Monte-Carlo Simulation Studies on the Superspin Structure of 3D Nanoparticle Supercrystals

Mauricio Cattaneo

Masterarbeit in Physik vorgelegt der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen im Juli 2018 angefertigt am Forschungszentrum Jülich GmbH, Peter Grünberg Institut bei

Prof. Dr. David DiVincenzo
Prof. Dr. Thomas Brückel
Acknowledgements

I would like to give my sincere thanks to Prof. Dr. Thomas Brückel and PD Dr. Oleg Petracic for providing me the opportunity to work on this thesis in the institute JCNS-2/PGI-4 at Forschungszentrum Jülich GmbH.

I want to thank Prof. Dr. David DiVincenzo who agreed to be the first referee of my thesis. I would like to thank him for his interest in my research and his concise and valuable input.

I owe special thanks to Oleg Petracic, Michael Smik and Xiao Sun for being my supervisors. This thesis would not have been done without their encouragement, guidance and patient support from the beginning till the end.

I found great motivation from discussions with the entire staff at JCNS, special mention to the JCNS Nanoparticle Club. The exchange with excellent researchers and different scientific background than my own has been very inspiring and has shaped this thesis.

Finally, I must offer my heartfelt thanks to my family and my friends who have supported me during this project. I would like to thank them for their support, love and understanding.
Abstract

Nanoparticle (NP) supercrystals constitute a fascinating novel type of material with tunable magnetic, electronic and optical properties [3, 8, 12]. By choosing different NP materials, e.g. ferromagnetic or antiferromagnetic, a variety of magnetic and eventually multifunctional properties might be achieved. Hereby, one major challenge is the deliberate control of the supercrystal structure and of the resulting physical properties. In simulations we are able to model the collective magnetic ground states from microscopic assumptions [3–5]. This thesis aims to develop a proper simulation methodology to deal with lattices of interacting magnetic NP moments. Such a study is a crucial step towards predicting magnetic ground states and energy landscapes as function of the supercrystal lattice type and as function of the individual NP properties. Immediate goals include the study of the influence of dipole-dipole interactions on superparamagnetism and the spin structure of supercrystals at low temperatures.
## Contents

1 Introduction 1

2 Theory of Magnetism 2
   2.1 Origin of Magnetism in Solid State Matter 2
       2.1.1 Bohr-van Leeuwen theorem 2
   2.2 Magnetic Interactions 4
       2.2.1 Dipole-Dipole Interaction 5
       2.2.2 Exchange Interaction 5
       2.2.3 Anisotropic Exchange or Dzyaloshinky-Moriya Interaction 6
   2.3 Types of Magnetic Behavior 7
       2.3.1 Diamagnetism 7
       2.3.2 Paramagnetism 8
       2.3.3 Collective Magnetism 8
           2.3.3.1 Ferromagnetism 9
           2.3.3.2 Ferrimagnetism 9
           2.3.3.3 Antiferromagnetism 9
       2.3.4 Geometrically Frustrated Systems and Spin Glasses 10

3 Methods 12
   3.1 Why Do Simulations in Physics 12
   3.2 Categories of Monte-Carlo Simulations 12
       3.2.1 Monte Carlo in Statistical Physics 13
       3.2.2 Markov Chain methods 13
   3.3 The Metropolis Algorithm 15
   3.4 Quality of Pseudo-random Numbers Picked on a 2-Sphere 18
       3.4.1 Uniformly Distributed Unit Vectors on a ($d-1$)-Sphere 20
       3.4.2 Examples of Vector Distribution 21
       3.4.3 MC simulations and Ergodicity 24

4 Ferromagnetic Nanoparticles and Supercrystals in the Non-interacting Limit 26
   4.1 Theory I 26
       4.1.1 Stoner-Wohlfarth Model 26
       4.1.2 Isotropic Paramagnet 27
       4.1.3 Potential Landscape with Non-vanishing Anisotropy 29
       4.1.4 ZFC-FC curves 31
       4.1.5 ac-Susceptibility 32
   4.2 Simulation results 34
       4.2.1 Isotropic Superparamagnetism 34
4.2.2 ZFC-FC Curves ................................................. 35
   4.2.2.1 High-Temperature Behavior ......................... 35
4.2.3 Influences on the Blocking Temperature ..................... 36
   4.2.3.1 Particle Number ..................................... 36
   4.2.3.2 Test Vector ........................................ 37
   4.2.3.3 Number of Monte-Carlo Steps per Measurement ....... 38
   4.2.3.4 Applied Magnetic Field ................................. 39
4.2.4 Hysteresis Plots ........................................... 40
4.2.5 ac-Susceptibility and Cole-Cole Plot .......................... 41
4.3 Summary and Discussion of Results ................................ 43

5 Ferromagnetic Nanoparticles with Dipole-Dipole Interaction 44
   5.1 Theory II .................................................. 44
   5.1.1 Computational Aspects .................................. 44
   5.1.2 Mermin-Wagner Theorem ................................ 44
   5.1.3 Antiferromagnetic Part of the Dipole-Dipole Interaction .... 45
   5.1.4 Finite and Infinite Systems ................................. 46
   5.1.5 Possible Approximations .................................. 47
   5.1.6 Onsager Reaction Field Method ............................ 47
      5.1.6.1 Motivation and Model ................................ 48
      5.1.6.2 Solution of the Magnetostatic Problem ............. 49
      5.1.6.3 Modelling parameters ................................ 54
      5.1.6.4 Cut-off Radius ...................................... 55
      5.1.6.5 Determination of the relative permeability ........ 56
      5.1.6.6 Consistency with non-interacting limit ............. 57
   5.2 Simulation Results .......................................... 59
      5.2.1 Groundstates in Periodic, Dipolar 3D Systems ........ 59
         5.2.1.1 Results in the Limit of Vanishing Magnetocrystalline
                   Anisotropy ........................................ 59
         5.2.1.2 Series of 'Groundstates' Depending on Different Pa-
                   rameters in the Onsager Approximation ............ 61
         5.2.1.3 Reconsidering Non-Vanishing Anisotropy ........... 68

6 Summary and Outlook ............................................. 70

A Detailed Calculations ........................................... 71
   A.1 Magnetostatic Derivation of the Onsager Reaction Field .... 71
      A.1.1 Laplace Equation in Azimuthal Symmetry ............... 71
      A.1.2 Solution to Our Boundary Value Problem ............... 73
### B Programming

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>C++ implementation of Metropolis algorithm</td>
<td>75</td>
</tr>
<tr>
<td>B.1.1</td>
<td>main.cpp</td>
<td>75</td>
</tr>
<tr>
<td>B.1.2</td>
<td>parameter.h</td>
<td>80</td>
</tr>
<tr>
<td>B.1.3</td>
<td>input-ini.h</td>
<td>81</td>
</tr>
<tr>
<td>B.1.4</td>
<td>str.h</td>
<td>85</td>
</tr>
<tr>
<td>B.1.5</td>
<td>perAux.h</td>
<td>96</td>
</tr>
<tr>
<td>B.1.6</td>
<td>rnd250.c</td>
<td>102</td>
</tr>
<tr>
<td>B.1.7</td>
<td>random-spd5.h</td>
<td>102</td>
</tr>
<tr>
<td>B.2</td>
<td>domainFinder.cpp</td>
<td>104</td>
</tr>
</tbody>
</table>

### C References

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>113</td>
</tr>
</tbody>
</table>
1 Introduction

NP supercrystals are regular arrangements of NPs in complete analogy to crystals in condensed matter. An example is given in figure 1.

![TEM micrographs and sketches of AB$_2$ superlattices (isostructural with intermetallic phase AlB$_2$, SG 191) of 11-nm $\gamma$-Fe$_2$O$_3$ and 6-nm PbSe NCs. [17]

The research on supercrystals started already several decades ago in the context of colloidal crystals on micron sized silica or polystyrene spheres [11]. In recent years, groups have succeeded in fabricating one, two or three dimensional arrangements of e.g. magnetic or semiconducting NPs using various self-organization techniques [3, 4, 8, 12]. The properties of these systems are determined both by the individual NPs and the interactions among them. Provided the composition and interaction of different NPs can be tuned, interesting behavior and novel applications may emerge [18].

But also from a fundamental point of view, such supercrystals of interacting magnetic nanoparticles are a fascinating subject. In particular NPs which consist of one magnetic single-domain per NP represent an interesting model system. The question whether systems with long-range interaction such as dipole-dipole interactions can exhibit long-range ordered ground states is not comprehensively answered in more than two dimensions. In fact, even the Mermin-Wagner theorem, answering the question for systems of up to two dimensions with a general ‘no’, has been challenged by e.g. Kosterlitz-Thouless transitions, via a divergence of the correlation length [9]. Albeit NP supercrystals have much larger length scales than usually considered in many-body theory, mesoscopic systems do provide very similar questions. The different length-scale and the dominance of the dipole-dipole interaction between the NP macromoments potentially leads to new possibilities of magnetic ordering within physical systems. In any case, the NP magnetic dipole-moments are subject to frustrated interactions.
2 Theory of Magnetism

Theoretical expectations of magnetic measurements will be the cornerstone of this chapter.

One can classify the magnetic phenomena into three main groups:

- Diamagnetism
- Paramagnetism
  - Localized Moments
  - Itinerant Moments
- Collective Magnetism
  - Ferromagnetism
  - Ferrimagnetism
  - Antiferromagnetism

2.1 Origin of Magnetism in Solid State Matter

2.1.1 Bohr-van Leeuwen theorem

Often, para- and diamagnetism are explained as an induction effect. It implies a theory of moving charges which can be treated within a classical atom model, and vector analysis as the significant mathematical language to understand magnetism. The general idea might be the following: The Larmor precession of the orbital angular momentum around the direction of the magnetic field induces an extra moment which according to Lenz’s law is directed oppositely to the orientation of the applied field. Trying to rigorously calculate any atomic magnetic moment in this setting will inevitably lead to a contradiction as we will see now:

Let a solid consist of identical ions and let it possess translational symmetry. We can then write the magnetization as

\[ M = \frac{N}{V} \langle m \rangle \]

where \( \mathbf{m} \) is the magnetic moment of the individual ion. \( N \) is the number of ions in the volume \( V \). If magnetism is a classical phenomenon, then each ion in our solid must offer a classical Hamiltonian function \( H \) and we have the following relations from statistical mechanics:
2. THEORY OF MAGNETISM

\[ m = -\nabla B_0 H \quad (2.1) \]

\[ \langle m \rangle = \frac{1}{Z} \int d^{3N_e}x \int d^{3N_e}p \ m e^{-\beta H} \quad (2.2) \]

\[ Z = \frac{1}{N_e! \hbar^{3N_e}} \int d^{3N_e}x \int d^{3N_e}p \ e^{-\beta H} \quad (2.3) \]

where

- \( Z \) is the classical partition function
- \( N_e \) is the number of electrons per ion
- \( \beta \equiv (k_B T)^{-1} \) is the inverse temperature
- \( \frac{1}{N_e! \hbar^{3N_e}} \int d^{3N_e}x \int d^{3N_e}p \ \equiv \iint_{\Gamma} \) is the normalized integration over the complete phase space spanned by \( N_e \) electrons (3 dimensions for both position and momentum variables).

We arrive at

\[ \langle m \rangle = \frac{1}{Z} \int d\gamma e^{-\beta H} m = -\frac{1}{Z} \int d\gamma e^{-\beta H} \nabla B_0 H \]

\[ = \frac{1}{\beta Z} \nabla B_0 \int d\gamma e^{-\beta H} \quad (2.4) \]

We therefore need to investigate the field dependence of the classical partition function. Noting that any magnetic field \( B_0 \) can be written as \( B_0 = \nabla \times A \), we rewrite our Hamilton function as

\[ H = \frac{1}{2m} \sum_{\alpha=1}^{3} \sum_{i=1}^{N_e} (p_i + eA_\alpha)^2 + H_{\text{int}}(r_1, \cdots, r_{N_e}) \]

where \( H_{\text{int}} \) is the term representing the electron interactions. Thereby, we separated the position- and momentum dependent parts of the total Hamilton function which
2. **THEORY OF MAGNETISM**

lets the partition function take the form

\[
Z = \frac{1}{N_e \hbar^{3N_e}} \int_{\Gamma_r} d^{3N_e} x e^{-\beta H_{\text{int}}(r_1, \ldots, r_{N_e})} \times \int_{\Gamma_p} d^{3N_e} p \exp \left( -\frac{\beta}{2m} \sum_{a=1}^{3} \sum_{i=1}^{N_e} ((p_i)_a + eA_a)^2 \right)
\]

Since the \( \Gamma_p \) integration lets every momentum coordinate run from \(-\infty \) to \( \infty \), the canonical momentum \( \tilde{p}_i := p_i + eA \) can be transformed linearly with arbitrary but constant \( A \), without changing the integration limits. Since any applied magnetic field \( B_0 \) would hence not alter the \( p \)-integration, we have

\[
\nabla_A Z = \nabla_{B_0} Z = 0 \quad (2.5)
\]

\[
\Rightarrow \langle m \rangle = 0 \quad (2.6)
\]

for any magnetic field.

We have therefore shown rigorously that, classically, there is no magnetism. There cannot be any discussion whether magnetism does exit, therefore we have shown that it must be an effect only understandable quantum mechanically. This is the famous Bohr-van Leeuwen theorem:

---

**Bohr-van Leeuwen Theorem**

Magnetism is a quantum mechanical effect. Strictly classically, there cannot be either dia-, para- or collective magnetism

---

For the purposes of this thesis however, this does not mean that we will proceed to argue strictly quantum mechanically. We can continue using classical or semiclassical models and calculations.

2.2 **Magnetic Interactions**

Different types of magnetic interactions are discussed in this section. We focus on interactions that are present in systems composed of magnetic moments of constant length.
2. Theory of Magnetism

2.2.1 Dipole-Dipole Interaction

The first interaction which might be expected to play a role is the magnetic dipolar interaction. Two magnetic dipoles $\mathbf{m}_1$ and $\mathbf{m}_2$ separated by $\mathbf{r}$ have an energy equal to

$$E_{\text{dip}} = \frac{\mu_0}{4\pi r^3} \left[ \mathbf{m}_1 \cdot \mathbf{m}_2 - \frac{3}{r^2} (\mathbf{m}_1 \cdot \mathbf{r})(\mathbf{m}_2 \cdot \mathbf{r}) \right]$$

$$= \frac{\mu_0}{4\pi r^3} \left[ \mathbf{m}_1 \cdot \mathbf{m}_2 - 3(\mathbf{m}_1 \cdot \hat{r})(\mathbf{m}_2 \cdot \hat{r}) \right]$$

with $\hat{r} \equiv \frac{\mathbf{r}}{r}$

which therefore depends on their separation and their degree of mutual alignment. If the magnetic moments in question are single electrons and we take distances at atomic length scales, the energies were equivalent to roughly 1K in temperature. Therefore, properties of condensed matter are conventionally not overly dependent on the dipole-dipole interaction except for those ordering at mK temperatures. On the other hand, this is a long-range interaction where the complete sample needs to be taken into account. For example, it is conceivable that the specific length scale of this interaction influences the emergence of magnetic domains or other phenomena of magnetic ordering.

2.2.2 Exchange Interaction

In atomic crystals or any conventional magnetic system, exchange interactions are usually responsible for long-range magnetic order. They are purely quantum mechanical in nature, but the underlying principle is electrostatics.

Consider a simple model with just two electrons which have spatial coordinates $\mathbf{r}_1$ and $\mathbf{r}_2$ respectively. The wave function for the joint state can be written as a product of single electron states, so that if the first electron is in state $\psi_a(\mathbf{r}_1)$ and the second electron is in state $\psi_b(\mathbf{r}_2)$, then the joint wave function is in $\psi_a(\mathbf{r}_1)\psi_b(\mathbf{r}_2)$.

Since electrons are fermions, the true wave function must be antisymmetric with respect to particle exchange. So the spin part of the wave function must either be an antisymmetric singlet state $\chi_S (S = 0)$ in the case of a symmetric spatial state or a symmetric triplet state $\chi_T (S = 1)$ in the case of an antisymmetric spatial state. Therefore we can write the wave function for the singlet case $\Psi_S$ and the triplet case $\Psi_T$ as

$$\Psi_S = \frac{1}{\sqrt{2}} [\psi_a(\mathbf{r}_1)\psi_b(\mathbf{r}_2) + \psi_a(\mathbf{r}_2)\psi_b(\mathbf{r}_1)] \chi_S$$

$$\Psi_T = \frac{1}{\sqrt{2}} [\psi_a(\mathbf{r}_1)\psi_b(\mathbf{r}_2) - \psi_a(\mathbf{r}_2)\psi_b(\mathbf{r}_1)] \chi_T$$
where both the spatial and spin parts of the wave function are included. If we assume the spin parts to be normalized, the energies of the two possible states are

\[
E_S = \int \Psi_S^* \mathcal{H} \Psi_S \, d^3r_1 \, d^3r_2 \\
E_T = \int \Psi_T^* \mathcal{H} \Psi_T \, d^3r_1 \, d^3r_2
\]

\[\Rightarrow E_S - E_T = 2 \int \psi_a^*(r_1) \psi_b^*(r_2) \mathcal{H} \psi_a(r_2) \psi_b(r_1) \tag{2.7}\]

We can construct a new effective Hamiltonian by using

\[
S_1 \cdot S_2 = \begin{cases} 
-\frac{3}{4} & S = 0 \text{ singlet} \\
\frac{1}{4} & S = 1 \text{ triplet}
\end{cases}
\]

The new Hamiltonian becomes

\[
\mathcal{H}' = \frac{1}{4} (E_S + 3E_T) - (E_S - E_T) \, S_1 \cdot S_2
\]

The interesting part is the non-constant spin dependent term. Defining the exchange constant \( J \) by

\[
J := \frac{E_S - E_T}{2} \tag{2.7}
\]

we define a new effective Hamiltonian

\[
\mathcal{H}^{\text{spin}} = -2J \, S_1 \cdot S_2 \tag{2.8}
\]

If \( J > 0, E_S > E_T \) and the triplet state \( S = 1 \) is favoured. If \( J < 0, E_T > E_S \) and the singlet state \( S = 0 \) is favoured.

The above derivation only holds for exactly 2 electrons and the generalization to many-body systems is not trivial. Nevertheless, the Hamiltonian (2.8) motivates the Heisenberg model

\[
\mathcal{H} = -\sum_{ij} J_{ij} S_i \cdot S_j \tag{2.9}
\]

with the factor 2 omitted to prevent counting pairs twice.

### 2.2.3 Anisotropic Exchange or Dzyaloshinsky-Moriya Interaction

This interaction originates from spin-orbit interactions within one magnetic ion. It can be understood as the exchange interaction between the excited state of one ion
2. THEORY OF MAGNETISM

and the ground state of the other.

\[ H_{\text{DM}} = \mathbf{D} \cdot (\mathbf{S}_1 \times \mathbf{S}_2) \]  \hspace{1cm} (2.10)

The vector \( \mathbf{D} \) vanishes when the crystal field has an inversion symmetry with respect to the centre between the two magnetic ions. However, in general \( \mathbf{D} \) may not vanish and then will lie parallel or perpendicular to the line connecting the two spins, depending on the symmetry. The form of the interaction is such that it tries to force \( \mathbf{S}_1 \) and \( \mathbf{S}_2 \) to be at right angles in a plane perpendicular to the vector \( \mathbf{D} \) in such an orientation as to ensure that the energy is negative. Its effect is therefore very often to cant the spins by a small angle. It commonly occurs in antiferromagnetics and results in a small ferromagnetic component of the moments which is produced perpendicular to the spin axis of the antiferromagnet. The effect is known as \textit{weak ferromagnetism}. It is found in, for example, \( \alpha\text{-Fe}_2\text{O}_3 \).

2.3 Types of Magnetic Behavior

We discuss magnetic behavior via the classification of magnetic materials. Specifically, we utilize the characteristic dependence of the magnetic susceptibility \( \chi \) on temperature, applied magnetic field and history:

2.3.1 Diamagnetism

Diamagnetism is defined by

\[ \chi^{\text{dia}} < 0 \quad \chi^{\text{dia}} = \text{const.} \]

The classical picture of diamagnetism being an induction effect has already been discussed in 2.1.1. Without discussing proper quantum mechanical treatments like Landau-Diamagnetism in crystals, we focus on phenomenology: Diamagnetism is a property displayed by \textit{all} materials. However, we only speak of diamagnetism if no other form of magnetism (para- or collective magnetism) is present since it will in general be the weaker effect.

Examples include

- Organic molecules
- Few metals like bismut, zinc, mercury
- Nonmetals like sulfur, iodine, silicon
- \textit{Meissner-Ochsenfeld} effect: Superconductors at \( T < T_c \) are perfect diamagnets: \( \chi^{\text{dia}} = -1 \)
As noted, in the presence of e.g. permanent magnetic dipoles, diamagnetism will be a negligible effect and it will not feature throughout this thesis.

2.3.2 Paramagnetism

Typically, one has

\[ \chi_{\text{para}} > 0 \quad \chi_{\text{para}} = \chi_{\text{para}}(T) \]

Essentially, paramagnetism is connected to the existence of permanent magnetic dipoles which try to, more or less, orient themselves along an applied auxiliary field \( H \). This competes with thermal motion, hence the temperature dependence of \( \chi_{\text{para}} \).

In materials, there are two possible origins of such permanent dipoles:

- Localized moments \( \rightarrow \) *Langevin paramagnetism*
- Itinerant moments (\( \sim \) quasi-free conduction electrons) \( \rightarrow \) *Pauli paramagnetism*

Generally, one has

\[ \chi_{\text{Langevin}} \gg \chi_{\text{Pauli}} \approx \text{const.} \]

and \( \chi_{\text{Pauli}} \) temperature independent to a first order approximation. Therefore, only Langevin paramagnetism will feature in this thesis, this however extensively.

2.3.3 Collective Magnetism

The susceptibility in this case is in general a complicated function of the applied field, temperature and the magnetic history of the sample:

\[ \chi_{\text{coll.}} = \chi_{\text{coll.}}(T, H, \text{history}) \]

Collective magnetism arises due to interactions between the permanent magnetic dipoles. Again, these permanent magnetic dipoles can be either localized or itinerant.

In the context of more standard solid state physics, the interaction in question is the quantum mechanical *exchange interaction* which does not have a classical analogy. As we will discuss later, this view is not sufficient in case of supercrystals comprised of magnetic nanoparticles. Instead, the origin for collective magnetism in our systems, will be the magnetic dipole-dipole interaction between the permanent magnetic moments of the nanoparticles.

Nevertheless, we recall other characteristic features of standard systems with collective magnetism: The exchange interaction leads to a critical temperature \( T^* \) below which there exists a *spontaneous magnetization*, i.e. a spontaneous ordering.
of dipoles that is not forced externally. Collective magnetism due to exchange interaction is conventionally divided into three sub-classes:

- Ferromagnetism
- Ferrimagnetism
- Antiferromagnetism

Above the critical temperature $T^*$, collective magnetic transitions into paramagnetism with the characteristic temperature dependence of the inverse susceptibility sketched in figure 2.

![Figure 2: Temperature dependence of the inverse magnetic susceptibility according to the Curie-Weiss law. Left: Paramagnetism, middle: Ferromagnetism, right: Antiferromagnetism](image)

### 2.3.3.1 Ferromagnetism

In this case the critical temperature is called the *Curie temperature*:

$$T^* = T_C$$

For temperatures $0 < T < T_C$ the permanent moments have a preferential orientation. At $T = 0$, all moments are oriented parallel to each other although $\mu_0 H_{\text{ext}} = 0$.

### 2.3.3.2 Ferrimagnetism

In this case the lattice of the system is divided into at least two sublattices $A, B$ with different absolute values of sublattice magnetizations $M_A, M_B$ such that

$$M_A \neq M_B \quad \text{and} \quad M_A + M_B \neq 0 \quad \text{for} \ T < T_C$$

### 2.3.3.3 Antiferromagnetism

Here, the critical temperature is called *Néel temperature*:

$$T^* = T_N$$
It is a special case of ferrimagnetism:

\[ |M_A| = |M_B| \neq 0 \quad \text{and} \quad M_A + M_B = 0 \quad \text{for} \quad T < T_C \]

The total magnetization \( M = M_A + M_B \) is therefore always zero in the absence of externally applied fields.

### 2.3.4 Geometrically Frustrated Systems and Spin Glasses

In realistic systems, more than one interaction or energy contribution determines the total energy of the system. The subsequent competition between interactions can lead to complex magnetic behavior that is not trivially following from the constituent interactions.

Even before considering the competition between independent interactions, 2-particle interactions in realistic (super)crystal structures will induce geometric frustration: In many lattices it is not possible to satisfy all the interactions in the system to find the ground state. Often this leads to the absence of a single unique ground state but a variety of low energy states.

We refer to this property that the system has no good way to choose which low energy configuration it must adopt as frustration. It is important to note that this phenomenon emerges with pretty much any magnetic interaction, including nearest-neighbour interactions such as the antiferromagnetic interaction:

![Figure 3: Kagome lattice as an example for a 2D geometry which leads to a frustrated system.](image)

On a square lattice it is easily possible to satisfy the requirement that nearest-neighbour spins must be antiparallel. However on a triangular lattice, things differ: If two adjacent spins are placed antiparallel, the third spin has no good choice
between aligning with one or the other spin. Locally, there exists no unique lowest-
energy state, but only two low-energy states that are equally unsatisfied.
This minimal frustrated system already shows metastability, hysteresis effects and
time-dependent relaxation towards equilibrium, all of which are phenomena absent
from the square lattice.
This example relies heavily on the fact that we considered models with low dimen-
sionality, i.e. \( \dim < 3 \). In three dimensions, frustration would not emerge from
a triangular or e.g. \textit{Kagome} lattice, but from a pyrochlore structure in which the
magnetic ions occupy a lattice of corner sharing tetrahedra. Here, there is no spin
order observable for any temperature, only a classical groundstate with macroscopic
degeneracy, sometimes described as \textit{cooperative paramagnetism}.

We can define a spin glass as a random, magnetic system with mixed interactions
classified by a random yet cooperative freezing of spins at a well defined freezing
temperature \( T_f \) below which a metastable frozen state appears without the usual
magnetic long-range ordering.
The decisive term in this definition is \textit{random}. Different types of randomness can
be imagined to transform a non-spin glass into a spin glass:

- Site-randomness
- Bond-randomness

Equally important is the presence of competing interactions as previously described.
Contributing features include magnetic anisotropy e.g. in amorphous magnets where
a random distribution of the \textit{easy-axes} implies random anisotropy.
3 Methods

3.1 Why Do Simulations in Physics

In many cases, models of ideal systems can be explored by theoretical methods, but they do not offer any physical realization so that no comparison to experiment is available. In many other cases, experimental realizations are too complex to be modelled by theoretical methods. In this situation the only possible test for an approximate theoretical solution is to compare with 'data' generated from a computer simulation.

Nuclear reactor meltdowns are a dramatic example: Although we want to know what the results of such events would be, we do not want to carry out experiments. There are also real physical systems which are sufficiently complex that they are not presently amenable to theoretical treatment. An example is the problem of understanding the specific behaviour of a system with many competing interactions and which is undergoing a phase transition. A model Hamiltonian/Hamilton function which is believed to contain all the essential features of the physics may be proposed, and its properties may then be determined from simulations. If the simulation disagrees with experiment, then a new Hamiltonian must be found. An important advantage of simulations is that different physical effects which are simultaneously present in real systems may be isolated and, through separate consideration by simulation, may provide a much better understanding.

The Monte Carlo method has had a considerable history in physics. As far back as 1949 a review of the use of Monte-Carlo simulations using 'modern computing machines' was presented by Metropolis and Ulam [15]. In addition to giving examples they also emphasized the advantages of the method. Of course, in the following decades the kinds of problems they discussed could be treated with far greater sophistication that was possible in the first half of the twentieth century. Nowadays, Monte-Carlo simulation methods have spread into different disciplines that have barely any connection to physics.

3.2 Categories of Monte-Carlo Simulations

A brief overview about Monte-Carlo methods is given. I present additional details about Markov-Chain methods because the research which is presented in this thesis was exclusively done via the Metropolis-Algorithm, the most famous representative of Markov-Chain Monte-Carlo methods.
3.2.1 Monte Carlo in Statistical Physics

Monte-Carlo methods are used throughout many physical and non-physical science branches. In physics, especially statistical mechanics, the following branches are of special interest:

- Monte-Carlo integration
- Importance sampling techniques, specifically Markov Chain methods
  - Local algorithms
  - Non-local algorithms

3.2.2 Markov Chain methods

The concept of Markov chains is central to those Monte-Carlo methods that are the most prominent in physics, especially solid state physics [10].

We define a stochastic process at discrete times labeled consecutively $t_1, t_2, t_3 \ldots$ for a system with a finite set of possible states $S_1, S_2, S_3, \ldots$, and we denote by $X_t$ the state the system is in at time $t$. We consider the conditional probability that $X_{t_n} = S_{i_n}$,

$$P \left( X_{t_n} = S_{i_n} | X_{t_{n-1}} = S_{i_{n-1}}, X_{t_{n-1}} = S_{i_{n-2}}, \ldots, X_{t_2} = S_{i_1} \right)$$

given that at the preceding time the system state $X_{t_{n-1}}$ was in state $S_{i_{n-1}}$, etc. Such a process is called a Markov process if this conditional probability is in fact independent of all states but the immediate predecessor, i.e.

$$P \left( X_{t_n} = S_{i_n} | X_{t_{n-1}} = S_{i_{n-1}} \right)$$

The corresponding sequence of states $\{X_t\}$ is called a Markov chain, and the above conditional probability can be interpreted as the transition probability to move from the state $i$ to state $j$,

$$W_{ij} = W(S_i \rightarrow S_j) = P(X_{t_n} = S_j | X_{t_{n-1}} = S_i)$$

We further require that

$$W_{ij} \geq 0 \quad \sum_j W_{ij} = 1$$

as usual for transition probabilities. We may then construct the total probability $P(X_{t_n} = S_j)$ that at time $t_n$ the system is in state $S_j$ as

$$P(X_{t_n} = S_j) = P(X_{t_n} = S_j | X_{t_{n-1}} = S_i) \cdot P(X_{t_{n-1}} = S_i)$$

$$= W_{ij} P(X_{t_{n-1}} = S_i)$$
The master equation conserves the change of this probability with time $t$ (treating time as a continuous rather than discrete variable and writing then $P(X_{t_n} = S_j) =: P(S_j, t)$)

$$\frac{dP(S_j, t)}{dt} = -\sum_i W_{ji}P(S_j, t) + \sum_i W_{ij}P(S_i, t)$$

(3.1)

Equation (3.1) can be considered as a 'continuity equation': The total probability is conserved at all times because

$$\sum_j P(S_j, t) \equiv 1 \quad \forall t \in \mathbb{R}$$

Furthermore, all probability of a state $i$ that is 'lost' by transition to state $j$ is gained in the probability of that state, and vice versa.

The Master equation therefore describes the balance of gain and loss processes:

The processes

$$S_j \rightarrow S_{i_1}$$
$$S_j \rightarrow S_{i_2}$$
$$S_j \rightarrow S_{i_3}$$
$$\ldots$$

are mutually exclusive. Hence the total probability for a move away from the state $j$ is simply the sum $\sum_i W_{ij}P(S_j, t)$.

We stress that equation (3.1) brings out the basic property of Markov processes:

**Basic Property of Markov Processes**

The knowledge of the state at time $t$ completely determines the future time evolution.

The main significance is that the importance sampling Monte Carlo process that will feature throughout this thesis via the Metropolis algorithm, can be interpreted as a Markov process if the following is true about the transition probabilities $W_{ij}$:

From now on we require that the transition probabilities satisfy the principle of detailed balance with the equilibrium probability $P_{eq}(S_j)$:

$$W_{ji}P_{eq}(S_j) = W_{ij}P_{eq}(S_i)$$

(3.2)

which will be fundamental for the Metropolis algorithm presented in section 3.3.

We already note that (3.2) implies

$$\frac{dP_{eq}(S_j, t)}{dt} \equiv 0 \quad \forall t \in \mathbb{R}$$
when put into (3.1) because all gain and loss terms cancel exactly. This is elementary for what one would understand by the term ‘equilibrium’ in context of a system transitioning between states [10].

### 3.3 The Metropolis Algorithm

From now on, we are only concerned with Monte-Carlo techniques as applied in statistical physics, specifically on-lattice models of systems offering permanent magnetic dipoles that display collective magnetism due to interactions.

In order to illustrate our discussion, we consider the Ising model.

The simple Ising model in zero applied field consists of spins which are confined to the sites of a lattice and which may have only the values +1 or −1. These spins interact with their nearest neighbors on the lattice with interaction constant $J$; the Hamiltonian for this model is given by

$$H = -J \sum_{i,j} \sigma_i \sigma_j \quad \sigma_i = \pm 1$$

The Ising model has been solved exactly in one and two dimensions so that Monte-Carlo results in these cases can be directly compared to theoretical expectations.

Next I present the classic Metropolis method.

Configurations are generated from a previous state using a transition probability which depends on the energy difference between the initial and final states. The sequence of states produced follows a time-ordered path, but the time in this case is referred to as ‘Monte Carlo time’. For relaxation models, such as we will assume are viable models for magnetization curves of nanoparticle supercrystals, the time-dependent behavior is described by a master equation like (3.1):

$$\frac{\partial P_n(t)}{\partial t} = -\sum_{n\neq m} [P_n(t)W_{n\rightarrow m} - P_m(t)W_{m\rightarrow n}]$$

(3.3)

where $P_n(t)$ is the probability of the system being in state $n$ at time $t$, and $W_{n\rightarrow m}$ is the transition rate for the process $n \rightarrow m$. We again identify the detailed balance from (3.2)

$$P_n(t)W_{n\rightarrow m} = P_m(t)W_{m\rightarrow n}$$

as a simple constraint that guarantees an equilibrium being realized as

$$\frac{\partial P_n(t)}{\partial t} \equiv 0$$

The probability of the $n$th state occurring in a classical system is given by

$$P_n(t) = \frac{1}{Z} \exp\left(-\frac{E_n}{k_BT}\right)$$

(3.4)
where $Z$ is the partition function. Outside of very simple cases like the Ising model discussed here, this expression is very difficult to evaluate, mostly because the partition function, i.e. knowledge about every possible state and its energy, is rarely ever known. However, one can avoid this difficulty by generating a Markov chain of states, i.e. generate each new state directly from the preceding state. If we produce the $n$th state from the $m$th state, the relative probability is the ratio of the individual probabilities and the denominator, the largely unknown $Z$ cancels. As a result, only the energy difference between the two states is needed, e.g.

$$\Delta E = E_n - E_m$$

The previous idea is possibly the most significant reason why Markov-Chain methods have been so successful in statistical physics because one can circumvent the arduous and ultimately not as interesting task of evaluating the partition function of a large, interacting system.

For the transition rates, any choice that satisfies detailed balance (3.2) is acceptable\(^1\). The first choice of rate which was used in statistical physics is the Metropolis form \cite{15}

$$W_{m\rightarrow n} = \begin{cases} \tau_0^{-1} \exp(-\Delta E/k_B T) & \Delta E > 0 \\ \tau_0^{-1} & \Delta E < 0 \end{cases} \quad (3.5)$$

where $\tau_0$ is the time required to attempt a spin-flip. The way the Metropolis algorithm is implemented can be described by a simple recipe, illustrated in figure 5

**Metropolis importance sampling Monte-Carlo scheme** \quad (3.6)

1. Choose an initial state
2. Choose a site $i$
3. Calculate the energy change $\Delta E$ which results if the spin at site $i$ is flipped
4. Generate a random number $r$ such that $0 < r < 1$
5. If $r < \exp(-\Delta E/k_B T)$, flip the spin
6. Go to the next site and go to 3.

\(^1\)From the derivation from the Master equation (3.1), detailed balance is sufficient but not necessary. It turns out however, that in practice only transition rates that do satisfy detailed balance are regularly used.
After a certain number of spins have been considered, the properties of the system are determined and added to the statistical average which is being kept. Note that the random number \( r \) must be chosen uniformly in the interval \([0, 1]\), and successive random numbers should be uncorrelated.

Obviously, this algorithm can be easily modified for use on different on-site models where the total energy of any given configuration can be calculated. Both the required high quality of random numbers and the precise nature of 'flipping' in the context of the more involved model we consider in the actual research part of this thesis will be thoroughly discussed in section 3.4.

'Monte-Carlo time' is usually measured in terms of Monte Carlo steps per site (MC-S/site) which corresponds to the consideration of every spin in the system once. With the algorithm from figure 5 states are generated with a probability proportional to (3.4) once the number of states is sufficiently large such that any initial transients from the early stages of the iterative loop are negligible. Then the desired averages

\[
\langle A \rangle = \sum_n P_n A_n
\]

of variables or observables \( A \) simply become arithmetic averages over the entire sam-
ple of states which is kept. Note that if an attempted spin-flip is rejected, the old state is counted again for the averaging.

We reiterate that the hallmark of the *Metropolis algorithm* is the specific choice of transition rates given in (3.5). Additional transitions can be imagined like Parallel Tempering which are used in order to accelerate convergence speeds of equilibrium averages for observables or the spin-configuration itself. Also, the choice how to flip spins or, more generally, choose a new configuration for consideration during a subsequent MCS, can be heavily altered compared to figure 5 where one just picks a site randomly.

Both types of modifications lead to algorithms that are not strictly *Metropolis* algorithms, but are still Markov-chain methods if they offer constraints ensuring possible equilibrium, most frequently via enforcing detailed balance [10].

### 3.4 Quality of Pseudo-random Numbers Picked on a 2-Sphere

The Metropolis algorithm as described in section 3.3 has been used extensively for Ising models where spins are restricted to exactly two states, *up* and *down*. In particular, this implies that it is clear how a local update to achieve a new configuration has to be carried out: Flipping one site to the other state.

In a Heisenberg-like model where each site carries a (super)spin that can assume every position on the 2 dimensional surface of a 3D unit sphere, it is far less clear how an update should be performed. Broadly speaking, 2 types of local updates in a Heisenberg-like on-lattice spin model are possible:

- The possible new states are uniformly distributed on the entire sphere without any bias from position of the original state.
- There is a probabilistic bias which part of the sphere is reachable within one Monte Carlo update.

In this thesis, we will exclusively utilize the second type of spin update. The implementation is depicted in figure 6:
3. METHODS

Figure 6: How spins are transformed or ‘flipped’ during one Monte Carlo step. A random vector \( d\mathbf{r} \) (red) is obtained via the Marsaglia method and added to the original \( \mathbf{r} \) (black). The result is normalized, yielding \( \mathbf{r}' \) (blue). In the end, one has both \( \mathbf{r} \) and \( \mathbf{r}' \) within a spherical shell with radius 1 (green).

**Incremental Transformation**

1. Generate an unbiased, random vector \( \tilde{d}\mathbf{r} \) that is uniformly distributed on a 3D unit sphere and stretch it if necessary with a scalar \( d_m \) such that \( d\mathbf{r} = d_m \tilde{d}\mathbf{r} \)

2. Add this random shift vector to the original \( \mathbf{r} \)

3. Normalize the result so that \( \mathbf{r}' \) is a unit vector.

The generation of a uniform distribution of random vectors on a spherical shell is discussed in section 3.4.1, we can assume for now that we have access to such random vectors of sufficient quality. This construction ensures that as long as \( d_m \) is not too large, \( \mathbf{r}' \) is biased towards not deviating too much from \( \mathbf{r} \). For example:

\[
d_m \leq 2 \quad \Rightarrow \quad \mathbf{r}' \neq -\mathbf{r} \quad \forall d\mathbf{r}
\]

which means that not every orientation is obtainable within one MCS. The \( d_m \) dependence of the \( \mathbf{r}' \) distribution is discussed in 3.4.2. In most cases\(^2\), we choose \( d_m = 1 \) which means that within 1 MCS there is a heavy bias for new orientations towards the original position. Figuratively, if the original position \( \mathbf{r} \) represents the north pole, then \( \mathbf{r}' \) is mathematically confined in the northern hemisphere while the equator is only asymptotically obtainable.

\(^2\)Justification and exemptions are given when needed
3. METHODS

3.4.1 Uniformly Distributed Unit Vectors on a \((d - 1)\)-Sphere

For arbitrary dimensions \(d \geq 1\) one can generate uniformly distributed vectors on the surface of the corresponding \((d - 1)\)-sphere:

- for \(d = 1\): set \([-1, 1]\) as boundary of the interval \([-1, 1]\)
- for \(d = 2\): full circle (as boundary, not area) with radius 1 and origin \((0, 0)\) in cartesian coordinates
- for \(d = 3\): spherical shell of the conventional 3-sphere with radius 1 and origin \((0, 0, 0)\) in cartesian coordinates

We can always do this with \(d\) random variables which have a Gaussian distribution in an arbitrary interval:

\[
\text{Gaussian } (x_1, x_2, \ldots, x_d) \mapsto \frac{1}{\sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}
\]

This \(d\)-vector will be uniformly distributed on the surface of a \(d\)-sphere.

For the special case \(d = 3\) that we are interested in however, there is a more elegant method by Marsaglia [13] which requires only 2 instead of 3 random numbers:

**Marsaglia Method**

1. Pick \(a\) and \(b\) from independent uniform distributions on \((-1, 1)\)
2. Reject points for which \(a^2 + b^2 \geq 1\)
3. From the remaining points
   - \(x = 2a\sqrt{1 - a^2 - b^2}\)
   - \(y = 2b\sqrt{1 - a^2 - b^2}\)
   - \(z = 1 - 2(a^2 + b^2)\)

The vectors \(
\begin{pmatrix} x \\ y \\ z \end{pmatrix}
\)
then have a uniform distribution on the surface of a unit sphere as needed in our spin-update scheme.
3.4.2 Examples of Vector Distribution

Next we demonstrate the influence of the length of the test vector $d_m$ onto the vector distribution of a modified vector after a certain number of steps according to our recipe in figure 6.

The following graphs correspond to this scenario:

1. $10^6$ unit vectors are initialized identically with orientation $(1,0,0)$
2. Choose $d_m = 2$.
3. With independent Marsaglia random vectors, perform the transformation.
   - once $\rightarrow$ graphs in figures 7, 8, 9.
   - 1000 times $\rightarrow$ graphs in figures 10, 11, 12.
4. Record the component distribution of the resulting vectors after all transformations are done.
5. Fill histograms with bins of width 0.002.

![component distribution](image)

Figure 7: Distribution of x-components after 1 loop
We observe that the x-component covers the complete interval $(-1, 1)$, but the distribution is skewed in favor of the original orientation $x \rightarrow 1$. The y- and z-components follow identical distributions, centered around 0. Note that the integrated y- and z-components are less than the x-component because one 'flip' cannot undo the non-uniform start distribution around $(1, 0, 0)$.
3. METHODS

Figure 10: Distribution of x-components after 1000 loops

Figure 11: Distribution of y-components after 1000 loops
3. METHODS

In this second set of graphs, we see that a sufficient number of transformation steps produces a distribution that is indistinguishable from a uniform distribution.

3.4.3 MC simulations and Ergodicity

We give a definition of ergodicity in the language of Monte-Carlo simulations on realistic machines with finite precision:

Definition: Ergodicity

In the context of a MC simulation, ergodicity means that the implementation of the algorithm ensures that all points in the simulated phase space of the system are eventually visited after a finite number of simulation steps.

Because of the finite nature of the precision of the machine, the phase space is also finite, albeit quite large. This is a subtle difference to the original definition of ergodicity in an uncountable phase space where each point merely is arbitrarily closely matched after sufficient, but also finite time.

In this thesis, we are interested in measurements taken at thermal equilibrium and therefore ergodicity is generally assumed on the level of microstates. This assumption is also referred to as ergodic hypothesis.

Ergodic Hypothesis

Thermodynamic systems evolve in a way that all energetically allowed regions in phase space are covered. The time that the trajectory stays in a particular region of phase space is proportional to the phase-space volume of this region.

Figure 12: Distribution of z-components after 1000 loops
The consequence of the above is that a necessary condition for our implementations is that one must choose a combination of $d_m$ and number of MCS that allows any microstate evolving to any other microstate. Additionally, on average there must not be any bias left in favor of phase space regions 'closer' to the original region. The last point implies that a distribution like in figure 10 is fine while a distribution like figure 7 is in violation of the ergodic hypothesis and thus cannot correspond to a system at thermal equilibrium.
4. Ferromagnetic Nanoparticles and Supercrystals in the Non-interacting Limit

The main goal of this thesis is to further the knowledge of dipolar magnetic systems. Because the interactions are long-range, a theoretical treatment is presently impossible. Comparing the simulation results with experiments will at least provide answers to questions like 'Do we know which energies are important' and 'Is the single-domain approximation valid'?

4.1 Theory I

Throughout section 4 we will not need to consider any interactions between the magnetic moments of the nanoparticles. The physical properties we simulate here are therefore comparatively easy to treat rigorously. This will be discussed next.

4.1.1 Stoner-Wohlfarth Model

Two energies will be considered in this chapter.

- The magnetostatic energy of one nanoparticle magnetic moment (‘superspin’) \( \mathbf{m} \) in an external magnetic field

\[
E_m = - \mathbf{m} \cdot \mathbf{B} = -m B \cos(\theta)
\]

- The magnetocrystalline anisotropy energy \( E_a = K V \sin^2(\delta) \)

which is quantified according to the Stoner-Wohlfarth model for uniaxial anisotropy constants.

\[
E = E(\theta, \delta) = -mB \cos(\theta) + KV \sin^2(\delta)
\]

or, in order to use the experimentally more common applied field strength \( \mathbf{H} \):

\[
E = -\mu_0 H m \cos(\theta) + KV \sin^2(\delta)
\]
4. Ferromagnetic Nanoparticles and SuperCrystals in the Non-Interacting Limit

4.1.2 Isotropic Paramagnet

First, we will ignore the second energy and try to recover the both analytically and experimentally well-known result for the magnetization curve of such a paramagnet in an external field. In nature, the vanishing of any magnetocrystalline anisotropy is not usual. The best analogy would be a material where the saturation magnetization is rather high compared to the anisotropy constant.

We define the total energy of the system being given as

\[ E = -\mathbf{m} \cdot \mathbf{B} \]

where the modulus \( |\mathbf{m}| = m \) is constant and \( \mathbf{B} = B \mathbf{\hat{e}}_z \). When calculating the expectation value of

\[ m_z = m \cos \theta \]

in the canonical ensemble, we therefore have to consider all possible orientations of \( \mathbf{m} \) on \( S^2 \), the surface of the 3-sphere, according to:

\[ \langle m_z \rangle = \frac{1}{Z} \int_{S^2} m_z \exp \left( \frac{\mathbf{m} \cdot \mathbf{B}}{k_B T} \right) d^2 r \]

with \( Z = \int_{S^2} \exp \left( \frac{\mathbf{m} \cdot \mathbf{B}}{k_B T} \right) d^2 r \)

\[ \varphi \text{-Symmetry} \Rightarrow \langle m_z \rangle = m \int_0^\pi \frac{\exp \left( \frac{m B \cos \theta}{k_B T} \right) \sin \theta \cos \theta d\theta}{\int_0^\pi \exp \left( \frac{m B \cos \theta}{k_B T} \right) \sin \theta d\theta} \]

where we used standard spherical coordinates \( (r, \theta, \varphi) \) and \( z = r \cos \theta \). Defining:

\[ x := \frac{m B}{k_B T} \quad v := \cos \theta \]

we get the easily solvable

\[ \langle m_z \rangle = m \frac{\int_1^{-1} ve^{xv} dv}{\int_1^{-1} e^{xv} dv} \quad (4.1) \]

The magnetic moments are indistinguishable. We therefore get for the total magnetization \( M \) if \( n \) is the number density of magnetic moments, i.e. the number of magnetic moments, each with modulus \( m \), per Volume

\[ M(T, B) = n \langle m_z \rangle = M_{\text{max}} L \left( \frac{m B}{k_B T} \right) \quad (4.2) \]
with the solution of (4.1), the *Langevin function* $L$

$$L(x) = \frac{1}{\tanh(x)} - \frac{1}{x} = \frac{x}{3} + O(x^3)$$

and the saturation magnetization, read maximum achievable magnetization

$$M_{\text{max}} = n m$$

For small fields $B$, the magnetic susceptibility can be written as

$$\chi = \frac{M}{H} \sim \frac{\mu_0 M}{B} = \frac{n\mu_0 m^2}{3k_B T}$$

In particular, we get $\chi \propto 1/T$ which is known as Curie’s law and is an important hallmark of paramagnetic systems or their analogies. We can rewrite the Curie law as

$$\chi = \frac{C}{T}$$

with the Curie constant $C$ which we here expect to be

$$C = \frac{n\mu_0 m^2}{3k_B}$$

or, in case that we have particles of finite volume $V_{\text{part}}$ and saturation magnetization *per particle volume* $M_s$ :

$$M_{\text{max}} = n M_s V_{\text{part}}$$

$$C = \frac{n\mu_0 (M_s V_{\text{part}})^2}{3k_B}$$
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

4.1.3 Potential Landscape with Non-vanishing Anisotropy

We will now also include the magnetocrystalline anisotropy energy. It is important to note that we thereby have introduced at least two additional parameters: The magnitude of the anisotropy constant $K$ and the distribution of the easy axes $\mathbf{k}$ where $E_a$ is minimal if $\pm \mathbf{m} \parallel \mathbf{k}$.

Figure 14: Langevin function and small-field approximation

Figure 15: Potential landscape in spherical coordinates without external field. Energy in arbitrary units, following the given parameter set.
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

\( \mu_0 H \neq 0 \)

\( \varphi \)

\[ 0 \]

\[ -\pi \]

\[ \pi \]

\[ 0 \]

\[ \pi \]

\[ 14 \]

\[ 12 \]

\[ 10 \]

\[ 8 \]

\[ 6 \]

\[ 4 \]

\[ 2 \]

\[ 0 \]

\[ -2 \]

\[ -4 \]

Figure 16: Potential landscape in spherical coordinates with external field

Figures 15 and 16 show the potential landscape for one superspin orientation in one setting of parameters in spherical coordinates \((\theta, \varphi)\):

- \(KV = 5\)  \(mB = 1.5\) (arbitrary, non-physical units)
- \(\hat{B} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\) \(\hat{k} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}\)
- \(E = E(\theta, \varphi) = KV \{1 - \cos^2[\delta(\theta, \varphi)]\} - mB \cos \theta\)
- \(\cos[\delta(\theta, \varphi)] = \frac{1}{\sqrt{2}}(\sin \theta \cos \varphi - \cos \theta)\)

Figures 17 and 18 show profiles for specific choices of \(\varphi\). Note that any external field breaks the \(\pi\)-periodicity in \(\theta\) into a \(2\pi\)-periodicity.

Figure 17: Profile of potential in figures 15, 16 for \(\varphi = 0\)
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

4.1.4 ZFC-FC curves

A very common experimental procedure is recording the Zero-field-cooled (ZFC) and Field-cooling (FC) curves. The sample undergoes (at least partly) the following path and typically shows a magnetization curve depending on $T$ as given in figure 19.

![Figure 18: Profile of potential in figures 15, 16 for $\varphi = 3\pi/4$](image)

![Figure 19: Idealized $M(T)$ curve with arbitrary units. Path 2, 3 are the ZFC, FC branches](image)
4. Standard Temperature-Sweep Magnetometric Experiment

1. The system is randomly generated at a sufficiently large starting temperature with $\mu_0 H = 0$ and cooled down

2. System is heated with a constant $\mu_0 H$ applied (ZFC curve)

3. without change in the applied field, system is cooled again (FC curve)

4. at $H = 0$, is now heated again

The fourth branch is most often omitted whereas the first may not have a name but is still essential for branches 2 and 3.

One of the characteristic quantities of these magnetization curves is the Blocking temperature $T_B$ which can (roughly) be defined as the temperature where during the ZFC-curve the magnetization reaches its maximum. The term blocking refers to the fact that above $T_B$, the system is expected to behave like a paramagnet. Since the involved magnetic moments are the nanoparticle superspins, this behavior is referred to as Superparamagnetism.

4.1.5 ac-Susceptibility

In case of ideal monodisperse nanoparticles as we mostly consider them, the relaxation time of particles obeys the Néel formula

$$\tau \simeq \tau_0 \exp \left(\frac{\theta}{T}\right) \quad \frac{\theta}{T} = \frac{KV}{k_B T}$$

with volume $V$ and uniaxial anisotropy constant $K$

we furthermore recall the static susceptibility (Langevin susceptibility) $\chi_0$

$$\chi_0 = \frac{(M_s V)^2}{3 k_B T}$$

with saturation magnetization $M_s$. Then, for an applied ac field, one finds

$$M(t) = H_0 \left(\chi' \cos \omega t + \chi'' \sin \omega t\right)$$

This implies that the magnetization will try to follow the applied field but would do so with a phase lag.

We present two cases of an ac-susceptibility experiment:

1. Apply a saturating field onto the system and have it relax before applying the sinusoidal $H_{ac}$

$\rightarrow \chi_{SAT}$
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

2. Have the sample cooled in zero-field and then switch on $H_{ac}$

$\rightarrow \chi_{ZFC}$

In the first case, one assumes that after switching off the saturating field, any single particle will undergo the following exponential decay (Néel relaxation)

$$M(t) = M_{eq} e^{-t/\tau}$$

This leads to the following real and imaginary part of the susceptibility in the model of Néel relaxation after saturation [1]:

$$\chi_\text{SAT}'(\omega) = \chi_0 \frac{1}{1 + (\omega \tau)^2}$$

$$\chi_\text{SAT}''(\omega) = \chi_0 \frac{\omega \tau}{1 + (\omega \tau)^2}$$

The second case also exhibits Néel relaxation as basis for the magnetization dynamics. However, an important difference is that as soon as any field is applied to the zero-field cooled sample, a net magnetization occurs instantaneously (as described in 4.1.4) which then leads to a temperature-independent nonzero contribution to $\chi_{ZFC}$.

The real-time result for the ZFC susceptibility $\chi$ becomes [1]

$$\chi_{ZFC}(t) = \frac{M_s^2}{3K} \left[ 1 + \frac{KV}{k_B T} \frac{1}{1 + e^{-t/\tau}} \right]$$

and a Fourier transformation yields

$$\chi_{ZFC}'(\omega) = \frac{M_s^2}{3K} \left[ 1 + \frac{KV}{k_B T} \frac{1}{1 + (\omega \tau)^2} \right] \quad (4.7)$$

$$\chi_{ZFC}''(\omega) = \frac{M_s^2}{3} \frac{V}{k_B T} \frac{\omega \tau}{1 + (\omega \tau)^2} \quad (4.8)$$

Only simulation results for the ZFC-ac susceptibility and the first setup will be presented. In future research however, the alternative setup may be of particular interest if samples with non-vanishing interactions are considered. Via eliminating $\omega t$ we obtain the following relationship between $\chi', \chi''$:

$$\left( \chi' - \frac{\alpha(2 + \sigma)}{2} \right)^2 + \chi''^2 = \left( \frac{\alpha \sigma}{2} \right)^2$$ \quad (4.9)

where

$$\alpha \equiv \frac{M_s^2}{3K} \quad \sigma \equiv \frac{KV}{k_B T}$$
we obtain the formula of a semi-circle.

This means that the so-called Cole-Cole plot of real vs. imaginary part of $\chi_{ac}$ is a perfect semi-circle in the first quadrant with its center coordinates

$$(c, 0) \equiv (\alpha(2 + \sigma)/2, 0)$$

and radius

$$r = \alpha\sigma/2$$

Furthermore, both the anisotropy constant and saturation magnetization can be extracted from such a circle plot

$$KV = 2k_B T \frac{r}{c - r}$$
$$M_s = \sqrt{\frac{k_B T}{V}} \sqrt{r}$$

4.2 Simulation results

Throughout the following chapters, we will almost exclusively consider *maghemite* ($\gamma$-Fe$_2$O$_3$) [2] nanoparticles with uniformly distributed easy axes distribution and realistic parameters as they are given in nature.

4.2.1 Isotropic Superparamagnetism

We study the limit of vanishing magnetocrystalline anisotropy for magnetic nanoparticles and aim to recover the behavior of an isotropic paramagnet.

To this end, at constant temperatures several magnetic fields are applied to the nanoparticles and the magnetization is recorded. Results are given in figure 21.
4. **FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT**

![Simulation results for the isotropic paramagnet and the fit of the Langevin expectation](image)

Figure 21: Simulation results for the isotropic paramagnet and the fit of the Langevin expectation

Fitting the theoretically expected Langevin function lets us recover the saturation magnetization that we implemented perfectly. We have therefore shown that our Monte-Carlo approach is in accordance with both theory and experiment.

4.2.2 **ZFC-FC Curves**

Several magnetometric simulations were carried out where appropriate temperature intervals were swept in the presence of static magnetic fields.

4.2.2.1 **High-Temperature Behavior**

We are interested in the behavior of the ZFC curve well above \( T_B \) where theory predicts the sample to behave like an isotropic paramagnet.

In particular, one expects the magnetization to follow a Curie-Weiss law

\[
M(T) = M_\infty + \frac{C}{T - T_C}
\]

with both \( M_\infty = 0 \) and \( T_C = 0 \) because anything other would indicate a non-vanishing permanent magnetization after the zero-field cooling or (anti-)ferromagnetic behavior. Also, the Curie constant \( C \) can be compared to equation (4.6)

There is nothing in our model allowing for either. Specifically, any \( T_C \neq 0 \) would indicate interactions among our superspins that would have to originate from errors in the implementation or a poor quality in the utilized pseudo-RNG.
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

We find that our computer experiment is in accordance with the expectation.

4.2.3 Influences on the Blocking Temperature

When doing a simulation, we still have to specify the following parameters:

- number of particles
- modulus of the test vector (see figure 6)
- number of Monte-Carlo steps (distributed among relaxation- and averaging-loops) per measurement point
- magnitude and direction of the applied magnetic field $\mu_0 H$
- distribution and magnitude of uniaxial anisotropy constants

Each of these parameters may influence the observed blocking temperature in our computer experiment. (As it turns out, all of the above do so, at least in the case of interacting, periodic systems which are discussed later).

4.2.3.1 Particle Number

Figure 23 shows simulations of the ZFC-curve for a different number of superspins in the sample but otherwise same parameters:

- Test vector modulus $|d_m| = 1$
- Monte-Carlo steps per point: $2000^{(\text{relaxation})} + 10^{(\text{averaging})}$
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

- applied field $\mu_0 H = 15 \text{ mT}$

![ZFC curves with varying number of particles](image)

Figure 23: ZFC curves with varying number of particles

We see that for low $N$, the statistical quality of the simulation is poor since for example no $T_B$ can be extracted well. Other than that, we do not observe different dynamics for greater $N$, but less noisy data. This is to be expected because we do not consider any interaction between particles in this chapter.

4.2.3.2 Test Vector

Figure 24 shows simulations of the ZFC-curve for different test vectors moduli in the sample but otherwise same parameters:

- number of magnetic moments: 1000
- Monte-Carlo steps per point: $2000^{(\text{relaxation})} + 10^{(\text{averaging})}$
- applied field $\mu_0 H = 15 \text{ mT}$
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

This is probably the most involved parameter study in 4.2.2. The analysis of the component distribution for the transformed vectors as demonstrated in 3.4.2 shows for all \(d_m\) that were considered here, that after 2000 MCS the vectors would be uniformly distributed in the absence of any potential. Yet we see dramatically different behaviour in terms of observed \(T_B\). We see an increase of \(T_B\) for decreasing \(d_m\), which is intuitive because a smaller \(d_m\) corresponds to a slower dynamic of any individual nanoparticle whereas the speed of temperature change is constant in all 11 plots shown. Especially the non-linear behaviour around \(d_m = 1\) however is only plausible if we recall this geometric insight: For this test-vector length, a jump 'from one pole to the equator' is barely possible. This is precisely the distance between the minima of the Stoner-Wohlfarth model. This means that for \(d_m > 1\), a jump from the energetically unfavourable equilibrium state to the favourable one is possible (though still not very likely) within one Monte-Carlo update, which seems counterintuitive for systems at low temperature. For smaller \(d_m\) where this quick channel is completely forbidden, which leads to significantly slower dynamics.

The geometric significance of the test-vector length must thus be taken into account when designing simulation parameters for other energy landscapes. A balance must be found between the economy of wasting too much computation time in uneventful parts of phase space (\(\leftrightarrow d_m\) chosen too low) and the phenomenon of spins leapfrogging potential landscapes at low temperatures (\(\leftrightarrow d_m\) chosen too big).

4.2.3.3 Number of Monte-Carlo Steps per Measurement
Figure 25 shows simulations of the ZFC-curve for different numbers of relaxation-Monte-Carlo steps per measurement point but otherwise same parameters, including averaging Monte-Carlo steps:
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

- number of magnetic moments: 1000
- Test vector modulus $|d_m| = 1$
- Monte-Carlo averaging steps per point: 10
- applied field $\mu_0 H = 15$ mT

![Figure 25: ZFC curves with varying number of Monte-Carlo steps (MCS)](image)

In contrast to the previous case, the number of MCS at fixed test-vector length is a well-behaving parameter. An increase of MCS yields lower $T_B$ which is the expected behaviour because any magnetic moment is allowed more time per temperature step to find its equilibrium. However, an increase of MCS leads to a convergent series of possible observations, precisely because our algorithm is based on Markov-Chains. In contrast to the previous example, the parameter change here does not constitute a different physical system being studied. For higher MCS, we enhance the statistical quality of our simulation.

4.2.3.4 Applied Magnetic Field

Figure 26 shows simulations of the ZFC-curve for different applied magnetic fields after cooling in zero-field, but otherwise same parameters.

- number of magnetic moments: 1000
- Test vector modulus $|d_m| = 1$
- Monte-Carlo steps per point: $2000^{(\text{relaxation})} + 10^{(\text{averaging})}$
The resulting series of ZFC curves shows, as expected, that $T_B$ decreases when the applied field is increased.

4.2.4 Hysteresis Plots

Alternatively to cooling and heating the sample at discrete applied magnetic fields, it is insightful to apply varying external fields at constant temperature on the sample after it has been cooled down at zero external field.

In the limit of vanishing anisotropy constants, this procedure will reproduce our previous results for the isotropic paramagnet. For the more realistic case of finite $KV$, we will encounter the well-known phenomenon of open hystereses.
Comparing hysteresis loops at different temperatures will give a relationship between coercivity and temperature. The coercive field is defined as the strength of the applied magnetic field required to reduce the magnetization of the sample after it had been driven to saturation.

Figure 27: Temperature dependence of observed coercivity. The hysteresis is symmetric as shown by the overlapping of the curves.

This behavior is also in accordance with experimental results. However, a theory of the precise analytical nature of the function $H_c = H_c(T)$ would have to involve the nature of modeled/implemented relaxation times $\tau$ as discussed in the context of Neél-Brown theory.

Another (technical) aspect is that the presented simulation results were achievable with far lesser computation time that comparable T-sweep simulations but with same statistical quality. Quite often it is observed that Metropolis-algorithm simulations at constant temperature yield much faster convergence.

Unfortunately, in the context of systems that feature frustration, such as most realistic magnetic systems do, experiments where the temperature is not held constant offer more insight.

4.2.5 ac-Susceptibility and Cole-Cole Plot

Finally, before moving on to interacting systems, we cover a third kind of sweep through parameter space. We want to investigate the AC-, or complex susceptibility $\chi_{ac}$ in the environment of a sinusoidal applied field at frequency $\omega$

$$\chi_{ac}(\omega) = \chi'(\omega) - i\chi''(\omega) \quad \chi', \chi'' \in \mathbb{R}$$
4. FERROMAGNETIC NANOPARTICLES AND SUPERCRYSTALS IN THE NON-INTERACTING LIMIT

Studying an explicitly time-dependent quantity is interesting from a technical point of view because we will directly make use of the interpretation of '1 Monte-Carlo step' as representative of a finite if extremely small period of time.

\[
\mathbf{H}_{\text{ext}} \equiv \mathbf{H}(t_{MC}) = H_0 \sin(\omega t_{MC})
\]

is the physical field we want to simulate. We introduced a 'Monte-Carlo time' \( t_{MC} \) which we can implement in a straightforward fashion in a computer experiment by a certain number of Monte-Carlo steps. For \( \chi', \chi'' \) we then have

\[
\chi'(\omega)H_0 = \frac{1}{N_{MC}} \sum_{t_{MC}=1}^{N_{MC}} M(t_{MC}) \sin(\omega t_{MC})
\]

\[
\chi''(\omega)H_0 = \frac{1}{N_{MC}} \sum_{t_{MC}=1}^{N_{MC}} M(t_{MC}) \cos(\omega t_{MC})
\]

Throughout this thesis we always consider magnetic samples that are considered linear and isotropic media. Thus, we can omit the vectors and \( \chi \) is scalar instead of a tensor of degree \( \geq 2 \).

As one can plainly see from the definition above, high frequencies are implemented via a larger number of MC steps. This approach works fine if the computational effort behind one MCS is small, but it will be a great obstacle when treating interacting systems.

Figure 28 shows the results for the temperature dependence of the real and imaginary part of \( \chi \) at fixed frequency.

![Figure 28: Real and imaginary part of the ac-susceptibility at \( N_{MCS} = 10^6 \)](image-url)
A frequency sweep at fixed temperature, conversely yields the Cole-Cole plot in Fig. 29:

![Cole-Cole plot with fitting of a circle function from (4.9)](image)

Figure 29: Cole-Cole plot and with fitting of a circle function from (4.9)

We find that our result confirms the expectation of a non-interacting system because a perfect semi-circle is observed. In contrast, an interacting system would show a strongly flattened circle.

### 4.3 Summary and Discussion of Results

Non-interacting systems have been comprehensively studied and we have collected sufficient evidence that our simulation approach is correctly implemented. We have found that simulation parameters such as the number of MCS per measurement have significant influence on any quantifiable observation.
5. Ferromagnetic Nanoparticles with Dipole-Dipole Interaction

From now on, we will consider nanoparticles forming supercrystals at realistic length scales ($10^{-8}$ to $10^{-7}$ m). Here, it is no longer a realistic assumption to consider the nanoparticles as non-interacting. Instead, our model of supercrystals will include magnetic dipole-dipole interactions as additional energy contribution.

Theoretical aspects of this model are discussed first, especially computational challenges arising from any 2-particle interaction and the comparatively sparse theoretical knowledge regarding 3D models with a rather complicated interaction like the dipole-dipole interaction.

Motivated by this, we discuss widely used approximative approaches towards these models and re-introduce a method originally used in molecular physics in order to apply a customized version for our models.

5.1 Theory II

As noted, the only difference compared to the previous section is the introduction of realistic 2-particle interactions in our models. However, this small change has major implications for code efficiency and other aspects which were rather simple in the case of non-interacting nanoparticles.

5.1.1 Computational Aspects

Recalling the Metropolis-Hastings algorithm from (3.6) we see that an obvious bottleneck for code efficiency is the estimation of the total energy change of the system. This effect is far larger in case of 2-particle interactions because it will imply a $\mathcal{O}(N^2)$-scaling with particle number, or correspondingly $\mathcal{O}(L^6)$-scaling with edge length $L$.

On the other hand, we note that this energy estimation is entirely deterministic. This is important because we can then at least utilize multi-threaded computations to speed this step up. This will be done and discussed in the sections which do not employ different approximative theories. We note however that, purely from a computer efficiency point of view, it will always pay off to reduce the amount of 2-particle interaction computations.

5.1.2 Mermin-Wagner Theorem

Another challenge in this thesis is the absence of comprehensive theoretical knowledge what a dipolar 3D system is expected to behave like. Although it is not the focus of this research, we recall an important result for sys-
tems of lower dimensionality, the Mermin-Wagner theorem [14]:

**Mermin-Wagner Theorem**

In 1D and 2D systems with sufficiently short-range interactions, no continuous symmetry can be broken spontaneously. This means that any thermal fluctuation offers sufficient perturbation destroying a possible ordering.

This result rigorously only applies to an isotropic Heisenberg ferromagnet, but generalizations towards various many-body systems exist [6]. This possesses rotational symmetry so that all the spin directions can be globally rotated without any additional energy cost. This means that long wavelength excitations, in which the spin state may deviate from its ground state value over a considerable distance, cost very little energy. Thus a fluctuation of the spins can be excited with very little energy cost. In one and two dimensions, they destroy the long range order. If, however, there is significant anisotropy there will be an energy cost associated with rotating the spins from their ground state value.

It turns out that the anisotropy energy penalty incurred by allowing these fluctuations increases with the square of $R$, the radius of the excitation, and hence the anisotropy energy will suppress all but the smallest of these non-linear fluctuations. It is the presence of such symmetry breaking fields which can stabilize long range order in two-dimensional systems. There is also a dipolar interaction between spins in real systems which, although much weaker than the exchange interaction, is anisotropic and can act in a similar way to suppress the growth of fluctuations.

### 5.1.3 Antiferromagnetic Part of the Dipole-Dipole Interaction

When writing down the full dipole-dipole energy between two magnetic moments $\mathbf{m}_1, \mathbf{m}_2$

$$E_{\text{dip}} = \mu_0 \frac{4\pi r^3}{3} [\mathbf{m}_1 \cdot \mathbf{m}_2 - 3(\mathbf{m}_1 \cdot \hat{r})(\mathbf{m}_2 \cdot \hat{r})]$$

$$= \mu_0 \frac{4\pi r^3}{3} \left(\mathbf{m}_1 \cdot \mathbf{m}_2 - 3\frac{\mu_0}{4\pi r^3} (\mathbf{m}_1 \cdot \hat{r})(\mathbf{m}_2 \cdot \hat{r})\right)$$

we can identify two contributions which we call the *antiferromagnetic* part $E_{\text{afm}}$ and residual

$$\tilde{E}_{\text{dip}} \equiv E_{\text{dip}} - E_{\text{afm}}$$

We can now imagine a hypothetical lattice model where the Hamilton function or Hamiltonian is completely defined by the 2-particle interaction given by $E_{\text{afm}} = \mu_0 \frac{4\pi r^3}{3} \left(\mathbf{m}_1 \cdot \mathbf{m}_2 - 3\frac{\mu_0}{4\pi r^3} (\mathbf{m}_1 \cdot \hat{r})(\mathbf{m}_2 \cdot \hat{r})\right)$.
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

\( \frac{\mu_0}{4\pi r^3} (\mathbf{m}_1 \cdot \mathbf{m}_2) \). We observe that this resembles the conventional Heisenberg model Hamiltonian

\[
\mathcal{H}_{\text{Heis}} = -\sum_{i,j} J_{ij} \mathbf{S}_1 \cdot \mathbf{S}_2
\]

where \( J_{ij} \) is the exchange integral between two spins \( \mathbf{S}_1, \mathbf{S}_2 \). The analogy we propose is therefore

\[
\begin{align*}
E_{\text{afm}} & \leftrightarrow \mathcal{H}_{\text{Heis}} \\
\mathbf{m}_i & \leftrightarrow \mathbf{S}_i \\
\frac{\mu_0}{4\pi r^3} & \leftrightarrow -J_{ij}
\end{align*}
\]

Most importantly, we see that our 'exchange integral'

\[
-\frac{\mu_0}{4\pi r^3} \simeq J_{ij} < 0 \quad \forall \text{ pairs } (i,j)
\]

which implies that antiferromagnetism is the expected type of collective magnetism if the system has the appropriate translational symmetry.

5.1.4 Finite and Infinite Systems

It is possible to study finite systems non-approximatively. By this we mean that one can in principal evaluate the complete energy of any interacting system as long as the number of considered magnetic moments is finite. Computational constraints will however restrict such simulations in terms of maximum number of moments in the system:

As discussed in section 5.1.1, the computational effort necessary to exactly calculate the necessary energy terms for each MCS scales with \( O(N^2) \). The simulation of one measurement (i.e. one instance of the parameters \( T \) and \( \mu_0 H_{\text{ext}} \)) for a system of \( N \sim 10^3 \) particles may take as long as 1 hour on machines that are available to me. Realistically, any obtainable sample of self-assembled nanoparticles in a superlattice will consist of no fewer than \( N \sim 10^9 \) contributing superspins which will already imply an estimated computation time factor of \( 10^{12} \). Therefore, what took 1 h before, would require several orders of magnitude longer.

This estimate ignores the fact that in the implementation used in this thesis (see appendix B.1.4) invariant parameters like site distances \( d \) and displacement vectors \( \hat{r} \) are calculated once and kept in working storage. For \( N \sim 10^9 \), this requires \( \sim 100 \) MB of RAM in the implementation given in the appendix. This could of course be cut down drastically if one sacrifices the flexibility of our code in terms of which superlattices can be put in, but it will still require \( \sim 10^3 \) KB of RAM and the same multiplication factor as before yields \( \sim 10^9 \) GB of required RAM. One could of
course perform simulations without putting these recurring parameters into storage, but this will in turn slow down the performance even further. Because of these harsh limitations, we will focus on ensembles of \( N < 10^5 \) superspins. These are either considered

1. finite systems where surface effects will prohibit any meaningful insight into bulk behavior
2. supercells that are continued indefinitely with periodic boundary conditions in order to simulate bulk behavior

This thesis puts more emphasis on the second branch. Additional assumptions and approximations are required to effectively study interacting systems to a similar degree as was done for non-interacting systems in section 4.2.

5.1.5 Possible Approximations

We present a selection of possible approaches to simulate interacting lattice models effectively. We focus on methods that are still within the Metropolis scheme (3.6). This means that we focus on approximations that only affect the calculation of energies.

- **Approximations for finite systems**
  - Cut-off

- **Approximations for infinite systems with PBC**
  - Ewald method
  - Cut-off
  - Cut-off + Mean Field (Onsager)

The idea of introducing a cut-off length beyond which interactions are neglected is that the loss in precision of the calculated energy is far outweighed by the gain in computation efficiency.

5.1.6 Onsager Reaction Field Method

We will now cover in detail the adaptation of a mean-field theory applied mostly in molecular physics, for the purpose of getting reasonable energy estimations. It will offer a vast reduction of the needed computation time. A derivation from a general idea which implies a classical magnetostatic problem will be presented.
The mathematical problem itself is exactly solvable and will thus be presented thoroughly. Emerging parameters and their physical significance will be discussed. Finally, both the achieved accuracy in simulations and the performance from an efficiency standpoint will be discussed.

5.1.6.1 Motivation and Model

In molecular physics, research on electric dipolar particles in fluids is an important topic with a long history. With electric dipolar fluids it is important to develop approximate methods to estimate energies stemming from two-particle interaction such as the electric dipole energy, similar to our simulation studies. The dipole-dipole interaction has the same form whether one talks about the energy of electric permanent dipoles $\mathbf{p}$ within an external electric field $\mathbf{E}$ or magnetic dipoles $\mathbf{m}$ in an applied $\mu_0\mathbf{H}$. Therefore, it appears reasonable to take previous work on approximative methods there as motivation for our work on magnetostatics.

Starting from the complete model with long-range interactions, the first step in the Onsager-Reaction-field method [16] is to introduce a cutoff radius $R$ which limits the spatial extension of any electric field $\mathbf{E}_d$ produced by an electric dipole

$$\mathbf{E}_d(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \frac{1}{r^3} (3(\mathbf{p} \cdot \hat{r})\hat{r} - \mathbf{p}) \quad \Rightarrow \quad \mathbf{E}_d^{\text{ORF}}(\mathbf{r}) = \Theta(R - r) \mathbf{E}_d(\mathbf{r}) \quad (5.3)$$

with the Heaviside function $\Theta$

$$\Theta(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 1 \end{cases}$$

In the context of computer simulations, this is already the most important step of the entire approximation because it implies that only a fraction of interaction terms have to be computed with $\mathbf{E}_d^{\text{ORF}}$ instead of $\mathbf{E}_d$.

On the other hand, taking this new model-interaction on its own is obviously too naive because one would quickly come to the conclusion that there were no qualitative difference between bulk behavior or the behavior of a tiny set of interacting dipoles.

The 2nd part of the ORF-method therefore provides an idea how to model the effect of the environment outside the cut-off sphere on the dipole in the center. The idea is to model these magnetic moments as a continuous, linear, isotropic, polarized medium and imagine it reacting towards a superdipole which sits at the exact center of the sphere which otherwise is comprised of vacuum. This medium will have a relative permittivity\(^3\) $\epsilon_r > 1$ which is a real scalar and is a quantity which can also be calculated when measuring the total polarization of the entire set of electric

\(^3\)More generally, $\epsilon_r \neq 1$ would suffice, but metamaterials with negative susceptibility $\chi_e \equiv \epsilon_r - 1$ are rarely considered.
dipoles in an external electric field.

From there, classical electrostatics will give an expectation for the reaction field $E_{RF}$, an additional electric field which is the response of the surrounding medium towards the presence of the dipole.

As stated, the magnetic field $B_d$ of a magnetic dipole has the same form as in the electric case, which is why we can just propose the following $B_d^{ORF}$ providing a cut-off for the magnetic field analogously to (5.3)

$$B_d(r) = \frac{\mu_0}{4\pi r^3} (3(m \cdot \hat{r})\hat{r} - m) \quad \Rightarrow \quad B_d^{ORF}(r) = \Theta(R - r) \cdot B_d(r) \quad (5.4)$$

As in the original case of electric dipolar molecules, we now have to make an estimate for any environment of the magnetic dipole beyond the cut-off radius $R$: We imagine a continuous, linear, isotropic, magnetized medium with the relative permeability $\mu_r > 1$ surrounding the cut-off sphere which is comprised of vacuum ($\mu_r = 1$) and a magnetic super-dipole at its center. This super-dipole again is the direct sum of all magnetic moments contained in the cut-off sphere. The model is illustrated in figures 30 and 31.

### 5.1.6.2 Solution of the Magnetostatic Problem

Vector calculus states that any curl-free vector field is the gradient of a scalar field\(^4\).

\(^4\)Provided that mathematical conditions like continuity, differentiability and correct asymptotic behavior for $r \to \infty$ are met.

Figure 30: First step of the ORF method: Introduce cut-off $R$ beyond which dipole-dipole interactions are not considered
This means
\[ \nabla \times \mathbf{E} = 0 \quad \Rightarrow \quad \exists \, \phi(\mathbf{r}) : -\nabla \phi = \mathbf{E} \quad (5.5) \]

Therefore, the solution of boundary problems for any stationary electric field \( \mathbf{E} \) and its potential \( \phi \) correspond to a given charge density \( \rho_e = \rho_e(\mathbf{r}) \) via
\[
\begin{align*}
\nabla \phi &= -\mathbf{E} \\
\text{div } \mathbf{E} &= \frac{\rho_e}{\epsilon_0} \\
\Rightarrow \quad \Delta \phi &= -\frac{\rho_e}{\epsilon_0}
\end{align*}
\]
which is the well-known Poisson equation. Any \( \phi(\mathbf{r}) \) that solves the Poisson equation and also satisfies boundary conditions given by materials etc. is the unique solution to such a problem.

Finding a solution to (5.6) is relatively easy if the problem has any symmetries. Because of the Maxwell equation \( \nabla \cdot \mathbf{B} = 0 \) there is no analogous scalar potential for magnetic fields. Instead, one has a not easily found vector potential \( \mathbf{A} \) such that \( \mathbf{B} = \nabla \times \mathbf{A} \). This is given via the Biot-Savart law
\[
\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi} \int_V \frac{\mathbf{j} \, dV \times \mathbf{r}'}{|\mathbf{r}'|^3}
\]
but cannot be evaluated easily since we first would have to find the correct static current density \( \mathbf{j} \) which represents our permanent magnetic dipole \( \mathbf{m} \).
It is therefore not immediately obvious that the derivation of an appropriate reaction field $B_{RF}$ will be as easy as in the dielectric case.

However, we realize that there is no free current density in our problem, only bound currents which induce magnetization, i.e. a dipole-moment density. This will facilitate the mathematics immensely in the following way:

We use the auxiliary field $H$ and a decomposition of the total charge current $j$ into bound and free part:

$$H = \frac{1}{\mu_0} B - M$$
$$j = j_{\text{free}} + j_{\text{bound}}$$
$$j_{\text{bound}} = \nabla \times M$$

We study the derivatives of $H$:

$$\nabla \cdot H = \frac{1}{\mu_0} \nabla \cdot B - \nabla \cdot M = -\nabla \cdot M$$
$$\nabla \times H = \frac{1}{\mu_0} \nabla \times B - \nabla \times M = j - j_{\text{bound}} = j_{\text{free}}$$

Since there is no free current, we have found analogously to (5.5)

$$\nabla \times H = 0 \Rightarrow \exists W(\mathbf{r}) : -\nabla W = H$$

Combining (5.11) and (5.10) we have found the

Poisson equation of the magnetic potential

$$\Delta W = \nabla \cdot M$$

just like (5.6) for electric fields. Now we can also make explicit use of our assumption that the outside medium is linear and isotropic:

$$M_i^{\text{linear}} = \sum_{j=1}^{3} \chi_{ij}^{(m)} H_j + O(H^2)$$
$$\chi_{ij}^{(m)} \text{ isotropic} = \chi_{m} \delta_{ij}$$
$$\Rightarrow M = \chi_{m} H$$
Most importantly for our calculations below, this implies that $\mathbf{B}$ and $\mathbf{H}$ are related to each other simply by

$$
\mathbf{B} \equiv \mu_0(\mathbf{H} + \mathbf{M}) = \mu_0\mathbf{H} + \mu_0\chi_m\mathbf{H} = \mu\mathbf{H} \quad (5.13)
$$

where

$$
\mu = \mu_0(1 + \chi_m) \equiv \mu_0 \mu_r \quad (5.14)
$$

If we assume that the surrounding medium with magnetization $M$ is 'roughly paramagnetic'

$$
1 < \mu_r \lesssim 100
$$

this assumption seems reasonable.

Very importantly, we hereby have a $\mathbf{H}_{ext}$ which is not constant throughout $\mathbb{R}^3$ like $\mathbf{B}_{ext}$ but piecewise defined as

$$
\mathbf{H}_{ext} = \mathbf{B}_{ext} \left\{ \begin{array}{ll}
\frac{1}{\mu_0} & r < R \\
\frac{1}{\mu_0 \mu_r} & r > R
\end{array} \right. \quad (5.15)
$$

In this very special case of magnetostatic problems, the boundary conditions given by the Maxwell equations look completely analogous to the electrostatic case via

$$
W \leftrightarrow \phi \\
\mathbf{H} \leftrightarrow \mathbf{E} \\
\mu_r \leftrightarrow \epsilon_r
$$

Finally, we arrive at the following set of differential equations and boundary conditions.

**Boundary conditions for the magnetic Poisson equation**

$$
W(r \to 0) \to \frac{\mathbf{m} \cdot \mathbf{r}}{4\pi r^3} \\
W(r \to \infty) \to -\mathbf{H}_{ext} \cdot \mathbf{r} \\
W(r = R) = \text{continuous} \\
\mathbf{H}_{in}^\perp = \mu_r \mathbf{H}_{out}^\perp \\
\mathbf{H}_{in}^\parallel = \mathbf{H}_{out}^\parallel
$$
Again, note that $\mathbf{m}$ now represents the total sum of dipole moments within $V_c$:

$$ m \equiv \sum_{V_c} m_i \tag{5.17} $$

With (5.16), the problem is unfortunately very unsymmetric because the angle between $\mathbf{m}$ and $\mathbf{H}_{\text{ext}}$ will take arbitrary values.

In order to be able to use techniques for problems with azimuthal boundary conditions, we split the problem in two parts where either the cumulated magnetic moment $\mathbf{m}$ or the external field $\mathbf{B}_{\text{ext}} = \mu \mathbf{H}_{\text{ext}}$ are a symmetry axis:

$$ W^I (r \to 0) \to \frac{\mathbf{m} \cdot \mathbf{r}}{4\pi r^3} $$

$$ W^I (r \to \infty) \to 0 $$

$$ W^I (r = R) = \text{continuous} $$

$$ H^I_{\text{in}} = \mu r H^I_{\text{out}} $$

$$ H^I_{\parallel} = H^I_{\text{out}} $$

$$ W^II (r \to 0) \to 0 $$

$$ W^II (r \to \infty) \to -\mathbf{H}_{\text{ext}} \cdot \mathbf{r} $$

$$ W^II (r = R) = \text{continuous} $$

$$ H^II_{\text{in}} = \mu r H^II_{\text{out}} $$

$$ H^II_{\parallel} = H^II_{\text{out}} $$

The explicit derivation makes use of techniques to solve the homogeneous Laplace equation

$$ \nabla^2 W = 0 $$

and is given in the appendix A.1.

The coordinate-free representation for the solution of the two problems is:

$$ W^I_{\text{in}} (\mathbf{r}) = \frac{\mathbf{m} \cdot \mathbf{r}}{4\pi} \left( \frac{1}{r^3} + \frac{1 - \mu_r}{R^3} \frac{1}{1 + 2\mu_r} \right) $$

$$ W^I_{\text{out}} (\mathbf{r}) = \frac{\mathbf{m} \cdot \mathbf{r}}{4\pi r^3} \frac{3}{1 + 2\mu_r} $$

$$ W^II_{\text{in}} (\mathbf{r}) = -\mathbf{H}_{\text{ext}} \cdot \mathbf{r} \frac{3\mu_r}{1 + 2\mu_r} $$

$$ W^II_{\text{out}} (\mathbf{r}) = \mathbf{H}_{\text{ext}} \cdot \mathbf{r} \left( \frac{R^3}{r^3} \frac{1 - \mu_r}{1 + 2\mu_r} - 1 \right) $$
and direct summation yields the solution of the original problem:

$$W(r) = \begin{cases} m \cdot r \frac{1}{4\pi} \left( \frac{1}{r^3} + \frac{1}{r^3} \frac{1 - \mu_r}{1 + 2\mu_r} \right) - H_{ext} \cdot r \frac{3\mu_r}{1 + 2\mu_r} & r < R \\ m \cdot r \frac{1}{4\pi r^3} \frac{3}{1 + 2\mu_r} + H_{ext} \cdot r \left( \frac{R^3}{r^3} \frac{1 - \mu_r}{1 + 2\mu_r} - 1 \right) & r > R \end{cases}$$

(5.18)

We can now identify the so-called reaction field that has a finite value at $r = 0$:

$$H_{RF} = - \left( m \frac{1 - \mu_r}{2\pi R^3 (1 + 2\mu_r)} - H_{ext} \frac{3\mu_r}{1 + 2\mu_r} \right)$$

(5.19)

Introducing $\mu_r$-depending constants

$$\gamma_m := \frac{1 - \mu_r}{1 + 2\mu_r}, \quad \gamma_h := -\frac{3\mu_r}{1 + 2\mu_r}$$

(5.20)

our estimate for the energy of one superspin $s$ due to the external field and the dipolar environment is therefore:

$$E_{RF} = - s \cdot B_{RF} = -\mu_0 s \cdot H_{RF}$$

$$E_{RF} = s \cdot \left( m \frac{\mu_0}{2\pi R^3} \frac{\gamma_m}{1 + 2\mu_r} + H_{ext} \frac{\mu_0}{\mu_0} \gamma_h \right)$$

$$E_{RF} = s \cdot \left( m \frac{\mu_0}{2\pi R^3} \frac{1 - \mu_r}{1 + 2\mu_r} - B_{ext} \frac{3\mu_r}{1 + 2\mu_r} \right)$$

(5.21)

Both the $m$- and $B_{ext}$ term have properties that should be discussed further.

We calculate $\mu_r$ via

$$\mu_r = 1 + \chi_m = 1 + \frac{M}{H_{ext}} = 1 + \frac{\mu_0 M}{B_{ext}}$$

(5.22)

where $M = |M|$ is the magnetization, i.e. the total magnetic dipole density which is assumed to be linearly dependent of $B_{ext} = |B_{ext}|$ throughout our approximation.

### 5.1.6.3 Modelling parameters

As a general rule, the reaction field encourages configurations within the cut-off sphere with large cumulative magnetic moment via a parallel auxiliary field because

$$-1/2 < \frac{1 - \mu_r}{1 + 2\mu_r} < 0 \quad \mu_r > 1$$

(5.23)

This term therefore indeed deserves the name ‘reaction field’ because this auxiliary field is the response of the imagined medium towards any net magnetic moment in the sphere.

The second term means that applied magnetic field acting on any single magnetic
moment is amplified because of the surrounding medium since

\[-\frac{3}{2} < \frac{-3\mu_r}{1 + 2\mu_r} < -1 \quad \mu_r > 1 \tag{5.24}\]

At first look, this seems like the Onsager approach leads to inconsistencies in the limit of non-interacting systems. This is ultimately not the case, as shown in 5.1.6.6.

### 5.1.6.4 Cut-off Radius

The radius \( R \) largely determines the computational effort that an energy estimate with this approximation still requires. Intuitively one would expect that a smaller cutoff radius implies an increase of the necessary correction, which is indeed the case. The fact that the correction is inversely proportional to the volume of the cut-off sphere means that it is also inversely proportional to the number of superspins that are treated exactly within our energy-estimation scheme which seems reasonable.

Because we are always dealing with a discrete dipole distribution, we must however specify how \( R \) is implemented. There will always be an interval of possible \( R \) values whose corresponding cut-off spheres would contain the same magnetic moments. Without additional criteria, \( R \) could thus be chosen arbitrary which would however limit comparability between simulations.

We found that a consistent and reasonable estimate would be choosing the greatest possible \( R \) that does not span additional sites outside the cut-off sphere. This implies that no internal site touches the boundary of the sphere as seen in figure 32, and the reaction field takes on the lowest possible absolute value. In any case, \( R \) has to

![Figure 32: Choice of \( R \) for reaction field estimation. Blue crosses signify magnetic sites. Both the cyan and salmon colored circles contain the exact same sites. They represent the smallest and biggest possible choices for \( R \)](image)
be chosen consistently in order to have comparable simulation results since e.g. a 10% increase of $R$ already means that the $m$-factor is cut in half.

Simulations were mostly done on $fcc$-super lattices which have a very high degree of spatial symmetry so that the computation of the best choice for $R$ needs to be carried out only once for any choice of 'modelling' cut-off.

5.1.6.5 Determination of the relative permeability

The recipe (5.22) given above has the advantage of being a self-consistent procedure because each magnetic moment will be updated regularly during the simulation of one measurement point and all under the same conditions. This means that the chosen $\mu_r$ indeed becomes the best choice self-fulfillingly.

This however holds only true if the global susceptibility remains roughly constant throughout these measurements. This is obviously not true e.g. during the process of a ZFC-FC curve where the applied magnetic field has only one value, yet the magnetization is dependent on the recent history and temperature of the sample.

The hitherto implemented scheme thus suffers at least from temporal retardation, which becomes an issue if the goal of the simulation is to pin down an exact value for the blocking temperature.

Even worse, in the absence of applied fields the recipe (5.22) becomes ill-defined and one has to either retroactively impose values from e.g. high-temperature values in the ZFC-curve or impose a theoretical expectation like the Langevin value for paramagnets

$$\mu_r^{\text{Langevin}} = 1 + \chi^{\text{Langevin}} = 1 + n\mu_0 \frac{(M_s V)^2}{3k_B T}$$

which we only used as a very first guess. In subsequent simulations, the value was indeed changed to be the value found for the tail-end of ZFC curves which let our $\mu_r$ converge towards $\mu_r \sim 15$ in zero-field.

The exact value of $\mu_r$ would of course be an interesting property/result in itself, however its influence of our reaction field is limited. The two prefactors in (5.20) are shown in figure 33.
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 33: Dependence of correction factors on relative permeability. Only $\mu_r > 1$ (solid) applies in our simulations

The prefactors vary only very slightly for $\mu_r \gtrsim 10$, and the applied-field correction term does not even apply in zero-field. Therefore, we can at least have confidence that the precise value of our assumed $\mu_r$ is of less importance than e.g. the validity of the assumptions regarding the material in (5.13)

5.1.6.6 Consistency with non-interacting limit

It is expected that the influence of the dipole-dipole interaction on the sample diminishes quickly with an increase of the lattice constant of the supercrystal $a_0$. This will be reflected automatically for the interactions that are considered in an exact matter within $V_c$. The $m$-factor in the reaction-field term also reflects an increase of the lattice constant automatically because $R$ per design scales automatically with any change of the lattice constant. The $B_{ext}$-factor does not scale directly with the spatial dimensions of the sample. The scale of the system is instead only influential via $\mu_r$ because the magnetization $M$ in (5.22) is the dipole moment density which means that if only the lattice constant increases without adding further magnetic dipoles, we have for $\mu_r$, $\gamma_m$, $\gamma_h$

$$\begin{align*}
\mu_r & \rightarrow 1 \\
\gamma_m & = \frac{1-\mu_r}{1+2\mu_r} \rightarrow 0 \quad \text{for } a_0 \rightarrow \infty \\
\gamma_h & = \frac{3\mu_r}{1+2\mu_r} \rightarrow 1
\end{align*}$$

This means that the $m$-term vanishes even faster than $\propto 1/R^3$. $\gamma_h \rightarrow 1$ means that there is no amplification via the magnetic medium anymore, which is also to be expected.

In total, we arrive for large lattice constants at a vanishing reaction field so that
only the superspins within $V_c$ will interact via dipole-dipole interaction and the unaltered externally applied field. As shown in figure 34, we get a smooth transition of the ZFC/FC curves towards the non-interacting case as soon as $\frac{a_0}{a} \sim 5$ where $a_0$ is chosen such that for $a \leq a_0$ the behavior is significantly different due to 2-particle interactions: $a_0 \simeq 25$ nm.

This means that the behavior of our approximative theory is largely correct in the non-interacting limit.

To understand what happens at $a = a_0$, we calculate the dipole-dipole energy of one spin in the environment of $\geq 100$ randomly oriented neighbouring spins. This component of the total potential that is independent from all anisotropy axes is given in figure 35.

![Figure 34: Convergence of ORF simulations of the ZFC curve towards the non-interacting limit](image)

The relevant information of this is that there is always exactly one global minimum produced by the dipole fields. Even local minima are only rarely observed. If one adds this potential to the known Stoner-Wohlfarth results in 4.1.3, one observes in most cases that additionally to a change in position of the entire potential landscape, that the potential wall between the two local minima increases. Since the blocking temperature is predominantly associated with this potential wall, one can already assume that the dipole-dipole interaction will increase the blocking temperature.
Figure 35: Potential landscape for a superspin in a 8x8x8 magnetic supercell (fcc). Plotted are energy differences compared to the energy for orientation $\theta = \pi/2$, $\varphi = 0$. Any cut-off of more than 30 contributing neighbours leads to an image that is indistinguishable at this resolution.

5.2 Simulation Results

We now present simulation results obtained by studying systems with interacting nanoparticles.

5.2.1 Groundstates in Periodic, Dipolar 3D Systems

We discuss a topic that is motivated more from pure statistical physics point of view, than in the context of self-assembled magnetic nanoparticles. We consider 3D crystal systems whose total energy is exclusively determined by the magnetic dipole-dipole interaction of magnetic moments positioned on the lattice sites. The aim is an investigation of a purely dipolar ground state in certain lattice structures.

5.2.1.1 Results in the Limit of Vanishing Magnetocrystalline Anisotropy

We realize that the question above is equivalent to the physically not viable case of self-assembled magnetic nanoparticles in the limit of vanishing magnetocrystalline anisotropy at each lattice site. Simulations can be therefore carried out in a straightforward fashion with our existing methods. We consider systems that are initiated at very high temperatures and then cooled down with zero applied field. We pay particular attention to the energy of the system and discuss the superspin landscape at $T \to 0$.

With the help of the self-developed program which is documented in B.2, we will produce figures like figure 36. In all of these figures, all spins are grouped into
sublattices that in most cases span the magnetic super cell and require all of their member sites to be parallel aligned within a certain margin of error.

Figure 36: 8x8x8 magnetic super cell \((fcc)\) at \(T = 1\). Spin distribution of one start configuration

In this thesis, I mostly show instances where these sublattices have the peculiar property of being essentially 2-dimensional so that one can find a perspective that makes the spin structure readable by a projection on a plane, as seen in figure 37.
5.2.1.2 Series of 'Groundstates' Depending on Different Parameters in the Onsager Approximation

We present several low energy states and their spin structure that were obtained with varying spin initializations. Although different cut-off radii were employed to further increase the variety of simulated configurations, the following selection of low-energy structures were found independently from these respective choices.
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 38: 2 sublattices, inspecific view

Figure 39: Rotated view of figure 38
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 40: 4 sublattices, inspecific view

Figure 41: Rotated view of figure 40
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 42: 4 sublattices, inspecific view

Figure 43: Rotated view of figure 42
Figure 44: 4-6 sublattices as a more complex configuration, rotated view

Figure 45: 8 sublattices, inspecific view
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 46: Rotated view of figure 45, yz-plane

Figure 47: Rotated view of figure 45, xz-plane
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 48: Rotated view of figure 45, xy-plane

Figure 49: Temperature dependence of the magnetization components for the sample as shown in both figures 42 and 43. $M_x$, $M_y$, $M_z$ are defined in (5.25). By these 'components', the 2-dimensional character of the low-energy state corresponds to the vanishing of 'MZ'
It is useful to define the following quantities:

\[ M_\alpha := \sum_{i=1}^{N} |m_i^\alpha| \quad \alpha = x, y, z \]  

(5.25)

This means that e.g. \( M_x = 0 \) if the \( x \)-component of the magnetization of each site is zero and not just randomly distributed in \((-1, 1)\). Therefore, \( M_x \) measures if the \( x \)-component of the magnetization vanishes completely, and the system therefore does not have a 3-dimensional, but a layered 2-dimensional magnetization landscape.

Apart from this often occurring reduction of dimensionality, it has always been observed that an even number of antiparallel sublattices emerges at low temperatures. However, we cannot call this behaviour antiferromagnetic because no spontaneous symmetry breaking is observed. Conversely, this 'phase transition' happens steadily but slowly in an arbitrarily large temperature region.

### 5.2.1.3 Reconsidering Non-Vanishing Anisotropy

We connect the previous chapter back to the more realistic setting of magnetic nanoparticles such as maghemite nanoparticles. We discuss how stable the findings of chapter 5.2.1.2 are with regards to non-zero anisotropy energies.

In short, employing realistic parameters of the maghemite nanoparticles in order to study the number and character of sublattices at low temperatures yields a clear but disappointing result: No long range order can be detected, meaning that the random anisotropy changes the local potential landscapes too strongly.

On a more positive note, we can therefore study ZFC curves quite well with the help of the ORF method because it is even more viable if there is no (short-range) magnetic order at the temperatures we consider. Simulation results are given in figures 50 and 51. Because of the more complicated potential landscape, it is necessary to make the simulations with a smaller test-vector length than in the non-interacting case. Otherwise, local potential minima maybe insufficiently scanned by our algorithm and we would find non-physical dynamics.

Both simulations are performed with \( d_m = 0.1 \ll 1 \), but the first plot shows that one must then also increase the number of Monte-Carlo steps per measurement: It has only been increased to 2000 from 1000 in the non-interacting examples earlier. The different shape and unrealistically high blocking temperature show that this simulation does not represent nature, and one has to increase the number of MCS even further. Of course, the real computation time also increases linearly.
5. FERROMAGNETIC NANOPARTICLES WITH DIPOLE-DIPOLE INTERACTION

Figure 50: Simulated ZFC curve for maghemite nanoparticles at $\mu_0 H = 8\text{mT}$ with and without dipole-dipole interaction. The observed difference in $T_B$ is highly exaggerated, as is the difference in shape. This is because the number of MCS is chosen too low and the blue curve does not represent a system that has found its thermal equilibrium at any point.

Figure 51: Simulated ZFC curve for maghemite nanoparticles at $\mu_0 H = 20\text{mT}$ with and without dipole-dipole interaction. In contrast to figure 50, the number of MCS per temperature step is increased by a factor of 10 with unaltered $d_m = 0.1$. The result is in far better accordance with experiments because the shape is better preserved and the difference in $T_B$ is only at about 100K.
6 Summary and Outlook

From the study of interacting particles, we have learned about the significance of simulation parameters like the test-vector length. The introduction of the Onsager Reaction Field method has been shown to be a valid approximation that can be used for magnetometric simulations like ZFC/FC curves in order to drastically reduce computational effort.

The presented research on purely dipolar systems in \textit{fcc}-geometry has shown peculiar magnetic behavior that is very much distinct from magnetic phenomena associated with short-range exchange interactions. Our research was very much restricted only towards pure bulk behaviour. Certainly surface effects are extremely important in conjunction with dipole-dipole interaction and need to be addressed in a next step. Also it appears quite likely that the magnetocrystalline anisotropy has a large impact when studying systems with long-range magnetic order. Therefore, the limit of vanishing anisotropy is not a satisfactory basis for understanding realistic behaviour of assemblies of nanoparticles.

Because our current algorithm and implementation does allow for any choice in lattice constant, saturation magnetization, anisotropy distribution etc, future work should be done by studying larger regions of parameter space.
A Detailed Calculations

A.1 Magnetostatic Derivation of the Onsager Reaction Field

The ORF is a consequence of the magnetic Poisson equation that was derived, including suitable boundary conditions, in 5.1.6.2. Our ‘magnetic sources’ \( \nabla \cdot \mathbf{M} \) are zero almost everywhere:

- Except for the magnetic dipole in the center, the inside of our cut-off sphere \( \{ \mathbf{r} \in \mathbb{R}^3 \mid 0 < r < R \} \) is modeled as vacuum, i.e. \( \mathbf{M} = 0 \).

- The outside \( \{ \mathbf{r} \in \mathbb{R}^3 \mid r > R \} \) is homogenously magnetized, i.e. \( \nabla \cdot \mathbf{M} = 0 \).

Therefore, it is more elegant to use the general solution of the homogenous Laplace equation which is easier to find, and account for everything else by finding special solutions at \( r = 0 \) and \( r = R \) via the boundary conditions. Since the solution to any boundary value problem as stated is unique, any solution we find, will automatically be the correct solution.

A.1.1 Laplace Equation in Azimuthal Symmetry

We are interested in the solution to the homogenous \textit{Laplace equation} in spherical coordinates:

\[
\Delta W(\mathbf{r}) = 0; \quad W(\mathbf{r}) = W(r, \theta, \varphi)
\]

\[
\frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial W}{\partial r} \right) + \frac{1}{r^2 \sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial W}{\partial \theta} \right) + \frac{1}{r^2 \sin^2 \theta} \frac{\partial^2 W}{\partial \varphi^2} = 0 \quad (A.1)
\]

We will immediately ignore the \( \varphi \)-part because we can always reduce our systems of interest to such with azimuthal symmetry. Furthermore, we make a separation ansatz:

\[
W(r, \theta, \varphi) = W(r, \theta) = \frac{u(r)}{r} P(\theta)
\]

Plugging this into (A.1) and multiplying by \( r^2/(uP) \) yields

\[
\frac{r^2}{u} \frac{d^2 u}{dr^2} + \frac{1}{P \sin \theta} \frac{d}{d\theta} \left( \sin \theta \frac{dP}{d\theta} \right) = 0 \quad (A.2)
\]

Therefore, both summands have to be simultaneously constant and opposite in sign. We therefore can define

\[
\frac{r^2}{u} \frac{d^2 u}{dr^2} =: \lambda \in \mathbb{R} \quad \Rightarrow \quad \frac{d^2 u}{dr^2} - \frac{\lambda}{r^2} u = 0 \quad (A.3)
\]

and

\[
x := \cos \theta \quad \Rightarrow \quad \frac{d}{dx} \left( \bullet \right) = -\frac{1}{\sin \theta} \frac{d}{d\theta} \left( \bullet \right) \quad (A.4)
\]
With these definitions we obtain from the second summand in (A.2) the ordinary differential equation

\[
\frac{d}{dx} \left[ (1 - x^2) \frac{dP}{dx} \right] + \lambda P = 0
\]

\[
(1 - x^2) \frac{d^2P}{dx^2} - 2x \frac{dP}{dx} + \lambda P = 0 \quad -1 \leq x \leq 1 \quad (A.5)
\]

At this point, we must stress that the solutions need to cover the complete domain \(x \in [-1, 1]\). We now define a function \(w_n(x)\)

\[
w_n(x) := -(x^2 - 1)^n \quad (A.6)
\]

\[
\Rightarrow (1 - x^2)w' + 2nxw = 0
\]

Differentiate \(n+1\) times \(\Rightarrow (1 - x^2)w^{(2+n)} - 2xw^{(n+1)} + n(n+1)w^{(n)} = 0 \quad (A.7)
\]

This means that we have found functions \(w'_n\) that solve our 2nd order ordinary differential equation (A.5) including the necessary condition that it is defined \(\forall x \in [-1, 1]\). These functions are therefore the only possible and unique solutions and we found that \(\lambda\) is not an arbitrary real number, but has to have the form

\[
\lambda = n(n + 1)
\]

so that our differential equation is the *Legendre Differential Equation*

\[
\frac{d}{dz} \left[ (1 - z^2) \frac{d}{dz} P_n(z) \right] + n(n + 1)P_n(z) = 0
\]

The so-called *Rodrigo form*

\[
P_n := w'_n = \frac{d^n}{dx^n}(x^2 - 1)^n
\]

that we recovered up to a normalization factor produces the *Legendre Polynomials* \(P_n(x)\)

\[
P_0(x) = 1
\]
\[
P_1(x) = x
\]
\[
P_2(x) = \frac{1}{2}(3x^2 - 1)
\]
\[
P_3(x) = \frac{1}{2}(5x^3 - 3x)
\]
\[
\vdots
\]
Since we have also found what $\lambda$ is, we can now investigate (A.3):

$$\frac{d^2 u}{dr^2} - \frac{n(n+1)}{r^2} u = 0;$$

The general solution to this for any given $n \in \mathbb{N}_0$ is

$$u(r) \equiv u_n(r) = a_n r^n + \frac{b_n}{r^{n+1}} \quad (A.8)$$

Together, the general solution of the Laplace equation for problems with azimuthal symmetry is

$$W(r) = W(r, \theta) = \sum_{l=0}^{\infty} \left( a_l r^l + \frac{b_l}{r^{l+1}} \right) P_l(\cos \theta) \quad (A.9)$$

$W(r)$ finite for $r = 0 \Rightarrow b_l = 0 \quad \forall l \geq 0$

$W(r) \to 0$ for $r \to \infty \Rightarrow a_l = 0 \quad \forall l \geq 0$

$$W(r) = A(r) \cdot \hat{r} \Rightarrow \nabla W(r) = \frac{d}{dr} A(r) \cdot \hat{r} + a \cdot A(r)$$

### A.1.2 Solution to Our Boundary Value Problem

We restate the two distinct boundary value problems from section 5.1.6.2 with azimuthal symmetry that we want to solve with the above.

$$\Delta W^{I/II} = \nabla \cdot M^{I/II}$$

$$W^I(r \to 0) \to \frac{m \cdot r}{4\pi r^3}$$  
$$W^I(r \to \infty) \to 0$$  
$$W^I(r = R) = \text{continuous}$$  
$$H^I_{in} = \mu_r H^I_{out}$$  
$$H^I_{in} = H^I_{out}$$

$$W^{II}(r \to 0) \to 0$$  
$$W^{II}(r \to \infty) \to -H_{ext} \cdot r$$  
$$W^{II}(r = R) = \text{continuous}$$  
$$H^{II}_{in} = \mu_r H^{II}_{out}$$  
$$H^{II}_{in} = H^{II}_{out}$$
Because the Legendre polynomials form an orthonormal set, one only needs to look at the powers or $\cos \theta$ that show up in the equations. This simplifies the problem to the following set of equations.

$$W_{\text{in}}(r) = \alpha \frac{\cos \theta}{r^2} + \beta r \cos \theta$$

$$W_{\text{out}}(r) = \gamma \frac{\cos \theta}{r^2} + \delta r \cos \theta$$

with $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ to be determined from the boundary conditions.
B Programming

B.1 C++ implementation of Metropolis algorithm

Here we give the C++ implementation that has given all simulation results that were presented in this thesis, except for the dynamic susceptibility measurements. To achieve this versatility of one programme, the user must set logical parameters declaring what part of the programme will be relevant when executed. Also, the user can still declare things when running the executable. It is written in the C++11 standard.

The basic structure of this implementation is taken from an older C programme with permission by Oleg Petracic. Both the transition to the more modern C++ realization and the treatment of 2-particle interactions, especially within ORF were written by the author.

B.1.1 main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <array>
#include <string>
#include <algorithm>
#include <thread>
#include <cstdlib>
#include <cmath>
#include "parameter.h" // parameter file
#include "rnd250.c" // random number generation
#include "random.cpp" // rng addition
#include "input.inp.h" // necessary programming stuff like arrays, file save
#include "perAux.h" // contains functions for periodic boundary conditions and ORF method
#include "ORF0.str.h" // functions with physical meaning
using namespace std;
```
int main()
{
    cout << "NN" estimate : " << NN << endl;
    cout << " Includeinel\-d\-interaction?";
    cin >> DIPOLAR;
    int edition;
    cout << " Edition : " ;
    cin >> edition;
    cout << endl;
    lattconst = lattconst; 0 ;
    int xx, StepNr, lx, lm, c ;
    xx = 1 ;
    xxmax = 1 ;
    reset_files(edition);
    if (PERIODIC)
    {
        if (type == "sc")
        {
            sc_generator_new(edge_N);
            max_N = pow(edge_N ,3) ;
        }
        if (type == "bcc")
        {
            bcc_generator_new(edge_N);
            max_N = 2 * pow(edge_N ,3) ;
        }
        if (type == "fcc")
        {
            fcc_generator_new(edge_N);
            max_N = 4 * pow(edge_N ,3) ;
        }
        if (type == "hcp")
        {
            hcp_generator_new(edge_N);
            max_N = 1 * pow(edge_N ,3) ;
        }
        makePeriodic();
    }
    else
    {
        if (type == "sc")
        {
            sc_generator_new(edge_N);
        }
        if (type == "bcc")
        {
            bcc_generator_new(edge_N);
        }
        if (type == "fcc")
        {
            fcc_generator_new(edge_N);
        }
        if (type == "hcp")
        {
            hcp_generator_new(edge_N);
        }
        if (SPHERICAL)
        {
            max_N = det_spheric();
            new_pos();
        }
        else
        {
            max_N = max N temp ;
            cout << max_N << "in cube.\n" ;
        }
    }
}

InitMain(edition);

auto t1 = chrono::high_resolution_clock::now();

vector<double> comp_t;
comp_t.push_back(0.);
double Et = 0. ;
// ************** Scans **************

for (int cfg = 0 ; cfg < CFG ; cfg++) // config loop
{
    InitArray();
```cpp
aniso(test);

T[xx] = Tpt[0];
Bx[xx] = Bxfix[0];
//BBx[xx] = Cx * Bx[xx] // OFF In para0 not necessary

for (c = 0; c < MR; c++) // sweep counter loop
{
    // one single temperature step at the beginning
    if (Tsteps[c] > 0)
    {
        T[xx] = Tpt[c];
        Bx[xx] = Bxfix[c];
        array<double, 2> arr = rel_perm(Bx[xx]);
        double mu_r = arr[0];
        double M = arr[1];
        for (lx = 0; lx < TRlxLoops[c]; lx++) // Relaxation loop
        {
            MCS_one(xx, Bx[xx], KB * T[xx], mu_r, M);
        }
        for (lm = 0; lm < TAvgLoops[c]; lm++) // Averaging loop
        {
            MCS_one(xx, Bx[xx], KB * T[xx], mu_r, M);
            Mx[xx] += measure();
        }
        Mx[xx] /= ((double) TAvgLoops[c]);
    }
    auto t2 = chrono::high_resolution_clock::now();
    chrono::duration<double, milli> fp_ms = t2 - t1;
    comp_t.push_back(fp_ms.count());
    spinsave(edition); // NOT advised here! GB files!
    filesave(edition, xx);
    double Et = E_filesave(edition, xx);
    detailed_measure(edition, xx);
    cout << "cfg: " << cfg + 1 << ",", T << "", B << "", M << "", Mx << "", Etot << "", Ettot >> "", Etorr << "", Err << "", Err1000;
    cout << ((comp_t[xx] - comp_t[xx - 1] - 1000) << ",", xx++;
    xxmax++;
    T[xx] = T[xx - 1] + dT[c];
}
```

```cpp
for (StepNr = 0; StepNr < Tsteps[c]; StepNr++)
{
    Bx[xx] = Bxfix[c];
    array<double, 2> arr = rel_perm(Bx[xx]);
    double mu_r = arr[0];
    double M = arr[1];
    for (lx = 0; lx < TRlxLoops[c]; lx++) // Relaxation loop
    {
        MCS_one(xx, Bx[xx], KB * T[xx], mu_r, M);
    }
    for (lm = 0; lm < TAvgLoops[c]; lm++) // Averaging loop
    {
```
B. PROGRAMMING

def MCS_one(xx, Bx[xx], KB*T[xx], mu, r, M):
    Me[xx] += measure();

M[xx] /= ((double) TAvgLoops[c]);

auto t2 = chrono::high_resolution_clock::now();

chrono::duration<double, milli> fp_ms = t2 - t1;

comp.push_back(fp_ms.count());

spinsave(edit);
// NOT ADVISED: GB files!

filesave(edit, xx)
Et = E_filesave(edit, xx);
detailed_measure(edit, xx);

cout << "cfg : cfg + 1 << " , T[xx] << " , Bx[xx] << " , Mx[xx] << " , Etot << " in ";
cout << (comp[xx] - comp[xx-1]) /1000 << "[ s]";

relPermeability << mu_r;
if (mu_r < 1)
{
    cout << "physical?!";
    //abort();
}
cout << endl;

//cout << "spin update\n";

xx++; xxmax++;

T[xx] = T[xx-1] + dT[c];

// cout << "start B sweep\n";
// ********** B-Scan ********** // Won't be activated here as Tsteps[] = 0;

Bx[xx] = Bxpt[0];
T[xx] = Tfix[0];

for (c = 0; c < MR; c++)
{
    // one single field step at the beginning
    // int status = 0; // variable 'status' and function 'countdown'
    // are strictly cosmetic: computation status bar
    if (Bsteps[c] > 0)
    {
        Bx[xx] = Bxpt[c];
        T[xx] = Tfix[c];
        arr = rel_perm(Bx[xx]);
        double mu_r = arr[0];
        double M = arr[1];
        //cout << "Single step, relaxation: ";
        for (lx = 0; lx < BRlxLoops[c]; lx++) // Relaxation loop
        {
            MCS_one(xx, Bx[xx], KB*T[xx], mu, r, M);
            //status = countdown(BRlxLoops[c], lx, status);
        }
        for (lm = 0; lm < BAvLoops[c]; lm++) // Averaging loop
        {
            MCS_one(xx, Bx[xx], KB*T[xx], mu, r, M);
            Me[xx] += measure();
        }
    }
    M[xx] /= ((double) TAvgLoops[c]);

auto t2 = chrono::high_resolution_clock::now();
chrono::duration<double, milli> fp_ms = t2-t1;
comp_t.push_back(fp_ms.count());

spinsave(edit); // NOT ADVISED: GB files!
filesave(edit, xx);
Et = E2filesave(edit, xx);
detailed_measure(edit, xx);

cout << "cfg: \n" << cfg + 1 << ", T\[xx\] << " , \Etot\n" << Bx[xx] << " , Mx\n"; 
cout << (comp_t[xx]-comp_t[xx-1])/1000 << "\n";
++xx;
++xxmax;
Bx[xx] = Bx[xx-1] + dBx[c];
}

// status = 0;
for (StepNr = 0; StepNr < Bsteps[c]; StepNr++)
{
T[xx] = Tfix[c];
array<double, 2> arr = rel_perm(Bx[xx]);
double mu_r = arr[0];
double M = arr[1];
//cout << "Relaxation progress: ";
for (lx = 0; lx < BRlxLoops[c]; lx++) // Relaxation loop
{
MCS_one(xx, Bx[xx], KB*T[xx], mu_r, M);
Mx[xx] += measure();
//status = countdown(BRlxLoops[c], lx, status);
}
for (lm = 0; lm < BAVgLoops[c]; lm++) // Averaging loop
{
//cout << "Schleife\n";
MCS_one(xx, Bx[xx], KB*T[xx], mu_r, M);
Mx[xx] /= (double) TAvgLoops[c];
//cout << "Averaging done\n";
auto t2 = chrono::high_resolution_clock::now();
chrono::duration<double, milli> fp_ms = t2-t1;
comp_t.push_back(fp_ms.count());
spinsave(edit); // NOT ADVISED: GB files!
filesave(edit, xx);
detailed_measure(edit, xx);

cout << "cfg: \n" << cfg + 1 << ", T\[xx\] << " , \Etot\n" << Bx[xx] << " , Mx\n"; 
cout << (comp_t[xx]-comp_t[xx-1])/1000 << "\n";
++xx;
++xxmax;
Bx[xx] = Bx[xx-1] + dBx[c];
}

// output.dat only updated after 1 sweep, NOT after every step as spin.dat!

// OFF spinatable();

} // \configuration loop
B.1.2 parameter.h

// Natural constants. Only to be changed for convenience!!
const double pi = M_PI; // pi
const double mu0 = pi * 4e-7; // vacuum permeability
const double eV = 1.60217662e-19; // electron charge for easy conversion Joule <-> electron Volt
const double hbar = 1.0545718e-34; // h bar
const double KV = 1.34e4; // Boltzmann constant

// Parameters that distinguish the sample/experiment and its dynamics
const double d_m = 1.; // text-rotation vector length
const double Mag = 0.386e+pi / 6 * pow(20e-9, 3); // Magnetic moment (saturation mag. * Volume)
const double KV_bcc = 1.344*pi / 6 * pow(20e-9, 3); // anisotropy constant * Volume

// Parameters with mostly computational rather than physical meaning
const int maxNtemp = 2016; // # spins being considered
const int maxN_temp = pow(edgeN, 3); // general bcc
const int maxN = 1000000; // maximum distance up to which dipole energy is calculated
const int loopN_2 = 590; // # MC Avgloops per step // ! ! in anisoBlock: loopN2 = loopN always ! !
const int double CUTOFF = 2; // maximum distance up to which dipole energy is calculated
const int BEST_CUTOFF; // largest radius containing some neighbours as above value

// Temperature T in Kelvin
B. PROGRAMMING

69 // and
70 // magnetic field B in T(esu)
71 // P o i n t s f o r T−sweeps
72 const double Tpt[MR+1] = {1000, 25, 1000, 1000};
73 const double dT[MR] = {-25, 25, -100};
74 const double Bfix[MR] = {0.0, 0.015, 0.015};
75 const int TRlxLoops[MR] = {loopNrRlx, loopNrRlx, loopNrRlx};
76 const int TAvgLoops[MR] = {loopNrAvg, loopNrAvg, loopNrAvg};
77
78 // P o i n t s f o r B−sweeps
79 const double Bxpt[MR+1] = {0.15, 0.15, 0.15, 0.15};
80 const double dBx[MR] = {0.005, -0.005, 0.005};
81 const double Tfix[MR] = {10, 10, 10};
82 const int BRlxLoops[MR] = {loopNrRlx, loopNrRlx, loopNrRlx};
83 const int BAvgLoops[MR] = {loopNrAvg, loopNrAvg, loopNrAvg};

B.1.3 input-init.h

1 // input-init.h
2 // based on inispd5tl.h
3 // computational necessities without physical meaning are defined here.
4 //
5 using namespace std;
6
7 // Spin positions
8 double POS[maxNtemp][3]; // Generates the sc lattice of spin positions.
9 bool intern[maxNtemp]; // flags if sites are within spherical cutoff
10 double NPOS[maxNtemp][3];
11
12 // Spin tables
13 double s[maxNtemp][3];
14 double sx[maxNtemp], sy[maxNtemp], sz[maxNtemp];
15
16 // Anisotropy easy axes
17 double k[maxNtemp][3];
18 double kx[maxNtemp], ky[maxNtemp], kz[maxNtemp];
19
20 void ini_config(); // initialize configuration
21 void InitArray(); // array initialization function -> CFG loop
22 void dist_matrix(); // calculate distance matrix for spin interaction
23 void r_vectors(); // calculate normalized distance vectors
24 void local_matrix(); // determine neighborhood positions for dipole interaction
25 void line_generator(); // generates line of positions
26 void sc_generator(); // sc generator
27 void bcc_generator(); // bcc generator
28 void fcc_generator(); // fcc generator
29 void sc_only(); // POS only with sc generated
30 double best_cutoff();
31
32 int Tsteps[MR], Bsteps[MR];
33 int xxmax;
34 double T[MAXDATA], Bx[MAXDATA];
35 double Mx[MAXDATA];
36 double TBsteps[CFG][5];
37
38 void InitMain(int ed) {
39  
40  int maxdat, i, xx;
41  int SEED_cluster;
42  if (ed < 1000)
43    SEED_cluster = SEED ;
44  else
45    SEED_cluster = SEED + ed;
46  seed250(SEED_cluster);
47  srand(SEED_cluster);
48  
49  maxdat = 0;
50  for (i = 0; i < MR; i++)
51    
52      //Tsteps[i] = abs( (int) ((Tpt[i]−Tpt[i+1])/dT[i]));
B. PROGRAMMING

63 // Pure B Sweep !!!
64 Tsteps[i] = abs((int)((Tpt[i]-Tpt[i+1])/dT[i]));  
65 Bsteps[i] = abs((int)((Bxpt[i]-Bxpt[i+1])/dBx[i]));  
66 //Bsteps[i] = 0; // Only T sweeps here !!
67 maxdat += Tsteps[i]+Bsteps[i];  
68 }
69
70 cout << (maxdat+1)*CFG << "Data points\n";  
71 if (((maxdat*CFG+1) > MAXDATA)  
72 {  
73   cout << "too many data points!!\n";  
74   abort();  
75 }
76
77 for (xx = 0; xx < MAXDATA; xx++)  
78 {  
79   Mx[xx] = 0.0;  
80   T[xx] = 0.0;  
81   Hx[xx] = 0.0;  
82   }
83
84 cout << "data variables initialized!\n";  
85 }
86
87 void InitArray()  
88 {  
89   cout << "initialize arrays...\n";  
90   ini_config();  
91   cout << "arrays initialized!\n";  
92   // If no dipole-dipole interaction is considered, below functions are NOT necessary  
93   cout << "calculate distances...\n";  
94   dist_matrix();  
95   cout << "distances calculated!\n";  
96   if (PERIODIC)  
97     BESTCUTOFF = best_cutoff();  
98   if (PRINT_ENERGIES or not PERIODIC)  
99     {  
100       cout << "calculate distance vectors...\n";  
101       rvectors();  
102       cout << "distance vectors calculated!\n";  
103     }
104   
105   cout << "calculate dipole neighbours\n";  
106   local_matrix();  
107   cout << "dipole neighbours determined!\n";  
108   /*  
109   */
110 }
111
112 void ini_config()  
113 {  
114   for (int i = 0; i < maxN; i++)  
115     {  
116       // Both spin and easy axis are stored as unit vectors because we need only the  
117       // angle between them via scalar product  
118       // Set initial spin vector directions  
119       Marsaglia(s[i]);  
120       sx[i] = s[i][0];  
121       sy[i] = s[i][1];  
122       sz[i] = s[i][2];  
123     // Set initial easy axes directions
B. PROGRAMMING

Marsaglia(k[i]);

kx[i] = k[i][0];
ky[i] = k[i][1];
kz[i] = k[i][2];

}
cout << "Initial spin directions set.\n"
cout << "Easy axes are set.\n"
}

// Two optional functions that are only useful when comparing magnetization to theoretical paramagnetic case!

double Langevin(double Ezee, double kbT)
{
    return (1/tanh(Ezee/kbT)−kbT/Ezee);
}

// Reset all output files

void reset_files(int ed)
{
    string data_file_cluster = data_file + to_string(ed) + ".dat";
    string table_file_cluster = table_file + to_string(ed) + ".dat";
    string Edata_file_cluster = Edata_file + to_string(ed) + ".dat";
    string Mdata_file_cluster = Mdata_file + to_string(ed) + ".dat";

    ofstream fout;
    fout.open(data_file_cluster, ios::trunc);
    fout.close();

    ofstream spinout;
    spinout.open(table_file_cluster, ios::trunc);
    spinout.close();

    ofstream anisout;
    anisout.open(aniso_file_cluster, ios::trunc);
    anisout.close();

    ofstream Efout;
    Efout.open(Edata_file_cluster, ios::trunc);
    Efout.close();

    ofstream Mfout;
    Mfout.open(Mdata_file_cluster, ios::trunc);
    Mfout.close();

    if (ONSDAT)
    {
        ofstream ons;
        ons.open("ons.dat", ios::trunc);
        ons.close();
    }

    ofstream blockout;
    blockout.open(blockT_file, ios::trunc);
    blockout.close();

    ofstream fout;
    fout.open(data_file_cluster, ios::app);
    //fout << "eg T B M(n)";
    fout << xx << "\t" << T[xx] << "\t" << Bx[xx] << "\t" << Mx[xx] << endl;
    fout.close();
}

// Save data file which contains magnetization at applied field, temperature

void filesave(int ed, int xx)
{
    string data_file_cluster = data_file + to_string(ed) + ".dat";
    ofstream fout;
    fout.open(data_file_cluster, ios::app);
    //fout << "eg T B M(n)";
    fout << xx << "\t" << T[xx] << "\t" << Bx[xx] << "\t" << Mx[xx] << endl;
    fout.close();
}

// Save spin positions. Every block of maxN rows corresponds to one row in data_file.
B. PROGRAMMING

```cpp
void spinsave(int ed)
{
    if (PRINTSPINS)
    {
        string table_file_cluster = table_file + to_string(ed) + ".dat";
        ofstream spinout;
        spinout.open(table_file_cluster, ios::app);
        // fout << "cgb T BMn";
        double X,Y,Z,VX,VY,VZ;
        for (int n = 0; n < maxN; n++)
        {
            VX = sx[n];
            VY = sy[n];
            VZ = sz[n];
            X = POS[n][0];
            Y = POS[n][1];
            Z = POS[n][2];
            /*
            // Operations on sx, POS in order to make the plots in gnuplot easier
            VX = 0.5*sx[n];
            VY = 0.5*sy[n];
            VZ = 0.5*sz[n];
            X = POS[n][0]/latt_const-0.5*VX;
            Y = POS[n][1]/latt_const-0.5*VY;
            Z = POS[n][2]/latt_const-0.5*VZ;
            */
            spinout << X << "\t" << Y << "\t" << Z << "\t" << VX << "\t" << VY << "\t" << VZ << endl;
            // fout << endl;
            spinout.close();
        }
    }
}

void anisosave(int ed)
{
    string aniso_file_cluster = aniso_file + to_string(ed) + ".dat";
    ofstream anisout;
    anisout.open(aniso_file_cluster);
    double X,Y,Z,VX,VY,VZ;
    for (int n = 0; n < maxN; n++)
    {
        VX = kx[n];
        VY = ky[n];
        VZ = kz[n];
        X = POS[n][0];
        Y = POS[n][1];
        Z = POS[n][2];
        /*
        VX = kx[n];
        VY = ky[n];
        VZ = kz[n];
        X = POS[n][0]/latt_const-0.5*VX;
        Y = POS[n][1]/latt_const-0.5*VY;
        */
    }
    // fout << endl;
    spinout.close();
}
```
\[ Z = \text{POS}[n][2] / \text{lat.c} \times \text{const} - 0.5 \times VZ; \]

\[
\text{anisout} \ll X << "\t" << Y << "\t" << Z << "\t" << VX << "\t" << VY << "\t" << VZ << end1;
\]

\[
\text{spinout} \ll \text{POS}[n][0] << "\t" << \text{POS}[n][1] << "\t" << \text{POS}[n][2] << "\t" << sx[n] << "\t" << sy[n] << "\t" << sz[n] << end1;
\]

\[
\text{cout} \ll "\text{Wrote easy axes table.}\n";
\]

\[
\text{anisout.close()};
\]

\[
\text{cout} \ll end1;
\]

\[
\text{cout} \ll "\text{end.}\n";
\]

B.1.4 str.h

// str.h

// functions and definitions that translate the physical structure of the simulated system are
// done here

using namespace std;

double dist[maxN][maxN];
double vVectors[maxN][maxN][3];

// function that calculates array dist[maxN][maxN]
void line_generator(int edge, double start[3], int direction, int start_index)
{
    int n = start_index;
    double a = 1.;
    int dir[3] = {0,0,0};
    dir[direction] = 1;
    for (int i = 0; i < edge; i++)
    {
        POS[n][0] = a*i+dir[0]+start[0];
        POS[n][1] = a*i+dir[1]+start[1];
        POS[n][2] = a*i+dir[2]+start[2];
        n++;
    }
}

void sc_generator(int edge, double start[3], int start_index)
{
    int n = start_index;
    double a = 1.;
    for (int i = 0; i<edge; i++)
    {
        for (int j = 0; j<edge; j++)
        {
            for (int k = 0; k<edge; k++)
            {
                POS[n][0] = a*i+start[0];
                POS[n][1] = a*j+start[1];
                POS[n][2] = a*k+start[2];
                n++;
            }
        }
    }
}

void sc_only(int outer_edge)
{
    double sc_start[3] = {0,0,0};
    sc_generator(outer_edge, sc_start, 0);
    if (maxN != pow(edgeN,3))
    {
        cout << "Error in sc generation, check site numbers!" << end1;
        abort();
    }
}

// end.
void bcc_generator(int outer_edge)
{
    double a = 1.0;
    double inner_start[3] = {0.5*a, 0.5*a, 0.5*a};
    double outer_start[3] = {0,0,0};
    if (outer_edge < 2 || maxN_temp != pow(edge_N, 3) + pow(edge_N-1, 3))
    {
        cout << "error in bcc generation, check site numbers!" << endl;
        abort();
    }
    sc_generator(outer_edge, outer_start, 0);
    int outer_index = pow(outer_edge, 3);
    sc_generator(outer_edge-1, inner_start, outer_index);
}

void fcc_generator(int outer_edge)
{
    double a = 1.0;
    double outer_start[3] = {0,0,0};
    if (outer_edge < 2 || maxN_temp != pow(edge_N, 3) + 3*edge_N*pow(edge_N-1, 2))
    {
        cout << "error in fcc generation, check site numbers!" << endl;
        abort();
    }
    sc_generator(outer_edge, outer_start, 0);
    int current_index = pow(outer_edge, 3);
    // generate lines in x-direction
    double current_start[3] = {0,0,0};
    for (int i = 0; i < outer_edge - 1; i++)
    {
        for (int j = 0; j < outer_edge - 1; j++)
        {
            current_start[1] = a*(0.5+i);
            current_start[2] = a*(0.5+j);
            line_generator(outer_edge, current_start, 0, current_index);
            current_index += outer_edge;
        }
        current_start[1] = 0;
        current_start[2] = 0;
    }
    // generate lines in y-direction
    for (int i = 0; i < outer_edge - 1; i++)
    {
        for (int j = 0; j < outer_edge - 1; j++)
        {
            current_start[0] = a*(0.5+i);
            current_start[2] = a*(0.5+j);
            line_generator(outer_edge, current_start, 1, current_index);
            current_index += outer_edge;
        }
        current_start[0] = 0;
        current_start[2] = 0;
    }
    // generate lines in z-direction
    for (int i = 0; i < outer_edge - 1; i++)
    {
        for (int j = 0; j < outer_edge - 1; j++)
        {
            current_start[0] = a*(0.5+i);
            current_start[1] = a*(0.5+j);
            line_generator(outer_edge, current_start, 2, current_index);
            current_index += outer_edge;
        }
        current_start[0] = 0;
        current_start[1] = 0;
    }
B. PROGRAMMING

// calculate the center of the object

array<double, 4> calc_cent_rad()
{
    array<double, 4> CENTER = {0, 0, 0, 0};
    for (int i = 0; i < max_N_temp; i++)
    {
        CENTER[0] += POS[i][0];
        CENTER[1] += POS[i][1];
        CENTER[2] += POS[i][2];
    }
    CENTER[0] /= max_N_temp;
    CENTER[1] /= max_N_temp;
    CENTER[2] /= max_N_temp;
    CENTER[3] = 0.5 * (edge_N - 1);
    return CENTER;
}

// check if a point is inside the sphere

bool in_sphere(double x, double y, double z, array<double, 4> center)
{
    double x2 = center[0];
    double y2 = center[1];
    double z2 = center[2];
    double R = center[3];
    double r = pow(x - x2, 2) + pow(y - y2, 2) + pow(z - z2, 2);
    return (r <= R*R);
}

// determine the number of points inside the sphere

int det_spheric()
{
    int MAX = 0;
    array<double, 4> CENTER = calc_cent_rad();
    fill_n(intern, max_N_temp, false);
    double x, y, z;
    for (int i = 0; i < max_N_temp; i++)
    {
        x = POS[i][0];
        y = POS[i][1];
        z = POS[i][2];
        //cout << x << " " << y << " " << z << " endl;"
        if (in_sphere(x, y, z, CENTER))
        {
            MAX++;
            intern[i] = true;
        }
    }
    cout << MAX << " of " << max_N_temp << " in sphere.\n";
    return MAX;
}

// new position function

void new_pos()
{
    int n = 0;
    for (int i = 0; i < max_N_temp; i++)
    {
        if (intern[i])
        {
            NPOS[n][0] = POS[i][0];
            NPOS[n][1] = POS[i][1];
        }
    }
}
N.POS[n][2] = POS[i][2];

n++;

}  

if (n != maxN)  
{  
    cout << "Error in spherical cut off!\n";
    abort();
}

// reset POS with arbitrary value (-42)
fill(POS[0], POS[0] + maxN*temp + 3, -42.);

// for relevant indices, fill POS with NPOS
for (int i = 0; i < maxN; i++)  
{  
    POS[i][0] = N.POS[i][0];
    POS[i][1] = N.POS[i][1];
    POS[i][2] = N.POS[i][2];
}

// calculation of the normalized distance vectors between sites -> rVectors[maxN][maxN][3]
void r_vectors()  
{  
    for (int i = 0; i < maxN; i++)  
    {  
        for (int j = i+1; j < maxN; j++)  
        {  
            double d = dist[i][j];
            double vec[3] = {POS[i][0] - POS[j][0], POS[i][1] - POS[j][1], POS[i][2] - POS[j][2]};
            double mod = sqrt(pow(vec[0], 2) + pow(vec[1], 2) + pow(vec[2], 2));
            vec[0] /= mod;
            vec[1] /= mod;
            vec[2] /= mod;

            // cout << "Test dist " << dist[i][j] << " \n" << dist[i][j] / mod << endl;
        }
    }
}
B. PROGRAMMING

```c
int j = 0;
// counter loop for case i>j
while(j < maxN)
{
    int i = j;
    while (i<maxN)
    {
        rVectors[i][j][0] = rVectors[j][i][0];
        rVectors[i][j][1] = rVectors[j][i][1];
        rVectors[i][j][2] = rVectors[j][i][2];
        i ++;
    }
    j ++;
}

// get the neighbours that are within the cutoff range and store their indices.
// Note that only i=0 gets all neighbours explicitly, neighbours for i>0 are partly contained
// for smaller i !!
void local_matrix()
{
    int min = maxN;  // completely optional:
    int max = 0;    // show the minimal/maximal number of neighbours that will be
                    // considered for E\_dipole
    // loop over all sites i
    for (int i = 0; i < maxN; i++)
    {
        int loc = 1;  // counts # neighbours for site i
        loc POS[i][0] = 0;  // 0th element contains number of neighbours, used in
                            // loc\_E\_dipole()
        // because of symmetry of E\_dipole, only upper triangle matrix needs to filled
        for (int j = i+1; j < maxN; j++) // only viable if loc\_E\_dipole() works with
                                       // corresponding structure!
        {
            // loc\_POS only filled with neighbouring sites within a distance of 'cutoff' around site i
            if (dist[i][j] <= cutoff)
            {
                loc\_POS[i][loc] = j;  // So, loc\_POS[a][b] = c reads: the
                                      // position vector of the bth neighbour of site a is stored as
                                      // cth element of the ORIGINAL POS array.
                loc ++;
                loc\_POS[i][0] += 1;
            }
            // Because of this construction, no dipole\_\_dipole pair will be either
            // counted double or forgotten!
        }
        if (loc > max)
        {
            max = loc;
        }
        if (loc < min)
        {
            min = loc;
        }
    }
}
```
cout << "Up to " << maxN-1 << " neighbours per site considered instead of original " << maxN-1 << endl;
*/
double dipole(int i, int j)
{
    double d = dist[i][j];
    double vec[3];
    vec[0] = rVectors[i][j][0];
    vec[1] = rVectors[i][j][1];
    vec[2] = rVectors[i][j][2];
    if (i==j)
    else
        return 0;
}

// this function calculates the sum of all dipole-dipole energies w/o cutoff!
double E_dipole()
{
    double Ed = 0;
    // int status;
    for (int i = 0; i<maxN; i++)
    {
        for (int j = i+1; j<maxN; j++) // upper triangle matrix: main diagonal entries must be 0, rest isn't determined because dipole pairs mustn't be counted twice
        {
            double d = dist[i][j];
            double vec[3];
            vec[0] = rVectors[i][j][0];
            vec[1] = rVectors[i][j][1];
            vec[2] = rVectors[i][j][2];
            cout << vec[2] << endl;
            // status = countdown(maxN, i, status);
        }
        return Ed;
    }

double E_dipole_quick(int j)
{
    double Ed = 0;
    // int status;
    for (int i = 0; i<maxN; i++)
    {
        if (i!=j)
        {
            double d = dist[i][j];
            double vec[3];
            vec[0] = rVectors[i][j][0];
            vec[1] = rVectors[i][j][1];
            vec[2] = rVectors[i][j][2];
            cout << vec[2] << endl;
            // status = countdown(maxN, i, status);
        }
    }
}
return Ed;

double E_dipole_total()
{
    double Ed = 0;
    omp_set_num_threads(numThreads);
    #pragma omp parallel for reduction(+:Ed)
    for (int i = 0; i < maxN-1; i++)
    {
        for (int j = i+1; j < maxN; j++)
        {
            Ed += dipole(i,j);
        }
    }
    Ed /= pow((edgeN-1)*latt_const,3);
    return Ed;
}

double E_aniso_total()
{
    double Ea = 0;
    omp_set_num_threads(numThreads);
    #pragma omp parallel for reduction(+:Ea)
    for (int n = 0; n < maxN; n++)
    {
        //Ea += KV*(1- pow(sx[n]*sx[n]+sy[n]*sy[n]+sz[n]*sz[n], 2));
        Ea += KV*(0- pow(sx[n]*sx[n]+sy[n]*sy[n]+sz[n]*sz[n], 2));
    }
    Ea /= pow((edgeN-1)*latt_const,3);
    return Ea;
}

double E_eez_total(double BBx)
{
    double Ezee = 0;
    omp_set_num_threads(numThreads);
    #pragma omp parallel for reduction(-:Ezee)
    for (int n = 0; n < maxN; n++)
    {
        Ezee += -Mag*BBx*sx[n];
    }
    Ezee /= pow((edgeN-1)*latt_const,3);
    return Ezee;
}

double E_surf_total()
{
    double Esurf = 0;
    omp_set_num_threads(numThreads);
    #pragma omp parallel for reduction(+:Esurf)
    for (int n = 0; n < maxN-1; n++)
    {
        for (int m = n+1; m < maxN; m++)
        {
            Esurf += sx[n]*sx[m] + sy[n]*sy[m] + sz[n]*sz[m];
        }
    }
    Esurf *= mu0*Mag*Mag / (2 * pow((edgeN-1)*latt_const, 6));
    return Esurf;
}

double E_dipole_quick_MULT(int j)
{
    double Ed = 0;
    //int Ntrunc = maxN/numThreads;
    //int status;
    omp_set_num_threads(numThreads);
    #pragma omp parallel for reduction(+:Ed)
    for (int i = 0; i < maxN; i++)
    {
        Ed += dipole(i,j);
    }
    return Ed;
double E_filesave(int ed, int xx)
{
  double Etotal = 0;

  if (PRINT_ENERGIES)
  {
    string data_file_cluster = Edata_file + to_string(ed) + ".dat";
    ofstream fout;
    fout.open(data_file_cluster, ios::app);
    // fout << "cgf T B M n";
    double Ed, Ea, Ezee, Esurf;
    Ed = E_dipole_total();
    Ea = E_aniso_total();
    Ezee = E_zee_total(Bx[xx]);
    Esurf = E_surf_total();
    Etotal = Ed + Ea + Ezee + Esurf;
    fout << xx << T[xx] << Bx[xx] << "t" << Ezee << "t" << Ea << "t" << Ed << "t" << Etotal << endl;
  }

  return Etotal;
}

detailed_measure(int ed, int xx)
{
  if (DETAILED_MEASURE)
  {
    double mx, my, mz, MX, MY, MZ;
    mx = 0;
    my = 0;
    mz = 0;
    MX = 0;
    MY = 0;
    MZ = 0;

    omp_set_num_threads(2);
    #pragma omp parallel for reduction(+:mx), reduction(+:my), reduction(+:mz),
      reduction(+:MX), reduction(+:MY), reduction(+:MZ)
    for (int n = 0; n < maxN; n++)
    {
      mx += sx[n]; // measure in x-direction
      my += sy[n];
      mz += sz[n];

      MX += abs(sx[n]); // measure total x-alignment
      MY += abs(sy[n]);
      MZ += abs(sz[n]);
    }

    m_x /= maxN;
    m_y /= maxN;
    m_z /= maxN;
    MX /= maxN;
    MY /= maxN;
    MZ /= maxN;

    string data_file_cluster = Mdata_file + to_string(ed) + ".dat";
    ofstream fout;
    fout.open(data_file_cluster, ios::app);
    // fout << "cgf T B M n";

    fout << xx << T[xx] << Bx[xx] << "t" << abs(mx) << "t" << abs(my) << "t" << abs(mz) << "t" << abs(MX) << "t" << abs(MY) << "t" << abs(MZ) << endl;

    fout.close();
  }
}
B. PROGRAMMING

608 609 /*
610 // calculate $E_{\text{dipole}}$ but at every site $i$, only the local neighbours determined in $\text{local_matrix()}$ are considered
611 */
612 double $\text{local}\_\text{dipole()}$
613 {
614  double $Ed = 0$;
615  for (int $i = 0$; $i < \text{max}\_N$; $i++$
616  {
617    int $max = \text{loc}\_\text{POS}[$i$][0]$; // exploit that we already calculated the
618    // number of neighbours = number of loops
619 // the upper triangle matrix is already implemented in $\text{local_matrix()}$, 
620 // therefore $j$ starts always at 1 as the 0th element is $\#(\text{neighbours})$, NOT an index!!
621    for (int $j = 1$; $j < \text{max}$; $j++$
622    {
623      int $l = \text{loc}\_\text{POS}[$i$][j]$; // list the 'old' index when all sites
624      // were considered so that we can still use $\text{dist[]}$ and
625      // exactly the same as in $E_{\text{dipole}}()$ from here.
626      double $d = \text{dist}[$i$][j]$;
627      double vec[3];
628      vec[0] = $\text{rVectors}[$i$][l][0]$;
629      vec[1] = $\text{rVectors}[$i$][l][1]$;
630      vec[2] = $\text{rVectors}[$i$][l][2]$;
631      // $Ed_{\text{old}} - \mu_0 \cdot \text{Mag} \cdot \text{Mag}/(4 \cdot \pi \cdot \text{pow}(d,3)) *$
633      // $Ed_{\text{new}} = \text{mu} \cdot \text{Mag} \cdot \text{Mag} / (3 \cdot \text{VOL} \cdot B)$;
634      // $Ed_{\text{new}} = \text{mu} \cdot \text{Max} \cdot \text{Mag} / (3 \cdot \text{VOL} \cdot B)$;
635      return $Ed$;
636  }
637  return $Ed$;
638 */
639
double $\text{measure()}$
640 {
641  double $mx = 0$; // measure in x-direction
642  for (int $n = 0$; $n < \text{max}\_N$; $n++$
643  {
644    $mx += sx[n]$;
645  }
646  return $mx/\text{max}\_N$;
647}

648 array<double,2> $\text{rel_perm}(\text{double } B)$
649 {
650  array<double,2> res;
651  double mu_r;$
652  double M;
653  double meas = $\text{measure()}$;
654  double $\text{VOL} = \text{pow}((\text{edge}N-1) \cdot \text{latt}\_\text{const},3)$;
655  if ($B == 0$)
656    $mu_r = \text{def}\_\text{mur}$;
657  else
658  {
659    // $z = \mu_0 \cdot \text{Max} \cdot \text{measure()} \cdot \text{Mag} / (3 \cdot \text{VOL} \cdot B)$;
660    $M = \text{max}\_N \cdot \text{meas} \cdot \text{Mag} / \text{VOL}$;
661    // $z = (1 / (1 + B \cdot \text{meas}) - 1)$;
662    $mu_r = 1 + m0M/B$;
663    // $\mu_r = B / (B - \mu_0 \cdot \text{Max} \cdot \text{measure()} \cdot \text{Mag})$;
664  }
B. PROGRAMMING

```c
// mu_r = def mu_r;
res[0] = mu_r;
return res;
}

double best_cutoff()
{
    double best_CUTOFF = edge_N - 1;
    double curr_dist;
    for (int i = 0; i < maxN - 1; i++)
    {
        for (int j = i + 1; j < maxN; j++)
        {
            curr_dist = dist[i][j]/latt_const;
            if (curr_dist > CUTOFF and curr_dist < best_CUTOFF)
                best_CUTOFF = curr_dist;
            // cout << curr_dist << "\t";
        }
    }
    best_CUTOFF -= 0.000001;
    cout << " cutoff is " << best_CUTOFF << " up from " << CUTOFF << endl;
    return best_CUTOFF;
}

vector<int> shuffle_list()
{
    vector<int> rand_list;
    for (int i = 0; i < maxN; i++)
        rand_list.push_back(i);
    random_shuffle(rand_list.begin(), rand_list.end());
    return rand_list;
}

void MCS_ALT(int xx, double BBx, double kt)
{
    int n;
    double E0, E1, E_a, modM, M_temp[3], dM[3];
    // int status;
    for (n = 0; n < maxN; n++)
    {
        E_a = -KV * pow(sx[n]*kx[n]+sy[n]*ky[n]+sz[n]*kz[n], 2); // Anistropy energy
        // E_d = loc_E_dipole(); // choose cutoff approximation for speed-up
        // E_d = E_dipole(); // choose to consider ALL sites for dipole i.
        E0 = -Mag*BBx*xx[n]; // field in z-direction (Zeeman energy)
        E0 += E_a; // sum of dipole energies
        M_temp[0] = sx[n];
        M_temp[1] = sy[n];
        M_temp[2] = sz[n];
        Marsaglia(dM);
        sx[n] += (d_m + dM[0]);
        sy[n] += (d_m + dM[1]);
        sz[n] += (d_m + dM[2]);
        modM = sqrt(pow(sx[n], 2)+pow(sy[n], 2)+pow(sz[n], 2));
        sx[n] /= modM;
        sy[n] /= modM;
        sz[n] /= modM;
        E_a = -KV * pow(sx[n]*kx[n]+sy[n]*ky[n]+sz[n]*kz[n], 2); // Anistropy energy
        // E_d = loc_E_dipole();
        // E_d = E_dipole();
        E1 = -Mag*BBx*xx[n]; // must be same (updated) calculation as E0
        E1 += E_a;
        if (E1 > E0)
        {
            if (rand0_1() > exp((E0-E1)/kt))
            {
                // do nothing
            }
        }
    }
```
void MCS_one(int sx, double BBx, double kt, double mu_r, double M)
{
    double E0, E1, E_a, E_d, mod_M, M_temp[3], d_M[3];
    vector<int> rand_list = shuffle_list(); // experimental DO NOT USE IN THIS VERSION
    // int status;
    for (int m = 0; m < max_N; m++)
    {
        E_d = 0;
        int n = rand_list[m]; // every spin updated exactly once per MCS_one call, but in a random
                     // fashion -> permutation
        if (int n == rnd250()%max_N);
        if (int n == m);
        E_a = -KV * pow(sx[n]*kx[n]+sy[n]*ky[n]+sz[n]*kz[n], 2); // Anistropy energy
        E1 = 0;
        // E_a *= (3*mu_r / (1+2*mu_r));
        if (DIPOLAR)
        {
            if (PERIODIC)
            {
                E1 = ORF(n, mu_r, BBx, M) ;
                E0 = 0;
                // E_a *= (3*mu_r / (1+2*mu_r));
            }
            else
            {
                E1 = E_dipole_quick_MULT(n);
                E0 = - Mag * BBx * sx[n]; // field in x-direction (Zeeman energy)
            }
        }
        else
        {
            E0 = - Mag * BBx * sx[n]; // field in x-direction (Zeeman energy)
        }
        E0 += E_a; // sum of dipole energies
        M_temp[0] = sx[n];
        M_temp[1] = sy[n];
        M_temp[2] = sz[n];
        Marsaglia(d_M);
        sx[n] += (d_m * d_M[0]);
        sy[n] += (d_m * d_M[1]);
        sz[n] += (d_m * d_M[2]);
        mod_M = sqrt(pow(sx[n], 2)+pow(sy[n], 2)+pow(sz[n], 2));
        sx[n] /= mod_M;
        sy[n] /= mod_M;
        sz[n] /= mod_M;
        E_d = 0;
        E_d = -KV * pow(sx[n]*kx[n]+sy[n]*ky[n]+sz[n]*kz[n], 2); // Anistropy energy
        E0 += E_d; // choose cutoff approximation for speed-up
        if (DIPOLAR)
        {
            if (PERIODIC)
            {
                E_d = ORF(n, mu_r, BBx, M) ;
                E1 = 0;
                // E_a *= (3*mu_r / (1+2*mu_r));
            }
            else
            {
                E0 = - Mag * BBx * sx[n]; // field in x-direction (Zeeman energy)
            }
        }
    }
}
\begin{verbatim}

{  E_d = E_dipoleQuick_MULT(n);
   E1 = - Mag * BBx * sx[n];  \textit{field in x-direction (Zeeman energy)}
}

else
   E1 = - Mag * BBx * sx[n];  \textit{field in x-direction (Zeeman energy)}

E1 += E_a;
E1 += E_d;
if (E1 > E0)
   {
      if (rand0_1() > \exp((E0-E1)/kt))
      {
         sx[n] = M_temp[0];
         sy[n] = M_temp[1];
         sz[n] = M_temp[2];
      }
   }

// status = countdown(maxN, n, status);

void aniso_test()
{
   int n;
   double t_kx = 0.0;
   double t_ky = 0.0;
   double t_kz = 0.0;
   double t_sx = 0.0;
   double t_sy = 0.0;
   double t_sz = 0.0;

   for (n = 0; n < maxN; n++)
   {
      t_kx += kx[n];
      t_ky += ky[n];
      t_kz += kz[n];
      t_sx += sx[n];
      t_sy += sy[n];
      t_sz += sz[n];
   }

   t_kx /= maxN;
   t_ky /= maxN;
   t_kz /= maxN;
   t_sx /= maxN;
   t_sy /= maxN;
   t_sz /= maxN;

   cout << "Avg easy axis components xyz: \" << t_kx << "\" << t_ky << "\" << t_kz << endl;
   cout << "Avg Superspin components xyz: \" << t_sx << "\" << t_sy << "\" << t_sz << endl;
}

B.1.5 perAux.h

1 // perAux.h
2 // additional structural functions for interacting periodic systems.
3 // intended for simulations with ORF method
4 // makes use of openMP for easily implemented multiprocessing
\end{verbatim}
for MC simulation
using namespace std;

const int perFactor = 3;
const int NN = 10 * (pow(edgeN + 2 * CUTOFF, 3) - pow(edgeN, 3)) + maxNtemp;
const double rho_bcc = 1. + pow(((double)(edgeN - 1)) / ((double)(edgeN)), 3);
const double rho_fcc = 1. + 3 * pow(((double)(edgeN - 1)) / ((double)(edgeN)), 2);
const int CUT_VOL = 100 * pow(CUTOFF / 1.5, 3) + 1;

double perPOS[NN][3];
double dist_per[maxNtemp][NN];
double rVectors_per[maxNtemp][NN][3];
int locPOS[maxNtemp][CUT_VOL];

int maxNper;
vector<int> perInd;

void sc_generator_new(int edge)
{
    int n = 0;
    double a = 1;
    for (int i = 0; i < edge; i++)
        for (int j = 0; j < edge; j++)
            for (int k = 0; k < edge; k++)
                POS[n][0] = 0 + a * i;
                POS[n][1] = 0 + a * j;
                POS[n][2] = 0 + a * k;
                n++;
}

void fcc_generator_new(int edge)
{
    int n = 0;
    double a = 1;
    for (int i = 0; i < edge; i++)
        for (int j = 0; j < edge; j++)
            for (int k = 0; k < edge; k++)
                POS[n][0] = 0 + a * i;
                POS[n][1] = 0 + a * j;
                POS[n][2] = 0 + a * k;
                n += 4;
}

void bcc_generator_new(int edge)
{
    int n = 0;
    double a = 1;
    for (int i = 0; i < edge; i++)
        for (int j = 0; j < edge; j++)
for (int k = 0; k < edge; k++)
{
    POS[n][0] = 0 + a * i;
    POS[n][1] = 0 + a * j;
    POS[n][2] = 0 + a * k;
    POS[n+1][0] = 0.5 + a * i;
    POS[n+1][1] = 0.5 + a * j;
    POS[n+1][2] = 0.5 + a * k;
    n += 2;
}

void hcp_generator_new (int edge)
{
    int n = 0;
    double a = 1.0;
    for (int i = 0; i < edge; i++)
    {
    for (int j = 0; j < edge; j++)
    {
        for (int k = 0; k < edge; k++)
        {
            POS[n][0] = (2 * i + ((j + k) % 2)) * a / 2.0;
            POS[n][1] = sqrt(3.0) * (j + 1.0 / 3.0 * (k % 2)) * a / 2.0;
            POS[n][2] = 2 * sqrt(6.0) / 3.0 * k * a / 2.0;
            n += 1;
        }
    }
}

bool inn_cube (double x, double y, double z, double marg)
{
    if (x < 0 + marg)
        return false;
    if (x > (edge_N - 1) - marg)
        return false;
    if (y < 0 + marg)
        return false;
    if (y > (edge_N - 1) - marg)
        return false;
    if (z < 0 + marg)
        return false;
    if (z > (edge_N - 1) - marg)
        return false;
    return true;
}

vector<int> find_edgeInd2()
{
    vector<int> EDGE;
    double x, y, z;
    for (int i = 0; i < max_N; i++)
    {
        x = POS[i][0];
        y = POS[i][1];
        z = POS[i][2];
        if (not inn_cube(x, y, z, CUTOFF))
            EDGE.push_back(i);
    }
    return EDGE;
}

bool copyable (double x, double y, double z, int cx, int cy, int cz)
{
    bool trivial = (cx == 0 and cy == 0 and cz == 0);
B. PROGRAMMING

169 if (inn_cube(x, y, z, -sqrt(3)*CUTOFF) and not trivial)
170 return true;
171 return false;
172 }
173
174 vector<array<double, 3>> copies(array<double, 3> original)
175 {
176 vector<array<double, 3>> CAND;
177 double x0 = original[0];
178 double y0 = original[1];
179 double z0 = original[2];
180 double x, y, z;
181 int sgn[3] = {-1, 0, 1};
182 array<double, 3> cand;
183 for (int i = 0; i < 3; i++)
184 {
185 for (int j = 0; j < 3; j++)
186 {
187 for (int k = 0; k < 3; k++)
188 {
189 x = x0 + edge_N*sgn[i];
190 y = y0 + edge_N*sgn[j];
191 z = z0 + edge_N*sgn[k];
192 if (copyable(x, y, z, sgn[i], sgn[j], sgn[k]))
193 {
194 cand[0] = x;
195 cand[1] = y;
196 cand[2] = z;
197 CAND.push_back(cand);
198 }
199 }
200 }
201 }
202 return CAND;
203 }
204 double distance(int i, int j)
205 {
206 double x1, x2, y1, y2, z1, z2, r;
207 x1 = perPOS[i][0];
208 y1 = perPOS[i][1];
209 z1 = perPOS[i][2];
210 x2 = perPOS[j][0];
211 y2 = perPOS[j][1];
212 z2 = perPOS[j][2];
213 r = sqrt(pow(x1-x2, 2)+pow(y1-y2, 2)+pow(z1-z2, 2));
214 return r;
215 }
216 void makePeriodic()
217 {
218 vector<int> EDGE;
219 EDGE = find_edgeInd2();
220 int len = EDGE.size();
221 int edgeInd;
222 cout << len << "of " << maxN << " in edge.\n";
223 if (len >= maxN - 1)
224 {
225  cout << "Superspins will interact with non copies\nnot physical!\n";
226  bool cont;
cout << "Continue regardless?";
if (not cont)
    abort();
}

array<double, 3> orig_pos;
vector<array<double, 3>> copy_vector;

int len2;
int PERlen = 0;
double per_x, per_y, per_z;

for (int i = 0; i < maxN; i++)
{
    perPOS[i][0] = POS[i][0];
    perPOS[i][1] = POS[i][1];
    perPOS[i][2] = POS[i][2];
    perInd.push_back(i);
}

for (int j = 0; j < len; j++)
{
    edgeInd = EDGE[j];
    orig_pos[0] = POS[edgeInd][0];
    orig_pos[1] = POS[edgeInd][1];
    orig_pos[2] = POS[edgeInd][2];
    copy_vector = copies(orig_pos);
    len2 = copy_vector.size();
    for (int i = 0; i < len2; i++)
    {
        per_x = copy_vector[i][0];
        per_y = copy_vector[i][1];
        per_z = copy_vector[i][2];
        perPOS[PERlen+maxN][0] = per_x;
        perPOS[PERlen+maxN][1] = per_y;
        perPOS[PERlen+maxN][2] = per_z;
        perInd.push_back(edgeInd);
        PERlen++;
    }
}

cout << "copies determined.\n";
cout << "Index conversion saved.\n";

int neighb, k;
double d;
onp_set_num_threads(numThreads);
#pragma omp parallel for private(k, neighb, d)
for (int i = 0; i < maxN; i++)
{
    // cout << "here\n";
    neighb = 0;
    for (k = 0; k < maxN + PERlen; k++)
    {
        d = distance(i, k);
        if (i != k and d <= CUTOFF)
        {
            neighb++;
            locPOS[i][neighb] = k;
            dist_per[i][k] = d; // no symmetrization necessary b/c k
runs through ALL indices

double vec[3] = (perPOS[i][0] - perPOS[k][0], perPOS[i][1] - perPOS[k][1], perPOS[i][2] - perPOS[k][2]);
double mod = sqrt(pow(vec[0], 2)+pow(vec[1], 2)+pow(vec[2], 2));
vec[0] /= mod;
vec[1] /= mod;
vec[2] /= mod;
rVectors_per[i][k][0] = vec[0];
rVectors_per[i][k][1] = vec[1];
rVectors_per[i][k][2] = vec[2];

locPOS[i][0] = neighb;

n = ORF(int i, double mu_r, double B, double M)
{
    double Ed = 0;
    double E_RF = 0;
    double sx1 = sx[i];
    double sy1 = sy[i];
    double sz1 = sz[i];
    double Sx = sx1;
    double Sy = sy1;
    double Sz = sz1;

    int len = locPOS[i][0];
    omp_set_num_threads(numThreads2);
    #pragma omp parallel for reduction(+:Ed), reduction(+:Sx), reduction(+:Sy), reduction(+:Sz)
    for (int j = 1; j <= len; j++)
    {
        int neighbInd1 = locPOS[i][j];
        int neighbInd2 = perInd[neighbInd1];
        double sx2 = sx[neighbInd2];
        double sy2 = sy[neighbInd2];
        double sz2 = sz[neighbInd2];
        Sx += sx2;
        Sy += sy2;
        Sz += sz2;

        double d = dist_per[i][neighbInd1];
        double vec[3];

        vec[0] = rVectors_per[i][neighbInd1][0];
        vec[1] = rVectors_per[i][neighbInd1][1];
        vec[2] = rVectors_per[i][neighbInd1][2];
    }

    E_RF = -sx1*(B-0) * Mag * (3*mu_r / (1+2*mu_r));
    E_RF += mu0*Mag*Mag * ((sx1*Sx + sy1*Sy + sz1*Sz) * (1-mu_r)/(2*mu_r+1)/(2*PI*(BEST_CUTOFF*lattice_const)^3));
    if (ONSDAT)
B. PROGRAMMING

```cpp
ofstream ons_file;
ons_file.open("ons.dat", ios::app);
ons_file << Ed << "t" << ERF << endl;
ons_file.close();
return Ed + E_RF;
```

B.1.6 rnd250.c

```cpp
#include "rnd.h"
struct st_rnd250 Rnd250;

void seed250 (long seed)
{
   Initialise den Zufallszahlsgenerator. Hierzu wird ein Modulo-
   Zufallszahlsgenerator benutzt. Um die lineare Unabhä ngigkeit der
   einzelnen Bits zu garantieren werden nachträglich noch Bitmasks
   einem Teil der Daten überlagert. Um die dadurch verursachten An-
   fangskorrelationen zu beseitigen werden die ersten Zufallszahlen
   verworfen.
   Parameter: seed --- Startwert
   seed = 1;

   for (i =0; i < 250; ++i)
   {
      k = seed / 127773;
      seed = 16807 * (seed - k*127773) - 2836 * k;
      if (seed < 0) seed += 0x7FFFFFFF;
      Rnd250.field[i] = seed;
   }

   k = 0x7FFFFFFF;
   j = 0x40000000;
   for (i=1; i<250; i+=8)
   { Rnd250.field[i] = (Rnd250.field[i] & k) | j; }

   Rnd250.point = 249;
   for (i=0; i<4711; ++i)
   { rnd250(); }
}
```

B.1.7 random-spd5.h

```cpp
const double PI = 3.1415926535897932385;
```
B. PROGRAMMING

double GaussDis(double, double, double);
const double realRND250_MAX = (double) MAXRAND250;

/∗ generate a random number in [0,1) ∗/
double rand0_0999()
{
  return (rnd250() / (realRND250_MAX+1.0));
}

/∗ generate a random number in (0,1) ∗/
double rand0_0001_0999()
{
  return ((rnd250() +1.0) / (realRND250_MAX+2.0));
}

/∗ generate a random number in [0,1] ∗/
double rand0_1()
{
  return (rnd250() / realRND250_MAX);
}

/∗ yields a 3D random unit vector ∗/
void Marsaglia(double ∗V)
{
  double rsq, y1, y2;
  do
  {
    y1 = rand0001_0999() * 2.0 - 1.0;
    y2 = rand0001_0999() * 2.0 - 1.0;
    rsq = y1 * y1 + y2 * y2;
  }
  while (rsq > 1.0);
  V[0] = 2.0 * y1 * sqrt((1.0 - rsq));
  V[1] = 2.0 * y2 * sqrt((1.0 - rsq));
  V[2] = 1.0 - 2.0 * rsq;
}

double GaussDis(double x, double avg, double sig)
{
  double fct;
  fct = (1.0/(sqrt(2.0*PI)*sig))*exp(-0.5*(x-avg)*(x-avg)/(sig*sig));
  return (fct);
}

/∗ yields a 3D unit vector, but with Gaussian distribution in y,z-coordinates <-> anisotropy axis distribution — easy axis in x-direction ∗/
void AnisotropyAxis(double ∗V)
{
  double rsq, y, z; y2, z2, ss;
  do
  {
    do
      {
        y = rand0_1() * 2.0 - 1.0;
        y2 = rand0_1();
      }
    while (y2 > GaussDis(y, y_avg, y_sig));
    do
      {
        z = rand0_1() * 2.0 - 1.0;
        z2 = rand0_1();
      }
B.2  domainFinder.cpp

This is the programme used to generate the visualizations of 3D spin landscapes that are presented in the later parts of the thesis. This programme finds sublattices in a data set of spin positions and -orientations which are parallel, and analyzes size and orientation of all these sublattices within one set. In conjunction with gnuplot, an entire set of spins can be visualized and via colorization by the programme, these images can be interpreted by the user even if thousands of spins in three dimensions are displayed at once.

Algorithm design and implementation in C++11 were done by the author.
data.open(filename);
std::string line;

for (int i = 0; i < num*maxN; i++)
{
    getline(data, line);
    double col1, col2, col3, col4, col5, col6;
    stringstream ss(line);
    ss >> col1 >> col2 >> col3 >> col4 >> col5 >> col6;
}

int counter = 0;
for (int j = num*maxN; j < (num+1)*maxN; j++)
{
    getline(data, line);
    double col1, col2, col3, col4, col5, col6;
    stringstream ss(line);
    ss >> col1 >> col2 >> col3 >> col4 >> col5 >> col6;
    POS[counter][0] = col1;
    POS[counter][1] = col2;
    POS[counter][2] = col3;
    VEC[counter][0] = col4;
    VEC[counter][1] = col5;
    VEC[counter][2] = col6;
    counter++;
}
data.close();

// calculate distance btw two positions
double distance(int i, int j)
{
    double x1, x2, y1, y2, z1, z2;
    x1 = POS[i][0];
    y1 = POS[i][1];
    z1 = POS[i][2];
    x2 = POS[j][0];
    y2 = POS[j][1];
    z2 = POS[j][2];
    return sqrt(pow(x1-x2, 2) + pow(y1-y2, 2) + pow(z1-z2, 2));
}

// scalar product btw two vectors
double scp(int i, int j)
{
    double vx1, vx2, vy1, vy2, vz1, vz2;
    vx1 = VEC[i][0];
    vy1 = VEC[i][1];
    vz1 = VEC[i][2];
    vx2 = VEC[j][0];
    vy2 = VEC[j][1];
    vz2 = VEC[j][2];
    return vx1*vx2+vy1*vy2+vz1*vz2;
}

// calculate center of mass of a given vector of sites
vector<double> POS_center(vector<int> sample)
{
    double center[3];
    vector<double> result;
    center[0] = 0;
    center[1] = 0;
    center[2] = 0;
    int len = sample.size();
    for (int i = 0; i < len; i++)
{ center[0]+=POS[sample[i][0]]; center[1]+=POS[sample[i][1]]; center[2]+=POS[sample[i][2]]; }
result.push_back(center[0]/len);
result.push_back(center[1]/len);
result.push_back(center[2]/len);
return result;
}

// calculate distance of two vectors (NOT indices)
double center_dist(vector<double> a, vector<double> b) {
  double x1, x2, y1, y2, z1, z2;
  x1 = a[0];
  y1 = a[1];
  z1 = a[2];
  x2 = b[0];
  y2 = b[1];
  z2 = b[2];
  return sqrt(pow(x1-x2,2) + pow(y1-y2,2) + pow(z1-z2,2));
}

// ignore vectors that aren't unit vectors.
void test_data() {
  for (int i = 0; i < maxN; i++)
    if (scp(i,i) < 0.99 or scp(i,i) > 1.01)
      { VEC[i][0] = 0;
        VEC[i][1] = 0;
        VEC[i][2] = 0;
        cout << i+1 << " ignored!\n";
      }
  }

// fill distance matrix
void dist_matrix()
  {
    // upper triangle matrix
    for (int i = 0; i < maxN-1; i++)
      { for (int j = i+1; j < maxN; j++)
          dist[i][j] = distance(i,j);
        }

    // symmetry i<>j
    for (int i = 0; i < maxN-1; i++)
      { for (int j = i+1; j < maxN; j++)
          dist[j][i] = dist[i][j];
        }
  }

// return list of neighbours within max_dist radius
vector<int> neighbours(int i, double max_dist)
  {
    vector<int> neighb;
    for (int j = 0; j < maxN; j++)
      { if (dist[i][j] <= max_dist)
          { // cout << i << " NB von " << j << endl;
            neighb.push_back(j);
          }
        }
    return neighb;
  }

// Calculate avg direction of a domain given its index in DOMAINS
vector<double> vec_mean(int domain)


```cpp
double new_mean[3];
new_mean[0] = 0;
new_mean[1] = 0;
new_mean[2] = 0;

vector<double> result;

int len = DOMAINS[domain].size();

for (int i = 0; i < len; i++)
{
    new_mean[0]+=VEC[DOMAINS[domain][i]][0];
    new_mean[1]+=VEC[DOMAINS[domain][i]][1];
    new_mean[2]+=VEC[DOMAINS[domain][i]][2];
}

/*
new_mean[0] = pow(new_mean[0], 1./len);
new_mean[1] = pow(new_mean[1], 1./len);
new_mean[2] = pow(new_mean[2], 1./len);
*/

double mod = sqrt(pow(new_mean[0], 2)+pow(new_mean[1], 2)+pow(new_mean[2], 2));

for (int i = 0; i < 3; i++)
    result.push_back(new_mean[i]/mod);

return result;
```

```cpp
// scalar product of a given vector old and a vector at index new_ind
double scp_vec_ind(vector<double> old, int new_ind)
{
    double vx1, vx2, vy1, vy2, vz1, vz2;
    vx1 = old[0];
    vy1 = old[1];
    vz1 = old[2];
    vx2 = VEC[new_ind][0];
    vy2 = VEC[new_ind][1];
    vz2 = VEC[new_ind][2];
    return vx1*vx2+vy1*vy2+vz1*vz2;
}
```

```cpp
// scalar product of two directly given vectors
double dom_dom(vector<double> a, vector<double> b)
{
    double vx1, vx2, vy1, vy2, vz1, vz2;
    vx1 = a[0];
    vy1 = a[1];
    vz1 = a[2];
    vx2 = b[0];
    vy2 = b[1];
    vz2 = b[2];
    return vx1*vx2+vy1*vy2+vz1*vz2;
}
```

```cpp
// return list of "good" neighbors among neighbors
vector<int> good_neighbours(vector<double> dom_mean, vector<int> neigh, double min_corr, int domain)
{
    int n;
    int len = neigh.size();
    vector<int> goodNs;
    for (int i = 0; i < len; i++)
    {
        n = neigh[i];
        // if (scp(n, j) >= min_corr and not forbidden[n])
        if (scp_vec_ind(dom_mean, n) >= min_corr and not forbidden[n])
        {
            forbidden[n] = true;
            DOMAINS[domain].push_back(n);
            // cout << n << " found as good neighbor\n";
            goodNs.push_back(n);
        }
    }
    return goodNs;
}
```
B. PROGRAMMING

291 } // flag if a site is isolated = w/o a single good neighbour = break recursive loop in
292
293 if (goodNs.size() == 0)
294 goodNs.push_back(-1);
295
296 return goodNs;
297 }

298 // Central algorithm: Recursively, find 'good neighbours' among connected and not yet domain-
299 associated vectors

300 void findDomain(int start, double max_dist, double min_corr, int domain)
301 {
302 vector<int> goodNs;
303 vector<int> neigh;
304 vector<double> dom_mean;
305
306 int gn_size;
307 int new_start;

308 // continuous update of what would be considered a good neighbour from avg(!!!)
309 orientation of current domain members
310 if (DOMAINS[domain].size() > 1)
311 dom_mean = vec_mean(domain);
312 else
313 {
314
315 dom_mean.push_back(VEC[start][0]);
316 dom_mean.push_back(VEC[start][1]);
317 dom_mean.push_back(VEC[start][2]);
318 }

319 neigh = neighbours(start, max_dist);
320 goodNs = good_neighbours(dom_mean, neigh, min_corr, domain);
321 gn_size = goodNs.size();
322
323 for (int i = 0; i < gn_size; i++)
324 {
325 new_start = goodNs[i];
326 if (new_start != -1)
327 {
328 // cout << " start findDomain from " << new_start << endl;
329 findDomain(new_start, max_dist, min_corr, domain);
330 }
331 }

332 // print a single vector's coordinates for gnuplot to ofile_name (!!!)
333 void print_coord(int i)
334 {
335 ofstream ofile;
336 ofile.open(ofile_name, ios::app);
337 double x, y, z, vx, vy, vz;
338 double scale = 0.5;
339 vx = scale * VEC[i][0];
340 vy = scale * VEC[i][1];
341 vz = scale * VEC[i][2];
342 x = POS[i][0] - 0.5 * vx;
343 y = POS[i][1] - 0.5 * vy;
344 z = POS[i][2] - 0.5 * vz;
345
346 ofile << x << "\t" << y << "\t" << z << "\t" << vx << "\t" << vy << "\t" << vz << endl;
347 ofile.close();

348 // print a single vector's coordinates for gnuplot to ofile2_name (!!!)
349 void print_coord_alt(int i)
350 {
351 ofstream ofile2;
352 ofile2.open(ofile2_name, ios::app);
353 double x, y, z, vx, vy, vz;
354 vx = VEC[i][0];
vy = VEC[i][1];

vx = VEC[i][2];

x = POS[i][0] - 0.5*vx;

y = POS[i][1] - 0.5*vy;

z = POS[i][2] - 0.5*vy;

ofile2 << x << "\t" << y << "\t" << z << "\t" << vx << "\t" << vy << "\t" << vs << endl;

ofile2.close();

}

// generate the complete gnuplot script to visualize all large domains

void gnu_script(vector<vector<int>> data)
{
    ofstream gnu;
    gnu.open(gnu_file);

    vector<int> param;

    int lines = data.size();

    for(int i = 0; i < lines; i++)
        param.push_back(data[i].size());

    gnu << "reset" << endl;
    gnu << "set view equal xyz" << endl;
    gnu << "set title " << endl;
    gnu << "plot " << endl;

    char c = '"';

    int von = 0;
    int bis = param[0]-1;

    for(int i = 0; i < lines; i++)
    {
        gnu << c << ofile_name << c;  
        gnu << von << c << bis << c;  
        gnu << title << c;  
        von += param[i];
        bis += param[i+1];
    }

    gnu << endl;
    gnu.close();
}

// MAIN FUNCTION

int main()
{

    int number;
    cout << "Which part?";
    cin >> number;
    // Some initializations.
    dcounter=0;

    ofstream ofile;
    ofstream ofile2;
    ofstream gnu;
    ofile.open(ofile_name, ios::trunc);
    ofile.close();
    ofile2.open(ofile2_name, ios::trunc);
    ofile2.close();
    gnu.open(gnu_file, ios::trunc);
    gnu.close();
    cout << "Output files reset!\n";

    vector<vector<int>> LARGEDOMAINS;
    vector<vector<double>> LARGEDOMAINs_AXIS;

452  
453 // read and test input file
454  
455 read_data(data_file, number);
test_data();
cout << "data file read...\n";
456  
457 // Every spin is eligible for a domain and all distances are stored
458 for (int i = 0; i < max_N; i++)
459  
460 {  
461     neg[i] = true;
462     forbidden[i] = false;
463 }
464  
465  
466 // Necessary initialization of vectors
467 vector<int> placeholder;
468  
469 // Call the central algorithm until every site has been assigned to an element of
470 // DOMAINS or has been found as isolated
471 for (int i = 0; i < max_N; i++)
472  
473 {  
474     cout << "Test " << i << endl;
475     if (not forbidden[i])
476  
477  
478     
479     
480     DOMAINS.push_back(placeholder);
dcounter++;
findDomain(i, MAX_DIST, MIN_CORR, dcounter);
481  
482 }
483  
484  
485  
486 cout << "How accurate?\n";
487 cin >> ACC;
488  
489 // From the partitioning, remove deviating spins and find LARGE domains (minimum number
490 // of members) and store them together with their orientation in LARGEDOMAINS
491 double ratio = 0;
492 int DCOUNTER = 0;
493  
494 double x, y, z;
495  
496 vector<double> dom_mean;
497 int DOMsize = DOMAINS.size();
498  
499 for (int i = 0; i < DOMsize; i++)
500  
501 
502 
503  
504  
505  
506  
507  
508  
509  
510 
511  
512  
513 
514 
515 
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531
cout << "Sublattice \omega \) << DCOUNTER << "\omega \) size: \) << len << "\omega \) direction: \) << x << "\omega \) << y << "\omega \) << z << endl;

for (int j = 0; j < len; j++)
    print_coord(v[j]);
// of file:
}
}

ratio /= max_N;
cout << DCOUNTER << "sublattices found! \n";
cout << "Sublattice in/out ratio = " << ratio << endl;

// Among the significant domains, find "interesting" domains, i.e. w/ anti-parallel or perpendicular orientation

int large_size = LARGEDOMAINS.size();

int pair_counter = 0; // count all pairs (perp, parall, anti-parall)
int anti_counter = 0; // count anti-pairs
int red_counter; // count almost parallel pairs
int RED_COUNTER = 0;
if (large_size >= 2)
{
    for (int i = 0; i < large_size - 1; i++)
    {
        red_counter = 0;

        for (int j = i+1; j < large_size; j++)
        {
            vector<double> a = LARGEDOMAINS_AXIS[i];
            vector<double> b = LARGEDOMAINS_AXIS[j];
            vector<int> v1 = LARGEDOMAINS[i];
            vector<int> v2 = LARGEDOMAINS[j];
            // if (center_dist(POS_center(v1), POS_center(v2)) <= min_size*MAX_DIST)
            if (dom_dom(a, b) <= -ACC * sqrt(0.5*(MIN_CORR+1)) || abs(dom_dom(a, b)) <= perp_corr)
            {
                pair_counter++;
                cout << i+1 << " vs " << j+1 << " interesting! \n";
                if (dom_dom(a, b) <= -ACC * sqrt(0.5*(MIN_CORR+1)))
                {
                    cout << "(anti-parall) \n";
                    anti_counter++;
                }
                else
                {
                    cout << "(perpendicular) \n";
                    /*
                        int len1 = v1.size();
                        int len2 = v2.size();
                        for (int k = 0; k < len1; k++)
                            print_coord_interesting(v1[k]);
                        for (int k = 0; k < len2; k++)
                            print_coord_interesting(v2[k]);
                        */
                }
            }
            else
            {
                if (dom_dom(a, b) >= +ACC * sqrt(0.5*(MIN_CORR+1)))
                {
                    pair_counter++;
                    cout << i+1 << " vs " << j+1 << " interesting! \n";
                    cout << "(almost_parall) \n";
                    red_counter++;
                }
            }
        }
    }
}
B. PROGRAMMING

```cpp
RED_COUNTER += red_counter;

if (2 * anti_counter == large_size - RED_COUNTER)
    cout << "!!! Each sublattice has exact antiparallel partner !!!\n";

// In case any large domains were found, produce the complete gnuscript for colored visualization

if (large_size > 1)
    { gnu_script(LARGEDOMAINS);
        for (int i = 0; i < max_N; i++)
            { if (neg[i])
                print_coord_alt(i);
            }
    } return 0;
```
C References


Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den