcal{S}$. Then the estimator for the observable $\mathcal{O}$ is:

$$\hat{\mathcal{O}} = \frac{\sum_{\mu\in\mathcal{S}_N}\mathcal{O}_\mu P_B^{-1}e^{-\beta E_\mu}}{\sum_{\mu\in\mathcal{S}_N} P_B^{-1}e^{-\beta E_\mu}} \tag{16}$$

weighted with the *Boltzmann weights* $P_B = \frac{1}{Z}e^{-\beta E_\mu}$, so that (16) cancels down to:

$$\hat{\mathcal{O}} = \frac{1}{N}\sum_{\mu\in\mathcal{S}_N}\mathcal{O}_\mu \tag{17}$$

Be aware that the states now have to be generated according to the *Boltzmann distribution.*

3.2 Markov chain

The Markov chain (named after Andrey Markov) denotes a discrete sequence of states from the countable state space $\mathcal{S}$, where every state $\mu^{(t+1)}$ at time $t+1$ only depends on the present state $\mu^{(t)}$, i.e.

$$P(\mu^{(t+1)} \in A|\mu^{(t)} = \nu, \mu^{(t-1)} \in A_{n-1}, ..., \mu^{(0)} \in A_0) = P(\mu^{(t+1)} \in A|\mu^{(t)} = \nu) \tag{18}$$

where $A_0, ..., A_{n-1}, A \subset \mathcal{S}$ and $\nu \in \mathcal{S}$ (cf. [1]). P denotes the probability to get from a given state to another state[3] in one time step.
In this thesis a time-homogeneous Markov chain will be used where the transition probability does not depend on t:

$$P(\mu^{(t+1)} = \nu|\mu^{(t)} = \mu) = P(\mu^{(t)} = \nu|\mu^{(t-1)} = \mu) \tag{19}$$

The transition probabilities should satisfy the properties of a probability distribution:

a) $P(\mu \to \nu) \geq 0, \forall \mu, \nu \in \mathcal{S}$

b) $\sum_{\nu \in \mathcal{S}} P(\mu \to \nu) = 1, \forall \mu \in \mathcal{S}$

In consequence, not every state has to be reachable from a current state within a single time step[4], but in a finite number of steps. The application of property b) above to the *detailed balance* condition (defined in (2)) leads to the probability to find the system in state ν at time $t+1$ (after equilibration)

$$P(\mu^{(t+1)} = \nu) = \sum_{\mu \in \mathcal{S}} P(\mu \to \nu) P(\mu^{(t)} = \mu) \tag{20}$$

3.3 Autocorrelation time

When it comes to simulating, only uncorrelated states need to be added to the Markov chain because not every time step simulated will create a 'really' new state. The object of this chapter is to find a function that describes how many time steps have to be simulated to guarantee that the state obtained is not correlated to the previous one. This function is called the integrated autocorrelation time τ_{int} (cf. [9]).
Therefore, the autocorrelation function for an arbitrary observable $\mathcal{O}$ is defined by

$$C(\mathcal{O}_i, \mathcal{O}_{i+t}) := \langle \mathcal{O}_i \mathcal{O}_{i+t} \rangle - \langle \mathcal{O}_i \rangle \langle \mathcal{O}_{i+t} \rangle \tag{21}$$

[3]This state may also be the current one again.
[4]$P(\mu \to \nu)$ can be zero!

Assuming that the result does not depend on the index i but only on the distance t, the definition cancels down to

$$C(\mathcal{O}_i, \mathcal{O}_{i+t}) \approx \langle \mathcal{O}_0 \mathcal{O}_t \rangle - \langle \mathcal{O} \rangle^2 = \langle (\mathcal{O}_0 - \langle \mathcal{O} \rangle)(\mathcal{O}_t - \langle \mathcal{O} \rangle) \rangle =: C_{\mathcal{O}}(t) \tag{22}$$

For great amount of data, $C_{\mathcal{O}}(t)$ can be estimated by $C_{\mathcal{O}}(t) \approx \frac{1}{N}\sum_{i=1}^{N} C(\mathcal{O}_i, \mathcal{O}_{i+t})$. With these equations the variance for an observable $\mathcal{O}$ can be approximated by

$$\begin{aligned}\sigma_{\mathcal{O}}^2 &= \left\langle \left(\frac{1}{N} \sum_{i=1}^{N} \mathcal{O}_i - \langle \mathcal{O} \rangle \right)^2 \right\rangle = \frac{1}{N^2} \left\langle \sum_{i,j=1}^{N} (\mathcal{O}_i - \langle \mathcal{O} \rangle)(\mathcal{O}_j - \langle \mathcal{O} \rangle) \right\rangle = \\ &= \frac{1}{N^2} \sum_{i,j=1}^{N} C(\mathcal{O}_i, \mathcal{O}_{i+t}) \approx \frac{1}{N^2} \sum_{i=1}^{N} \sum_{t=-\infty}^{+\infty} C_{\mathcal{O}}(t) = \\ &= 2\frac{C_{\mathcal{O}}(0)}{N} \left(\frac{1}{2} + \sum_{t=1}^{\infty} \frac{C_{\mathcal{O}}(t)}{C_{\mathcal{O}}(0)} \right) = 2\tau_{\mathcal{O},int} \frac{C_{\mathcal{O}}(0)}{N}\end{aligned} \tag{23}$$

where $\tau_{\mathcal{O},int} := \frac{1}{2} + \sum_{t=1}^{\infty} \frac{C_{\mathcal{O}}(t)}{C_{\mathcal{O}}(0)}$.
As $C_{\mathcal{O}}(0)$ represents the variance of all data obtained, the number of independent measurements can be calculated by applying (23):

$$N_{independent} = \frac{C_{\mathcal{O}}(0)}{\sigma_{\mathcal{O}}^2} = \frac{N}{2\tau_{\mathcal{O},int}} \tag{24}$$

Hence, the distance between two uncorrelated states needs to be at least $2\tau_{\mathcal{O},int}$.

3.4 Bootstrap method

As there will be calculations on *secondary quantities* like the magnetic susceptibility and the specific heat capacity in the simulation, the bootstrap method will now be discussed in order to be able to calculate the standard deviations of these quantities.
Assume that an observable $\mathcal{O}$ was estimated by computing its estimator $\hat{\mathcal{O}}$ for a sample of size N. First of all, generate M[5] pseudo samples $\mathcal{O}^{(1)}, ...\mathcal{O}^{(M)}$ by randomly choosing N data points of the original sample[6] and add them to these pseudo samples (cf. [10]). Now calculate the quantities $\hat{\mathcal{O}}^{(i)}$ for $i = 1, ..., M$. The standard deviation of $\hat{\mathcal{O}}$ is then given by

$$\sigma_B(\hat{\mathcal{O}}) = \sqrt{\frac{1}{M} \sum_{i=1}^{M} (\hat{\mathcal{O}}^{(i)} - \hat{\mathcal{O}})^2} \tag{25}$$

[5]Choose a large M for better precision.
[6]It is possible to draw one data point more than once!

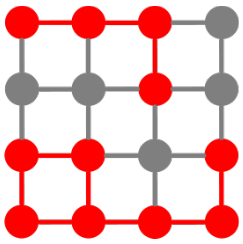

Figure 2: Possible spin cluster with bonds on a 4×4 lattice (with periodic boundary condition).

4 The two-dimensional XY model

The XY-model (or planar model) is one of the special cases of the *n-vector* (or $\mathcal{O}(n)$) model with $n = 2$, where n describes the degrees of freedom. Other models would be the Ising model ($n = 1$) and the Heisenberg model ($n = 3$). This differentiation was introduced by H.E. Stanley (see [19]).
This chapter will depict the theoretical aspects and fundamentals as well as the practical results when it comes to simulating the model on a two-dimensional lattice.

4.1 Preface

Consider a d-dimensional lattice Λ (in this case $d = 2$) with a spin $\vec{s_i} \in \Lambda$ on each site. Each spin is represented by a unit-length vector $\vec{s_i} = \begin{pmatrix} cos(\theta_i) \\ sin(\theta_i) \end{pmatrix}$ containing the angle $\theta_i \in [0, 2\pi)$. The Hamiltonian of the system is given by:

$$\mathcal{H} = -\sum_{i \neq j} J_{ij} \vec{s_i} \cdot \vec{s_j} - \sum_i B_i cos(\theta_i) = -J \sum_{\langle i,j \rangle} cos(\theta_i - \theta_j) - B \sum_i cos(\theta_i) \tag{26}$$

In order to simplify the problem two assumptions have been made here. On the one hand the coupling of two spins was set to be $J_{ij} = J$ for all pairs of nearest neighbours $\langle i, j \rangle$ or otherwise $J_{ij} = 0$. On the other hand the magnetic field was set constant on the whole lattice, i.e. $B_i = B \ \forall_{i \in \Lambda}$.

4.2 Wolff algorithm

The Wolff algorithm is named after Ulrich Wolff who published it in 1989 (cf. [20] & [21]). The algorithm is a cluster algorithm which means that not just a single spin is updated in one Monte-Carlo step but even a whole group (cluster) of spins. Figure 2 shows a possible constellation for such a cluster. The usage of these clusters allows numerical simulations to perform much better around the phase transition[7] and thus avoid the *critical slowing down*.

[7]For further information on the Kosterlitz-Thouless transition see [8].

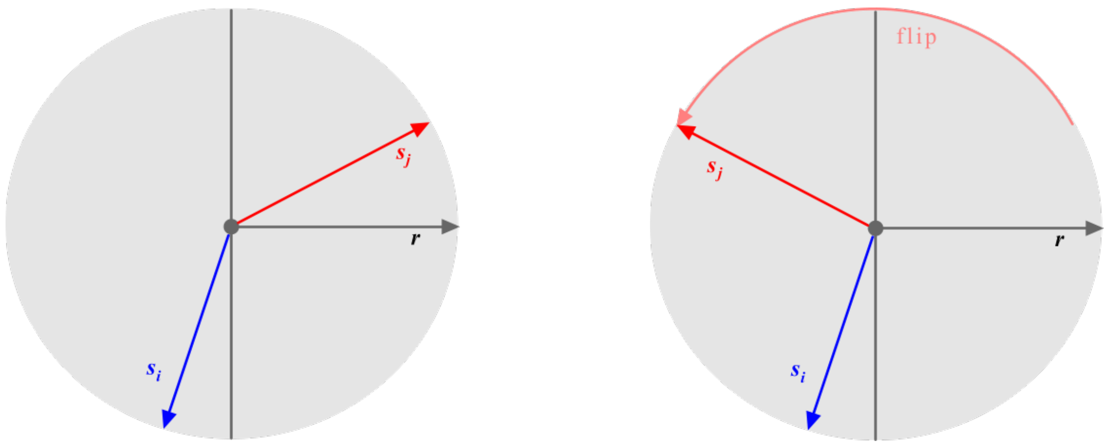

Figure 3: Spin s_j before (left) and after (right) the flip.

4.2.1 Description

A spin flip (shown in figure 3) is defined by the operator $R(\vec{r})$:

$$R(\vec{r})\vec{s_i} = \vec{s_i} - 2(\vec{s_i} \cdot \vec{r})\vec{r} \tag{27}$$

Hence, the spin $\vec{s_i} \in \Lambda$ is reflected with respect to the orthogonal to $\vec{r}$.
A single cluster can now be constructed by applying the following steps:

a) Randomly choose a seed spin $\vec{s_i} \in \Lambda$ and a reference direction $\vec{r} \in [0, 2\pi)$.

b) Add $\vec{s_i}$ to the cluster $c \subseteq \Lambda$ and flip the spin $\vec{s_i} \rightarrow R(\vec{r})\vec{s_i}$.

c) Consider all nearest neighbours $\vec{s_j} \notin c$ of $\vec{s_i}$ and activate the bond $\langle i, j \rangle$ with probability

$$\begin{aligned} P(\vec{s_i}, \vec{s_j}) &= 1 - exp\left\{min\left[0, \beta\vec{s_i} \cdot (1 - R(\vec{r}))\,\vec{s_j}\right]\right\} \\ &= 1 - exp\left\{min\left[0, 2\beta(\vec{r} \cdot \vec{s_i})(\vec{r} \cdot \vec{s_j})\right]\right\} \end{aligned} \tag{28}$$

If $\langle i, j \rangle$ is activated, then $\vec{s_j}$ is added to the cluster and $\vec{s_j}$ is flipped.

d) Repeat *c*) for all spins $\vec{s_i}$ that were newly appended to the cluster.

4.2.2 Properties

This algorithm needs to hold some properties which guarantee that the system is properly updated in every Monte-Carlo step. These properties are detailed balance and ergodicity. In the following two paragraphs it is verified that the Wolff algorithm complies with both of them.

4.2.2.1 Detailed Balance

The algorithm guarantees detailed balance (cf. [20]):

$$\begin{aligned} \frac{P\left(\{\vec{s_i}\} \to \{\vec{s_i}'\}\right)}{P\left(\{\vec{s_i}'\} \to \{\vec{s_i}\}\right)} &= \prod_{\langle i,j\rangle \in \partial c} \frac{1 - P\left(R(\vec{r})\vec{s_i}, \vec{s_j}\right)}{1 - P\left(R(\vec{r})\vec{s_i}', \vec{s_j}'\right)} \\ = exp\left\{\beta \sum_{\langle i,j\rangle \in \partial c} \vec{s_i} \cdot \left[R(\vec{r}) - 1\right] \vec{s_j}\right\} &= exp\left\{\beta \sum_{\langle x,y\rangle} \left(\vec{s_i}' \cdot \vec{s_j}' - \vec{s_i} \cdot \vec{s_j}\right)\right\} \\ = e^{\beta \Delta E} = \frac{\frac{1}{Z} e^{\beta E'}}{\frac{1}{Z} e^{\beta E}} &= \frac{P(\{\vec{s_i}'\})}{P\left(\{\vec{s_i}\}\right)} \end{aligned} \tag{29}$$

where $\Delta E := E' - E$ and ∂c is the sum of all bonds $\langle i, j\rangle$ with $\vec{s_i} \in c$ and $\vec{s_j} \notin c$.

4.2.2.2 Ergodicity

The system is ergodic if this algorithm now guarantees that any arbitrary state of the system can be reached in a finite number of steps. This is actually the case because it is possible to build clusters that consist of a single spin. When it comes to flipping the cluster, this spin is mirrored about the orthogonal of an arbitrary vector $\vec{r}$ and is consequently able to point in any direction desired. Therefore it takes at most $|\Lambda|$ of these cluster flips to get from any state to another one (c.f. [20]).

4.3 Critical exponents

The objective of this chapter (cf. [12]) is to find a law that describes the critical behaviour of the XY-model. Therefore one often finds a physical property F that behaves depending on the reduced temperature $\epsilon = \frac{T - T_c}{T_c}$ like

$$F(\epsilon) = a\epsilon^{\varphi}(1 + b\epsilon^{x} + ...), \qquad a, b \in \mathbb{R}, x > 0 \tag{30}$$

where φ denotes a critical exponent. When T converges to T_c and accordingly $\epsilon \to 0$ the ϵ-terms in brackets vanish and hence

$$F(\epsilon) \sim \epsilon^{\varphi} \tag{31}$$

As this critical exponent depends on the direction of approximation to T_c, one defines φ' for $T \nearrow T_c$ and φ for $T \searrow T_c$. With these definitions the magnetic susceptibility and the specific heat capacity can be written as follows:

$$\chi \sim \begin{cases} (-\epsilon)^{-\gamma'}, & T \nearrow T_c \\ \epsilon^{-\gamma}, & T \searrow T_c \end{cases} \tag{32}$$

$$C_V \sim \begin{cases} (-\epsilon)^{-\alpha'}, & T < T_c \\ \epsilon^{-\alpha}, & T > T_c \end{cases} \tag{33}$$

The magnetization only needs one critical exponent because it is not defined at high temperatures:

$$M \sim (-\epsilon)^{\beta} \tag{34}$$

The correlation length (which will be discussed in 4.4.4) can either be defined through ν and ν' or η and the correlation function $G(r)$:

$$\xi \sim \begin{cases} (-\epsilon)^{-\nu'}, & T \nearrow T_c \\ \epsilon^{-\nu}, & T \searrow T_c \end{cases} \tag{35}$$

$$G(r) \approx \frac{1}{r^{d-2+\eta}}, \qquad T = T_c \tag{36}$$

with lattice dimension d.
By adding the critical exponent δ for critical isotherms, these exponents are related through the following inequalities:

$$\begin{aligned} \alpha' + 2\beta + \gamma' &\geq 2 \\ \alpha' + \beta(1+\delta) &\geq 2 \\ \beta(\delta - 1) &\leq \gamma' \end{aligned} \tag{37}$$

The exponents are almost equal for every thermodynamic system and they only depend on the lattice dimension d, the spin dimension n and the range of spin interaction. As the systems gain complexity with every lattice and spin dimension, analytical solutions for high dimensional systems become difficult or maybe even impossible. Numerical simulation is the most important tool for approximating the critical exponents in those cases.

4.4 Simulation

Now that the theoretic aspects were discussed an executable program has to be set up which allows the user to simulate the XY-model and to obtain data on several observables. The following chapter describes the procedure of simulation and its results around the critical temperature of the Kosterlitz-Thouless transition $T_c \approx 0.894$ (c.f. [4]). Note that this temperature is only valid for an infinitely large lattice. On the simulated lattices below there will occur *finite-size effects* that distort the results and the simulated T_c will be higher than the T_c of the infinite case. The results are illustrated in the interval $T \in (0, 2]$.

4.4.1 Implementation

The simulation[8] starts with generating a $L \times L$ lattice where every node is given a spin. This is a random number between 0 and 1 representing the interval $[0, 2\pi)$ of all possible configurations. This procedure is called a *hot start*, in contrast to a *cold start* where all spins are set in the same direction.
Now the temperature is set to T_{max}, the highest temperature to be simulated and the system starts equilibrating by performing a given number of cluster flips. In order to build and flip these clusters the Wolff algorithm is applied. For the implementation the algorithm was slightly modified so that all spin flips can be done when the whole cluster is build. After every cluster flip the magnetization and energy density is saved in an array and - at certain temperatures - also saved to *XY-equilibration.dat*.
With these data obtained in the equilibration it is now possible to determine the integrated autocorrelation time for magnetization and energy. This is the point where the production run starts. Similarly to the equilibration run, cluster flips are performed using the Wolff algorithm and several quantities are saved in an array, taking into account the integrated autocorrelation time. Among these observables are magnetization and energy density, cluster size and $\vec{s_i} \cdot \vec{s_j}$ $(\forall_{|\vec{s_i} - \vec{s_j}| = r})$ for calculations on the spatial correlation.
After a given number of runs the data is saved to several files:

XY-model.dat:	Contains average magnetization density with error, average energy density with error, average magnetic susceptibility with error, average specific heat capacity with error, average cluster size with error, integrated autocorrelation time of magnetization and energy.
XY-plot.dat:	Contains position and direction of every spin in the lattice. This save only occurs at T_{max}, $T = 1.05$ and T_{min}.
XY-vortices.dat:	Contains the positions of the vortex centres. These are saved at T_{max}, $T = 1.05$ and T_{min}.
XY-correlation.dat:	Contains $\langle \vec{s_i} \cdot \vec{s_j} \rangle$ for every possible distance r between two spins $\vec{s_i}$ and $\vec{s_j}$.

The errors for the primary quantities (magnetization density, energy density and cluster size) were obtained by simply calculating the standard error as shown in (6). Errors in respect to secondary quantities (magnetic susceptibility and specific heat capacity)

[8] Whose *C* code can be found in the appendix.

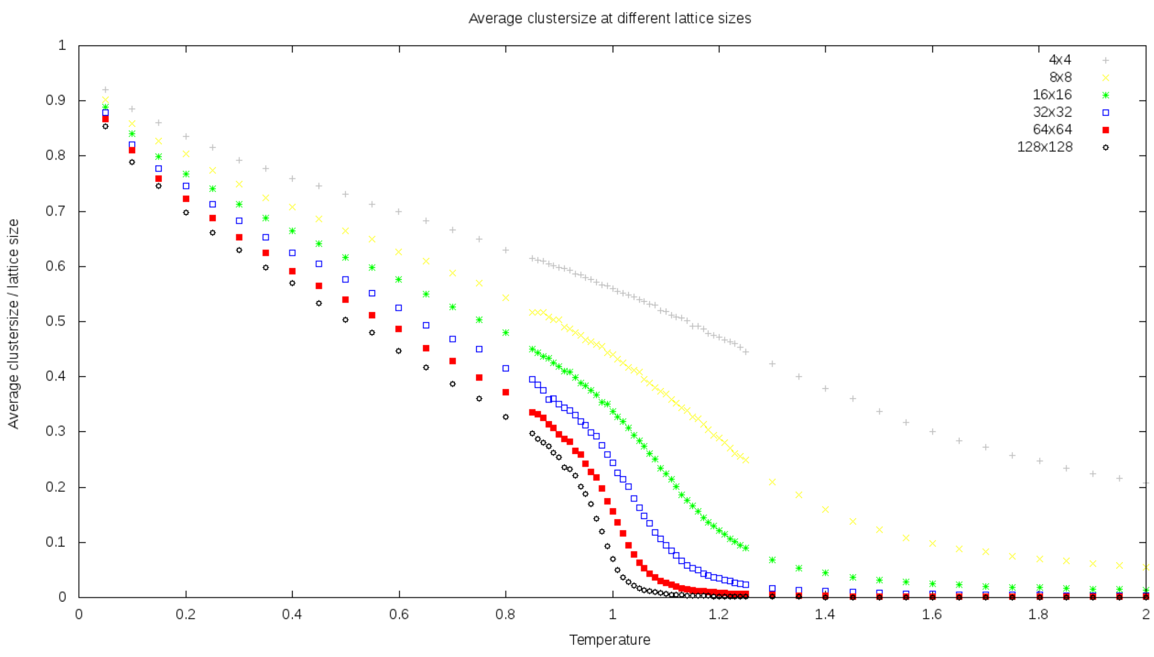

Figure 4: Average size of clusters (in relation to the lattice size) near T_c at different lattice sizes.

were done by applying the bootstrap method. Now the temperature is reduced by ΔT and the simulation repeats itself from equilibration to saving the data until the minimum temperature T_{min} is reached.

4.4.2 Parameters

The executable takes six arguments. The first one is the lattice length L. The lattice will then have a size of $L \times L$. Typically L is chosen between 2^2 and 2^7 in this thesis. The next two arguments regulate the number of cluster flips while equilibrating and the total number of independent points in the Markov chain (per temperature). Both are meant to be about 10000 but of course they can be higher for better precision.
The last three arguments concern the temperature: T_{min}, T_{max} and ΔT. They determine which interval of temperatures is covered and ΔT fixes the distance between two temperatures. In the vicinity of the critical temperature ($T \in [0.85, 1.25]$) ΔT is divided by five to obtain a smoother curve in the figures.
The external magnetic field that appears in (26) is switched off, i.e. $B = 0$, and the Boltzmann constant is set $k_B = 1$. For research on the ferromagnetic XY-model the coupling constant has to be positive[9], more precisely $J = 1$.

4.4.3 Observables

The simulation's behaviour is essentially influenced by the probability to add a certain spin to the cluster (as described in (28)). At low temperatures, this probability

[9] $J < 0$ for the anti-ferromagnetic model.

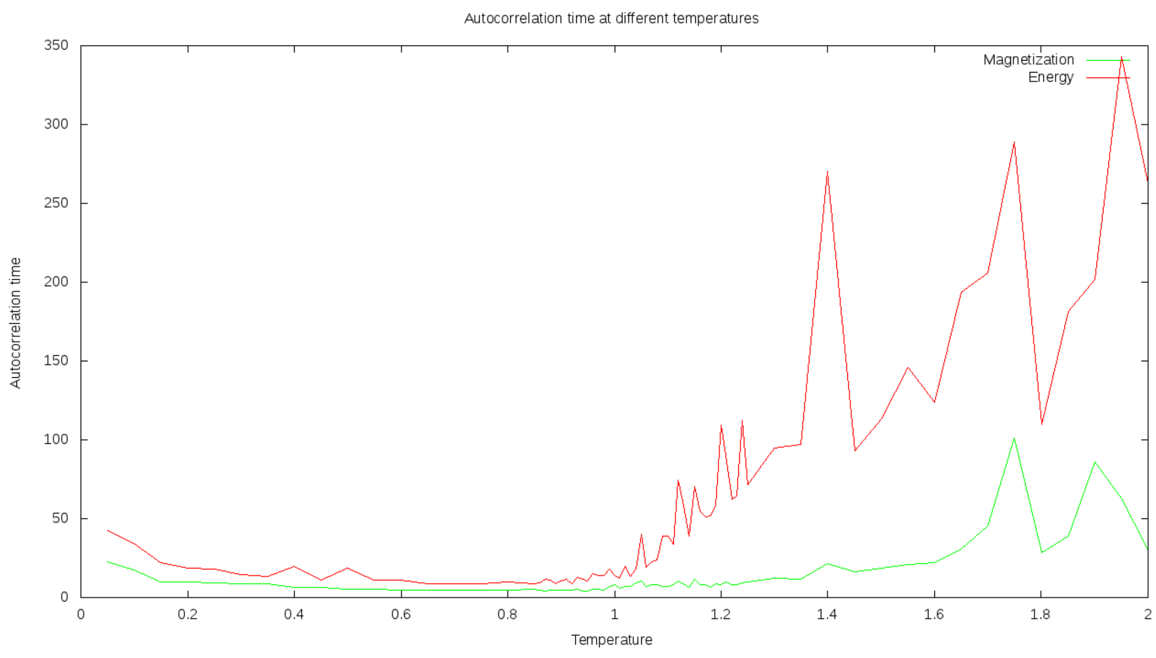

Figure 5: Integrated autocorrelation time for energy and magnetization near T_c at different lattice sizes.

approximates 1, so that almost every spin in one hemisphere is added to the cluster. Hence, cluster sizes nearly as large as the lattice size may be reached at temperatures near zero. Figure 4 shows that this is actually the case.
In contrast to that, $P(\vec{s_i}, \vec{s_j})$ approximates zero when temperatures above T_c are reached. Therefore, clusters will mostly be composed of a single spin and it takes more cluster flips to reach a new independent configuration. That is why the integrated autocorrelation time becomes larger when the temperature is increased (as pictured in figure 5). This discrepancy of the system's behaviour and the phase transition in between is examined in this chapter by viewing the performance of several observables. The data for the figures shown in this chapter were obtained from simulations with the parameters shown in table 1 and they are in agreement with those specified in [4].

L	$N_{equilibration}$	$N_{independent}$	T_{min}	T_{max}	ΔT
4	10000	50000	0.05	2	0.05
8	10000	25000	0.05	2	0.05
16	10000	50000	0.05	2	0.05
32	10000	10000	0.05	2	0.05
64	10000	10000	0.05	2	0.05
128	10000	20000	0.05	2	0.05

Table 1: Parameters chosen for various lattice sizes.

4.4.3.1 Magnetization

The magnetization per spin for any configuration μ is calculated as follows:

$$M_\mu = \left| \frac{1}{|\Lambda|} \sum_{i \in \Lambda} cos(\theta_i - \bar{\theta}) \right| \tag{38}$$

where $\bar{\theta}$ is the average spin angle.
Figure 6 shows that at low temperatures the magnetization density draws near 1. This behaviour can be explained by looking at the average cluster size which almost equals the lattice size. So nearly all spins are simultaneously flipped towards the same direction. The lattice's image is painted in figure 7.
In the area around the critical temperature T_c smaller clusters are formed and therefore the spins become more and more disordered until they are uniformly distributed over the whole interval $[0, 2\pi)$ as the average cluster size converges to 1.

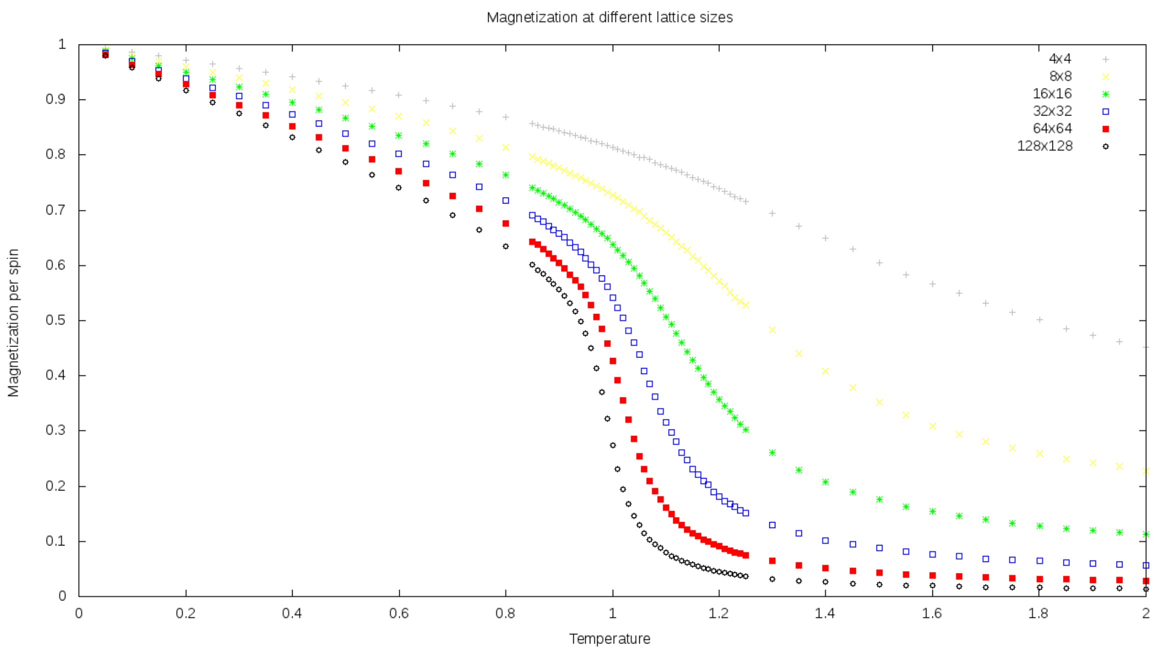

Figure 6: Magnetization density near T_c at different lattice sizes.

4.4.3.2 Magnetic susceptibility

The magnetic susceptibility is calculated as described in (11), omitting the factor $\frac{1}{V}$. This leads to figures 8 & 9. Towards the critical temperature the graph rises very rapidly. This behaviour can be traced back to the fact that the susceptibility essentially depends on the magnetization's variance which becomes very large around T_c due to cluster sizes. Those large fluctuations are exhibited in figure 10 where the magnetization is plotted at the first 10000 Monte-Carlo steps.

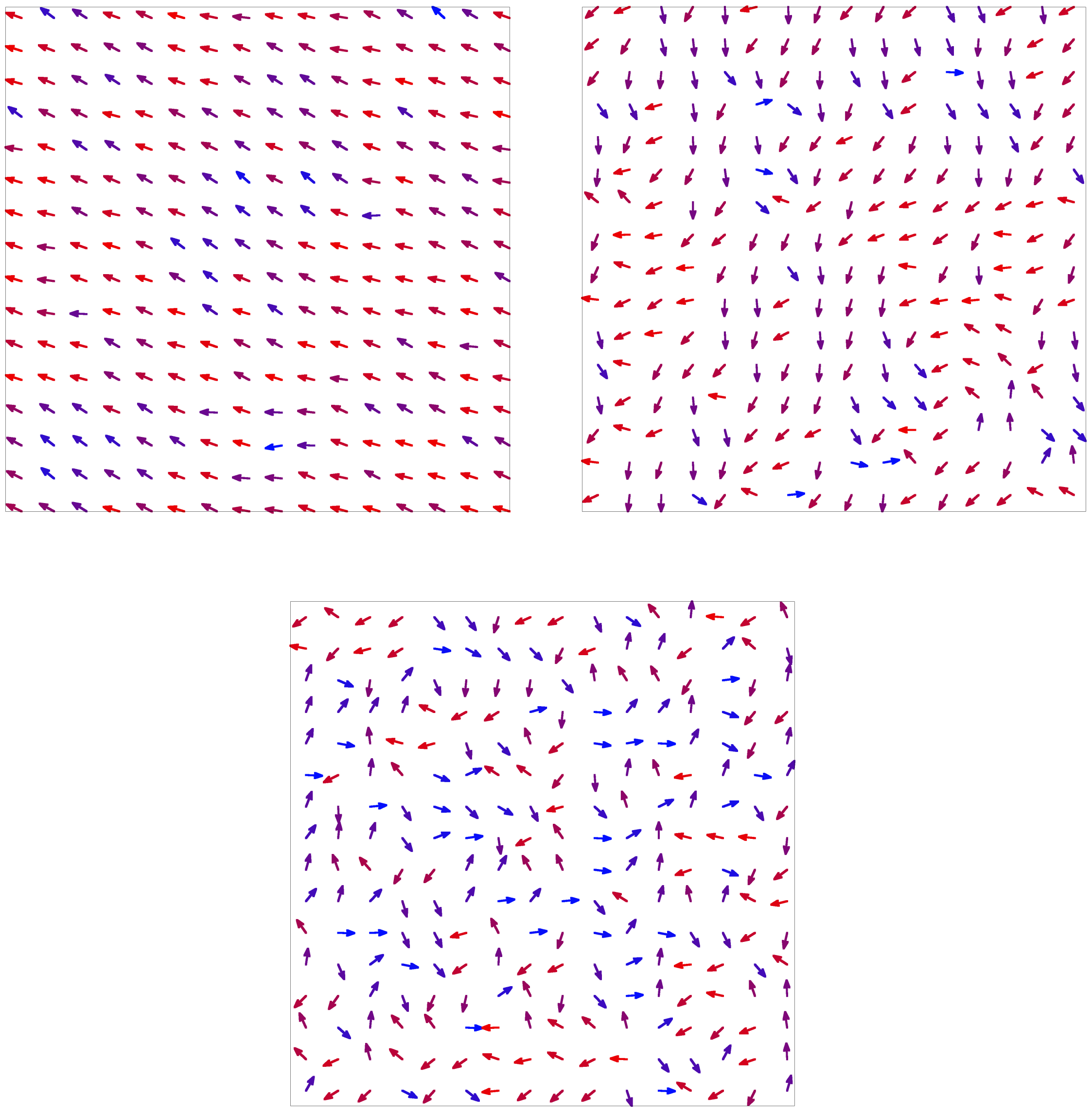

Figure 7: Spin configurations on a 16×16 lattice at $T = 0.05$ (left), $T = 1.05$ (right) and $T = 2$ (bottom).

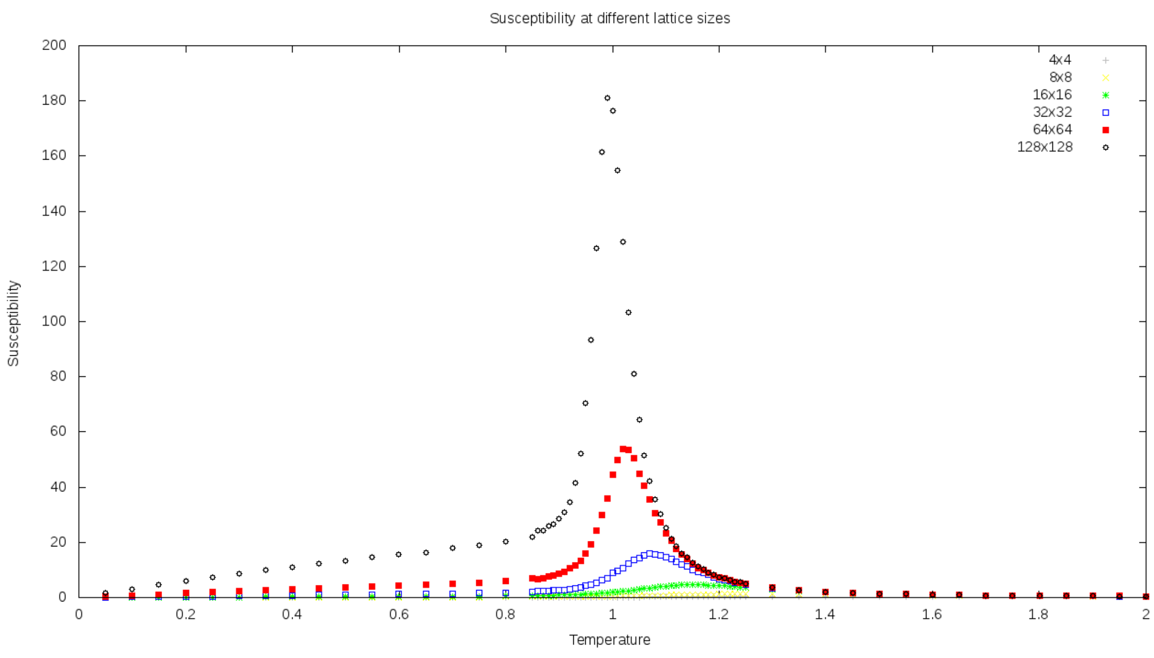

Figure 8: Magnetic susceptibility near T_c at different lattice sizes.

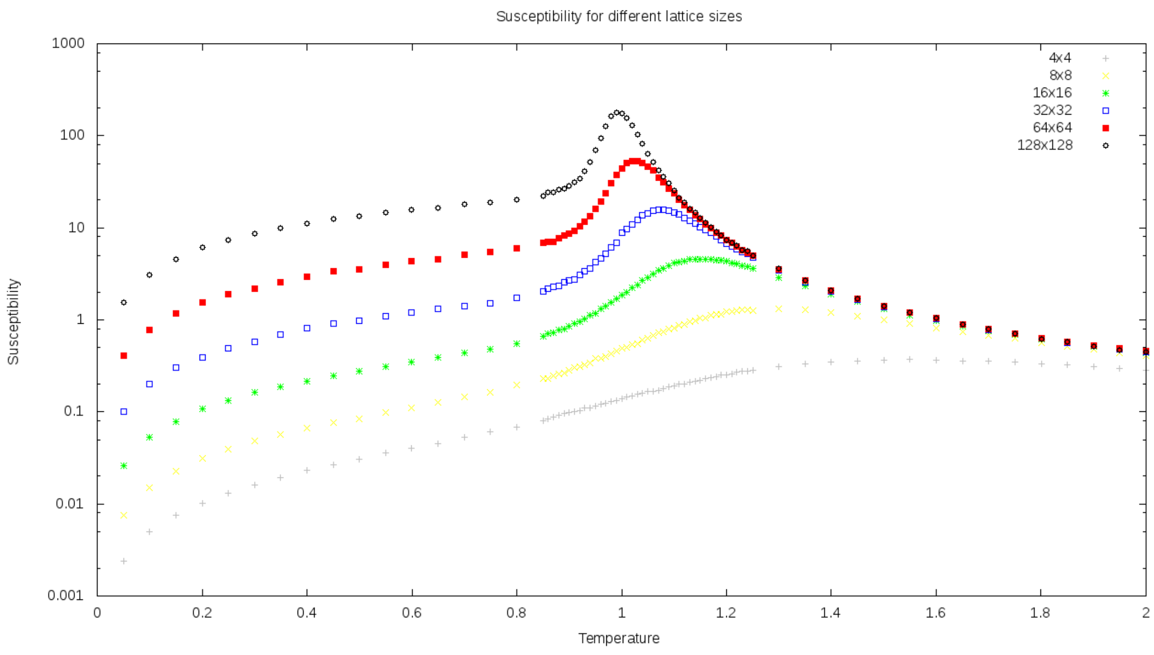

Figure 9: Magnetic susceptibility near T_c at different lattice sizes with logarithmic scale.

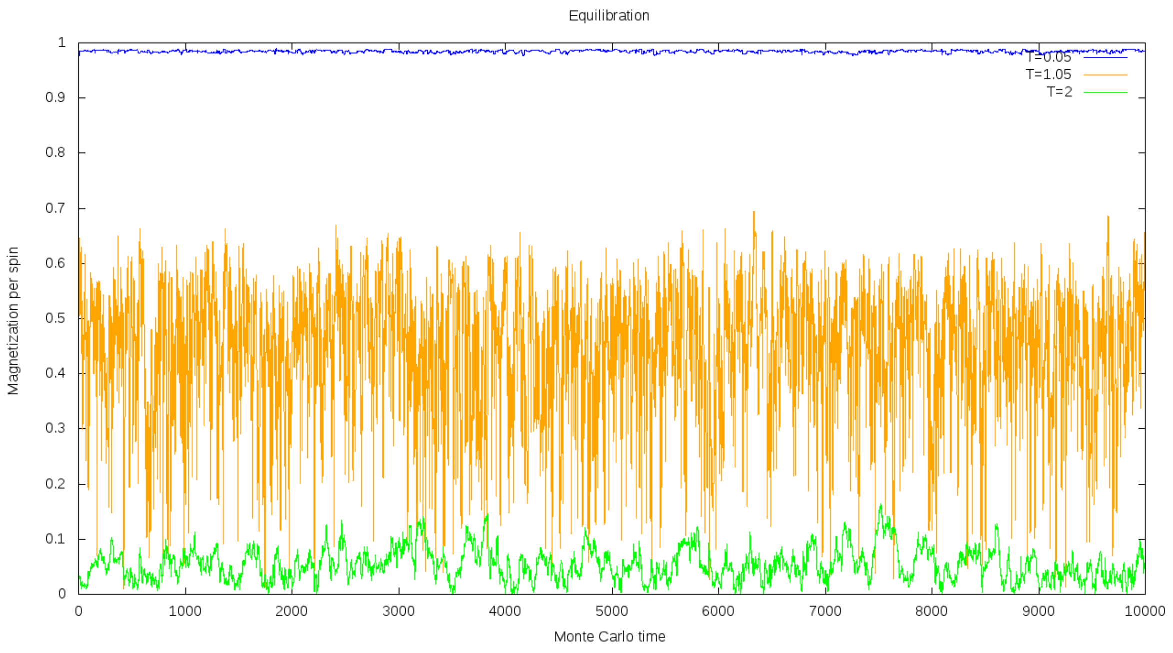

Figure 10: Magnetization density during equilibration at different temperatures (Monte Carlo time equals cluster flips).

4.4.3.3 Energy

The energy density is calculated by looking at the orientation of every spin pair towards each other. Parallel spin pairs cause an energy loss of $-2J$ whereas anti-parallel pairs cause an energy gain of $2J$. Since almost all spin pairs are parallel at low temperatures, the system's energy in figure 11 converges to -2. With increasing temperature the energy increases as the spins loose their common orientation.

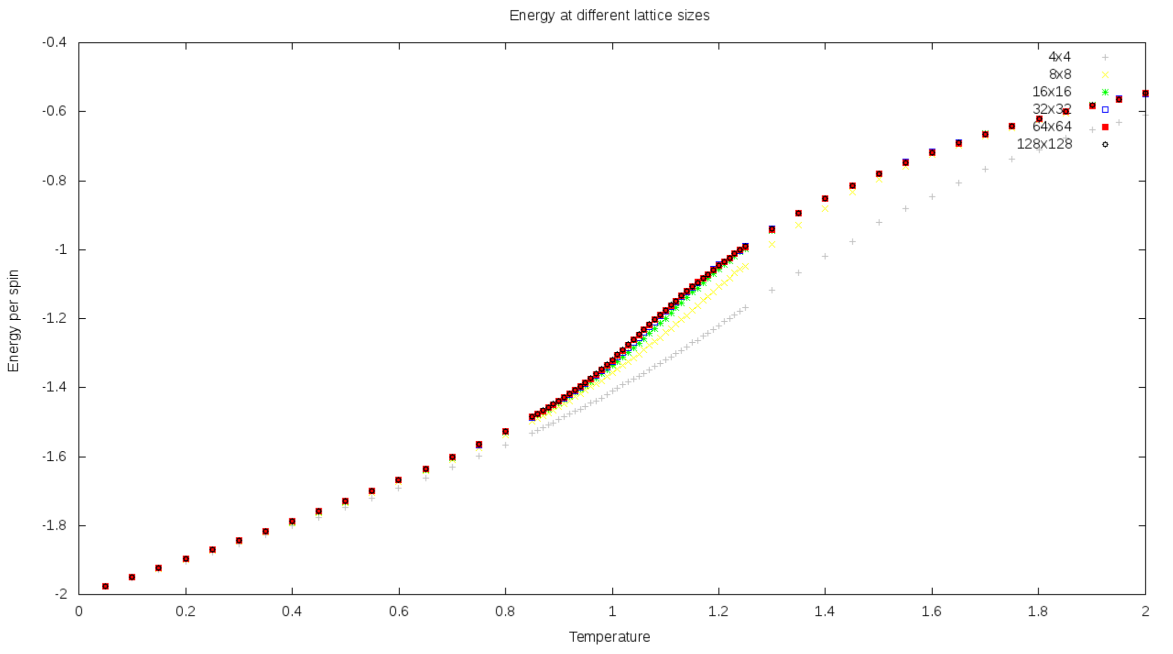

Figure 11: Energy density near T_c at different lattice sizes.

4.4.3.4 Specific heat capacity

The specific heat capacity is calculated as described in (14). Its values at different lattice sizes are nearly all the same (figure 14). The peak around T_c again (as already specified in 4.4.3.2) results from the system's critical behaviour when the clusters become smaller.

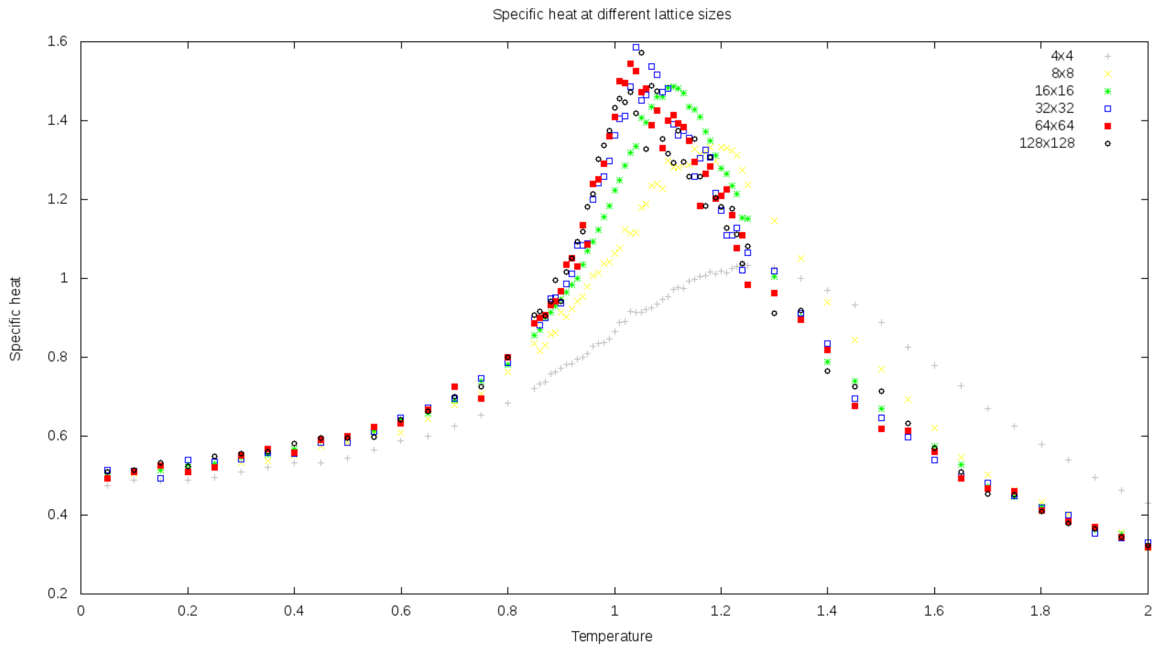

Figure 12: Specific heat near T_c at different lattice sizes.

4.4.4 Correlation length

As the cluster size is important for the system's behaviour one would like to have a measure for this size. The correlation length ξ can be seen as a measure that indicates how far in the lattice space one spin is correlated to another. In the simulation this length was derived from the spatial correlation function (cf. [5])

$$G(r) = \langle \vec{s_i} \cdot \vec{s_j} \rangle = \frac{1}{N_r} \sum_{|\vec{x_i} - \vec{x_j}| = r} cos(\theta_i - \theta_j) \tag{39}$$

where $\vec{x_i}$ is the position of spin $\vec{s_i}$ and N_r is the number of spins with distance r.
Furthermore, this function behaves like (cf. [6])

$$G(r) \sim exp\left(-\frac{r}{\xi}\right) \tag{40}$$

so that $G(r)$ can be calculated in the simulation and then fitted to the exponential law (figure 13).
In the end the correlation lengths can be plotted (see figure 14) and compared. The

plot indicates a *finite-size effect*: the larger the lattice the larger ξ can become, as the correlation length obviously cannot outrun the lattice size.

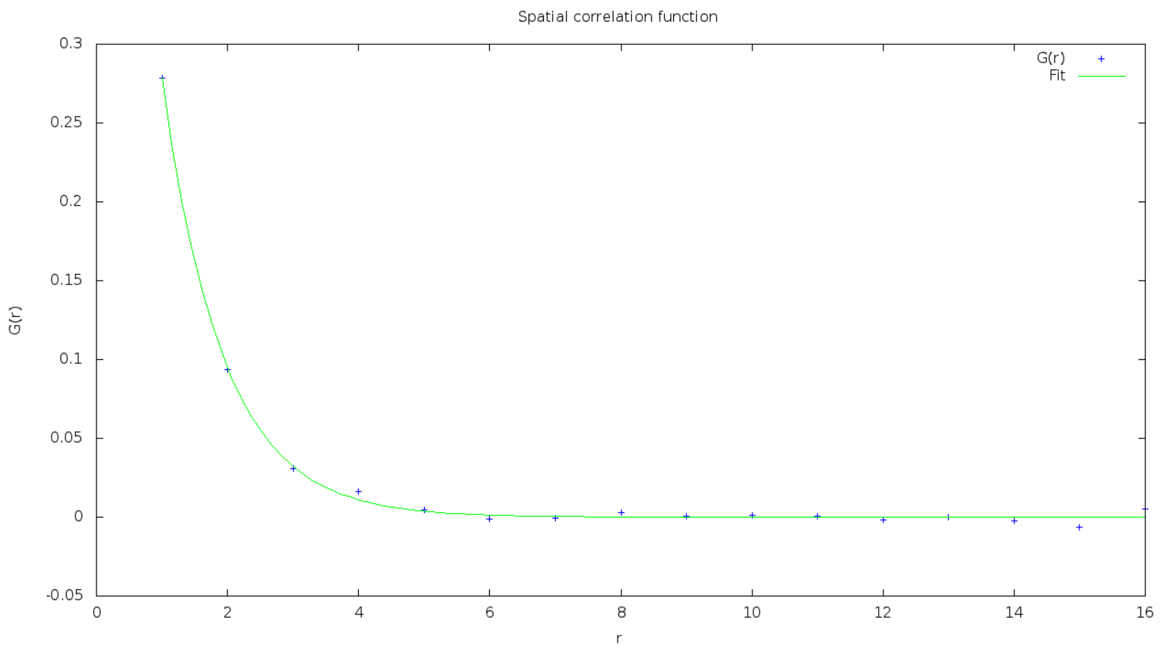

Figure 13: Plot of the spatial correlation function with fit.

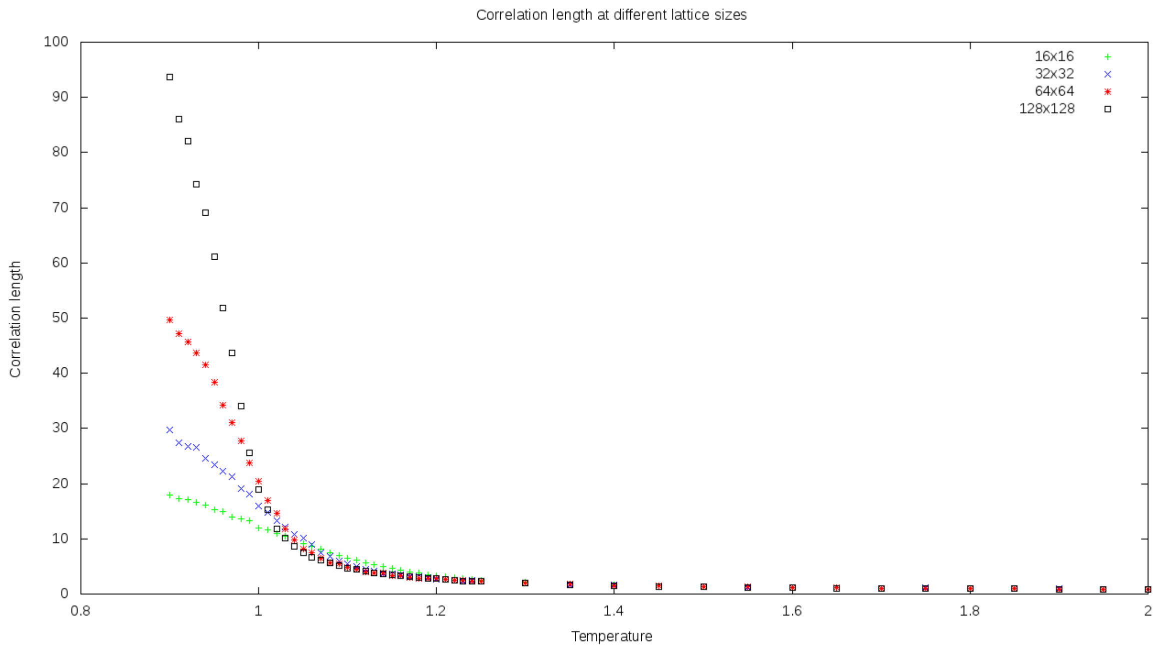

Figure 14: Correlation length at T_c and higher temperatures at different lattice sizes.

4.4.5 Vortices

At $T > T_c$ one observes the occurrence of vortices which destroy the system's long-range order. In order to determine these vortices, the vorticity q of a certain region is given by

$$\oint_C d\theta(\vec{r}) = 2\pi q \tag{41}$$

where C is the boundary of the region (c.f. [7]).
To identify the centres of such vortices it is helpful to only consider curves C that run around the square of four nearest neighbours in clockwise direction and sum up the angular changes. This was numerically done and visualised in figure 16. The vortices can be split in two groups: clockwise ($q = 1$) and counter-clockwise ($q = -1$) that both occur only at $T > T_c$ and in the same quantities (see figure 15).
The energy of a system with an isolated vortex and no external magnetic field can be approximated by

$$E \approx 2\pi J \; log\left(\frac{R}{\kappa}\right) \tag{42}$$

with the system's radius R and the lattice spacing κ.

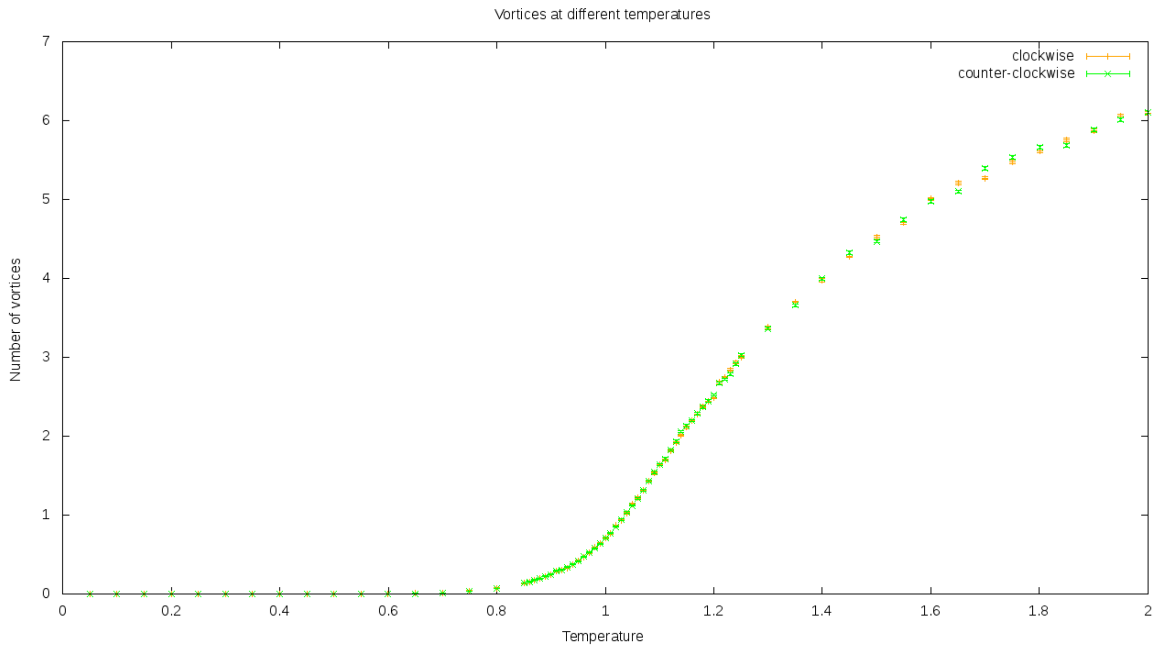

Figure 15: Number of vortices on a 64×64 lattice.

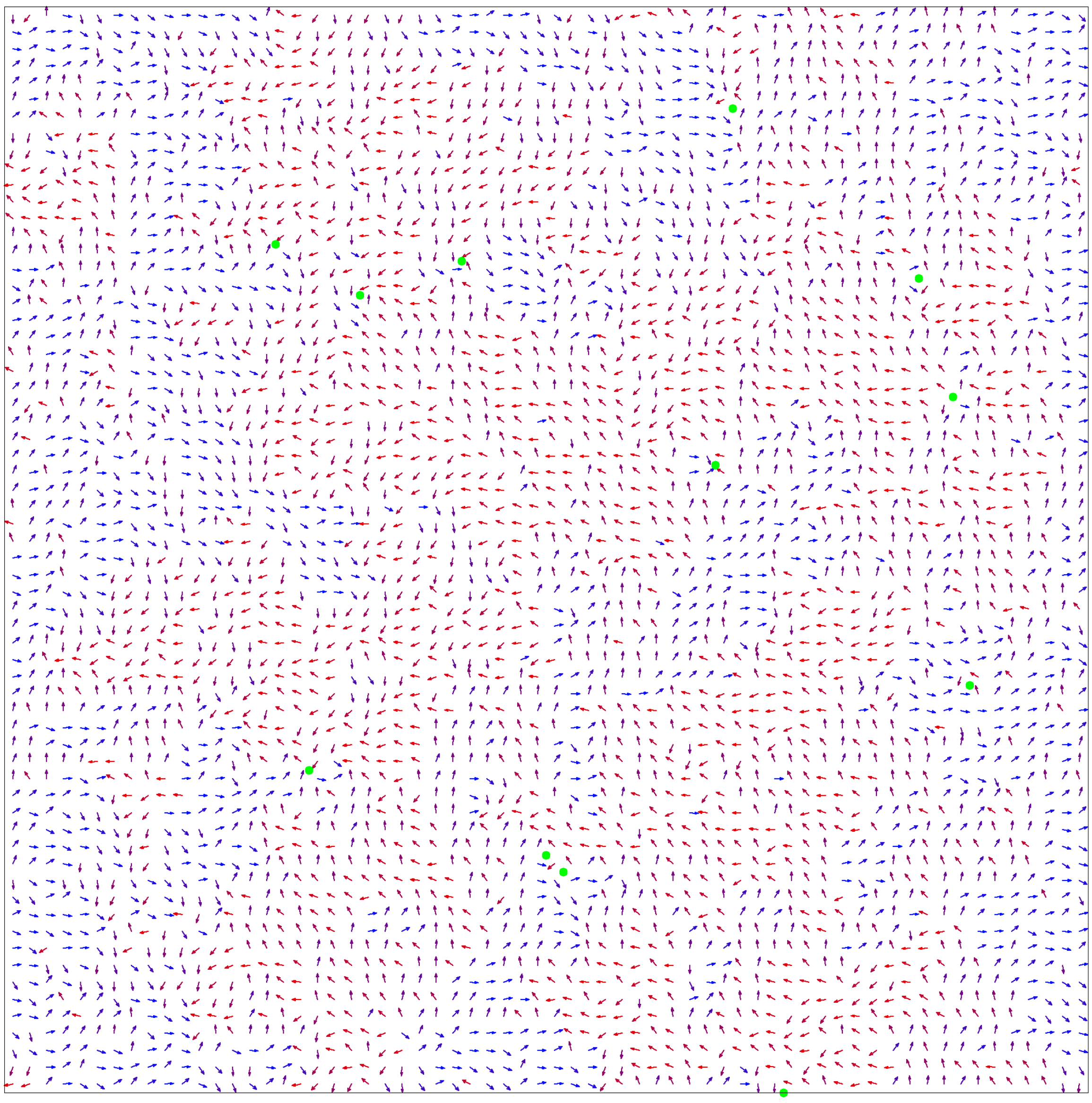

Figure 16: 64×64 lattice at $T = 1.05$ with vortex centres (marked as green dots).

5 Summary

This thesis briefly summarised the most important principles of thermodynamics and statistics of the XY-model. Technical terms were introduced which are essential for further examinations of the model. Chapter 3 dealt with techniques that help to set up a simulation and improve the simulation's results. The last part introduced the XY-model and the Wolff algorithm in detail. The simulation for the XY-model was demonstrated by describing the procedure and parameters that were used to obtain the results given in the last sections.
Every observable examined suggested a phase transition at a certain critical temperature. Several figures showed the differences of simulations on diverse lattice sizes from 4×4 to 128×128. The system's actual states were also depicted for certain temperatures. Further investigations were made on the correlation that indicated a massive increase of the correlation length towards the critical temperature. The last examination concentrated on vortices. It was confirmed that they occur only at temperatures above T_c and that clockwise and counter-clockwise vortices annihilate each other.

Bibliography

[1] D. Gamerman and H. F. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, chapter Markov Chains, pages 113–114. Chapman and Hall/CRC, 2006.

[2] W. Grimus. *Einführung in die Statistische Physik und Thermodynamik: Grundlagen und Anwendungen*, chapter 1.6.1 Wärmeaustausch, pages 23–24. Oldenbourg Verlag München, 2010.

[3] W. Grimus. *Einführung in die Statistische Physik und Thermodynamik: Grundlagen und Anwendungen*, chapter 4.1 Zustandssummen und thermodynamische Potentiale, pages 68–69. Oldenbourg Verlag München, 2010.

[4] R. Gupta and C. F. Baillie. Critical behavior of the two-dimensional xy model. *Physical Review Letters B*, 45(6):2883–2898, 1992.

[5] W. Janke. Monte carlo simulations of spin systems. http://www.physik.uni-leipzig.de/∼janke/Paper/spinmc.pdf, last visited 2.04.2015.

[6] L. Kadanoff. *Statistical Physics - Statics, Dynamics and Renormalization*, chapter 11.5 Spatial Correlations, pages 232–237. World Scientific, 2000.

[7] J. M. Kosterlitz. The critical properties of the two-dimensional xy model. *Journal of Physics C: Solid State Physics*, 7:1046–1060, 1974.

[8] J. M. Kosterlitz and D. J. Thouless. Ordering, metastability and phase transitions in two-dimensional systems. *Journal of Physics C: Solid State Physics*, 6:1181–1203, 1973.

[9] C. B. Lang. Data survival guide.
http://physik.uni-graz.at/∼cbl/teaching/dsg/node7.html, last visited 2.04.2015.

[10] K. Mingh and M. Xie. Bootstrap: A statistical method. http://www.stat.rutgers.edu/home/mxie/rcpapers/bootstrap.pdf, last visited 2.04.2015.

[11] W. Nolting. *Grundkurs Theoretische Physik 6 - Statistische Physik*, chapter 1.4.2 Freie Energie, pages 70–72. Springer, 2007.

[12] W. Nolting. *Grundkurs Theoretische Physik 6 - Statistische Physik*, chapter 4.2.1 Kritische Exponenten, pages 297–303. Springer, 2007.

[13] A. B. Owen. *Monte Carlo theory, methods and examples*, chapter 9.1 Basic importance sampling. 2013.

[14] F. Reif. *Grundlagen der Physikalischen Statistik und der Physik der Wärme*, chapter 2.3 Grundlegende Postulate, pages 64–70. Walter de Gruyter, 1976.

[15] F. Reif. *Grundlagen der Physikalischen Statistik und der Physik der Wärme*, chapter 1.3 Mittelwerte, page 15. Walter de Gruyter, 1976.

[16] D. V. Schroeder. *An Introduction to Thermal Physics*, chapter 6.2 Average Values, pages 230–231. Addison Wesley Longman, 2000.

[17] D. V. Schroeder. *An Introduction to Thermal Physics*, chapter 1.6 Heat Capacities, page 28. Addison Wesley Longman, 2000.

[18] F. Schwabl. *Statistische Mechanik*, chapter 6.1 Dichtematrix, Thermodynamik, pages 274–276. Springer, 2006.

[19] H. E. Stanley. Dependence of critical properties on dimensionality of spins. *Physical Review Letters*, 20:589–592, 1968.

[20] U. Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62(4):361–364, 1989.

[21] U. Wolff. Collective monte carlo updating in a high precision study of the x-y model. *Nuclear Physics B*, 322:759–774, 1989.

Appendix

```
/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////// THE 2D XY-MODEL ////////////////////////////////////////
////////////////////////////////////// (with Wolff algorithm) //////////////////////////////////////
//////////////////////////////////////// by Moritz Fink ////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <gsl/gsl_rng.h> //compile: "gcc -lgsl -lgslcblas -o XY -O3 XY.c"

#define J 1 //J=+1 (ferromagnetic), J=-1 (anti-ferromagnetic)
#define PI 3.14159265
#define PEAK 1.05
#define eps 0.005

/////////////////////////////////////////////////////////////////////////////////////////

//Neighbour function
int right_neighbour(int i, int L)
{
  if ((i+1)%L == 0) return i+1-L;
  else return i+1;
}

int left_neighbour(int i, int L)
{
  if ((i+1)%L == 1) return i+L-1;
  else return i-1;
}

int upper_neighbour(int i, int L)
{
  if (i < L) return i+L*(L-1);
  else return i-L;
}

int lower_neighbour(int i, int L)
{
  if (i+1 > L*(L-1)) return i-L*(L-1);
  else return i+L;
}

/////////////////////////////////////////////////////////////////////////////////////////

double angle(double position1, double position2)
//Returns the shortest angle between position1 and position2
{
  double path;

  if(position2 > position1) path = position2-position1;
  else path = position2-position1+1.0;

  if(path > .5) path = path-1.0;

  return path;
}

/////////////////////////////////////////////////////////////////////////////////////////

double translate(double position, double dx)
//Translates the spin counter clockwise by dx
{
  position += dx;
  if (position > 1) position -= 1.0;
  if (position < 0) position += 1.0;
  return position;
}

/////////////////////////////////////////////////////////////////////////////////////////

int flipCluster(int L, gsl_rng* rng, double* spin, double beta)
//Creates and flips a cluster and returns the cluster's size
{
  int i, j, k;
  int nr_new, i_max, clusterlength, start;
```

```c
  int inCluster, same_area;
  int nbr[4];
  int cluster[L*L];
  double normal_vec;
  double a;
  double p; //Probability of adding a spin to the cluster
  double s_in; //seed spin * normal vector
  double s_jn; //spin_j * normal vector
  const double twoPi = 2*PI;

  cluster[0] = (int)(gsl_rng_uniform(rng)*L*L); //Choose seed spin
  nr_new = 1; //Number of newly added spins
  clusterlength = 1;

  normal_vec = gsl_rng_uniform(rng); //Choose normal vector
  a = fabs(normal_vec-spin[cluster[0]]); //Angle between normal vector and seed spin

  if(a >= .25 && a < .75) //Normal vector and seed spin not in the same hemisphere
  {
    same_area = 0;
  }
  else same_area = 1; //Normal vector and seed spin are in the same hemisphere

  do
  {
        start = clusterlength-nr_new; //Smallest index of a newly added spin
        i_max = nr_new;
        nr_new=0;

    for(i=0; i<i_max; i++) //Look at every new spin
    {
      s_in = cos(twoPi * angle(spin[cluster[start+i]],normal_vec));

      nbr[0] = right_neighbour(cluster[start+i],L);
      nbr[1] = left_neighbour(cluster[start+i],L);
      nbr[2] = upper_neighbour(cluster[start+i],L);
      nbr[3] = lower_neighbour(cluster[start+i],L);

      for(j=0; j<4; j++) //Look at the four neighbours
      {
        inCluster = 0;

        for(k=0; k<clusterlength; k++) if(nbr[j]==cluster[k])
                //Is the neighbour already in the cluster?
        {
          inCluster = 1; //Neighbour already in cluster
          break;
        }

        if(!inCluster) //Neighbour not in cluster
        {
          a = fabs(normal_vec-spin[nbr[j]]);
          //Distance between normal vector and neighbour

          if((a < .25 || a >= .75) && same_area || a >= .25 && a < .75 && !same_area)
          // Neighbour is in hemisphere of normal vector and seed spin OR
          // neighbour is in the other hemisphere and hence in the one of the seed spin
          {
            s_jn = cos(twoPi * angle(spin[nbr[j]],normal_vec));
            p = 1 - exp(-2*beta*J*s_in*s_jn);

            if(gsl_rng_uniform(rng)<=p)
            {
              nr_new++;
              cluster[clusterlength] = nbr[j]; //Add neighbour to cluster
              clusterlength++;
            }
          }
        }
      }
    }
  } while(nr_new != 0); //Repeat as long as spins were added

  //Flip cluster
  for(i=0; i<clusterlength; i++)
  {
    a = angle(spin[cluster[i]],normal_vec);
    if(a >= 0) spin[cluster[i]] = translate(spin[cluster[i]],2*(a-.25)); //Flip spin
    else spin[cluster[i]] = translate(spin[cluster[i]],2*(.25+a)); //Flip spin
  }

  return clusterlength;
```

```c
}

////////////////////////////////////////////////////////////////////////////////////////////////////

double standard_deviation(double *y, int N)
//Returns the standard deviation
{
  int i;
  double x, x2;

  x = 0;
  x2 = 0;

  for(i=0; i<N; i++)
  {
    x += y[i];
    x2 += y[i]*y[i];
  }

  x /= N;
  x2 /= N;

  return sqrt(x2 - x*x);
}

////////////////////////////////////////////////////////////////////////////////////////////////////

double Magnetization_Density(double* spin, int L)
//Returns the absolute value of magnetization per spin
{
  int i;
  const double twoPi = 2*PI;
  double M, dx, dy, average_spin;

  dx = 0;
  dy = 0;

  for(i=0; i<L*L; i++)
  {
    dx += cos(twoPi*spin[i]);
    dy += sin(twoPi*spin[i]);
  }

  average_spin = atan2(dy,dx)/twoPi;

  M=0;
  for(i=0; i<L*L; i++) M += cos(twoPi*angle(spin[i],average_spin));

  return fabs(M/(L*L));
}

double Energy_Density(double* spin, int L)
//Returns the energy per spin
{
  int i;
  const double twoPi = 2*PI;
  double E;

  E=0;
  for (i=0; i<L*L; i++)
  {
    E -= J*cos(twoPi*angle(spin[i],spin[right_neighbour(i,L)]));
    E -= J*cos(twoPi*angle(spin[i],spin[upper_neighbour(i,L)]));
  }

  return E/(L*L);
}

////////////////////////////////////////////////////////////////////////////////////////////////////

int c_vortices(double* spin, int L, int* c_vortex_centers)
//Returns the amount of clockwise vortices and saves its positions
{
    int i, index, amount;
    double a;

    index = 0;
    amount = 0;

    for(i=0; i<L*L; i++)
    {
        a = angle(spin[i],spin[right_neighbour(i,L)]);
        if(a > 0.15 && a < 0.35)
        {
            a = angle(spin[right_neighbour(lower_neighbour(i,L),L)],spin[lower_neighbour(i,L)]);
            if(a > 0.15 && a < 0.35)
            {
                a = angle(spin[lower_neighbour(i,L)],spin[i]);
                if(a > 0.15 && a < 0.35)
                {
                    amount++;
                    c_vortex_centers[index++] = i;
                }
            }
        }
    }

    c_vortex_centers[index] = -1;

    return amount;
}

int cc_vortices(double* spin, int L, int* cc_vortex_centers)
//Returns the amount of counter-clockwise vortices and saves its positions
{
    int i, index, amount;
    double a;

    index = 0;
    amount = 0;

    for(i=0; i<L*L; i++)
    {
        a = angle(spin[i],spin[right_neighbour(i,L)]);
        if(a > -0.35 && a < -0.15)
        {
            a = angle(spin[right_neighbour(lower_neighbour(i,L),L)],spin[lower_neighbour(i,L)]);
            if(a > -0.35 && a < -0.15)
            {
                a = angle(spin[lower_neighbour(i,L)],spin[i]);
                if(a > -0.35 && a < -0.15)
                {
                    amount++;
                    cc_vortex_centers[index++] = i;
                }
            }
        }
    }

    cc_vortex_centers[index] = -1;

    return amount;
}

/////////////////////////////////////////////////////////////////////////////////////////////

double specific_heat(double *E, int N, double beta, double T)
//Returns the specific heat per spin
{
  double x;

  x = standard_deviation(E,N);

  return (beta/T) * x*x;
}

double susceptibility(double *M, int N, double beta)
//Returns the magnetic susceptibility per spin
{
  double x;

  x = standard_deviation(M,N);

  return beta * x*x;
}

/////////////////////////////////////////////////////////////////////////////////////////////

double autocorr_func(double* y, int t, int N)
{
  int i;
  double a,b,c;

  a = 0;
```

```c
  b = 0;
  c = 0;

  for(i=0; i<N-t; i++)
  {
    a += y[i]*y[i+t];
    b += y[i];
    c += y[i+t];
  }

  a /= N-t;
  b /= N-t;
  c /= N-t;

  return a-(b*c);
}

double autocorr_time(double* y, int N)
//Returns the integrated autocorrelation time
{
  int i;
  double C_0, zeta, ro;

  zeta = 0;
  C_0 = autocorr_func(y,0,N);

  for(i=1; i<N; i++)
  {
    ro = autocorr_func(y,i,N);
    if(ro > 0) zeta += ro;
    else break;
  }

  return zeta/C_0 + 0.5;
}

//////////////////////////////////////////////////////////////////////////////////////////////////

double average(double* y, int N)
//Returns the average
{
  int i;
  double average;

  average = 0;
  for(i=0; i<N; i++) average += y[i];
  average /= N;

  return average;
}

//////////////////////////////////////////////////////////////////////////////////////////////////

double bootstrap(double *y,int N, int M, gsl_rng *rng, double beta,double T, int length,int quantity)
//Apply bootstrap method on specific heat (quantity=0) or magnetic susceptibility (quantity=1)
{
  int i,j;
  double Q[M], S[N], average, stddev, var;

  for(i=0; i<M; i++)
  {
    for(j=0; j<N; j++) S[j] = y[(int)(gsl_rng_uniform(rng)*N)];
    if(quantity==0) Q[i] = specific_heat(&S[0],N,beta,T);
    if(quantity==1) Q[i] = susceptibility(&S[0],N,beta);
  }

  average=0;
  for(i=0; i<M; i++) average += Q[i];
  average /= M;

  stddev=0;
  for(i=0; i<M; i++)
  {
    var = Q[i] - average;
    stddev += var*var;
  }
  stddev /= M-1;
  stddev = sqrt(stddev);

  return stddev;
}

```

```c
///////////////////////////////////////////////////////////////////////////////////////////

void spatial_correlation(double *spin, int L, double *sisj, gsl_rng *rng)
//G(i,j) = <s_i*s_j>
{
  int i;
  int left, right, up, down;
  int seed;
  double twoPi = 2*PI;

  seed = (int)(gsl_rng_uniform(rng)*L*L); //Choose seed spin
  left=seed; right=seed; up=seed; down=seed;

  for(i=0; i<L/2; i++) //Look at all neighbours of seed spin
  {
    left = left_neighbour(left,L);
    right = right_neighbour(right,L);
    up = upper_neighbour(up,L);
    down = lower_neighbour(down,L);

    sisj[i] += cos(twoPi*angle(spin[seed],spin[left])) + cos(twoPi*angle(spin[seed],spin[right]));
    sisj[i] += cos(twoPi*angle(spin[seed],spin[up])) + cos(twoPi*angle(spin[seed],spin[down]));
    //sisj[i] += spin[seed] * (spin[left] + spin[right] + spin[up] + spin[down]);
  }
}

///////////////////////////////////////////////////////////////////////////////////////////

void status(double T, int current_step, int max_step)
//Prints the current status of the simulation
{
    time_t rawtime;
    int percent;

    time(&rawtime);
    printf("\n%s", ctime(&rawtime));
    if(max_step != 0)
    {
        percent = (int)(100*current_step/max_step);
        printf("T = %g\t Markov-Chain: %d/%d (%d%%)\n", T, current_step, max_step, percent);
    }
}

///////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////

int main(int argc, char **argv)
{
  if(argc < 7)
      {
          printf("Need six arguments: <L> <warmup_steps> <max_steps> <T_min> <T_max> <T_step>");
          return 1;
      }

  int i,j;
  int L=atoi(argv[1]); //Length of lattice
  int t; //Monte Carlo time
  int warmup_steps=atoi(argv[2]), max_steps=atoi(argv[3]); //MC time for equilibration and production
  int step_mag, step_en; //Number of independent data
  int spatial_count;
  int status_count;

  int cc_vortex_centers[L*L];
  int c_vortex_centers[L*L];

  double two_Pi = 2*PI;
  double T, T_min=atof(argv[4]), T_max=atof(argv[5]), T_step=atof(argv[6]); //Temperatures
  double beta; //beta=1/T
  double autocorr_time_mag, autocorr_time_en; //Autocorrelation times for magnetization and energy
  double tenPercent;

  double average_mag, error_mag, average_en, error_en;
  double magnetic_susceptibility, error_sus, spec_heat, error_heat;
  double average_size, error_size, average_clockwise, average_counter;
  double error_clockwise, error_counter;

  double Magnetization[(int)(fmax(max_steps,warmup_steps))];
  double Energy[(int)(fmax(max_steps,warmup_steps))];
  double clustersize[(int)(fmax(max_steps,warmup_steps))];

  double clockwise[(int)(fmax(max_steps,warmup_steps))];
  double counter_clockwise[(int)(fmax(max_steps,warmup_steps))];

  double sisj[L/2]; //For calculation on the spatial correlation

  double spin[L*L]; //Current spin configuration

  char filename[200];
  char equilibrationfilename[200];
  char plotfilename[200];
  char correlationfilename[200];
  char vortexfilename[200];

  gsl_rng *rng; //Pointer for random number generator

  FILE* file; //File for various quantities
  FILE* equilibrationfile; //File with data from equilibration
  FILE* plotfile; //File for plotting the lattice
  FILE* correlationfile; //File with correlation function
  FILE* vortexfile;

/////////////////////////////////////////////////////////////////////////////////////////////////

  //Create files (Files with same name will be deleted!)
  sprintf(filename,"XY-model(L=%i,warmup_steps=%i,max_steps=%i,T_min=%g,T_max=%g,T_step=%g).dat",
    L, warmup_steps, max_steps, T_min, T_max, T_step);
  file = fopen(filename,"w");
  fclose(file);

  sprintf(equilibrationfilename,
          "XY-equilibration(L=%i,warmup_steps=%i,max_steps=%i,T_min=%g,T_max=%g,T_step=%g).dat",
          L, warmup_steps, max_steps, T_min, T_max, T_step);
  equilibrationfile = fopen(equilibrationfilename,"w");
  fclose(equilibrationfile);

  sprintf(plotfilename,"XY-plot(L=%i,warmup_steps=%i,max_steps=%i,T_min=%g,T_max=%g,T_step=%g).dat",
    L, warmup_steps, max_steps, T_min, T_max, T_step);
  plotfile = fopen(plotfilename,"w");
  fclose(plotfile);

  sprintf(correlationfilename,
          "XY-correlation(L=%i,warmup_steps=%i,max_steps=%i,T_min=%g,T_max=%g,T_step=%g).dat",
          L, warmup_steps, max_steps, T_min, T_max, T_step);
  correlationfile = fopen(correlationfilename,"w");
  fclose(correlationfile);

  sprintf(vortexfilename,
          "XY-vortices(L=%i,warmup_steps=%i,max_steps=%i,T_min=%g,T_max=%g,T_step=%g).dat",
          L, warmup_steps, max_steps, T_min, T_max, T_step);
  vortexfile = fopen(vortexfilename,"w");
  fclose(vortexfile);

/////////////////////////////////////////////////////////////////////////////////////////////////

    //Configuration of GSL
  rng = gsl_rng_alloc(gsl_rng_mt19937); //initialize rng
  long seed = time(NULL); //set seed
  gsl_rng_set(rng,seed);

/////////////////////////////////////////////////////////////////////////////////////////////////

  //Initialize lattice
  for (i=0; i<L*L; i++) spin[i]=gsl_rng_uniform(rng); //set spin randomly between 0 and 1

/////////////////////////////////////////////////////////////////////////////////////////////////

    tenPercent = max_steps/10;

    printf("Start: ");
    status(T_max,0,0);

  for(T=T_max; T>=T_min-eps; T-=(fabs(T-PEAK)<.2 ? T_step/5 : T_step))
    //Generates more points near the critical temperature
  {
      status_count = 0;
    beta=1/T;

    for(i=0; i<L/2; i++) sisj[i]=0;

    // EQUILIBRATION
    printf("\n\nT = %g:\tEquilibration ",T);

    equilibrationfile = fopen(equilibrationfilename,"a");
```

```c

    for (t=0; t<warmup_steps; t++)
    {
      //Build and flip cluster
      clustersize[t] = flipCluster(L, rng, spin, beta);

      //Calculate and save magnetization and energy
      Magnetization[t] = Magnetization_Density(spin,L);
      Energy[t] = Energy_Density(spin,L);

      if(fabs(T-T_min)<eps || fabs(T-PEAK)<eps || fabs(T-T_max)<eps)
            {
                fprintf(equilibrationfile,"%g\t%d\t%g\t%g\n",T,t,Magnetization[t],Energy[t]);
            }
    }

    fclose(equilibrationfile);

    printf("done\n");

    /*------------------------------------------------------------------*/
    //Calculate autocorrelation times
    autocorr_time_mag = autocorr_time(Magnetization,warmup_steps);
    autocorr_time_en = autocorr_time(Energy,warmup_steps);

    printf("Autocorrelation time: %g\n", autocorr_time_mag);

    step_mag = 0;
    step_en = 0;
    /*------------------------------------------------------------------*/

    // PRODUKTIONSLAUF
    t=0;
    spatial_count=0;

    do
    {
      //Build and flip cluster
      clustersize[step_mag] = flipCluster(L, rng, spin, beta);

      //Calculate and save magnetization
      if(t%((int)(2*autocorr_time_mag)+1)==0 && t!=0) //Only save independent data
      {
          if(step_mag >= status_count*tenPercent)
                {
                    status(T,step_mag,max_steps);
                    status_count++;
                }

        Magnetization[step_mag] = Magnetization_Density(spin,L);
        counter_clockwise[step_mag] = (double)cc_vortices(spin,L,cc_vortex_centers);
        clockwise[step_mag] = (double)c_vortices(spin,L,c_vortex_centers);
        //vortices(spin,L,&clockwise,&counter_clockwise,vortex_centers);
        step_mag++;

        //Calculate correlation
        spatial_correlation(spin, L, sisj, rng);
        spatial_count++;
      }

      //Calculate and save energy
      if(t%((int)(2*autocorr_time_en)+1)==0 && t!=0) //Only save independent data
      {
        Energy[step_en] = Energy_Density(spin,L);
        step_en++;
      }

      t++;
    } while (step_mag < max_steps && step_en < max_steps);

    // EVALUATION

    printf("\nEvaluating results ");

    average_mag = average(Magnetization,step_mag);
    error_mag = standard_deviation(Magnetization,step_mag)/sqrt(step_mag);

    average_en = average(Energy,step_en);
    error_en = standard_deviation(Energy,step_en)/sqrt(step_en);

    magnetic_susceptibility = L*L*susceptibility(Magnetization, step_mag, beta);
    error_sus = L*L*bootstrap(Magnetization, step_mag, 20000, rng, beta, T, L, 1);

    spec_heat = L*L*specific_heat(Energy, step_en, beta, T);
    error_heat = L*L*bootstrap(Energy, step_en, 20000, rng, beta, T, L, 0);

    average_size = average(clustersize, step_mag)/L/L;
    error_size = standard_deviation(clustersize,step_mag)/sqrt(step_mag)/L/L;

    average_clockwise = average(clockwise,step_mag);
    error_clockwise = standard_deviation(clockwise,step_mag)/sqrt(step_mag);

    average_counter = average(counter_clockwise,step_mag);
    error_counter = standard_deviation(counter_clockwise,step_mag)/sqrt(step_mag);

    for(i=0; i<L/2; i++) sisj[i] /= 4*spatial_count; //Calculate <s_i*s_j>

    // SAVE
    file = fopen(filename,"a");
    fprintf(file,"%g\t%f\t%e\t%f\t%e\t%e\t%e\t%e\t%e\t%f\t%e\t%g\t%g\t%g\t%g\t%g\t%g\n",T,
      average_mag,error_mag,average_en,error_en,magnetic_susceptibility,error_sus,
      spec_heat,error_heat,average_size,error_size,autocorr_time_mag,autocorr_time_en,
      average_clockwise,error_clockwise,average_counter,error_counter);
    fclose(file);

    if(fabs(T-T_min)<eps || fabs(T-PEAK)<eps || fabs(T-T_max)<eps)
    {
      plotfile = fopen(plotfilename,"a");
      fprintf(plotfile,"\nT=%g\n",T);
      //Save vectors by printing (x,y) and (dx,dy)
      for(i=0; i<L; i++) for(j=0; j<L; j++) fprintf(plotfile,"%d\t%d\t%g\t%g\t%g\n",
                      j,L-1-i,0.5*cos(two_Pi*spin[i*L+j]),0.5*sin(two_Pi*spin[i*L+j]),spin[i*L+j]);
      fclose(plotfile);

      vortexfile = fopen(vortexfilename,"a");
      fprintf(vortexfile,"\nT=%g\n",T);
      for(i=0; i<L*L; i++)
            {
                if(c_vortex_centers[i] != -1) fprintf(vortexfile,"%g\t%g\n",
                            c_vortex_centers[i]%L + 0.5, L - (int)(c_vortex_centers[i]/L) - 1.5);
                else break;
            }
            for(i=0; i<L*L; i++)
            {
                if(cc_vortex_centers[i] != -1) fprintf(vortexfile,"%g\t%g\n",
                            cc_vortex_centers[i]%L + 0.5, L - (int)(cc_vortex_centers[i]/L) - 1.5);
                else break;
            }
      fclose(vortexfile);
    }

    correlationfile = fopen(correlationfilename,"a");
    fprintf(correlationfile,"\n%g:\n",T);
    for(i=0; i<L/2; i++) fprintf(correlationfile,"%d\t%g\n",i+1,sisj[i]);
    fclose(correlationfile);

    printf("done\n\n");
  }

  return 0;
}
```

Listing 1: XY.c

T	M	SE_M	E	SE_E	χ	SE_χ	C_V	SE_{C_V}
2	0.028078	0.000144795	-0.546954	0.000191675	0.000104829	1.567e-06	7.78872e-05	1.17043e-06
1.95	0.029385	0.00015467	-0.564251	0.000407953	0.000122682	1.86989e-06	8.42962e-05	2.65743e-06
1.9	0.029801	0.000155118	-0.582103	0.000217946	0.000126639	1.89911e-06	9.0159e-05	1.52039e-06
1.85	0.030769	0.000160182	-0.600521	0.000281058	0.000138693	2.07889e-06	9.38001e-05	2.09641e-06
1.8	0.031853	0.000167289	-0.620853	0.000185908	0.000155475	2.4773e-06	0.000100304	1.46386e-06
1.75	0.033045	0.000170703	-0.643001	0.000282314	0.000166512	2.46397e-06	0.000112401	2.44928e-06
1.7	0.034819	0.000181126	-0.666228	0.000308001	0.000192979	2.91307e-06	0.000114231	2.75274e-06
1.65	0.036186	0.000191847	-0.691103	0.000307082	0.000223062	3.50531e-06	0.000120156	2.74904e-06
1.6	0.038243	0.000201058	-0.717699	0.000187167	0.000248105	3.9386e-06	0.000136843	1.92467e-06
1.55	0.040667	0.00021247	-0.747502	0.000383649	0.000291248	4.36151e-06	0.000149729	4.26589e-06
1.5	0.042984	0.000226231	-0.779253	0.000529285	0.000341203	5.18383e-06	0.000150904	5.81295e-06
1.45	0.047194	0.000247598	-0.814276	0.000659431	0.000422792	6.15639e-06	0.000165046	7.91255e-06
1.4	0.051432	0.000267894	-0.852946	0.000984955	0.000512621	7.70723e-06	0.000199967	1.4107e-05
1.35	0.057045	0.000297416	-0.894259	0.000434321	0.000655233	9.33564e-06	0.000218495	6.86582e-06
1.3	0.064379	0.000334472	-0.941034	0.00071755	0.000860549	1.2499e-05	0.000234894	1.16552e-05
1.25	0.075202	0.000387251	-0.991559	0.000582119	0.0011997	1.76839e-05	0.000240077	9.74109e-06
1.24	0.077513	0.000400259	-1.00003	0.000652604	0.00129199	1.88869e-05	0.000270614	1.18043e-05
1.23	0.080505	0.000415788	-1.01181	0.00110145	0.00140552	2.04886e-05	0.000263025	1.94419e-05
1.22	0.083538	0.000432533	-1.0236	0.000603976	0.00153349	2.31852e-05	0.000283566	1.15227e-05
1.21	0.086679	0.000449472	-1.03546	0.000879217	0.00166963	2.39257e-05	0.000299367	1.69389e-05
1.2	0.091019	0.000464353	-1.04649	0.000755157	0.00179686	2.63753e-05	0.000295428	1.57269e-05
1.19	0.094689	0.000486171	-1.05964	0.000734789	0.00198624	2.76173e-05	0.000293577	1.44403e-05
1.18	0.099087	0.000508491	-1.0721	0.00108142	0.00219121	3.19011e-05	0.00031328	2.53396e-05
1.17	0.10305	0.0005317	-1.08359	0.00102726	0.00241628	3.47006e-05	0.000309125	2.10516e-05
1.16	0.108971	0.000552636	-1.09447	0.000790034	0.00263282	3.69304e-05	0.000288977	1.59073e-05
1.15	0.11488	0.000586115	-1.10819	0.00101917	0.00298722	4.23459e-05	0.000316518	2.23552e-05
1.14	0.121457	0.000625457	-1.12065	0.000875988	0.00343155	4.85077e-05	0.000329474	1.9103e-05
1.13	0.1301	0.00065588	-1.13347	0.00119954	0.00380689	5.20133e-05	0.000338058	2.53553e-05
1.12	0.137836	0.000693633	-1.14884	0.000905827	0.00429578	5.86943e-05	0.000340141	2.13159e-05
1.11	0.148868	0.000746004	-1.16167	0.000515581	0.00501371	6.62855e-05	0.000345413	1.15353e-05
1.1	0.160304	0.000788924	-1.1751	0.000545445	0.0056582	7.41077e-05	0.000341768	1.25069e-05
1.09	0.176507	0.000853722	-1.19032	0.000549436	0.00668662	8.70806e-05	0.000324976	1.29828e-05
1.08	0.191104	0.000898026	-1.20271	0.000693132	0.00746714	9.30563e-05	0.00034805	1.6796e-05
1.07	0.210082	0.000963465	-1.21776	0.000586494	0.00867537	0.000106446	0.000339198	1.40065e-05
1.06	0.230455	0.00102454	-1.23287	0.000408999	0.00990258	0.000114533	0.000361626	1.08159e-05
1.05	0.254755	0.00107224	-1.24719	0.000461476	0.0109495	0.000125804	0.00035928	1.15614e-05
1.04	0.286184	0.00113312	-1.26138	0.000460598	0.0123458	0.000141037	0.000372675	1.1984e-05
1.03	0.320806	0.00115957	-1.27743	0.00039704	0.0130545	0.000156788	0.000376976	1.01444e-05
1.02	0.355729	0.00115718	-1.29208	0.000344189	0.0131281	0.000171341	0.000365168	9.2189e-06
1.01	0.392028	0.00110867	-1.3071	0.000352839	0.0121698	0.000184129	0.000366127	8.92051e-06
1	0.426136	0.00104247	-1.32129	0.000325243	0.0108675	0.000189348	0.000344323	8.53412e-06
0.99	0.458979	0.000929283	-1.33468	0.000358098	0.0087229	0.000169742	0.000332198	9.27186e-06
0.98	0.485431	0.000846849	-1.3491	0.000309717	0.00731789	0.000162004	0.00031532	7.89143e-06
0.97	0.506984	0.0007563	-1.36097	0.000231612	0.0058968	0.000134256	0.000305421	5.91941e-06
0.96	0.528318	0.000673812	-1.37411	0.000304943	0.0047294	0.000121774	0.000302702	7.81801e-06
0.95	0.546192	0.000609207	-1.38621	0.000275892	0.00390667	9.64784e-05	0.000264995	6.72916e-06
0.94	0.561047	0.000554572	-1.39776	0.000252191	0.00327181	8.68528e-05	0.000276829	6.35032e-06
0.93	0.573556	0.000511143	-1.40814	0.00022373	0.00280933	6.31864e-05	0.000251578	5.27996e-06
0.92	0.583363	0.000489009	-1.41834	0.000217742	0.00259924	5.80088e-05	0.00025672	5.39028e-06
0.91	0.594864	0.000454083	-1.42846	0.000224049	0.00226584	4.57308e-05	0.000252535	5.5122e-06
0.9	0.604269	0.000439411	-1.43886	0.000216698	0.00214535	4.39179e-05	0.000236182	5.33994e-06
0.89	0.613307	0.000417167	-1.44851	0.000228946	0.00195538	3.92463e-05	0.000230152	5.50584e-06
0.88	0.621289	0.000409352	-1.45828	0.000196065	0.0019042	3.6463e-05	0.000227502	4.66975e-06
0.87	0.629604	0.00038499	-1.46766	0.000197671	0.00170365	2.89543e-05	0.000221206	4.73308e-06
0.86	0.638104	0.000371221	-1.47615	0.000217048	0.00160239	2.77023e-05	0.000219625	5.56262e-06
0.85	0.643676	0.000376294	-1.48485	0.000176688	0.00166585	2.84432e-05	0.000216046	4.46217e-06
0.8	0.675554	0.000338917	-1.52693	0.000178777	0.00143581	3.35056e-05	0.000195413	4.37972e-06
0.75	0.70211	0.000310312	-1.56519	0.000132302	0.00128392	2.18523e-05	0.000169716	3.26877e-06
0.7	0.726247	0.000291992	-1.60121	0.000141247	0.00121799	2.22132e-05	0.000176991	3.72783e-06
0.65	0.749446	0.000269804	-1.63559	0.000123088	0.00111991	1.98427e-05	0.000162982	3.3607e-06
0.6	0.771165	0.000249547	-1.66774	0.000105358	0.0010379	1.84098e-05	0.000154172	3.10772e-06
0.55	0.792612	0.000231063	-1.69897	0.00010639	0.000970726	1.7305e-05	0.00015199	3.35475e-06
0.5	0.812736	0.000207008	-1.72962	9.0473e-05	0.000857045	1.47815e-05	0.000146452	3.0128e-06
0.45	0.832722	0.00019332	-1.75913	6.88394e-05	0.000830501	1.52281e-05	0.000143991	2.55335e-06
0.4	0.852418	0.000168875	-1.78769	6.90965e-05	0.000712972	1.27441e-05	0.000136396	2.8714e-06
0.35	0.87128	0.000150481	-1.81591	6.79328e-05	0.000646986	1.17091e-05	0.000138785	3.31028e-06
0.3	0.890576	0.000128027	-1.84351	3.92961e-05	0.000546367	9.53668e-06	0.000134807	2.17573e-06
0.25	0.909355	0.000107949	-1.87064	3.38814e-05	0.000466116	8.22987e-06	0.000127156	2.16811e-06
0.2	0.927794	8.65235e-05	-1.89723	3.64433e-05	0.000374316	6.77954e-06	0.00012451	2.77239e-06
0.15	0.946082	6.49724e-05	-1.92351	2.65722e-05	0.000281428	4.82671e-06	0.000128351	2.804e-06
0.1	0.964179	4.73859e-05	-1.94937	1.11612e-05	0.000191871	3.63475e-06	0.000124573	1.76218e-06
0.05	0.982161	2.20048e-05	-1.97484	8.60863e-06	9.6842e-05	1.76205e-06	0.0001205	2.59826e-06

Table 2: Results of the simulation on a 64×64 lattice in detail.

Affirmation

Eidesstattliche Erklärung

Ich habe die Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Außerdem bestätige ich hiermit, dass die vorgelegten Druckexemplare und die vorgelegte elektronische Version der Arbeit identisch sind, dass ich über wissenschaftlich korrektes Arbeiten und Zitieren aufgeklärt wurde und dass ich von den in § 26/27 Abs. 5 vorgesehenen Rechtsfolgen Kenntnis habe.

$Unterschrift:$ $Ort, Datum:$