Unterschrift BetreuerIn

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Noise Sources and Analysis of Heat Assisted Recording

ausgeführt am Institut für Festkörperphysik
der Technischen Universität Wien

unter der Anleitung von
Assoc.Prof. Privatdoz. Dipl.-Ing. Dr.techn. Dieter Suess

durch

Florian Slanovc

Dornbacher Straße 119,
1170 Wien

18. Dezember 2018

Unterschrift StudentIn

Zusammenfassung

Die Festplattenindustrie ist ständig auf der Suche nach Technologien, welche die Schreib- und Lesegeschwindigkeiten erhöhen, Materialkosten senken oder Speicherdichte erhöhen. Da Weiterentwicklungen oft nicht alle dieser Vorteile in gleichem Maße abdecken, ist es erforderlich, auch auf den gewünschten Anwendungsbereich zu berücksichtigen. Oft ist es das Ziel, möglichst viel Speicher mit langer Lebenszeit auf engstem Raum preiswert unterbringen zu können. Heat Assisted Magnetic Recording (HAMR) ist eine Technik, mit der Bits auf magnetische Speichermedien geschrieben werden können. Dabei wird das magnetische Material kurzzeitig an bzw. über die Curie-Temperatur erhitzt, um die Ummagnetisierung zu erleichtern. Der Vorteil gegenüber einem herkömmlichen Schreibprozess, in welchem die Ummagnetisierung ausschließlich durch das angelegte externe Magnetfeld geschieht, ist eine feinere räumliche Auflösung und damit verbundene größere lineare Dichte des Bitmusters. Hierbei können unter anderem auch herkömmliche granulare Speichermedien verwendet werden, was die Materialkosten in Grenzen hält.

Damit eine derartige Technologie auch für die praktische Anwendung verwendet werden kann, ist es nötig, die Qualität des erhalten Bitmusters bei einem Schreibprozess zu untersuchen. Damit die Bits im Leseprozess problemlos detektiert werden können, ist eine entsprechend hohe Signal-to-noise ratio (SNR) erforderlich. Das Ziel dieser Arbeit ist es, die Einflüsse verschiedenster Materialeigenschaften der magnetischen Körner auf die SNR zu erkunden. Diese Resultate sollen helfen, bei der Auswahl von Materialzusammensetzungen die höchstmögliche SNR zu erzielen.

Das charakteristischen Verhalten eines zylindrischen magnetischen Korns mit bestimmtem Durchmesser in einem HAMR-Schreibprozess kann mittels eines 2D-Phasendiagramms beschrieben werden. Er beinhaltet die Ummagnetisierungswahrscheinlichkeit des Korns in Abhängigkeit von Maximaltemperatur des Hitzepulses und Phasenddifferenz zwischen Hitzepuls und angelegtem Magnetfeld. Im Rahmen dieser Arbeit werden die Ummagnetisierungswahrscheinlichkeiten auf ein granulares Medium übertragen und anschließend der Leseprozess simuliert, um aus dem erhaltenen Signal die SNR zu bestimmen. Da diese in Abhängigkeit der Gestalt des Phasendiagramms entsprechend variiert, wurde mithilfe eines mathematischen Modells basierend auf den charakteristischen Parametern die Gestalt des Phasendiagramms modelliert. In diesem Modell lassen sich die Parameter jeweils unabhängig voneinander variieren und damit deren Einfluss auf die SNR gezielt untersuchen. Die erhaltenen Resultate geben Auskunft darüber, welche Gestalt ein entsprechendes Phasendiagramm haben sollte, um damit Bits mit einer ansprechenden SNR

schreiben zu können.

Abgesehen von diesen Untersuchungen in Kapitel 6 beschäftigt sich die Arbeit auch mit den theoretischen Modellen zur numerischen Simulation von Ummagnetisierungswahrscheinlichkeiten eines einzelnen magnetischen Korns in Kapitel 3 und der dafür erforderlichen Zeitintegrationsverfahren von stochastischen Differentialgleichungen in Kapitel 4. In Anhang A ist außerdem ein mathematischer Exkurs zu räumlichen Isotropie von Bilinearformen gemacht. In Anhang C und D findet man eine Dokumentation des Sourcecodes, der für die Übertragung der Ummagnetisierungswahrscheinlichkeiten aus dem Phasendiagramm auf ein granulares Medium und anschließende SNR-Berechnung mittels des SNR-Calculator von SEAGATE verwendet werden kann.

Abstract

The hard disk industry is constantly on the lookout for technologies that increase write and read speeds, reduce material costs or increase storage density. Since the further developments often do not cover all these demands to the same extent, it is necessary to consider also the desired field of application. It is often the goal to store as much memory as possible with a long service life in the smallest space and at a reasonable price. Heat Assisted Magnetic Recording (HAMR) is a technique for writing bits onto magnetic storage media. The magnetic material is briefly heated to or above the Curie temperature to facilitate remagnetization. The advantage over a conventional writing method, where the magnetization is performed exclusively by the applied external magnetic field, is a finer spatial resolution and the associated higher linear density of the bit pattern. Among other things, conventional granular storage media can also be used, which keeps material costs within reasonable limits.

For such a technology to be applied in practice, it is necessary to examine the quality of the bit pattern obtained during a write operation. A high signal-to-noise ratio (SNR) is required so that the bits can be easily recognized during the reading process. The aim of this work is to investigate the influence of different material properties of the magnetic grains on the SNR. These results should help to achieve the highest possible SNR when selecting the material composition.

The characteristic behavior of a cylindrical magnetic grain with a certain diameter in a HAMR writing process can be described with a 2D phase diagram. It contains the probability that the grain will be magnetized depending on the maximum temperature of the thermal pulse and the phase difference between the thermal pulse and the applied magnetic field. In this paper, the probabilities of remagnetization are transferred to a granular medium and then the reading process is simulated to determine the SNR from the received signal. Since this varies according to the shape of the phase diagram, the diagram was modelled using a mathematical model based on the characteristic parameters. In this model, the parameters can be varied independently of each other and their influence on the SNR can be specifically investigated. The results obtained provide information on how a phase diagram should look in order to be able to write bits with an attractive SNR.

In addition to these investigations in Chapter 6, the work also deals with the theoretical models for the numerical simulation of the remagnetization probabilities of a single magnetic grain in Chapter 3 and the time integration methods of stochastic differential equations required for this in Chapter 4. Appendix A also contains a

mathematical excursus on the spatial isotropy of bilinear forms. Appendices C and D contain a documentation of the source code which can be used for the transfer of the magnetization probabilities from the phase diagram to a granular medium and the subsequent SNR calculation with the SNR calculator of SEAGATE.

Contents

Chapter 1

Introduction

Heat Assisted Magnetic Recording (HAMR) is a potential writing procedure for hard disks, which might be practicable to reach an even higher storage density than classical perpendicular writing. Whereas in perpendicular writing just a magnetic field is used to polarize the bits (see figure 1), HAMR uses an additional laser pulse to heat the material up to its Curie temperature to ease the writing process. As shown in [11] this technique allows to write the bits very precisely what leads to a higher storage density.

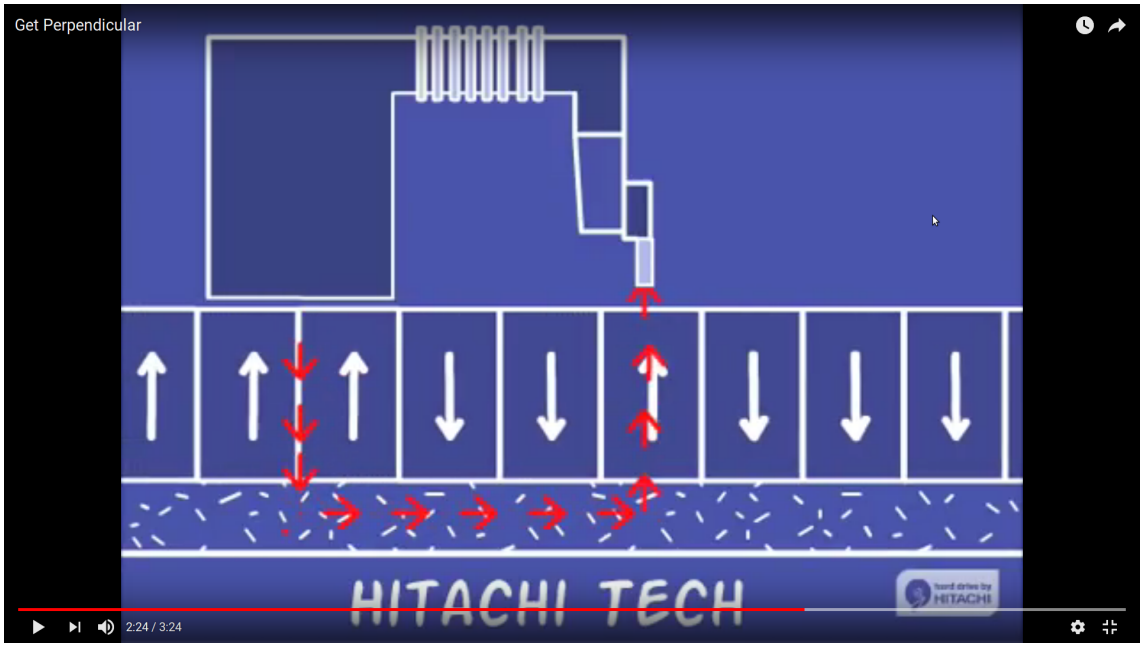


Figure 1.1: Visualization of a classical writing head in perpendicular writing. The writer applies a magnetic field to the storage medium and polarizes the magnetic material in the required direction. The screen shot is taken from a promotion video for the perpendicular writing technique made by Hitachi Global Storage Technologies with the title "Get Perpendicular". It was uploaded on July 9th, 2008 and can be reached under `https://www.youtube.com/watch?v=xb_PyKuI7II`[1].

The goal of this thesis is the investigation of the resulting signal-to-noise-ratio (SNR) in a writing process in HAMR. A high SNR is a quality feature of written bits because it is the ratio of the mean magnetic signal and the variance of repeated writing processes which are detected by the reader system (see [15]).
Therefore the tasks are:

- Simulation of a writing process for a single grain and investigation of the switching probability.
- Extension of switching probability calculation to a media containing a multitude of grains.
- Reading the magnetic signal of the material and calculating the SNR.
- Investigate, how the material properties affect the switching probabilities and therefore the resulting SNR.

We also describe the theoretical background where the upper steps are based on.

[1] last access on November 16th, 2017

Chapter 2

Units and constants

In the introductory tables we present the most important variables with units and physical constants, which are going to be used in this thesis. We mainly choose SI-units. Variables in bold letters are three dimensional vectors.

variable	description	unit
μ_S	atomic magnetic moment	J/T
J	exchange energy between atoms	J
k_1	anisotropy constant in z direction	J
γ	gyromagnetic ratio	$1/(\mathrm{s}^{-1}\mathrm{T}^{-1})$
λ	damping constant	1
$\vec{\boldsymbol{H}}$	magnetic field	T

constant	description	value
k_B	Boltzmann constant	$1.38064853 \cdot 10^{-23}$ J/K
μ_B	Bohr magneton	$9.274009995 \cdot 10^{-24}$J/T

Chapter 3

Modelling and theoretical background

In this chapter we give a summary over the theoretical background of our simulations. We focus on the mention of all parameters we are going to use in the numerical simulations afterwards, as well as the necessary assumptions and approximations. The source of this chapter is mainly [1, sections 2-4], others are explicitly mentioned. The main assumption is the validity of the extended atomistic 3D-Heisenberg spin model.

3.1 Spin Hamiltonian

To deal with the energy balance of a magnetic material we need to find a suitable Hamilton operator that describes the magnetic interaction between the spins of the atoms electrons. As it is cited in [1, section 2], there are three dominating parts which have to be considered, namely

- the exchange interaction $\mathcal{H}_{\text{exc}}$,
- the magnetic anisotropy $\mathcal{H}_{\text{ani}}$,
- the interaction with an external magnetic field $\mathcal{H}_{\text{app}}$.

So the total Hamiltonian $\mathcal{H}$ can be written as

$$\mathcal{H} = \mathcal{H}_{\text{exc}} + \mathcal{H}_{\text{ani}} + \mathcal{H}_{\text{app}}. \tag{3.1.1}$$

In the following sections we describe those the individual parts in more detail. From now on we assume that every local spin moment is written as a unit vector $\vec{\boldsymbol{S}}_i = \begin{pmatrix} S_x^i & S_y^i & S_z^i \end{pmatrix}^T$ which points in the direction of the actual atomic moment $\vec{\boldsymbol{\mu}}_s$ induced by the spin.

3.1.1 Exchange interaction

As discussed in [1, section 2.1], the exchange term $\mathcal{H}_{\text{exc}}$ is the clearly dominating term in (3.1.1) in the temperature range 300-1200 K. In general the exchange term

can be written in the form

$$\mathcal{H}_{\text{exc}} = -\sum_{i \neq j} \vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j, \tag{3.1.2}$$

where $\boldsymbol{J} \in \mathbb{R}^{3\times 3}$ describes the tensorial components of the exchange interaction in Joules [J] and the sum is taken over all atoms in the material. For our applications it will be sufficient to make the following assumptions:

- The exchange term only contains summands that describe the interaction between nearest neighbors:
  In most materials the nearest neighbors interaction is clearly dominating the interaction between atom pairs which are further apart in the lattice. Therefore the sum in (3.3.1) is taken only over $i \neq j$ of nearest neighbors atoms and not of arbitrary atoms in the material. We use the notation $\sum_{i,j}^{\text{nn}}$ for that.

- The exchange interaction is isotropic in the direction of space:
  The single summands in (3.3.1) then only depend on the angular orientation of $\vec{\boldsymbol{S}}_i$ and $\vec{\boldsymbol{S}}_j$ and can therefore be written as a multiple of their scalar product in the form $J_{ij}\vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j$ with a scalar $J_{ij} \in \mathbb{R}$. [1]

- The scalar value J_{ij} in the previous point has the same value for all couples of nearest neighbors atoms:
  This assumption may be true in simple lattices of quite high symmetric and few or similar atoms in the crystal basis. Then the value of J_{ij} is independent of the specific atoms indices i and j and can therefore be replaced by a global parameter J in (3.3.1).

Under consideration of the previous assumptions we can finally rewrite the exchange Hamiltonian in (3.3.1) as

$$\mathcal{H}_{\text{exc}} = -J \sum_{i,j}^{\text{nn}} \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j. \tag{3.1.3}$$

3.1.2 Magnetic anisotropy

The exchange interaction in the final form (3.1.3) depends only on the relative orientation between the spins, therefore a global rotation of all spins in the system yields the same energy, what makes the system thermally very unstable. The thermal stability of ferromagnetic systems is usually archived by a globally preferred spin direction - the so called easy axis. It is modeled by a direction in form of a constant unit vector $\boldsymbol{e}_z$, which points (without loss of generality) in the direction of the z-axis. Spins which are not parallel (or antiparallel) to $\boldsymbol{e}_z$ should be energetically disadvantaged, therefore we write the term in the form

$$\mathcal{H}_{\text{ani}} = -k_1 \sum_i (\vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{e}}_z)^2 \tag{3.1.4}$$

with an uniaxial anisotropy constant k_1 in Joules [J] per atom and a sum over all atoms i.

[1] The proof of this theorem can be found in appendix A.1.3.

3.1.3 External magnetic field

The interaction with an external magnetic field $\vec{\boldsymbol{H}}_{ext}$ can be described by

$$\mathcal{H}_{\text{ext}} = -\mu_S \sum_i \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{H}}_{ext} \tag{3.1.5}$$

where we have already assumed that the norm of the atomic magnetic moment μ_S in Joules per Tesla [J/T] is independent of i and therefore constant for all atoms. Note that we adopt the notation of [1] and therefore here and in the following the magnetic field $\vec{\boldsymbol{H}}$ is always measured in Tesla [T] which leads to a Hamiltonian measured in Joule [J].

3.2 Landau-Lifshitz-Gilbert (LLG) equation

For our intended simulations, we need to investigate the time dependent behavior of the spins $\vec{\boldsymbol{S}}_i$. Therefore we use the atomistic LLG equation, which is given in [1, (13)] by

$$\frac{\partial \vec{\boldsymbol{S}}_i}{\partial t} = -\frac{\gamma}{1+\lambda^2}\left[\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{H}}^i_{\text{eff}} + \lambda \vec{\boldsymbol{S}}_i \times \left(\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{H}}^i_{\text{eff}}\right)\right], \tag{3.2.1}$$

with describes the time dependent motion of the spin i in an effective field $\vec{\boldsymbol{H}}^i_{\text{eff}}$. This field is given by the total Hamiltonian $\mathcal{H}$ in (3.1.1) and a thermal field

$$\vec{\boldsymbol{H}}^i_{\text{th}} = \vec{\boldsymbol{\Gamma}}(t)\sqrt{\frac{2\lambda k_B T}{\gamma \mu_s \Delta t}}$$

that describes the thermal fluctuations as a Gaussian white noise term (see also [1, 4.2]). The three components of $\vec{\boldsymbol{\Gamma}}(t)$ are Gaussian distributions with mean of zero and time dependent standard deviations. Increasingly fluctuations at higher temperatures are therefore described via increasing standard variations. The total effective field thus can be written as

$$\vec{\boldsymbol{H}}^i_{\text{eff}} = -\frac{1}{\mu_s}\frac{\partial \mathcal{H}}{\partial \vec{\boldsymbol{S}}_i} + \vec{\boldsymbol{H}}^i_{\text{th}}. \tag{3.2.2}$$

The time dependent spin motion described by (3.2.1) contains two distinct physical effects:

- The precession of the magnetization arises due the quantum mechanical interaction of an atomic spin with the applied field (first term in (3.2.1)).
- The relaxation of the magnetization represented by a coupling of the magnetization to a heat bath. It forces the magnetization vectors to align in the direction of the field with a coupling strength determined by the microscopic damping parameter λ.

3.3 Landau-Lifshitz-Bloch (LLB) equation

As in [2] described, the LLB equation summarizes the microscopic spins to a macrospin of a nanoparticle and describes its average spin polarization $\vec{\boldsymbol{m}} := \vec{\boldsymbol{M}}/M_s^0$, where $\vec{\boldsymbol{M}}$ is the total magnetization of the particle and M_s^0 is the saturation magnetization value at temperature $T = 0$ K. The motion of $\vec{\boldsymbol{m}}$ is then described by the following equation:

$$\begin{aligned}\frac{\partial \vec{\boldsymbol{m}}}{\partial t} =& \gamma \left[\vec{\boldsymbol{m}} \times \vec{\boldsymbol{H}}_{\text{eff}}\right] + \frac{|\gamma|\alpha_{\parallel}}{m^2}\left(\vec{\boldsymbol{m}} \cdot \vec{\boldsymbol{H}}_{\text{eff}}\right)\vec{\boldsymbol{m}} \\ &- \frac{|\gamma|\alpha_{\perp}}{m^2}\left\{\vec{\boldsymbol{m}} \times \left[\vec{\boldsymbol{m}} \times \left(\vec{\boldsymbol{H}}_{\text{eff}} + \vec{\boldsymbol{\eta}}_{\perp}\right)\right]\right\} + \vec{\boldsymbol{\eta}}_{\parallel}\end{aligned}$$

For the dimensionless damping parameters in parallel and longitudinal direction $\alpha_{\parallel}$ and $\alpha_{\perp}$ holds

$$\alpha_{\parallel} := \lambda \frac{2T}{3T_C}$$

and

$$\alpha_{\perp} := \begin{cases} \lambda\left(1 - \frac{T}{3T_C}\right) & T < T_C \\ \alpha_{\parallel} & T \geq T_C \end{cases}$$

with the Curie-Temperature T_C. Similar to the previous chapter, the effective field $\vec{\boldsymbol{H}}_{\text{eff}}$ contains various components as described in [10, (5)], e.g. exchange and anisotropy field $\vec{\boldsymbol{H}}_{\text{exc}}$ and $\vec{\boldsymbol{H}}_{\text{ani}}$. Those have the form

$$\vec{\boldsymbol{H}}_{\text{exc}} = \begin{cases} \frac{1}{2\widetilde{\chi}_{\parallel}}\left(1 - \frac{m^2}{m_e^2}\right)\vec{\boldsymbol{m}}, & T \lesssim T_C \\ -\frac{1}{\widetilde{\chi}_{\parallel}}\left(1 + \frac{3}{5}\frac{T_C}{T - T_C}m^2\right)\vec{\boldsymbol{m}}, & T \gtrsim T_C. \end{cases} \tag{3.3.1}$$

where m_e is the zero-field equilibrium spin polarization for a given temperature and with the assumption of an easy axis in z-direction

$$\vec{\boldsymbol{H}}_{\text{ani}} = -\frac{1}{\widetilde{\chi}_{\perp}}(m_x\vec{\boldsymbol{e}}_x + m_y\vec{\boldsymbol{e}}_y) \tag{3.3.2}$$

with magnetic susceptibilities $\widetilde{\chi}_{\parallel}$ and $\widetilde{\chi}_{\perp}$. The additional fluctuation terms $\vec{\boldsymbol{\eta}}_{\perp}$ and $\vec{\boldsymbol{\eta}}_{\parallel}$ are represented by random numbers with suitable distributions conform to the fluctuation-dissipation theorem (given in [2, (6)]). The variations to those random numbers are, among others, directly proportional to the temperature T and indirectly proportional to the volume V of the particle.

Chapter 4

Time integration in VAMPIRE

Thus (3.2.1) is a time dependent stochastic differential equation, it is important to choose a proper time integration method to calculate the spin motions numerically. In the following we present two methods, which are provided by VAMPIRE. We therefore investigate the stability of the time integration methods of the whole LLG equation

$$\frac{\partial \vec{\boldsymbol{S}}_i}{\partial t} = \underbrace{-\frac{\gamma}{1+\lambda^2}\left[\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{H}}^i_{\text{eff}}(\boldsymbol{S}, r_i) + \lambda \vec{\boldsymbol{S}}_i \times \left(\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{H}}^i_{\text{eff}}(\boldsymbol{S}, r_i)\right)\right]}_{=:\vec{\boldsymbol{f}}_i(\boldsymbol{S}, r_i)}, \tag{4.0.1}$$

with describes the time dependent motion of the normed spin vector $\vec{\boldsymbol{S}}_i$ in an effective field (compare to (3.2.2))

$$\vec{\boldsymbol{H}}^i_{\text{eff}}(\boldsymbol{S}, r_i) := -\frac{1}{\mu_s}\frac{\partial \mathcal{H}}{\partial \vec{\boldsymbol{S}}_i}(\boldsymbol{S}) + \vec{\boldsymbol{H}}^i_{\text{th}}(r_i),$$

which depends on

- all spins $\boldsymbol{S} := (\vec{\boldsymbol{S}}_1, \vec{\boldsymbol{S}}_2, \ldots, \vec{\boldsymbol{S}}_N)$ in the system (via exchange interaction),
- (pseudo) stochastic thermal fluctuations, which we want to represent in this notation by a seed-like value $r_i \in \mathbb{N}_0$ for $i = 1, \ldots, N$ in the sense that same values of r_i stand for same thermal fields $\vec{\boldsymbol{H}}^i_{\text{th}}(r_i)$ for the i-th spin.

Details to this physical formulation can be found in [1, sections 4.1,4.2].
In VAMPIRE there are two integration methods given:

- Heun's method
- the midpoint method

In the following we are going to investigate the differences between these two methods and their behavior concerning stability.

4.1 Heun's method

4.1.1 General case

The classical time integration step in Heun's method for an ordinary first order differential equation of the form

$$y'(t) = f\big(t, y(t)\big), \quad y(0) = y_0 \tag{4.1.1}$$

can be written in the form

$$y_{n+1} = y_n + \frac{1}{2}h\big(f(t_n, y_n) + f(t_{n+1}, y'_{n+1})\big), \tag{4.1.2}$$

where $h = t_{n+1} - t_n$ is the time step size, y_n and y_{n+1} are the numerical solutions at time t_n respectively t_{n+1} and

$$y'_{n+1} = y_n + hf(t_n, y_n)$$

is in fact a step of the explicit Euler method.

4.1.2 VAMPIRE realization

Basically the Heun implementation in VAMPIRE is quite similar to the general case. We describe each step and give references to the original source code in Appendix B.1 (see also [1, section 4.3]). We describe the initial spin states at time t_n with $\vec{\boldsymbol{S}}_i$ for $i = 1, \ldots, N$ and calculate the spin state at time t_{n+1} in the numerical solution in the following four steps:

- For every spin, the seeds $r_1, \ldots, r_N$ are fixed and an Euler step is done in the form

  $$\vec{\boldsymbol{S}}'_i := \vec{\boldsymbol{S}}_i + h\vec{\boldsymbol{f}}_i(\boldsymbol{S}, r_i) \qquad \forall i = 1, \ldots, N.$$

  Note that $\vec{\boldsymbol{H}}^i_{\text{eff}}$ in $\vec{\boldsymbol{f}}_i$ is depending on $\boldsymbol{S} = (\vec{\boldsymbol{S}}_1, \ldots, \vec{\boldsymbol{S}}_N)$ for all $i = 1, \ldots, N$, so the initial state of the spin directions is taken for every $i = 1, \ldots, N$ (see lines 200-213).

- Every result of the previous Euler step is normalized, i.e.

  $$\vec{\boldsymbol{S}}^{0'}_i := \frac{1}{|\vec{\boldsymbol{S}}'_i|}\vec{\boldsymbol{S}}'_i \qquad \forall i = 1, \ldots, N$$

  (see lines 215-220).

- With the combination of all new calculated and normalized spin directions $\boldsymbol{S}^{0'} := (\vec{\boldsymbol{S}}^{0'}_1, \ldots, \vec{\boldsymbol{S}}^{0'}_N)$ we finally do the Heun step in the form

  $$\vec{\boldsymbol{S}}''_i := \vec{\boldsymbol{S}}_i + \frac{1}{2}h\big(\vec{\boldsymbol{f}}_i(\boldsymbol{S}, r_i) + \vec{\boldsymbol{f}}_i(\boldsymbol{S}^{0'}, r_i)\big)$$

  (see lines 251-266). Note that r_i has not changed compared to the Euler step.

- It remains to normalize the final spin direction by

$$\vec{\boldsymbol{S}}_i^{0''} := \frac{1}{|\vec{\boldsymbol{S}}_i''|}\vec{\boldsymbol{S}}_i'' \qquad \forall i = 1, \ldots, N$$

(see lines 268-273), which is finally the spin state at time t_{n+1} for all $i = 1, \ldots, N$ in the numerical solution.

4.2 Midpoint method

4.2.1 General case

The implicit midpoint method for the differential equation in (4.1.1) is given by

$$y_{n+1} = y_n + hf\left(t_{n+1/2}, \frac{y_n + y_{n+1}}{2}\right) \tag{4.2.1}$$

where $t_{n+1/2} := t_n + h/2$. This equation gives just an implicit definition of y_{n+1}, so it has to be solved either analytically or numerically.

4.2.2 Preliminary remarks for the VAMPIRE implementation - Cayley transform

First, we can rewrite the function $\vec{\boldsymbol{f}}_i$ in (4.0.1) with the additional definition

$$\vec{\boldsymbol{g}}_i(\boldsymbol{S}, r_i) := -\frac{\gamma}{1+\lambda^2}\left[\vec{\boldsymbol{H}}^i_{\text{eff}}(\boldsymbol{S}, r_i) + \lambda\left(\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{H}}^i_{\text{eff}}(\boldsymbol{S}, r_i)\right)\right]$$

in the separated form

$$\vec{\boldsymbol{f}}_i(\boldsymbol{S}, r_i) = \vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{g}}_i(\boldsymbol{S}, r_i). \tag{4.2.2}$$

With those definitions we can write the implicit midpoint rule for our differential equation (4.0.1) in the form

$$\vec{\boldsymbol{S}}_i' = \vec{\boldsymbol{S}}_i + \frac{h}{2}(\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}_i') \times \vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \boldsymbol{S}')/2, r_i\big), \tag{4.2.3}$$

with the initial state spin states $\vec{\boldsymbol{S}}_i$ at time t_n and the final spin states $\vec{\boldsymbol{S}}_i'$ at time t_{n+1} for all $i = 1, \ldots, N$. A directly iterative solver might be inefficient, because $\vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \boldsymbol{S}')/2, r_i\big)$ contains the information orientation of the whole spin system in the initial state $\boldsymbol{S}$ and the final state $\boldsymbol{S}'$. The idea is now to substitude $\boldsymbol{S}'$ in the term $\vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \boldsymbol{S}')/2, r_i\big)$ by a good approximation $\widetilde{\boldsymbol{S}}$, which might be computed in a convenient predictor step. Afterwards the resulting equation

$$\vec{\boldsymbol{S}}_i' = \vec{\boldsymbol{S}}_i + \frac{h}{2}(\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}_i') \times \vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \widetilde{\boldsymbol{S}})/2, r_i\big). \tag{4.2.4}$$

is solved for $\vec{\boldsymbol{S}}_i'$ by using the ***Cayley transform***. So before we deal with the VAMPIRE implementation, we start with an introduction and some properties of this ***Cayley transform***, which will be used afterwards. It is a summary and supplement of [5, section 2].

Lemma 4.2.1. *Let $\vec{\boldsymbol{G}} \in \mathbb{R}^3$ be a fixed vector. Then the linear mapping*

$$\begin{aligned} \mathbb{R}^3 &\longrightarrow \mathbb{R}^3 \\ \vec{\boldsymbol{S}} &\longmapsto \vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} \end{aligned}$$

can be uniquely written as a matrix-vector-product with a skew-symmetric matrix $\boldsymbol{G}$, i.e. there holds

$$\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} = \boldsymbol{G}\vec{\boldsymbol{S}} \qquad \forall \vec{\boldsymbol{S}} \in \mathbb{R}^3$$

with a matrix $\boldsymbol{G} \in \mathbb{R}^{3\times 3}, \boldsymbol{G}^T = -\boldsymbol{G}$.

Proof. The definition of cross-products and matrix-vector-multiplication leads to

$$\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} \times \begin{pmatrix} G_1 \\ G_2 \\ G_3 \end{pmatrix} = \begin{pmatrix} S_2G_3 - S_3G_2 \\ S_3G_1 - S_1G_3 \\ S_1G_2 - S_2G_1 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & G_3 & -G_2 \\ -G_3 & 0 & G_1 \\ G_2 & -G_1 & 0 \end{pmatrix}}_{=:\boldsymbol{G}} \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix}$$

□

With the short-cut $\vec{\boldsymbol{G}}_i := \vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \widetilde{\boldsymbol{S}})/2, r_i\big)$ and the corresponding matrix $\boldsymbol{G}_i$ (see the previous lemma) we can rewrite (4.2.4) in the form

$$\vec{\boldsymbol{S}}'_i = \vec{\boldsymbol{S}}_i + \frac{h}{2}\boldsymbol{G}_i(\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}'_i).$$

Note again, that $\boldsymbol{G}_i$ only depend on $\boldsymbol{S}_i$ and eventually on a predictor-step, but not on $\boldsymbol{S}'$. Algebraic transformations of the equation above lead to

$$\left(\mathbb{1} - \frac{1}{2}\boldsymbol{G}_i\right)\vec{\boldsymbol{S}}'_i = \left(\mathbb{1} + \frac{1}{2}\boldsymbol{G}_i\right)\vec{\boldsymbol{S}}_i. \tag{4.2.5}$$

The following lemma deals with the explicit solution $\vec{\boldsymbol{S}}'_i$ of this equation.

Lemma 4.2.2. *Let $\boldsymbol{G} \in \mathbb{R}^{3\times 3}$ be a skew-symmetric matrix and $\vec{\boldsymbol{G}}$ the corresponding vector (see lemma 4.2.1). Then the following statements hold:*

(i) $\mathbb{1} - \boldsymbol{G}$ *is invertible.*

(ii) $(\mathbb{1} - \boldsymbol{G})^{-1}(\mathbb{1} + \boldsymbol{G})$ *is orthogonal.*

(iii) $(\mathbb{1} - \boldsymbol{G})^{-1}(\mathbb{1} + \boldsymbol{G}) = \mathbb{1} + \frac{2}{1+\vec{\boldsymbol{G}}^2}\left(\boldsymbol{G} + \boldsymbol{G}^2\right)$

Proof. (i):
Assume that $\mathbb{1}-\boldsymbol{G}$ is not invertible. Then we can find a $\vec{\boldsymbol{S}} \in \mathbb{R}^3\backslash\{0\}$ with $(\mathbb{1}-\boldsymbol{G})\vec{\boldsymbol{S}} = 0$. Because of the skew-symmetry of $\boldsymbol{G}$, we have

$$\vec{\boldsymbol{S}}^T\boldsymbol{G}\vec{\boldsymbol{S}} = -\vec{\boldsymbol{S}}^T\boldsymbol{G}^T\vec{\boldsymbol{S}} = -\left(\vec{\boldsymbol{S}}^T\boldsymbol{G}^T\vec{\boldsymbol{S}}\right)^T = -\vec{\boldsymbol{S}}^T\boldsymbol{G}\vec{\boldsymbol{S}}$$

(note that $\vec{\boldsymbol{S}}^T \boldsymbol{G}^T \vec{\boldsymbol{S}} \in \mathbb{R}$ is a scalar and therefore invariant under transposition), which implies $\vec{\boldsymbol{S}}^T \boldsymbol{G} \vec{\boldsymbol{S}} = 0$. Using that, we get

$$0 = \vec{\boldsymbol{S}}^T (\mathbb{1} - \boldsymbol{G}) \vec{\boldsymbol{S}} = |\vec{\boldsymbol{S}}|^2 - 0 = |\vec{\boldsymbol{S}}|^2,$$

what is a contradiction to the assumption $\vec{\boldsymbol{S}} \in \mathbb{R}^3 \backslash \{0\}$.
(ii):
By using the skew-symmetry of $\boldsymbol{G}$, we get

$$\left((\mathbb{1} - \boldsymbol{G})^{-1}(\mathbb{1} + \boldsymbol{G})\right)^T (\mathbb{1} - \boldsymbol{G})^{-1}(\mathbb{1} + \boldsymbol{G}) = (\mathbb{1} - \boldsymbol{G})(\mathbb{1} + \boldsymbol{G})^{-1}(\mathbb{1} - \boldsymbol{G})^{-1}(\mathbb{1} + \boldsymbol{G}).$$

Because of

$$(\mathbb{1} + \boldsymbol{G})(\mathbb{1} - \boldsymbol{G}) = \mathbb{1} - \boldsymbol{G}^2 = (\mathbb{1} - \boldsymbol{G})(\mathbb{1} + \boldsymbol{G})$$

the two inverse matrices in the middle of the upper product commutate and therefore the result is the identity matrix $\mathbb{1}$.
(iii):
First we note the well-known vector identity

$$(\vec{\boldsymbol{A}} \times \vec{\boldsymbol{B}}) \times \vec{\boldsymbol{C}} = \vec{\boldsymbol{B}}(\vec{\boldsymbol{A}} \cdot \vec{\boldsymbol{C}}) - \vec{\boldsymbol{A}}(\vec{\boldsymbol{B}} \cdot \vec{\boldsymbol{C}}). \tag{4.2.6}$$

We could finished the proof by showing

$$(\mathbb{1} + \boldsymbol{G})\vec{\boldsymbol{S}} = (\mathbb{1} - \boldsymbol{G}) \left(\mathbb{1} + \frac{2}{1 + \vec{\boldsymbol{G}}^2} \left(\boldsymbol{G} + \boldsymbol{G}^2 \right) \right) \vec{\boldsymbol{S}} \qquad \forall \vec{\boldsymbol{S}} \in \mathbb{R}^3.$$

By rewriting the matrix-vector-products as cross-products according to lemma 4.2.1 and using the upper vector identity, we get for the right hand side

$$\begin{aligned}
& (\mathbb{1} - \boldsymbol{G}) \left(\mathbb{1} + \frac{2}{1 + |\vec{\boldsymbol{G}}|^2} \left(\boldsymbol{G} + \boldsymbol{G}^2 \right) \right) \vec{\boldsymbol{S}} \\
\overset{\text{lem. 4.2.1}}{=} \; & (\mathbb{1} - \boldsymbol{G}) \left(\vec{\boldsymbol{S}} + \frac{2}{1 + \vec{\boldsymbol{G}}^2} \left(\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} + (\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \times \vec{\boldsymbol{G}} \right) \right) \\
\overset{(4.2.6)}{=} \; & (\mathbb{1} - \boldsymbol{G}) \left(\vec{\boldsymbol{S}} + \frac{2}{1 + \vec{\boldsymbol{G}}^2} \left(\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} + \vec{\boldsymbol{G}}(\vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{G}}) - \vec{\boldsymbol{S}} \vec{\boldsymbol{G}}^2 \right) \right) \\
= \; & \vec{\boldsymbol{S}} + \frac{2}{1 + \vec{\boldsymbol{G}}^2} \left(\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} + \vec{\boldsymbol{G}}(\vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{G}}) - \vec{\boldsymbol{S}} \vec{\boldsymbol{G}}^2 \right) \\
& - \vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} - \frac{2}{1 + \vec{\boldsymbol{G}}^2} \left((\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \times \vec{\boldsymbol{G}} + \underbrace{(\vec{\boldsymbol{G}} \times \vec{\boldsymbol{G}})}_{=0} (\vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{G}}) - (\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \vec{\boldsymbol{G}}^2 \right) \\
\overset{(4.2.6)}{=} \; & \frac{1}{1 + \vec{\boldsymbol{G}}^2} \Big(\vec{\boldsymbol{S}} + \vec{\boldsymbol{S}} \vec{\boldsymbol{G}}^2 + 2\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} + \cancel{2\vec{\boldsymbol{G}}(\vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{G}})} - \cancel{2\vec{\boldsymbol{S}} \vec{\boldsymbol{G}}^2} \\
& \qquad - \vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}} - (\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \vec{\boldsymbol{G}}^2 - \cancel{2\vec{\boldsymbol{G}}(\vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{G}})} + \cancel{2\vec{\boldsymbol{S}} \vec{\boldsymbol{G}}^2} + 2(\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \vec{\boldsymbol{G}}^2 \Big) \\
= \; & \frac{1}{\cancel{1 + \vec{\boldsymbol{G}}^2}} \left(\vec{\boldsymbol{S}} \cancel{(\mathbb{1} + \vec{\boldsymbol{G}}^2)} + (\vec{\boldsymbol{S}} \times \vec{\boldsymbol{G}}) \cancel{(\mathbb{1} + \vec{\boldsymbol{G}}^2)} \right) \\
= \; & (\mathbb{1} + \boldsymbol{G}) \vec{\boldsymbol{S}}
\end{aligned}$$

what finishes the proof. □

We can apply the results of the previous lemma in (4.2.5) for the case $\boldsymbol{G} = \frac{1}{2}\boldsymbol{G}_i$ to achieve the following three facts:

- There exists the explicit solution of (4.2.5)

$$\vec{\boldsymbol{S}}_i' = \left(\mathbb{1} - \frac{1}{2}\boldsymbol{G}_i\right)^{-1}\left(\mathbb{1} + \frac{1}{2}\boldsymbol{G}_i\right)\vec{\boldsymbol{S}}_i. \tag{4.2.7}$$

- The transformation matrix

$$\left(\mathbb{1} - \frac{1}{2}\boldsymbol{G}_i\right)^{-1}\left(\mathbb{1} + \frac{1}{2}\boldsymbol{G}_i\right)$$

  is orthogonal, so it holds $|\vec{\boldsymbol{S}}_i| = |\vec{\boldsymbol{S}}_i'|$. This is indeed a nice behaviour of this integration method, because $\vec{\boldsymbol{S}}_i$ remains normalized in every time-integration step.

- We get a practical method to calculate $\vec{\boldsymbol{S}}_i'$ in the form

$$\vec{\boldsymbol{S}}_i' = \vec{\boldsymbol{S}}_i + \frac{1}{1 + \frac{1}{4}\vec{\boldsymbol{G}}_i^2}\left(\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{G}}_i + \frac{1}{2}(\vec{\boldsymbol{S}}_i \times \vec{\boldsymbol{G}}_i) \times \vec{\boldsymbol{G}}_i\right). \tag{4.2.8}$$

4.2.3 VAMPIRE realization

After the preliminary work in the previous section we now turn to the implementation-steps in VAMPIRE. References are given to the source code in appendix B.2. We describe the initial spin states at time t_n with $\vec{\boldsymbol{S}}_i$ for $i = 1, \ldots, N$ and calculate the spin state at time t_{n+1} in the numerical solution in the following two steps:

- First, the equation

$$\vec{\boldsymbol{S}}_i' = \vec{\boldsymbol{S}}_i + \frac{h}{2}(\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}_i') \times \vec{\boldsymbol{g}}_i(\boldsymbol{S}, r_i)$$

  is solved to $\vec{\boldsymbol{S}}_i'$ with the use of the ***Cayley transform*** in (4.2.8) (see lines 138-141). Note that this equation is equivalent to (4.2.3) for the choice $\widetilde{\boldsymbol{S}} := \boldsymbol{S}$.

- Then

$$\vec{\boldsymbol{S}}_i'' = \vec{\boldsymbol{S}}_i + \frac{h}{2}(\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}_i'') \times \vec{\boldsymbol{g}}_i\big((\boldsymbol{S} + \boldsymbol{S}')/2, r_i\big)$$

  is again solved with the ***Cayley transform*** to get $\vec{\boldsymbol{S}}_i''$(see lines 180-183), which is the spin state at time t_{n+1} for all $i = 1, \ldots, N$ in the numerical solution. This equation correspond to (4.2.3) for the choice $\widetilde{\boldsymbol{S}} := \boldsymbol{S}'$, which is the result of the first step.

4.3 Comparison of the time integration methods

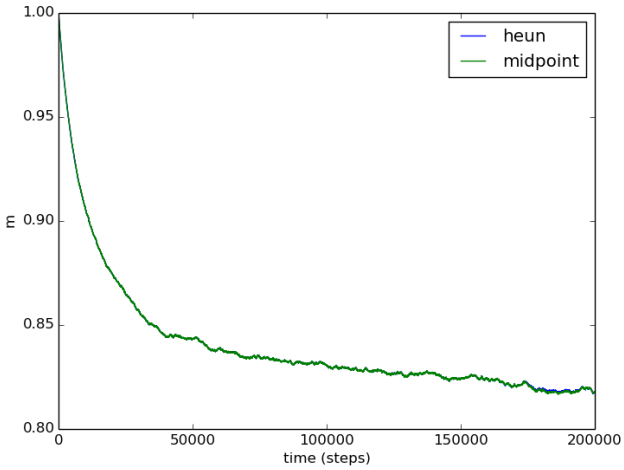
In the previous sections 4.1 and 4.2, we described Heun's method and the midpoint method as two possible time integration methods for the LLG equation. Both are implemented in the form of a predictor-corrector method. We now want to compare the two versions to find the convenient method for our propose. It is expected that the quality of the time integration method strongly depends on the chosen time step. If it is too large, the result may be unstable and the numerical solution does not converge to the real solution. It is also known that explicit time integration methods (as the Heun' method) usually have a much smaller stability regions compared to implicit time integration methods (as the midpoint method). We want to investigate, if this behavior can also be observed for the numerical solution of the stochastic differential equation of Sec. 3.2 in VAMPIRE.

The graphs in Fig. 4.1 show the development of the zero-field equilibrium spin polarization m_e (see Sec. 3.3) calculated in VAMPIRE over increasing time steps with a damping constant $\gamma = 0.02$ (for chosen parameter set see Tab. 4.1). Since m_e is calculated via a mean value over different time steps, we expect a asymptotic approach to the "real" value of m_e with increasing amount of time steps. In (c) one can clearly observe that the midpoint method shows convergence to a similar value compared to (a) or (b), whereas the result of Heun's method also converges, but to a different value.
It should also be noted that the convergence behavior depends strongly on the chosen value for the damping constant γ. For the case $\gamma = 0.1$ in Fig. 4.2 we observe a quite suitable behavior of both time integration methods up to $\Delta t = 10^{-15}$ s. If we decrease the time step further to $\Delta t = 10^{-14}$ s, Heun's method completely collapses, whereas the midpoint method remains fairly stable, even though the observed limit does not seem to be completely right.

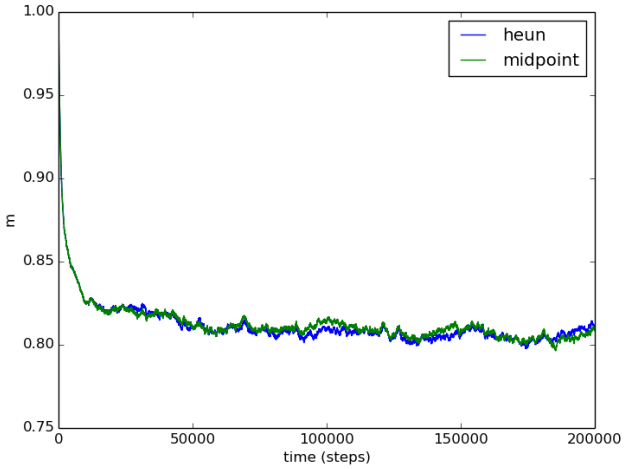
parameter	value	unit
J	$6.72 \cdot 10^{-21}$	J
μ_S	1.85	μ_B
k_1	$8.624 \cdot 10^{-23}$	J
unit cell size	2.40	Å
cylinder diameter	5	nm
cylinder height	8	nm

Table 4.1: Parameters used for the VAMPIRE simulation in Fig. 4.1 and 4.2.

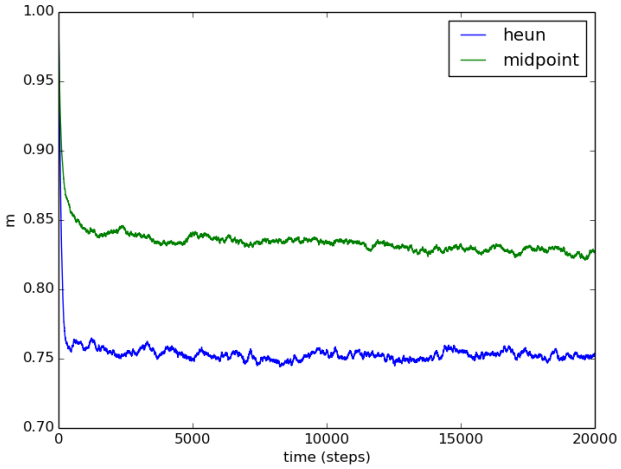
Taking those results into account, the midpoint method seems to be the much more stable integration method – especially for simulations with time step values which are quite "on the edge" in terms of numerical stability.



(a)

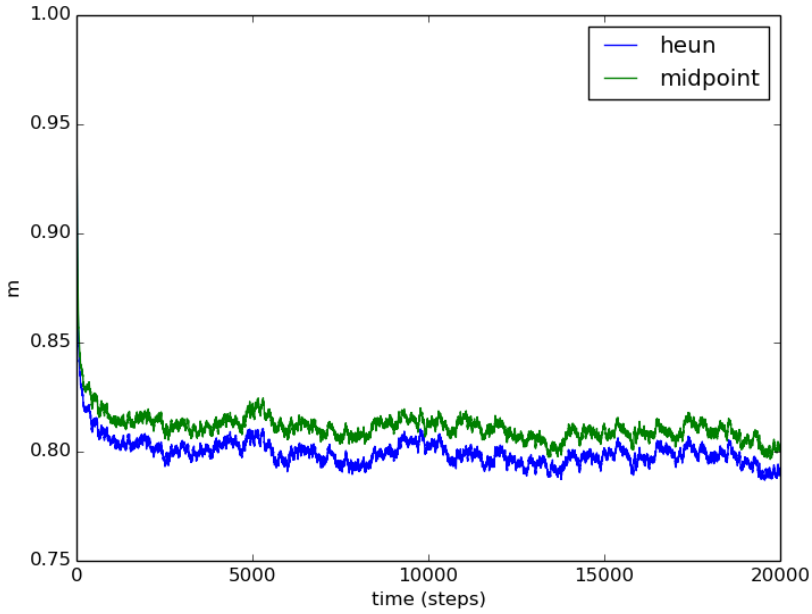


(b)

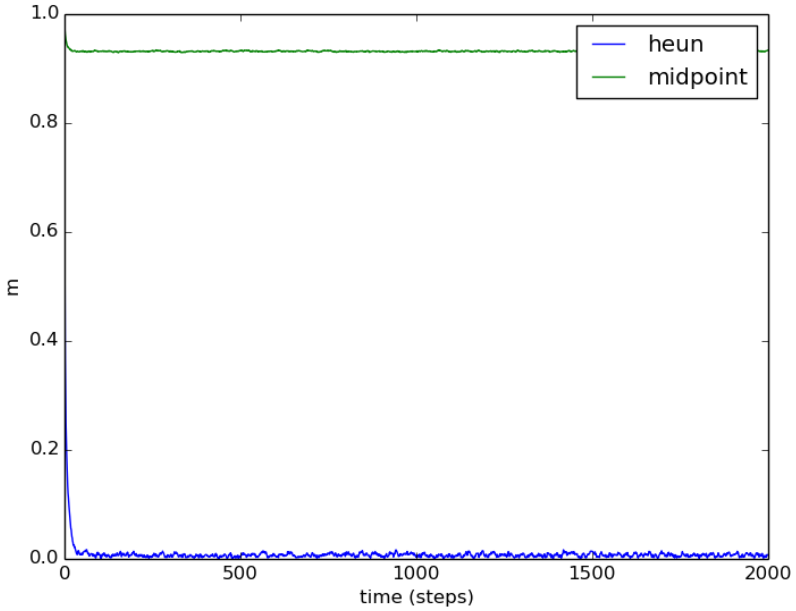


(c)

Figure 4.1: m_e as a function of the total amount of time steps for: $\Delta t = 10^{-17}$ s in (a), $\Delta t = 10^{-16}$ s in (b) and $\Delta t = 10^{-15}$ s in (c) with $\gamma = 0.02$ in each case.



(a)



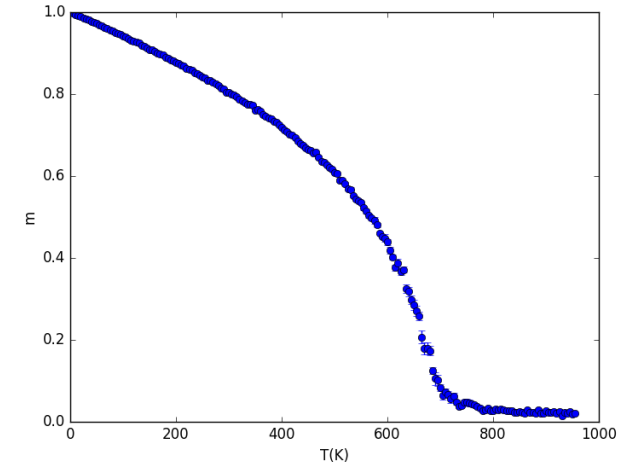
(b)

Figure 4.2: m_e as a function of the total amount of time steps for: $\Delta t = 10^{-15}$ s in (a) and $\Delta t = 10^{-14}$ s in (b) with $\gamma = 0.1$ in each case.

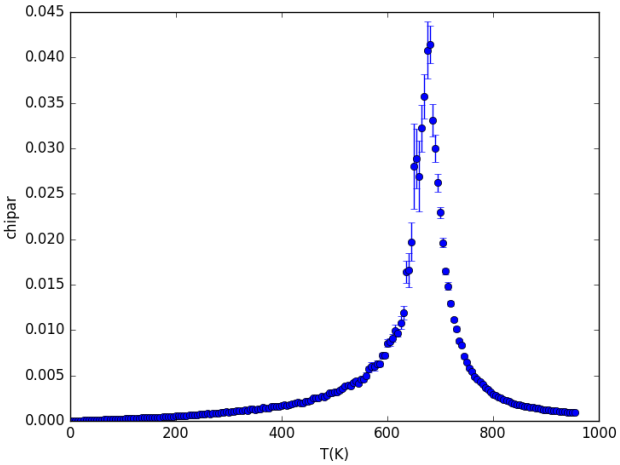
Chapter 5

Switching probability for magnetic grains

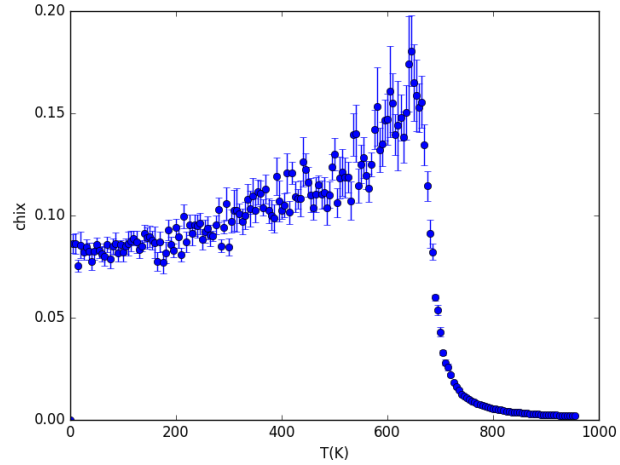
In this short chapter we give the main idea of the method to calculate the switching probability of a magnetic grain under the circumstances of an applied heat pulse and external magnetic field, as it is performed in a writing process in HAMR. The same technique is used in [10]. The idea is to use the (in comparison to the LLG equation) much computationally cheaper LLB equation from Sec. 3.3 to simulate the time dependent behavior of the magnetization. The disadvantage of the LLB equation is the explicitly required magnetic susceptibility $\widetilde{\chi}_{\parallel}$ respectively $\widetilde{\chi}_{\perp}$ and zero-field equilibrium spin polarization m_e (see (3.3.1) and (3.3.2)). Those depend on the material parameters of the grain and can be achieved by first integrating the LLG equation with VAMPIRE. This has to be done only once for each magnetic grain with a certain material composition and size. For a grain diameter of 5 nm and the identical parameter choice as in Table 6.1 (except of T_C, which is not required as input parameter), we receive the results in Fig. 5.1.
In the second step the resulting temperature dependent values for $\widetilde{\chi}_{\parallel}, \widetilde{\chi}_{\perp}$ and m_e are used in a LLB-simulation to calculate the grains' switching probabilities as a function of the writing temperature and the phase difference between the applied heat spot and magnetic field, which corresponds to a certain down-track position of a grain in a writing process. Even though such a phase diagram needs many different repeated numerical solutions of the stochastic LLB equation (for different writing temperatures, phase differences and also repetitions with only various seeds to approximate the probability distribution), the runtime is easily manageable via parallel computing of the independent simulations. Due to the fast runtime of the LLB-simulations the phase diagram can be computed with a much higher resolution compared to a full atomistic simulation with VAMPIRE. A possible resulting phase diagram can be observed in Fig. 6.3 or [11, Fig. 2].



(a)



(b)



(c)

Figure 5.1: Temperature dependent material parameters from the atomistic simulations in VAMPIRE. In order to receive a good statistic for the stochastic result, every data point is the mean value of 10 simulations with different seeds.

Chapter 6

Transition Jitter, Switching Probability and Curvature Reduction dependency of the Signal-to-Noise Ratio in Heat-Assisted Magnetic Recording

The content of this chapter presents the main results of this thesis and is therefore provided with the formal requirements of a scientific paper, which is intended to be published in a convenient journal in the near future. It attaches to the previous chapter and investigates the form of the resulting phase plot in terms of suitability for a bit writing process. The source code of the required simulations is shown in Appendix D.

6.1 Introduction

The quality of a written bit pattern on granular media is mostly determined by the signal-to-noise ratio (SNR). A high SNR means a sharp edge between neighboring bits and depends either on suitable magnetic material properties of the grains to provide a good switching probability in the writing process but also on the size and position distribution of the grains in the granular medium. There are different methods to calculate the switching probability of a grain model during heat-assisted magnetic recording (HAMR), which is subject to a heat pulse and an external magnetic field. One method is solving the stochastic Landau-Lifshitz-Gilbert (LLG) equation for each atom of the grain with VAMPIRE (see [1]), or solving the stochastic coarse-grained Landau-Lifshiftz-Bloch (LLB) equation (see [10]). Depending on the down-track position x and off-track position y, the repetition of a switching trajectory for a given parameter set results in the switching probability of a grain. Calculating the probability for various x and y yields a phase diagram of the write process. This process is illustrated in Fig. 6.5. Instead of using the off-track direction y, we plot the peak temperature T_{peak}. The resulting diagram contains much more information then the footprint, because there is no need to specify the write

temperature beforehand. For each write temperature T_{peak} the off-track direction y can be easily determined under the assumption of a Gaussian heat pulse via the formula in Eq. (6.3.1). What we receive is the grain's switching probability P as a function of its down-track position d (relative to the applied magnetic field and heat pulse) and the peak temperature T_{peak} of the heat pulse with justifiable computational afford. The received phase diagrams (as in [11, Fig.2]) for $P(d, T_{\text{peak}})$ allow us to simulate writing processes of bit pattern on granular media as we will describe in Sec. 6.3. The advantage of this approach of the writing process is that the phase diagram has to be created only once and the switching probability can afterwards be extracted for an arbitrary amount of magnetic grains with no further computational afford. The disadvantage is that every grain is regarded individually and therefore it is not obvious how to take stray-field interactions into account. This can be done by adding a variation to the Curie Temperature T_C. The phase plots contain much information about the size and characteristics of the magnetic grains as well as the parameters of the writing process (velocity, external applied field etc.). In [7] for instance, the shape of the phase diagram in dependence of the composition of a bi-layer material is investigated. Although such a phase diagram contains a lot of information about a single grain, it a priori tells very little about the resulting SNR of the read-back signal of a bit series. Hence, one should systematically investigate the influences of changes of the phase diagram on the final SNR. In this paper we will develop a mathematical model of a phase diagram, which allows to vary certain parameters and perform writing processes with the resulting diagrams. The read-back signal then gives some indication of the potentially SNR-improvement. The mathematical formulation of the phase plot is presented in Sec. 6.2. In Sec. 6.3, we will describe the simulation of the writing and read-back processes and Sec. 6.4 will summarize and discuss the results of the received SNR and compare those to theoretical formulas.

6.2 Mathematical model of a phase plot

In heat-assisted magnetic recording (HAMR) the switching probability of a magnetic grain is given as a phase diagram. As in [11, Fig. 2] the area of the highest switching probability has a C-like shape in the $d - T_{\text{peak}}$-plane. In the following we use that optical configuration to define an analytical function, which allows us to model such a phase plot.

6.2.1 Model parameters

We use eight parameters that fully determine the shape of the phase plot:

- down-track-jitter: σ_d [nm]
- off-track-jitter: σ_o [nm]
- maximum switching probability: $P_{\max}$
- half maximum temperature: F [K]

- bit length: b [nm]
- curvature: p_1 [nm/K^2]
- position in T_{peak}-direction: p_2 [K]
- position in d-direction: p_3 [nm]

6.2.2 Mathematical model

We now define the model function P in three steps with two help functions h_1 and h_2. Note that those functions depend on the upper parameters. First the down-track- and off-track-jitter is modeled as the slope of the probability along a cut through the phase diagram for fixed d and T_{peak}, respectively. As in [7, eq. (4)] we use the Gaussian cumulative distribution function

$$\Phi(x, \mu, \sigma) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right) \tag{6.2.1}$$

with

$$\lim_{x\to-\infty} \Phi(x) = 0, \lim_{x\to+\infty} \Phi(x) = 1, \Phi(\mu) = \frac{1}{2} \tag{6.2.2}$$

and σ determining the slope of Φ. We write

$$h_1(T, d) := \sqrt{P_{\max}} \cdot \Phi(d, 0, \sigma_d) \cdot \Phi(T, F, \sigma_o) \tag{6.2.3}$$

and receive a function as in Fig. 6.1 (a) that models the down-track-jitter σ_d via the vertical and the off-track-jitter σ_o via the horizontal contour sharpness. The variable $\sqrt{P_{\max}}$ determines the maximum function value, i.e. the formal limit

$$\lim_{T,d\to\infty} h_1(T, d) = \sqrt{P_{\max}}. \tag{6.2.4}$$

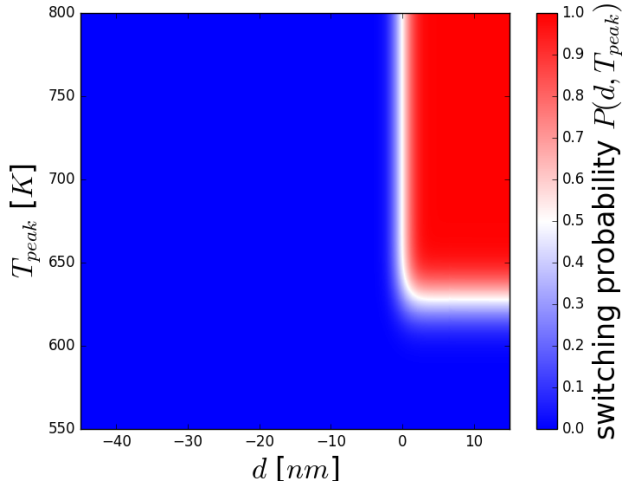
If a bit is written, the switching probability should again decrease after a certain writing distance in down-track direction, therefore we additionally multiply a mirrored and shifted function h_1 in the form

$$h_2(T, d) := h_1(T, d) \cdot h_1(T, b - d) \tag{6.2.5}$$

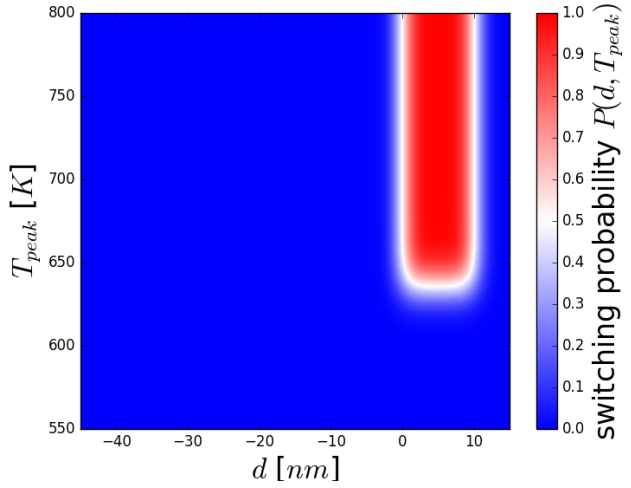
and receive a function graph as in Fig. 6.1 (b). The maximum function value is $P_{\max}$ but note, that this only holds in the limit for $b, T \to \infty$, so $P_{\max}$ might never be actually reached. Finally we receive the complete model via transformation of the bit into parabolic shape via

$$P(T, d) := h_2\Big(T, d - \big(p_1(T - p_2)^2 - p_3\big)\Big) \tag{6.2.6}$$

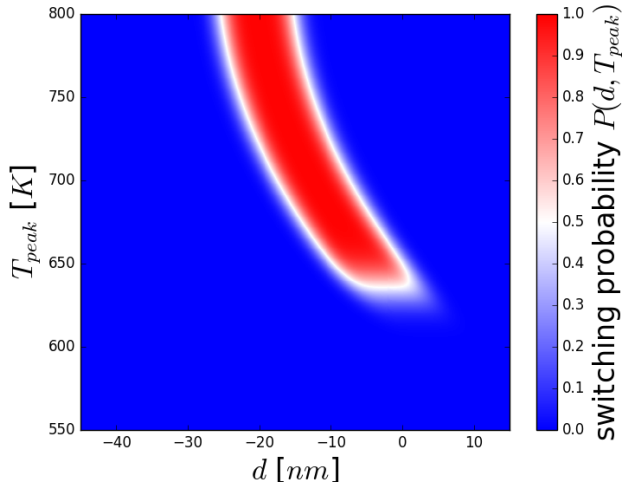
to get the C-like curvature as in Fig. 6.1 (c), which is usually observed (as in [11, Fig.2]). The impact of the model parameters defined in Sec. 6.2.1 on the shape of the model function can be visually observed in Fig. 6.2.



(a)



(b)



(c)

Figure 6.1: (a) and (b) show the graph of the help functions h_1 respectively h_2. (c) the final model phase plot can be observed.

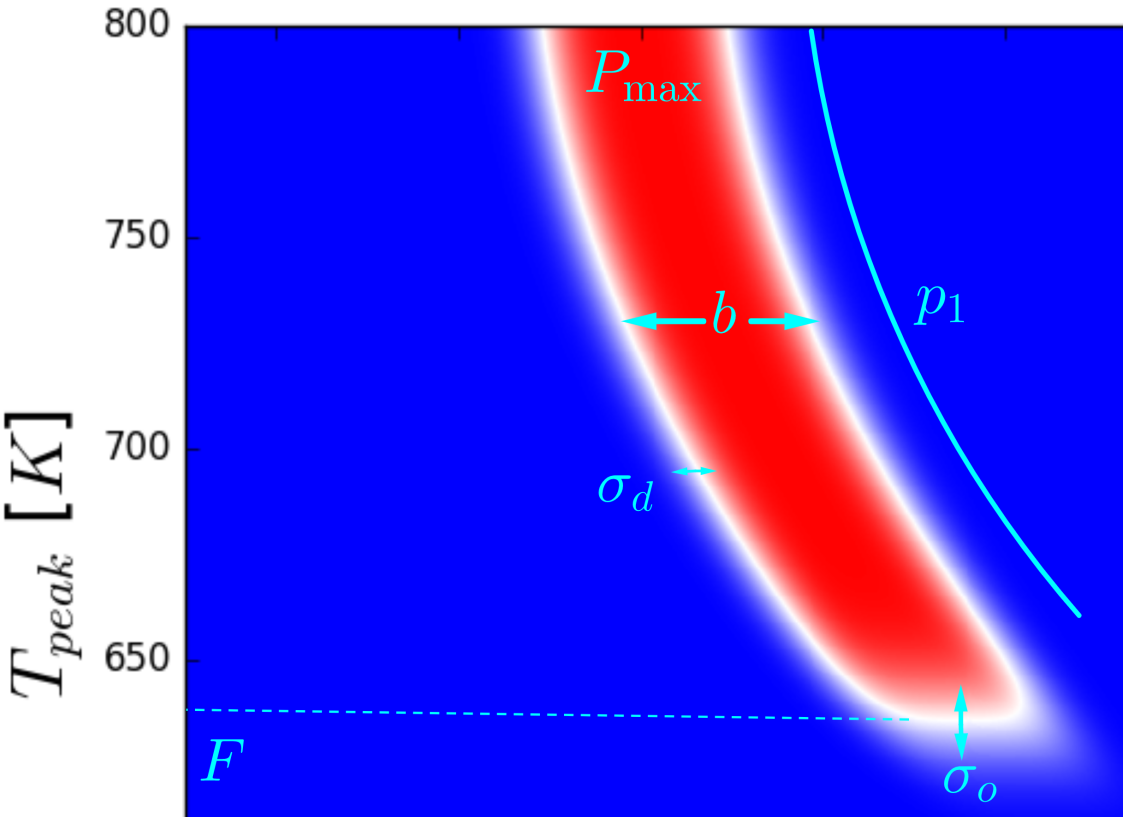


Figure 6.2: Detailed impact of the different model parameters in Sec. 6.2.1 visualized via an enlarged view of Fig. 6.1 (c).

K_1 [J/m^3]	$6.6 \cdot 10^6$
$J_{k,l}$ [J/link]	$6.72 \cdot 10^{-21}$
μ_s [μ_B]	1.6
J_s [T]	1.35
a [nm]	0.24
λ	0.02
T_C [K]	698.69

Table 6.1: Used material parameters in the LLB model

parameter	min value	max value
σ_d [nm]	0.01	4.00
$P_{\max}$	0.64	1.00
b [nm]	4.0	12.0
curvature reduction [%]	0	100

Table 6.2: Range of variation for the model parameters.

6.2.3 Reference system and variation of the parameters

As already mentioned we aim to investigate the influence of changes of the described switching probability phase plots on the resulting SNR. Of course, it is desirable to start all variations from a realistic basic parameter set. Therefore we used the LLB model as in [10] with the material parameters that can be seen in Table 6.1, to compute such a reference phase plot.
We choose a grain height of 8 nm, an applied external magnetic field of 0.8 T tilted with an angle of 22° with a duration of 0.67 ns and a moving Gaussian pulse with velocity $v = 15$ m/s and a full width at half maximum (FWHM) of 60 nm, what leads to a maximum thermal gradient of 11 K/nm. A resulting phase diagram can be seen in Fig. 6.3. The parameters of the plots in Fig. 6.1 were determined via a least square fit of the simulated diagram in Fig. 6.3. The results of the fit for different grain sizes are given in Table 6.3. According to Sec. 6.2.1, we have in total eight parameters, who determine the model phase plot. The investigated parameter variations were taken in the ranges of Table 6.2.
The grain diameter is furthermore varied between 4 and 8 nm. Note that the basic parameter set for every grain size is taken from a fit of the simulated LLB phase diagram and is therefore different for each grain size (see Table 6.3). In Sec. 6.4 we investigate the influence of $\sigma_d, P_{\max}, b$ and p_1 on the read-back SNR. In Fig. 6.4 (a)-(d), those four parameters are varied to visualize their influence in comparison to the initial choice of parameters in Fig. 6.1 (c).

	grain size				
	4 nm	5 nm	6 nm	7 nm	8 nm
σ_d [nm]	2.51	2.03	1.74	1.50	1.37
$P_{\max}$	0.993	0.995	0.993	0.997	1.000
b [nm]	10.2	10.2	10.3	10.2	10.1
p_1 [10^{-4} nm/K^2]	3.28	3.88	4.33	4.89	5.16

Table 6.3: Parameter set that was evaluated via least square fit of the simulated phase diagrams for grain sizes from 4 to 8 nm.

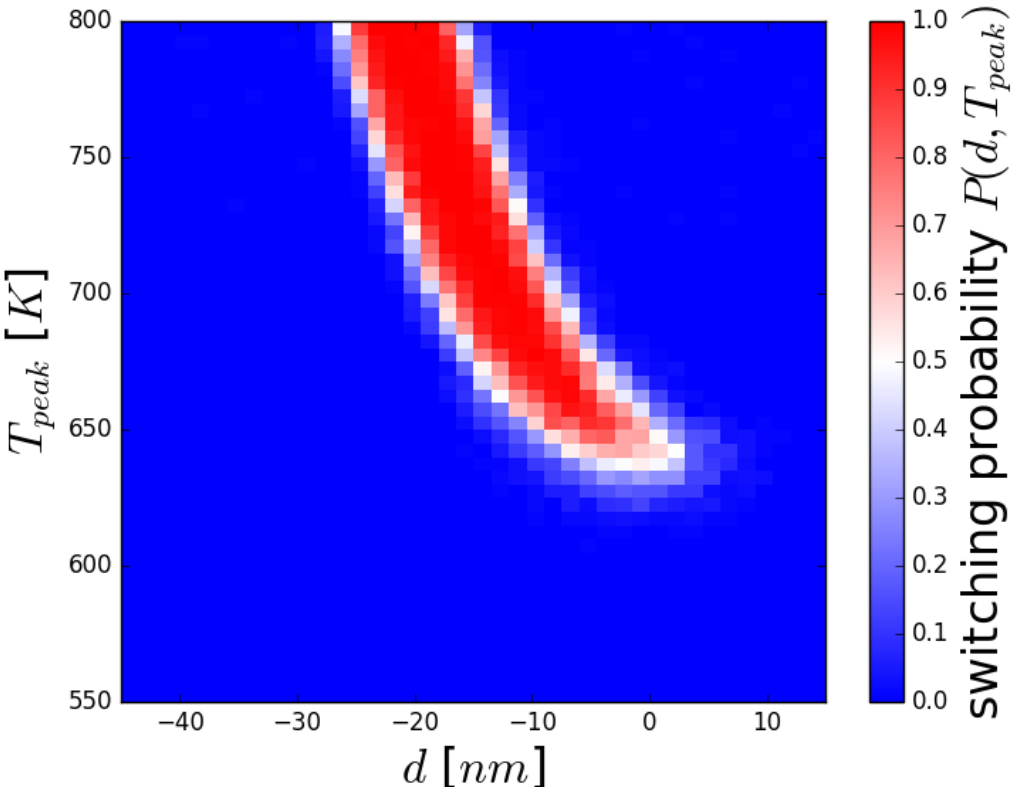


Figure 6.3: Simulated phase diagram with the LLB model for grain diameter $D = 7$ nm and material parameters in Table 6.1.

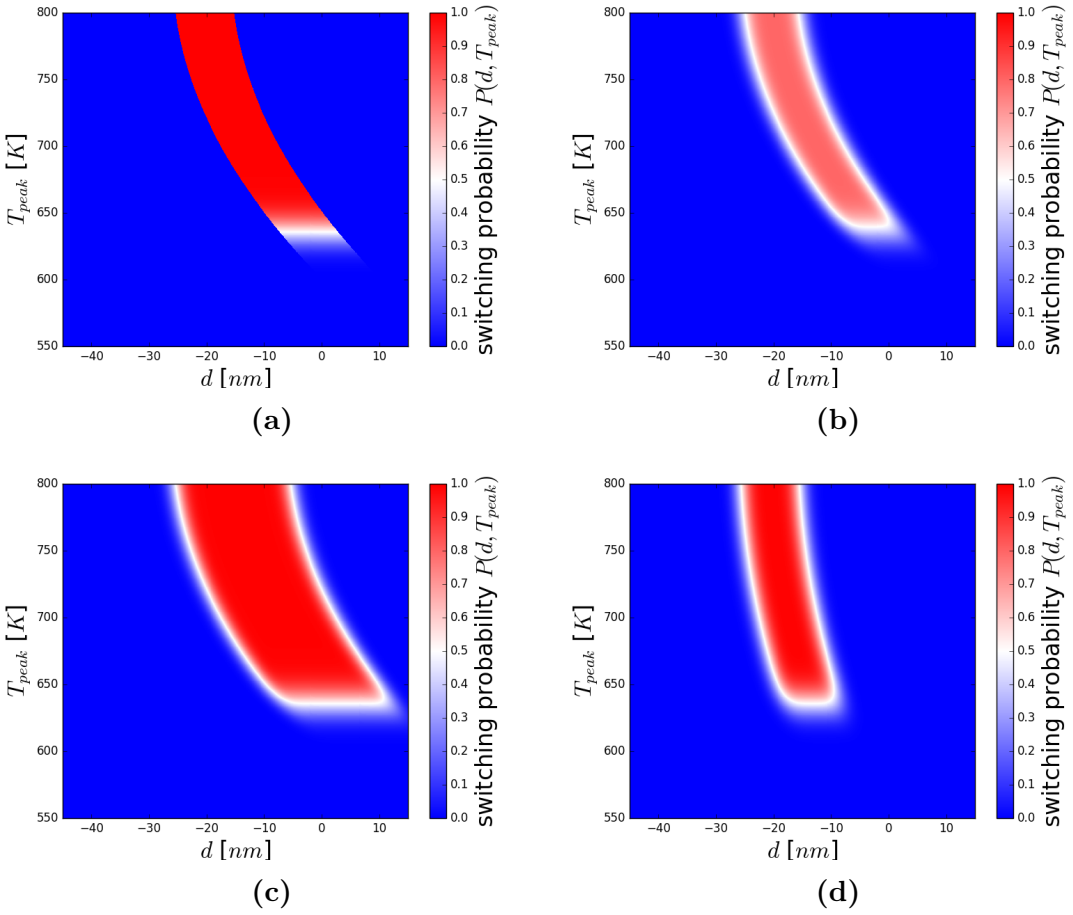


Figure 6.4: In comparison with Fig. 6.1, each picture shows only one changed parameter value. The phase plot in (a) has a reduced down-track jitter $\sigma_d = 0.0001$ nm, (b) a reduced $P_{\max} = 0.8$, (c) an extended bit-length of $b = 20$ nm and (d) a reduced curvature parameter p_1 by 60%.

6.3 Bit pattern on granular media

6.3.1 Writing process

We aim to use a phase plot, which determines the switching probability of a single cylindrical grain, to write bit pattern on granular media. Therefore we assume a Gaussian heat pulse moving across the medium with velocity $v = 15$ m/s, a full width at half maximum (FWHM) of 60 nm and a peak-temperature of $T_C + 60$ K, which is slightly dependent on the Curie temperature T_C at different grain sizes, which is about $T_C \approx 700$ K. The writing is done by mapping the switching probabilities of the corresponding phase plot to the grains of the granular medium according to their positions. The mapping is justified due to the assumption that every grain is approximated by a cylinder with a certain diameter that is subject to a Gaussian heat pulse, whose peak temperature T_{peak} depends on the off-track position y of the grain (see Fig. 6.5) via the formula

$$T_{\text{peak}}(y) = (T_{\max} - T_{\min}) \cdot \exp\left(-\frac{y^2}{2\sigma^2}\right) + T_{\min}, \tag{6.3.1}$$

where $T_{\min}$ and $T_{\max}$ are the overall minimum and maximum temperatures of the whole heat pulse and $\sigma = \text{FWHM}/\sqrt{8\ln 2}$ its standard deviation. We neglect a spatially varying temperature within a single grain and therefore assume, that the whole grain volume receives the same Gaussian pulse. The stray-field is not taken into account directly and thus different grains do not influence the switching probability of each other. The impact of the stray-field is only considered via an added variation to T_C.
We further assume a randomly initialized magnetization of the grains of the granular medium, where the diameters of all grains can be assumed identically (see Fig. 6.6, top) and calculate the switching probability for every grain within the chosen bit series of [3]:

(-1 +1 +1) -1 -1 -1 -1 -1 +1 +1 +1 -1 -1 +1 -1 -1 -1 +1 -1
+1 -1 +1 +1 +1 +1 -1 +1 +1 -1 +1 -1 -1 +1 +1 (-1 -1 -1),

where -1 represents a down and +1 an up bit. The 2×3 bits in the brackets are used for padding and the remaining 31 bits in-between represent the desired bit series. According to the probability in the phase plot, the actual magnetization direction of every grain is determined by a random number generator. Therefore an uniformly distributed pseudo random number $r \in [0, 1)$ is created. If the probability p fulfills $p > r$, the magnetization of the grain is changed according to the direction of the applied field. Otherwise it remains unchanged in the previous direction. This writing process is repeated for every switch of the magnetic field in the order of a real writing process from left to right on the medium. The result can be seen in Fig. 6.6, bottom.

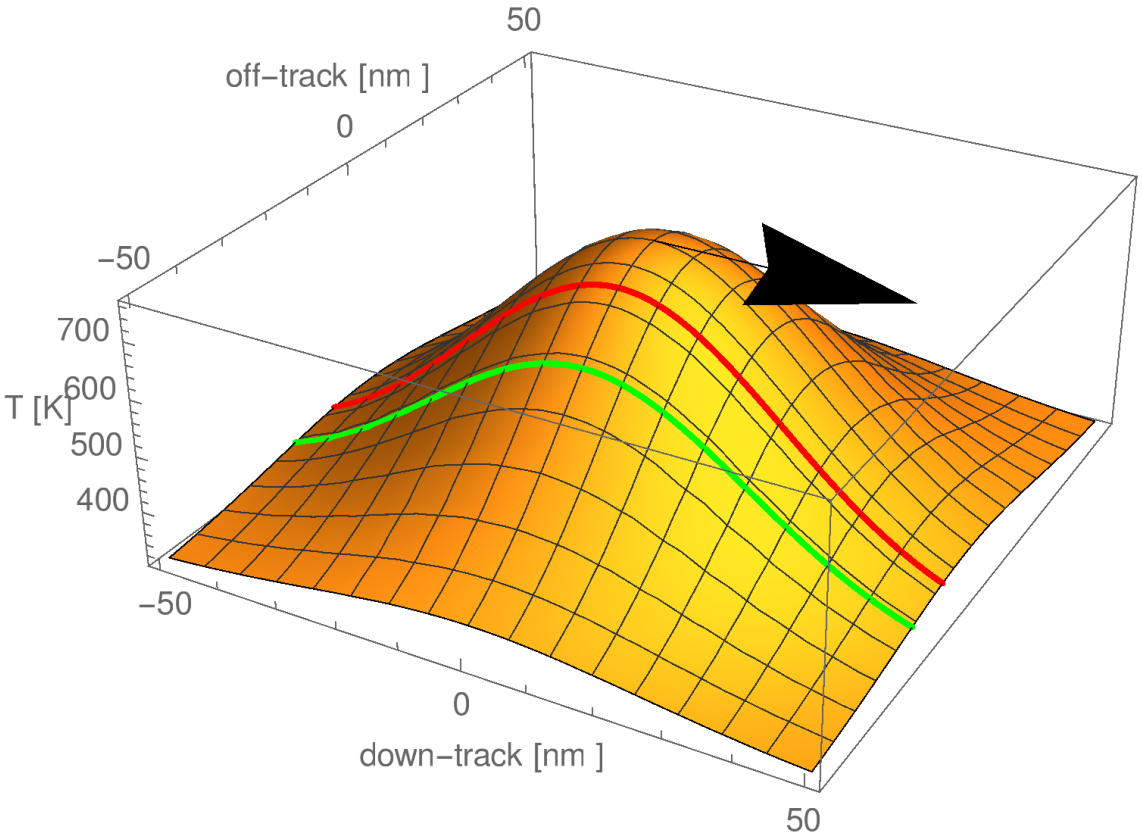


Figure 6.5: Visualization of the Gaussian heat pulse that moves across the granular medium in the direction of the arrow. Together with an applied magnetic field it performs the writing process. The red and green curve demonstrate that grains receive different peak temperatures depending on their off-track position.

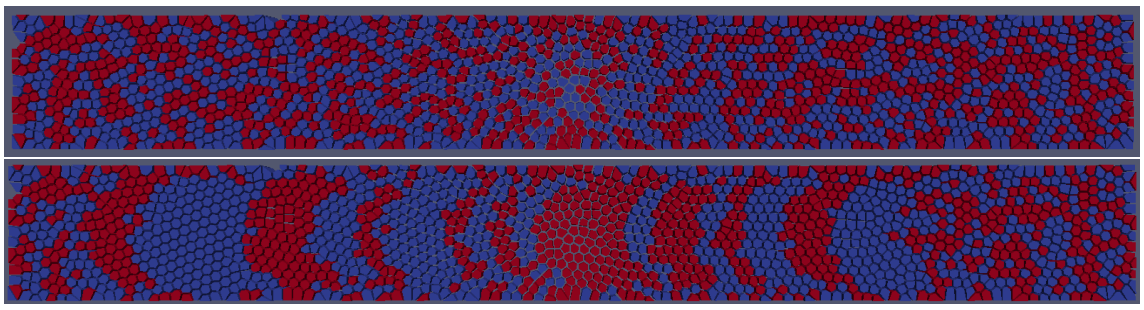


Figure 6.6: Top: Randomly initialized granular medium (500 × 60 nm and a thickness of 8 nm) with grain diameter of 4 nm and 1 nm gap between neighboring grains. The colors distinguish magnetizations in up or down direction (blue: down, red: up). Bottom: Granular medium after the simulated writing process.

6.3.2 Reading process

The reader module is defined via its sensitivity function as in [8]. The potential is illustrated in Fig. 6.7 (a). The integral in [8, (1)]

$$\int H \cdot M \, \mathrm{d}V_m$$

of the reader's sensitivity function H and the media's magnetization M is assumed to have negligible dependence on the z-direction. Therefore it degenerates to an area integral, which is computed via a discrete sum over the data points in the sensitivity function, multiplied with $-1, +1$ oder 0 depending on whether the data point is within a grain with magnetization in down-, up-direction or a gap. Moving the sensitivity function in 0.5 nm steps across the medium gives us the detected read-back signal of the whole bit pattern as in Fig. 6.7 (b).

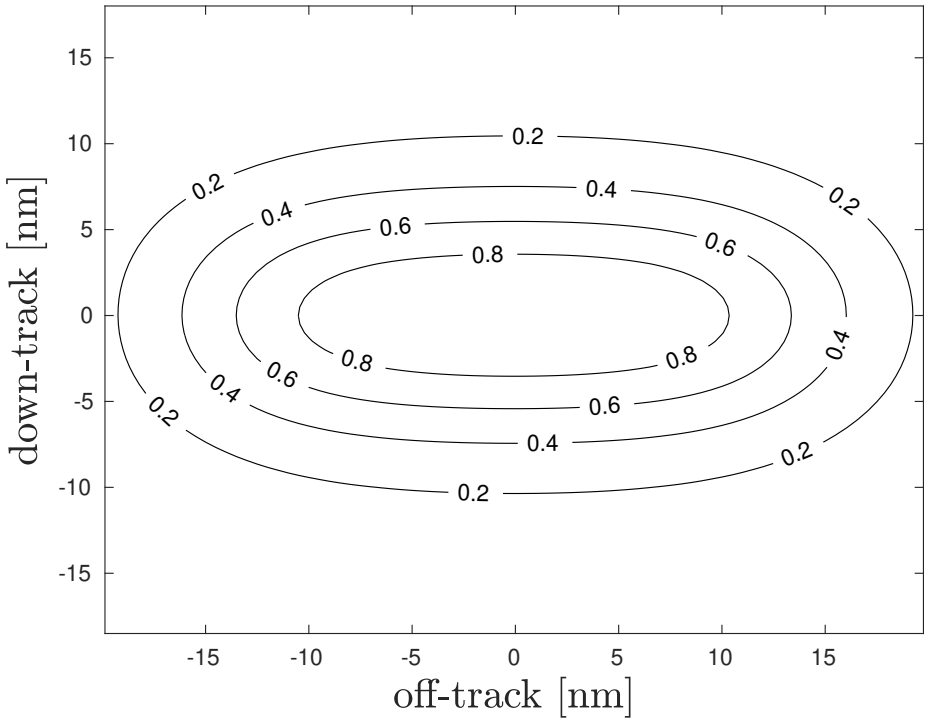
6.3.3 SNR calculation

By repeating the writing and reading process on 50 different randomly initialized granular media, we are able to calculate the SNR with the methods of [3]. To be compatible with the SNR calculator, the data points of every read-back signal are therefore interpolated to 9 samples per bit. Fig. 6.8 exemplary shows the resulting curve shape of the detected signals.

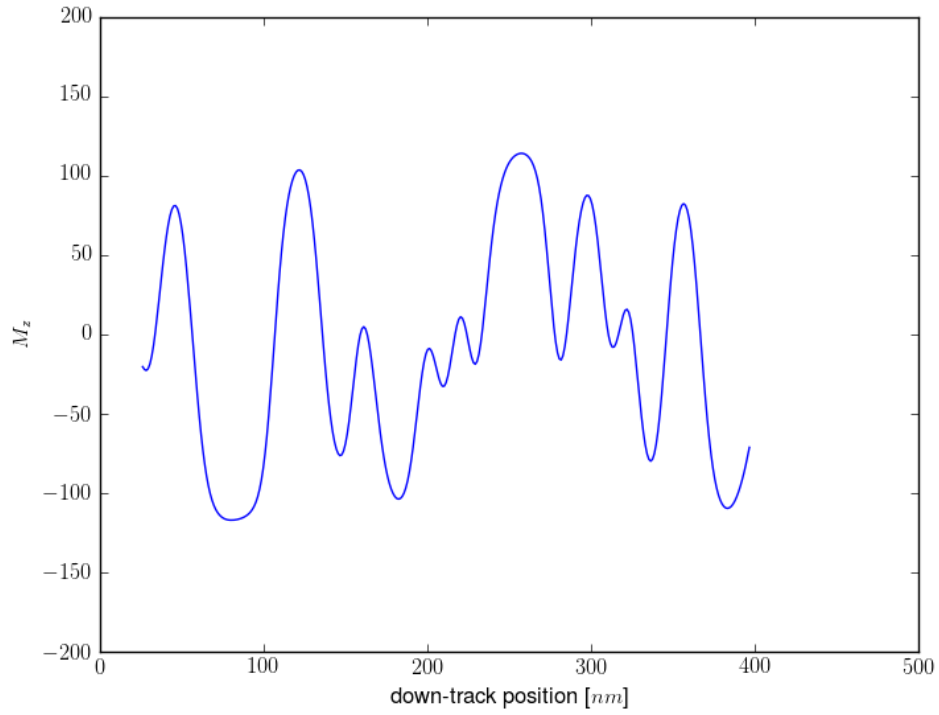
6.4 Results and Discussion

6.4.1 SNR curves

For the different parameter variations, the resulting SNR curves are plotted in Fig. 6.9. For selected values for the parameters, Fig. 6.10 - 6.13 visualize the corresponding bit pattern. The impact of the varied parameters on the SNR can clearly be observed. The curves describing the dependence on bit length, $P_{\max}$ and the down-track-jitter σ_d demonstrate a significant slope. The bit length is indirect proportional to the linear density on the bits, so it is usually very desirable to choose it as low as possible. The results show nevertheless a sharp degressive decrease of the SNR for low bit lengths, so compromises are necessary. $P_{\max}$ and σ_d are clearly two parameters with significantly impact on the SNR, so it is recommendable to consider those values in terms of material optimization as in [7]. The slopes of the curves $\Delta\mathrm{SNR}/\Delta P_{\max}$ and $\Delta\mathrm{SNR}/\Delta\sigma_d$ give an indication about the profit of improvements concerning $P_{\max}$ and σ_d. Within the chosen reference system, the curvature reduction does not influence the SNR strongly. The main reason of this observation might be the chosen ratio between the FWHM of the heat pulse in the writing process (see Sec. 6.2.3 and Fig. 6.5) and the reader width (see Fig. 6.7 (a)). For larger reader width or smaller FWHM a bigger influence is expected.



(a)



(b)

Figure 6.7: (a): Conture plot of the sensitivity function with a data point resolution of 0.5 nm in down- and off-track-direction. (b) Example for a read-back curve determined by the reader in form of the sensitivity function in (a) across the granular medium in Fig. 6.6.

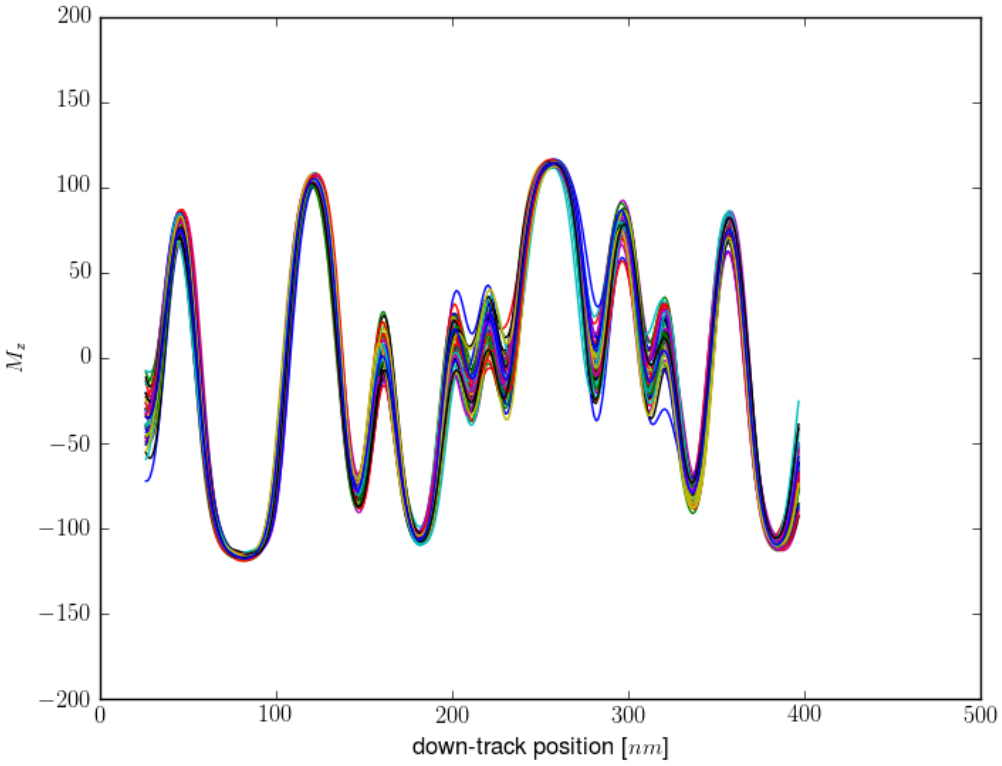


Figure 6.8: The read-back signal of 50 bit pattern written with the same model phase plot, but on different granular media. The shape of the curves is used for the SNR calculation of the signal.

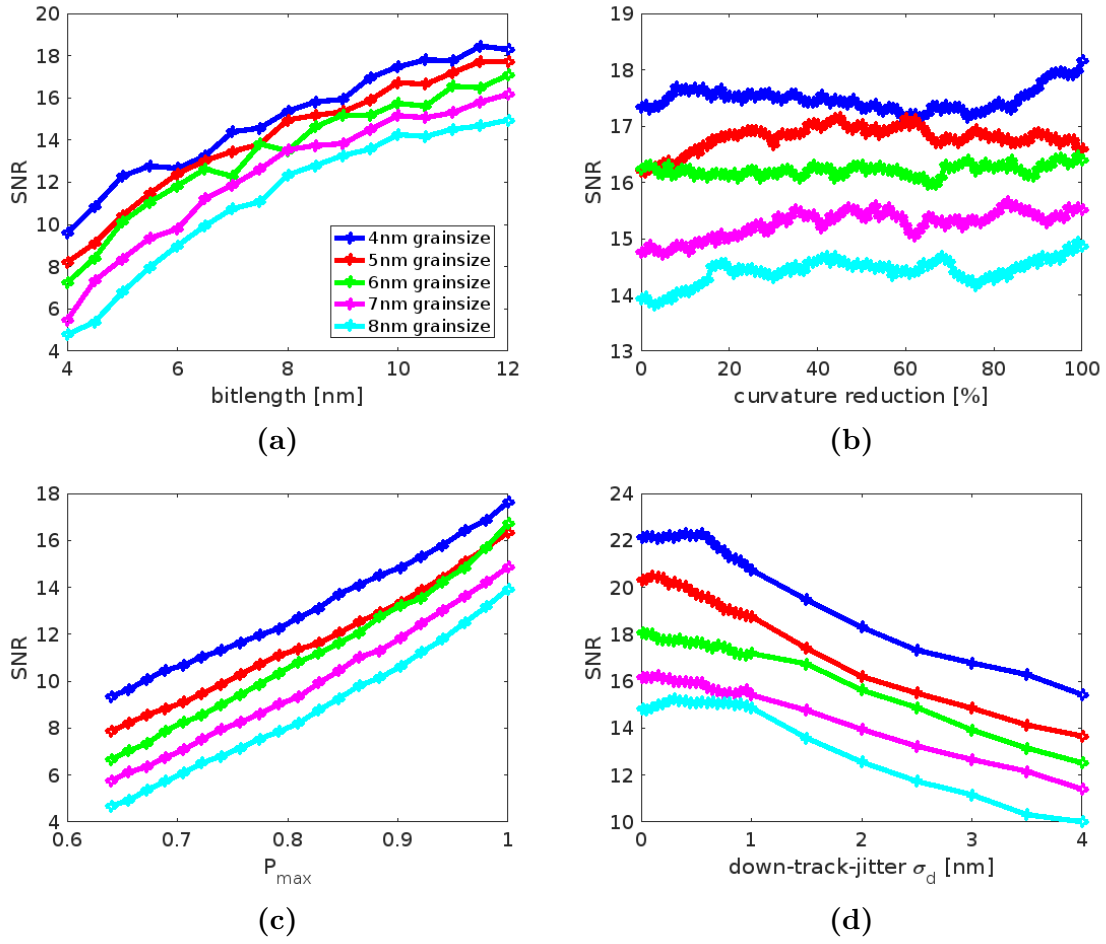


Figure 6.9: Results of the SNR calculation for various parameters.

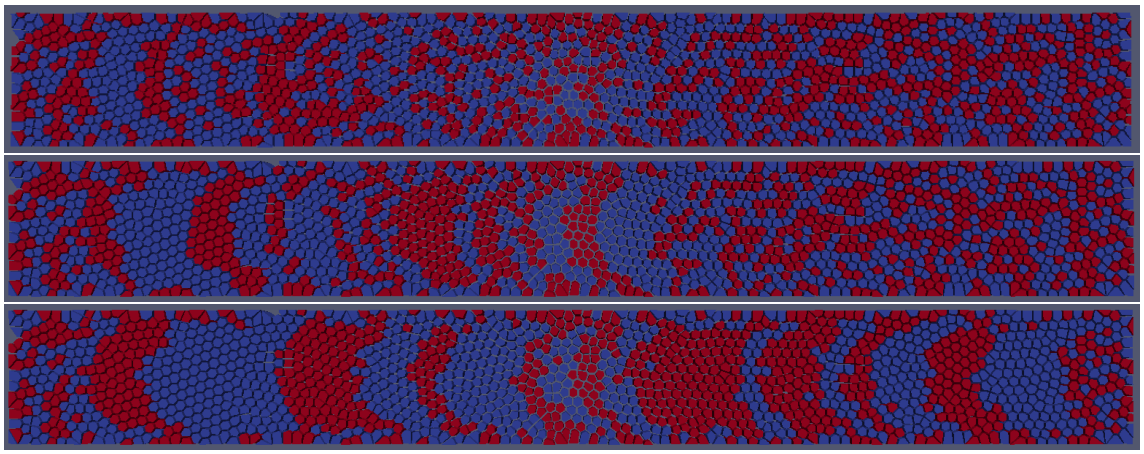


Figure 6.10: Bit pattern for bit lengths of $4, 7$ and 12 nm (top to bottom).

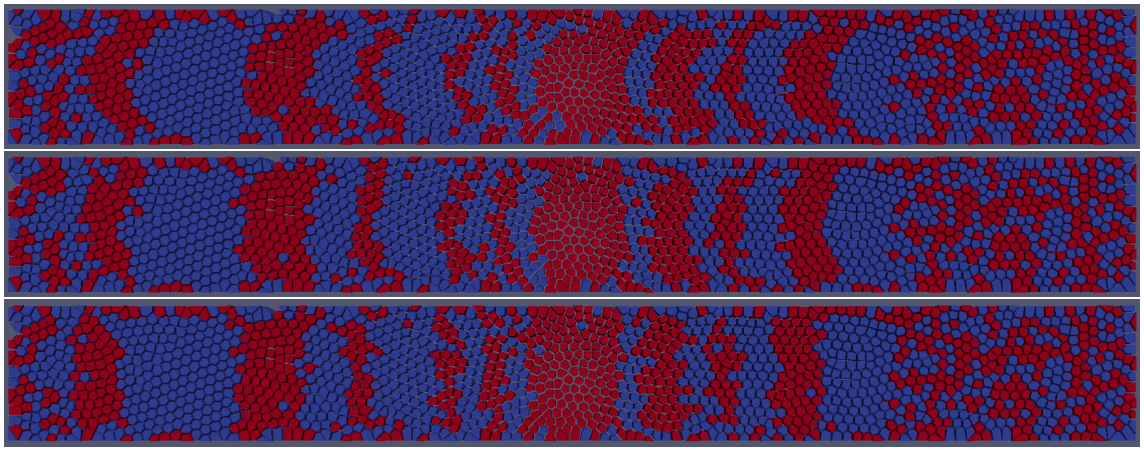


Figure 6.11: Bit pattern for curvature reductions of $0, 50$ and 100 % (top to bottom).

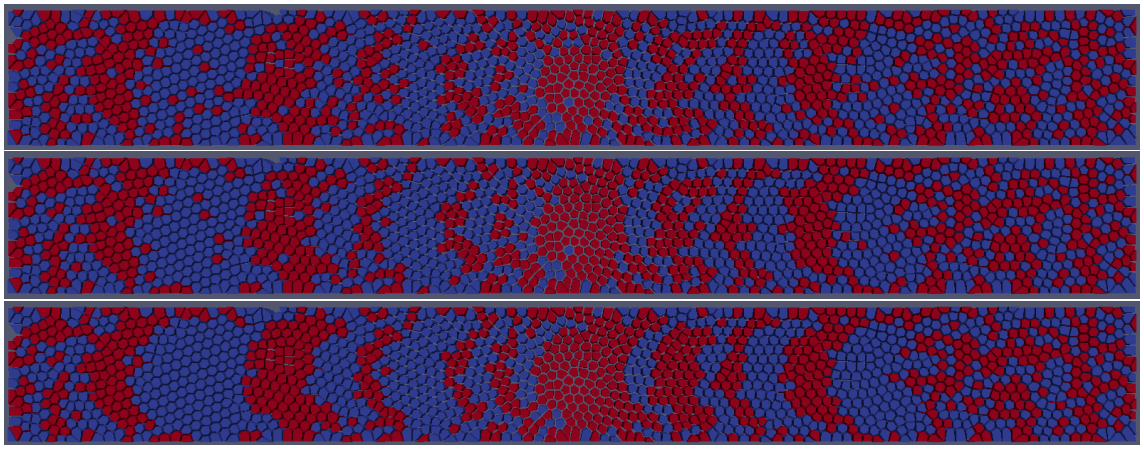


Figure 6.12: Bit pattern for $P_{\max} = 0.64, 0.81$ and 1.00 (top to bottom).

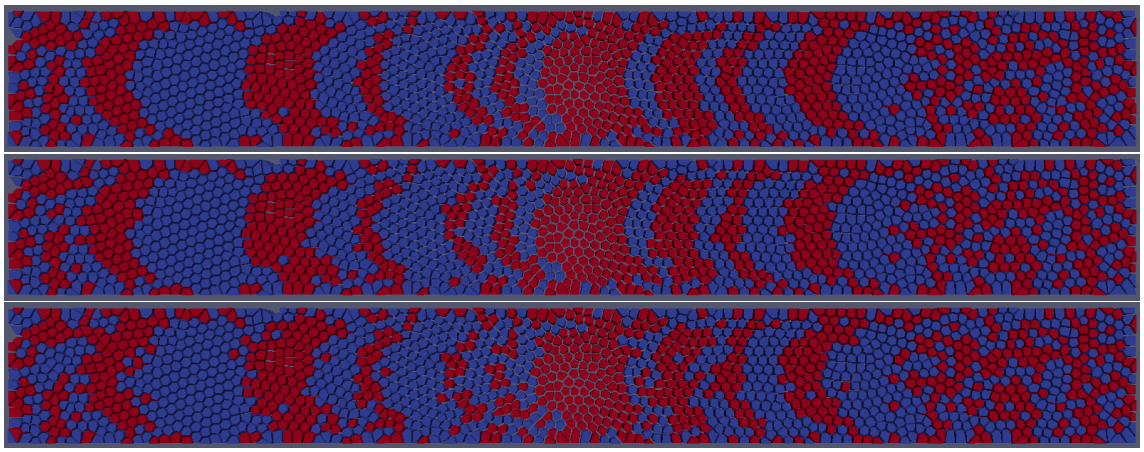


Figure 6.13: Bit pattern for down-track-jitters of $0.01, 2.00$ and 4.00 nm (top to bottom).

6.4.2 Comparison with theory

In [9, (2.4)] the SNR-dependence on the writing-jitter σ_w is given by

$$SNR \propto \sigma_w^{-2} \tag{6.4.1}$$

where the proportionality constant depends, inter alia, on the bit length, which we assume to be constant for the moment. As in [13, (4)], the writing-jitter can be separated into two independent parts:

- the down-track-jitter σ_d, that originates from the probability distribution in the phase diagram
- the grain-jitter σ_g, that comes from the grain distribution in the granular medium

The independence ensures that the total jitter of these two parts can be written as

$$\sigma_w = \sqrt{\sigma_d^2 + \sigma_g^2}. \tag{6.4.2}$$

Note that this consideration actually holds only for one down-track-row of grains and the right-hand side in Eq. (6.4.1) should therefore be multiplied with the average number of grains in cross-track direction (as in [4]). However we can include that factor in the proportionality constant for fixed grain size. In literature (see e.g. [14], [9, Sec. 2.1.3]), this value is also commonly designated as the magnetic transition parameter a. The grain-jitter σ_g usually depends on the mean and the variation of the grain sizes. In our simulations, we assumed the distribution to be very sharp, i.e. we neglect the dependence on the variations and assume all grains to have approximately the same size D. In this case σ_g only depends on D and a very simplified model with square grains in [13] shows

$$\sigma_g = \frac{D}{\sqrt{12}}. \tag{6.4.3}$$

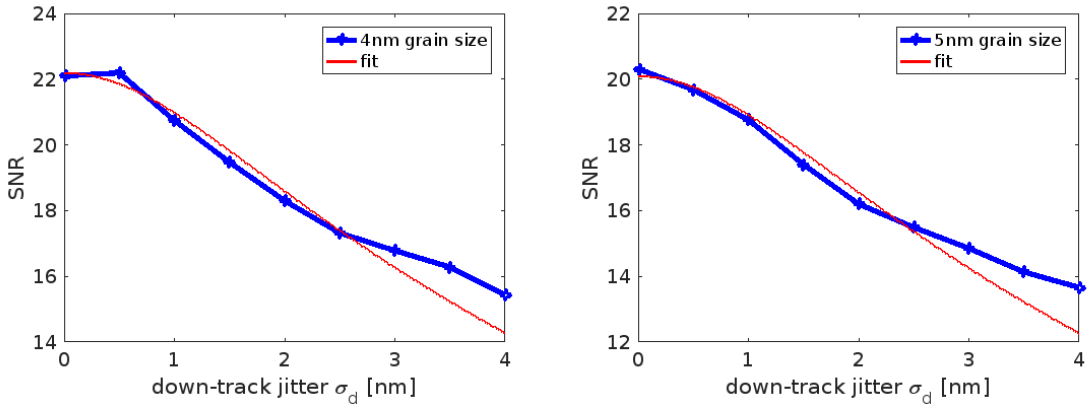
We assume σ_g to be slightly larger, due to the following reasons:

- In the model, D denotes the side length of the square, not the diameter of the quadratic grain.
- Polygons with (mostly) more than 4 sides cannot be aligned that close to each other in a row as squares do in [13, Fig. 1].

Therefore we chose a second fit parameter f_2 (apart from the proportionality constant f_1) and fit Eq. (6.4.1) in the final form

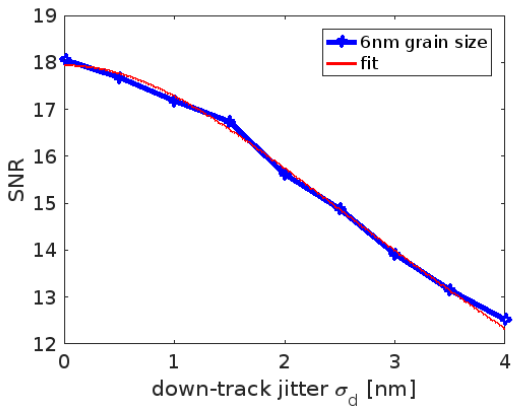
$$SNR(f_1, f_2, \sigma_d) = f_1 \cdot \left(\sigma_d^2 + f_2^2 \cdot \frac{D^2}{12}\right)^{-1}. \tag{6.4.4}$$

The results for the grain sizes $D = 4, 5, 6, 7, 8$ nm are shown in Fig. 6.14. For the fitting parameter f_2, we archive the values in Table 6.4.
The plots show that the trend of the fit curve is quite suitable for those right values of f_2, which are nevertheless quite close to 1.

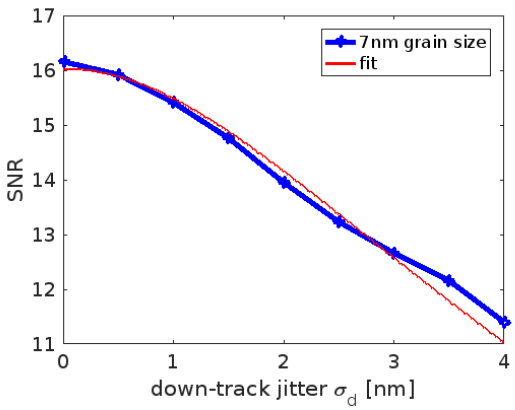


(a)

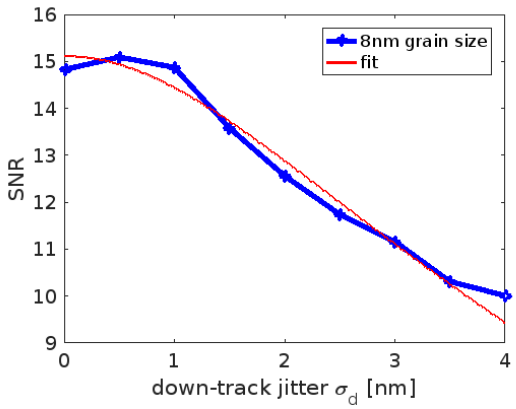
(b)



(c)



(d)



(e)

Figure 6.14: Fitting curve of the SNR calculation for various down-track-jitter parameter.

	grainsize				
	4 nm	5 nm	6 nm	7 nm	8 nm
f_2	1.52	1.23	1.42	1.34	1.05

Table 6.4: Optimal values of the correction factor of the formula in Eq. (6.4.3) determined by least square fit.

6.5 Conclusion

As the phase diagram (switching probability as a function of peak temperature and down-track position) allows us to write bit pattern on granular media, the main goal of this paper was to investigate the influence of different characteristics of the phase diagram on the SNR value. This influence is mostly nontrivial, therefore we created a mathematical model that allows us to vary individual parameters to show their impact on the total SNR of written bit series. Within this model we chose different values for the bit length, curvature, maximum switching probability and down-track-jitter and plotted the SNR dependency of those values. The main results showed, that the curvature reduction does not significantly increase the SNR for the chosen dimensions of heat pulse and reader-width. Variation of the bit length, $P_{\max}$ and down-track-jitter showed on the other hand a distinct influence. For considerations as in [7], where the material composition is optimized for a minimal jitter, the slope in our curves are useful to give an indication about the SNR-influence of such improvements, without performing a whole expensive simulation for SNR-calculation with a bit series on granular media. Finally we showed some comparison with theoretical formulas, which is also an application of the shown procedure. A computationally cheap empirical validation of different formulas was possible, even though we had to append an additional correction factor that we included to our fit parameters. This was necessary to receive a proper fit and therefore a better coincidence to theory.

Acknowledgment

The authors would like to thank the Austrian Science Fund (FWF) under grant No. I2214-N20 for financial support. The computational results presented have been achieved using the Vienna Scientific Cluster (VSC). Further thanks to Stephanie Hernández for providing the SNR calculator from SEAGATE, which we gratefully used for the SNR calculation of the read-back signal and Prof. Randall Victora for providing the sensitivity function of the reader.

Appendix A

Mathematical background

A.1 The shape of $\boldsymbol{J}_{ij}$

In this section we give the proof, that for isotropic space the term in (3.3.1) can be written as

$$\vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j = J_{ij} \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j \tag{A.1.1}$$

with a scalar value J_{ij} on the right hand side. Although we use this equation only in $\mathbb{R}^3$, we give a general proof for dimensions $d \geq 3$. We start with a mathematical definition of isotropy in space.

Definition A.1.1. *The bilinear and scalar function*

$$\begin{aligned} \mathbb{R}^d \times \mathbb{R}^d &\longrightarrow \mathbb{R} \\ (\vec{\boldsymbol{S}}_i, \vec{\boldsymbol{S}}_j) &\longmapsto \vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j \end{aligned}$$

is called ***isotropic in space****, if it is invariant under orthogonal transformations with determinant* $= +1$*, i.e.*

$$\vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j = (\boldsymbol{Q}\vec{\boldsymbol{S}}_i)^T \boldsymbol{J}_{ij} (\boldsymbol{Q}\vec{\boldsymbol{S}}_j) \qquad \forall \boldsymbol{Q} \in SO(d)$$

what is equivalent to the matrix $\boldsymbol{J}_{ij}$ *fulfilling*

$$\boldsymbol{J}_{ij} = \boldsymbol{Q}^T \boldsymbol{J}_{ij} \boldsymbol{Q} \qquad \forall \boldsymbol{Q} \in SO(d).$$

The following lemma is based on [6, equation (18)], where a formula of a general rotation in $\mathbb{R}^d$ is given. We use it to proof the main theorem afterwards.

Lemma A.1.2. *For* $d \geq 3$ *and two unit column vectors* $\vec{\boldsymbol{S}}_i, \vec{\boldsymbol{S}}_j \in \mathbb{R}^d, \vec{\boldsymbol{S}}_i \neq \vec{\boldsymbol{S}}_j$ *we define the quadratic matrix*

$$\boldsymbol{R} := \mathbb{1} - 2\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 2\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T,$$

with $\vec{\boldsymbol{r}} := (\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j)/\|\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\|$ *and* $\vec{\boldsymbol{p}}$ *an arbitrary unit vector with* $\vec{\boldsymbol{p}} \perp \vec{\boldsymbol{S}}_i$ *and* $\vec{\boldsymbol{p}} \perp \vec{\boldsymbol{S}}_j$*. Then* $\boldsymbol{R}$ *performs a rotation that fulfills* $\boldsymbol{R}\vec{\boldsymbol{S}}_i = \vec{\boldsymbol{S}}_j$ *and* $\boldsymbol{R}\vec{\boldsymbol{S}}_j = \vec{\boldsymbol{S}}_i$ *(see Fig. A.1).*

Proof. 1st step:
In [6] a matrix in the shape of $\boldsymbol{R}$ is derived as rotation matrix, so we skip to prove $\det(\boldsymbol{R}) = +1$. However we can show easily that this is in fact an orthogonal matrix by verifying $\boldsymbol{R}^T\boldsymbol{R} = \mathbb{1}$. First we remark that $\boldsymbol{R}^T = \boldsymbol{R}$ because

$$\boldsymbol{R}^T = \left(\mathbb{1} - 2\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 2\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T\right)^T = \mathbb{1}^T - 2(\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T)^T - 2(\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T)^T = \mathbb{1} - 2\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 2\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T = \boldsymbol{R}$$

and so we get

$$\begin{aligned}\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R} &= \left(\mathbb{1} - 2\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 2\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T\right)\left(\mathbb{1} - 2\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 2\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T\right) \\ &= \mathbb{1} - 4\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T - 4\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T + 4\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T + 4\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T + 4\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T + 4\vec{\boldsymbol{p}}\vec{\boldsymbol{p}}^T\vec{\boldsymbol{r}}\vec{\boldsymbol{r}}^T.\end{aligned}$$

The choice of $\vec{\boldsymbol{p}}$ with $\vec{\boldsymbol{p}} \perp \vec{\boldsymbol{S}}_i$ and $\vec{\boldsymbol{p}} \perp \vec{\boldsymbol{S}}_j$ implies $\vec{\boldsymbol{p}} \perp \vec{\boldsymbol{r}}$ and therefore $\vec{\boldsymbol{r}}^T\vec{\boldsymbol{p}} = 0 = \vec{\boldsymbol{p}}^T\vec{\boldsymbol{r}}$, so the last two summands vanish. Both $\vec{\boldsymbol{r}}$ and $\vec{\boldsymbol{p}}$ are unit vectors with norm $= 1$, so $\vec{\boldsymbol{r}}^T\vec{\boldsymbol{r}} = 1 = \vec{\boldsymbol{p}}^T\vec{\boldsymbol{p}}$ and the remaining terms are canceling out. We have finally shown $\boldsymbol{R}^T\boldsymbol{R} = \mathbb{1}$.
2nd step:
We first calculate the squared norm of $\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j$:

$$\|\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\|^2 = (\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j)^T(\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j) = \vec{\boldsymbol{S}}_i{}^T\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_i{}^T\vec{\boldsymbol{S}}_j - \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i + \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_j \quad \text{(A.1.2)}$$

The unit vectors $\vec{\boldsymbol{S}}_i$ and $\vec{\boldsymbol{S}}_j$ fulfill

$$\vec{\boldsymbol{S}}_i{}^T\vec{\boldsymbol{S}}_i = \|\vec{\boldsymbol{S}}_i\|^2 = 1 = \|\vec{\boldsymbol{S}}_j\|^2 = \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_j$$

and the scalar value $\vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i$ is identical with its transposed, so we get

$$\vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i = \left(\vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i\right)^T = \vec{\boldsymbol{S}}_i{}^T\vec{\boldsymbol{S}}_j. \quad \text{(A.1.3)}$$

Using these identities in (A.1.2), we finally get

$$\|\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\|^2 = 2\left(1 - \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i\right).$$

We now use this result to calculate

$$\begin{aligned}\boldsymbol{R}\vec{\boldsymbol{S}}_i &= \left(\mathbb{1} - \frac{2}{\|\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\|^2}\left(\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\right)\left(\vec{\boldsymbol{S}}_i{}^T - \vec{\boldsymbol{S}}_j{}^T\right)\right)\vec{\boldsymbol{S}}_i \\ &= \left(\mathbb{1} - \frac{\cancel{2}}{\cancel{2}\left(1 - \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i\right)}\left(\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\right)\left(\vec{\boldsymbol{S}}_i{}^T - \vec{\boldsymbol{S}}_j{}^T\right)\right)\vec{\boldsymbol{S}}_i \\ &= \vec{\boldsymbol{S}}_i - \frac{1}{\cancel{\left(1 - \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i\right)}}\left(\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\right)\cancel{\left(1 - \vec{\boldsymbol{S}}_j{}^T\vec{\boldsymbol{S}}_i\right)} \\ &= \vec{\boldsymbol{S}}_i - \left(\vec{\boldsymbol{S}}_i - \vec{\boldsymbol{S}}_j\right) = \vec{\boldsymbol{S}}_j.\end{aligned}$$

The identity $\boldsymbol{R}\vec{\boldsymbol{S}}_j = \vec{\boldsymbol{S}}_i$ can be proved analogously using (A.1.3). □

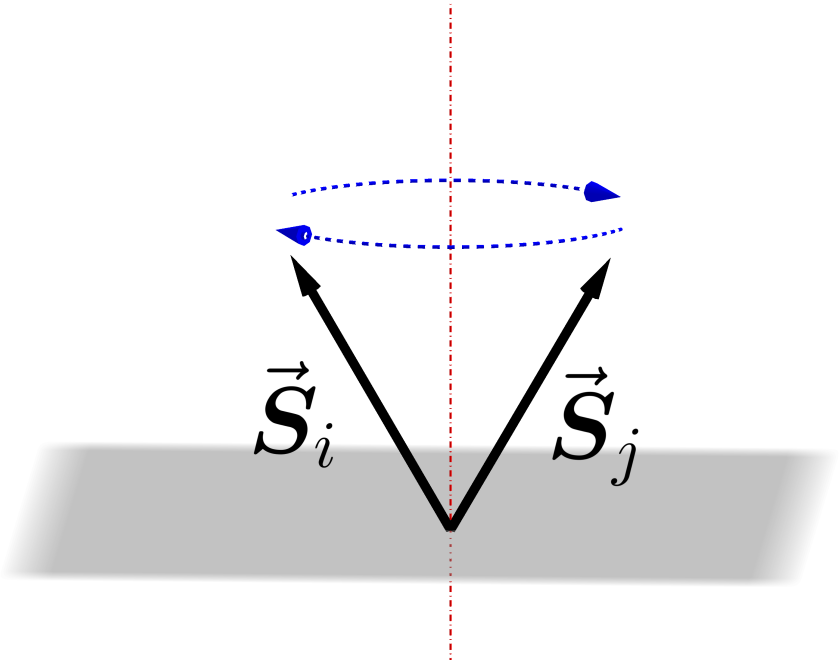


Figure A.1: Illustration of the rotation $\boldsymbol{R}$ that exchanges $\vec{\boldsymbol{S}}_{\boldsymbol{i}}$ and $\vec{\boldsymbol{S}}_{\boldsymbol{j}}$. for $d \geq 3$.

Theorem A.1.3. *Let the dimension of space be $d \geq 3$. Then isotropy in space in definition A.1.1 holds if and only if there exists a scalar value J_{ij} with* (A.1.1) *for all $\vec{\boldsymbol{S}}_{\boldsymbol{i}}, \vec{\boldsymbol{S}}_{\boldsymbol{j}} \in \mathbb{R}^d$.*

Proof. 1st step:
If (A.1.1) holds with a scalar value J_{ij}, we get isotropy in space immediately, because every matrix $\boldsymbol{Q} \in SO(d)$ fulfills $\boldsymbol{Q}^T\boldsymbol{Q} = \mathbb{1}$ and therefore

$$(\boldsymbol{Q}\vec{\boldsymbol{S}}_i)^T\boldsymbol{J}_{ij}(\boldsymbol{Q}\vec{\boldsymbol{S}}_j) = J_{ij}(\boldsymbol{Q}\vec{\boldsymbol{S}}_i)\cdot(\boldsymbol{Q}\vec{\boldsymbol{S}}_j) = J_{ij}\vec{\boldsymbol{S}}_i^T\underbrace{\boldsymbol{Q}^T\boldsymbol{Q}}_{=\mathbb{1}}\vec{\boldsymbol{S}}_j = J_{ij}\vec{\boldsymbol{S}}_i\cdot\vec{\boldsymbol{S}}_j = \vec{\boldsymbol{S}}_i^T\boldsymbol{J}_{ij}\vec{\boldsymbol{S}}_j. \qquad \checkmark$$

2nd step:
We now assume that isotropy in space holds. With lemma A.1.2 for every $\vec{\boldsymbol{S}}_{\boldsymbol{i}}, \vec{\boldsymbol{S}}_{\boldsymbol{j}} \in \mathbb{R}^d$ a rotation matrix $\boldsymbol{R}$ can be constructed with

$$\vec{\boldsymbol{S}}_i^T\boldsymbol{J}_{ij}\vec{\boldsymbol{S}}_j = (\underbrace{\boldsymbol{R}\vec{\boldsymbol{S}}_i}_{=\vec{\boldsymbol{S}}_j})^T\boldsymbol{J}_{ij}(\underbrace{\boldsymbol{R}\vec{\boldsymbol{S}}_j}_{=\vec{\boldsymbol{S}}_i}) = \vec{\boldsymbol{S}}_j^T\boldsymbol{J}_{ij}\vec{\boldsymbol{S}}_i.$$

If we choose Cartesian unit vectors for $\vec{\boldsymbol{S}}_{\boldsymbol{i}}$ and $\vec{\boldsymbol{S}}_{\boldsymbol{j}}$ in this equation, we see that $\boldsymbol{J}_{ij}$ has to be symmetrical.
3rd step:
Whereas $\boldsymbol{J}_{ij}$ is symmetrical, it is a well known fact from linear algebra, that $\boldsymbol{J}_{ij}$ is orthogonally diagonalizable, i.e. there exists a matrix $\boldsymbol{Q} \in SO(d)$ with $\boldsymbol{Q}^T\boldsymbol{J}_{ij}\boldsymbol{Q} = \boldsymbol{D}_{ij}$ where $\boldsymbol{D}_{ij}$ is a diagonal matrix. Isotropy in space implies that $\boldsymbol{J}_{ij} = \boldsymbol{D}_{ij}$, therefore $\boldsymbol{J}_{ij}$ is a diagonal matrix.
4th step:

If $\vec{e}_k$ is the k-th canonical unit vector, there holds

$$\vec{e}_k^T \boldsymbol{J}_{ij} \vec{e}_k = (\boldsymbol{J}_{ij})_{kk}$$

whereas $(\boldsymbol{J}_{ij})_{kk}$ is the k-th diagonal entry of the diagonal matrix $(\boldsymbol{J}_{ij})_{kk}$. We can now construct again a matrix $\boldsymbol{R}$ as is lemma A.1.2, where $\vec{\boldsymbol{S}}_{\boldsymbol{i}}, \vec{\boldsymbol{S}}_{\boldsymbol{j}}$ are arbitrary canonical unit vectors $\vec{e}_k$ and $\vec{e}_\ell$. Then the isotropy in space implies

$$(\boldsymbol{J}_{ij})_{kk} = \vec{e}_k^T \boldsymbol{J}_{ij} \vec{e}_k = (\boldsymbol{R}\vec{e}_k)^T \boldsymbol{J}_{ij} (\boldsymbol{R}\vec{e}_k) = \vec{e}_\ell^T \boldsymbol{J}_{ij} \vec{e}_\ell = (\boldsymbol{J}_{ij})_{\ell\ell}$$

and this equality holds for any k, ℓ. So all diagonal elements of $\boldsymbol{J}_{ij}$ are equal and the matrix is a multiply of the unit matrix, i.e. $\boldsymbol{J}_{ij} = J_{ij}\mathbb{1}$. Therefore for arbitrary $\vec{\boldsymbol{S}}_i, \vec{\boldsymbol{S}}_j \in \mathbb{R}^d$ holds

$$\vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j = J_{ij} \vec{\boldsymbol{S}}_i^T \mathbb{1} \vec{\boldsymbol{S}}_j = J_{ij} \vec{\boldsymbol{S}}_i^T \vec{\boldsymbol{S}}_j = J_{ij} \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j$$

what concludes the proof. □

Note A.1.4. *An remarkable fact of the previous theorem is the necessity of the assumption $d \geq 3$. The case $d = 1$ is not interesting, because every matrix degenerates to a scalar and therefore the equivalent statements of theorem A.1.3 are both trivially fulfilled, but for $d = 2$ the theorem is not valid. The graphical illustration of this fact is, that for $d = 2$ it is not possible for two given arbitrary unit vectors to find a rotation that exchanges the vectors as in Fig. A.1. The critical point is the postulation of determinant $= +1$ in definition A.1.1. If we allow the determinant to be $= -1$ we can perform the exchange via a reflection as in Fig. A.1. We deal with that case in the following.*

The advantage of the case $d = 2$ is that we have the simple general form of orthogonal transformations, which we present in the following lemma:

Lemma A.1.5. *In the case $d = 2$ every orthogonal transformation with determinant equal to $+1$ can be written in the form*

$$\boldsymbol{Q} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{A.1.4}$$

with $\alpha \in [0, 2\pi)$. This matrix describes a rotation around the origin with angle α. □

Proof. First we assume

$$\boldsymbol{Q} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

as arbitrary 2×2 matrix with $a, b, c, d \in \mathbb{R}$. Orthogonality leads to

$$\mathbb{1} = \boldsymbol{Q}^T \boldsymbol{Q} = \begin{pmatrix} a^2 + c^2 & ab + cd \\ ab + cd & b^2 + d^2 \end{pmatrix}$$

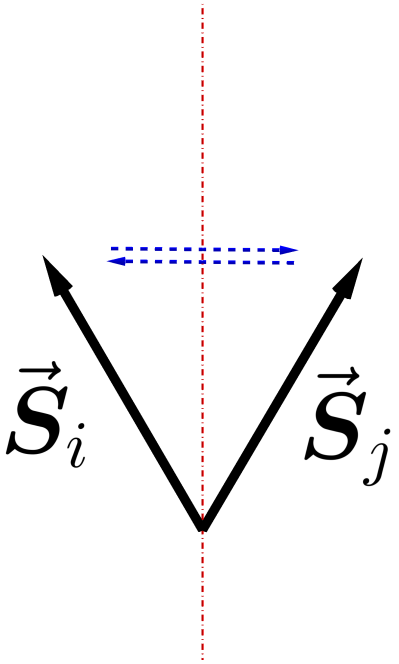


Figure A.2: Illustration of a reflection that exchanges $\vec{S}_i$ and $\vec{S}_j$ for $d = 2$ instead of the rotation in figure A.1.

and $\det(\boldsymbol{Q}) = ad - bc = 1$, so we have four conditions to the entries:

$$a^2 + c^2 = 1 \tag{A.1.5}$$
$$b^2 + d^2 = 1 \tag{A.1.6}$$
$$ab + cd = 0 \tag{A.1.7}$$
$$ad - bc = 1 \tag{A.1.8}$$

If we take the sum $b \cdot$ (A.1.7) $+ d \cdot$ (A.1.8) and the difference $d \cdot$ (A.1.7) $- b \cdot$ (A.1.8) we get

$$ab^2 + ad^2 = d \iff a(b^2 + d^2) = d \overset{\text{(A.1.6)}}{\iff} a = d,$$
$$cd^2 + b^2c = -b \iff c(b^2 + d^2) = -b \overset{\text{(A.1.6)}}{\iff} b = -c,$$

so the shape of $\boldsymbol{Q}$ is

$$\boldsymbol{Q} = \begin{pmatrix} a & -c \\ c & a \end{pmatrix}$$

and because (A.1.5) is the equation of a unit circle in the coordinates a and c we can parametrize it by setting $a = \cos\alpha$ and $c = \sin\alpha$ with an arbitrary $\alpha \in [0, 2\pi)$. □

Proposition A.1.6. *Let the dimension of space be $d = 2$. Then isotropy in space in definition A.1.1 holds if there exists a scalar value J_{ij} with* (A.1.1) *for all $\vec{S}_i, \vec{S}_j \in \mathbb{R}^d$ but the conversely implication is not true, i.e. there exists a matrix $\boldsymbol{J}_{ij} \in \mathbb{R}^2$ that fulfills the requirements of isotropy in space of definition A.1.1 but* (A.1.1) *is not true for any $J_{ij} \in \mathbb{R}$.*

Proof. The first implication can be shown in the same way as it was done in the 1st step of the proof of theorem A.1.3. As a counterexample for the opposite, we define

the matrix

$$\boldsymbol{J}_{ij} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

If we transform it with an arbitrary matrix $\boldsymbol{Q}$ in the shape of (A.1.4), we get[1]

$$\begin{aligned}
\boldsymbol{Q}^T \boldsymbol{J}_{ij} \boldsymbol{Q} &= \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \\
&= \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -\sin\alpha & -\cos\alpha \\ \cos\alpha & -\sin\alpha \end{pmatrix} \\
&= \begin{pmatrix} -\cos\alpha\sin\alpha + \cos\alpha\sin\alpha & -\cos^2\alpha - \sin^2\alpha \\ \sin^2\alpha + \cos^2\alpha & \cos\alpha\sin\alpha - \cos\alpha\sin\alpha \end{pmatrix} \\
&= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \boldsymbol{J}_{ij},
\end{aligned}$$

i.e. $\boldsymbol{J}_{ij}$ is isotropic in space. Nevertheless (A.1.1) is not fulfilled because for $\vec{\boldsymbol{S}}_i = \vec{\boldsymbol{e}}_1$ and $\vec{\boldsymbol{S}}_j = \vec{\boldsymbol{e}}_2$ there holds

$$\vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -1 \quad \neq \quad 0 = J_{ij} \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = J_{ij} \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j$$

for any scalar J_{ij}. □

Proposition A.1.7. *If we skip the condition "determinant = +1" in definition A.1.1 and also allow orthogonal matrices with determinant = −1, i.e. reflections of space, then theorem A.1.3 also holds in the case $d = 2$.*

Proof. With proposition A.1.6 is remains to show:
Isotropy in space plus the invariance under orthogonal transformations with determinant = −1 imply the existence of a scalar value J_{ij} with (A.1.1) for all $\vec{\boldsymbol{S}}_{\boldsymbol{i}}, \vec{\boldsymbol{S}}_{\boldsymbol{j}} \in \mathbb{R}^2$.

We proof that by using the invariance of

$$\boldsymbol{J}_{ij} = \begin{pmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{pmatrix}$$

under selected orthogonal transformations $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$. First we define

$$\boldsymbol{Q}_1 := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

[1] If we regard the chosen $\boldsymbol{J}_{ij}$ as a rotation in the shape of (A.1.4) for $\alpha = \pi$ the result of the calculation is not surprising because $\boldsymbol{Q}^T \boldsymbol{J}_{ij} \boldsymbol{Q}$ is the composition of rotations with angles $-\alpha, \pi$ and α which yields a rotation with π. Instead of $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ every matrix in the shape of $\begin{pmatrix} j_{11} & -j_{12} \\ j_{12} & j_{11} \end{pmatrix}$ with arbitrary $j_{11}, j_{12} \in \mathbb{R}$ could also be taken as a counterexample.

After lemma A.1.5 with $\alpha = \pi$, this matrix is orthogonal with determinant $= +1$. Isotropy in space implies

$$\begin{pmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{pmatrix} = \boldsymbol{J}_{ij} = \boldsymbol{Q}_1^T \boldsymbol{J}_{ij} \boldsymbol{Q}_1 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} j_{12} & -j_{11} \\ j_{22} & -j_{21} \end{pmatrix} = \begin{pmatrix} j_{22} & -j_{21} \\ -j_{12} & j_{11} \end{pmatrix},$$

so $j_{11} = j_{22}$ and $-j_{12} = j_{21}$ is forced and the matrix can be written in the form

$$\boldsymbol{J}_{ij} = \begin{pmatrix} j_{11} & j_{12} \\ -j_{12} & j_{11} \end{pmatrix}.$$

We have already mentioned in footnote 1 that this matrix is invariant under every orthogonal transformations with determinant $= +1$. So in the following it is indispensable to select an orthogonal matrix with determinant $= -1$, e.g.

$$\boldsymbol{Q}_2 := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

If we again use the assumed invariance of $\boldsymbol{J}_{ij}$ under such a transformation, we get

$$\begin{pmatrix} j_{11} & j_{12} \\ -j_{12} & j_{11} \end{pmatrix} = \boldsymbol{J}_{ij} = \boldsymbol{Q}_2^T \boldsymbol{J}_{ij} \boldsymbol{Q}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j_{11} & j_{12} \\ -j_{12} & j_{11} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j_{12} & j_{11} \\ j_{11} & -j_{12} \end{pmatrix} = \begin{pmatrix} j_{11} & -j_{12} \\ j_{12} & j_{11} \end{pmatrix}.$$

So $j_{12} = -j_{12}$, what is only possible if $j_{12} = 0$ and $\boldsymbol{J}_{ij} = j_{11}\mathbb{1}$ is a multiple of the unitary matrix. Therefore for arbitrary $\vec{\boldsymbol{S}}_i, \vec{\boldsymbol{S}}_j \in \mathbb{R}^2$ holds

$$\vec{\boldsymbol{S}}_i^T \boldsymbol{J}_{ij} \vec{\boldsymbol{S}}_j = j_{11} \vec{\boldsymbol{S}}_i^T \mathbb{1} \vec{\boldsymbol{S}}_j = j_{11} \vec{\boldsymbol{S}}_i^T \vec{\boldsymbol{S}}_j = j_{11} \vec{\boldsymbol{S}}_i \cdot \vec{\boldsymbol{S}}_j$$

what concludes the proof with $J_{ij} := j_{11}$. □

Appendix B

VAMPIRE-Sourcecode

The following two C++-codes contain the time integration methods in VAMPIRE, which are discussed in Chapter 4.

B.1 Heun's method

```
//------------------------------------------------------------------------------
//
//  Vampire - A code for atomistic simulation of magnetic materials
//
//  Copyright (C) 2009-2012 R.F.L.Evans
//
//  Email:richard.evans@york.ac.uk
//
//  This program is free software; you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation; either version 2 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful, but
//  WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
//  General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program; if not, write to the Free Software Foundation,
//  Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
//
// ----------------------------------------------------------------------
//
///
/// @file
/// @brief Contains the LLG (Heun) integrator
///
/// @details The LLG integrator...
///
```

```
31 /// @section notes Implementation Notes
32 /// This is a list of other notes, not related to functionality but
       rather to implementation.
33 /// Also include references, formulae and other notes here.
34 ///
35 /// @section License
36 /// Use of this code, either in source or compiled form, is subject to
       license from the authors.
37 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2010.
       All Rights Reserved.
38 ///
39 /// @section info File Information
40 /// @author  Richard Evans, richard.evans@york.ac.uk
41 /// @version 1.0
42 /// @date    07/02/2011
43 /// @internal
44 /// Created:    05/02/2011
45 /// Revision:   ---
46 ///=====================================================================
47 ///
48
49 // Standard Libraries
50 #include <cmath>
51 #include <cstdlib>
52 #include <iostream>
53
54 // Vampire Header files
55 #include "atoms.hpp"
56 #include "errors.hpp"
57 #include "LLG.hpp"
58 #include "material.hpp"
59
60 //Function prototypes
61 int calculate_spin_fields(const int,const int);
62 int calculate_external_fields(const int,const int);
63
64 namespace LLG_arrays{
65
66   // Local arrays for LLG integration
67   std::vector <double> x_euler_array;
68   std::vector <double> y_euler_array;
69   std::vector <double> z_euler_array;
70
71   std::vector <double> x_heun_array;
72   std::vector <double> y_heun_array;
73   std::vector <double> z_heun_array;
74
75   std::vector <double> x_spin_storage_array;
76   std::vector <double> y_spin_storage_array;
77   std::vector <double> z_spin_storage_array;
78
79   std::vector <double> x_initial_spin_array;
80   std::vector <double> y_initial_spin_array;
81   std::vector <double> z_initial_spin_array;
82
```

```
83    bool LLG_set=false; ///< Flag to define state of LLG arrays (
        initialised/uninitialised)
84
85  }
86
87  namespace sim{
88
89  /// @brief LLG Initialisation function
90  ///
91  /// @details Resizes arrays used for Heun integration
92  ///
93  /// @section License
94  /// Use of this code, either in source or compiled form, is subject to
        license from the authors.
95  /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2011.
        All Rights Reserved.
96  ///
97  /// @section Information
98  /// @author  Richard Evans, richard.evans@york.ac.uk
99  /// @version 1.0
100 /// @date    07/02/2011
101 ///
102 /// @return EXIT_SUCCESS
103 ///
104 /// @internal
105 /// Created:    07/02/2011
106 /// Revision:   ---
107 ///=====================================================================
108 ///
109 int LLGinit(){
110
111   // check calling of routine if error checking is activated
112   if(err::check==true){std::cout << "sim::LLG_init has been called" <<
        std::endl;}
113
114   using namespace LLG_arrays;
115
116   x_spin_storage_array.resize(atoms::num_atoms,0.0);
117   y_spin_storage_array.resize(atoms::num_atoms,0.0);
118   z_spin_storage_array.resize(atoms::num_atoms,0.0);
119
120   x_initial_spin_array.resize(atoms::num_atoms,0.0);
121   y_initial_spin_array.resize(atoms::num_atoms,0.0);
122   z_initial_spin_array.resize(atoms::num_atoms,0.0);
123
124   x_euler_array.resize(atoms::num_atoms,0.0);
125   y_euler_array.resize(atoms::num_atoms,0.0);
126   z_euler_array.resize(atoms::num_atoms,0.0);
127
128   x_heun_array.resize(atoms::num_atoms,0.0);
129   y_heun_array.resize(atoms::num_atoms,0.0);
130   z_heun_array.resize(atoms::num_atoms,0.0);
131
132   LLG_set=true;
133
134     return EXIT_SUCCESS;
```

```
135 }
136
137 /// @brief LLG Heun Integrator Corrector
138 ///
139 /// @callgraph
140 /// @callergraph
141 ///
142 /// @details Integrates the system using the LLG and Heun solver
143 ///
144 /// @section License
145 /// Use of this code, either in source or compiled form, is subject to
    license from the authors.
146 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2011.
    All Rights Reserved.
147 ///
148 /// @section Information
149 /// @author  Richard Evans, richard.evans@york.ac.uk
150 /// @version 1.0
151 /// @date    07/02/2011
152 ///
153 /// @return EXIT_SUCCESS
154 ///
155 /// @internal
156 /// Created:    05/02/2011
157 /// Revision:   ---
158 //=====================================================================
159 ///
160 int LLG_Heun(){
161
162   // check calling of routine if error checking is activated
163   if(err::check==true){std::cout << "sim::LLG_Heun has been called" <<
      std::endl;}
164
165   using namespace LLG_arrays;
166
167   // Check for initialisation of LLG integration arrays
168   if(LLG_set==false) sim::LLGinit();
169
170   // Local variables for system integration
171   const int num_atoms=atoms::num_atoms;
172   double xyz[3];     // Local Delta Spin Components
173   double S_new[3];   // New Local Spin Moment
174   double mod_S;    // magnitude of spin moment
175
176   // Store initial spin positions
177   for(int atom=0;atom<num_atoms;atom++){
178     x_initial_spin_array[atom] = atoms::x_spin_array[atom];
179     y_initial_spin_array[atom] = atoms::y_spin_array[atom];
180     z_initial_spin_array[atom] = atoms::z_spin_array[atom];
181   }
182
183   // Calculate fields
184   calculate_spin_fields(0,num_atoms);
185   calculate_external_fields(0,num_atoms);
186
187   // Calculate Euler Step
```

```
188   for(int atom=0;atom<num_atoms;atom++){
189
190     const int imaterial=atoms::type_array[atom];
191     const double one_oneplusalpha_sq = mp::material[imaterial].
      one_oneplusalpha_sq; // material specific alpha and gamma
192     const double alpha_oneplusalpha_sq = mp::material[imaterial].
      alpha_oneplusalpha_sq;
193
194     // Store local spin in Sand local field in H
195     const double S[3] = {atoms::x_spin_array[atom],atoms::y_spin_array[
      atom],atoms::z_spin_array[atom]};
196     const double H[3] = {atoms::x_total_spin_field_array[atom]+atoms::
      x_total_external_field_array[atom],
197                          atoms::y_total_spin_field_array[atom]+atoms::
      y_total_external_field_array[atom],
198                          atoms::z_total_spin_field_array[atom]+atoms::
      z_total_external_field_array[atom]};
199
200     // Calculate Delta S
201     xyz[0]=(one_oneplusalpha_sq)*(S[1]*H[2]-S[2]*H[1]) + (
      alpha_oneplusalpha_sq)*(S[1]*(S[0]*H[1]-S[1]*H[0])-S[2]*(S[2]*H[0]-
      S[0]*H[2]));
202     xyz[1]=(one_oneplusalpha_sq)*(S[2]*H[0]-S[0]*H[2]) + (
      alpha_oneplusalpha_sq)*(S[2]*(S[1]*H[2]-S[2]*H[1])-S[0]*(S[0]*H[1]-
      S[1]*H[0]));
203     xyz[2]=(one_oneplusalpha_sq)*(S[0]*H[1]-S[1]*H[0]) + (
      alpha_oneplusalpha_sq)*(S[0]*(S[2]*H[0]-S[0]*H[2])-S[1]*(S[1]*H[2]-
      S[2]*H[1]));
204
205     // Store dS in euler array
206     x_euler_array[atom]=xyz[0];
207     y_euler_array[atom]=xyz[1];
208     z_euler_array[atom]=xyz[2];
209
210     // Calculate Euler Step
211     S_new[0]=S[0]+xyz[0]*mp::dt;
212     S_new[1]=S[1]+xyz[1]*mp::dt;
213     S_new[2]=S[2]+xyz[2]*mp::dt;
214
215     // Normalise Spin Length
216     mod_S = 1.0/sqrt(S_new[0]*S_new[0] + S_new[1]*S_new[1] + S_new[2]*
      S_new[2]);
217
218     S_new[0]=S_new[0]*mod_S;
219     S_new[1]=S_new[1]*mod_S;
220     S_new[2]=S_new[2]*mod_S;
221
222     //Writing of Spin Values to Storage Array
223     x_spin_storage_array[atom]=S_new[0];
224     y_spin_storage_array[atom]=S_new[1];
225     z_spin_storage_array[atom]=S_new[2];
226   }
227
228   // Copy new spins to spin array
229   for(int atom=0;atom<num_atoms;atom++){
230     atoms::x_spin_array[atom]=x_spin_storage_array[atom];
```

```
231     atoms::y_spin_array[atom]=y_spin_storage_array[atom];
232     atoms::z_spin_array[atom]=z_spin_storage_array[atom];
233   }
234
235   // Recalculate spin dependent fields
236   calculate_spin_fields(0,num_atoms);
237
238   // Calculate Heun Gradients
239   for(int atom=0;atom<num_atoms;atom++){
240
241     const int imaterial=atoms::type_array[atom];;
242     const double one_oneplusalpha_sq = mp::material[imaterial].
      one_oneplusalpha_sq;
243     const double alpha_oneplusalpha_sq = mp::material[imaterial].
      alpha_oneplusalpha_sq;
244
245     // Store local spin in Sand local field in H
246     const double S[3] = {atoms::x_spin_array[atom],atoms::y_spin_array[
      atom],atoms::z_spin_array[atom]};
247     const double H[3] = {atoms::x_total_spin_field_array[atom]+atoms::
      x_total_external_field_array[atom],
248                          atoms::y_total_spin_field_array[atom]+atoms::
      y_total_external_field_array[atom],
249                          atoms::z_total_spin_field_array[atom]+atoms::
      z_total_external_field_array[atom]};
250
251     // Calculate Delta S
252     xyz[0]=(one_oneplusalpha_sq)*(S[1]*H[2]-S[2]*H[1]) + (
      alpha_oneplusalpha_sq)*(S[1]*(S[0]*H[1]-S[1]*H[0])-S[2]*(S[2]*H[0]-
      S[0]*H[2]));
253     xyz[1]=(one_oneplusalpha_sq)*(S[2]*H[0]-S[0]*H[2]) + (
      alpha_oneplusalpha_sq)*(S[2]*(S[1]*H[2]-S[2]*H[1])-S[0]*(S[0]*H[1]-
      S[1]*H[0]));
254     xyz[2]=(one_oneplusalpha_sq)*(S[0]*H[1]-S[1]*H[0]) + (
      alpha_oneplusalpha_sq)*(S[0]*(S[2]*H[0]-S[0]*H[2])-S[1]*(S[1]*H[2]-
      S[2]*H[1]));
255
256     // Store dS in heun array
257     x_heun_array[atom]=xyz[0];
258     y_heun_array[atom]=xyz[1];
259     z_heun_array[atom]=xyz[2];
260   }
261
262   // Calculate Heun Step
263   for(int atom=0;atom<num_atoms;atom++){
264     S_new[0]=x_initial_spin_array[atom]+mp::half_dt*(x_euler_array[atom
      ]+x_heun_array[atom]);
265     S_new[1]=y_initial_spin_array[atom]+mp::half_dt*(y_euler_array[atom
      ]+y_heun_array[atom]);
266     S_new[2]=z_initial_spin_array[atom]+mp::half_dt*(z_euler_array[atom
      ]+z_heun_array[atom]);
267
268     // Normalise Spin Length
269     mod_S = 1.0/sqrt(S_new[0]*S_new[0] + S_new[1]*S_new[1] + S_new[2]*
      S_new[2]);
270
```

```
271       S_new[0]=S_new[0]*mod_S;
272       S_new[1]=S_new[1]*mod_S;
273       S_new[2]=S_new[2]*mod_S;
274
275       // Copy new spins to spin array
276       atoms::x_spin_array[atom]=S_new[0];
277       atoms::y_spin_array[atom]=S_new[1];
278       atoms::z_spin_array[atom]=S_new[2];
279     }
280
281     return EXIT_SUCCESS;
282 }
283
284 /// @brief LLG Heun Integrator (CUDA)
285 ///
286 /// @callgraph
287 /// @callergraph
288 ///
289 /// @details Integrates the system using the LLG and Heun solver
290 ///
291 /// @section License
292 /// Use of this code, either in source or compiled form, is subject to
      license from the authors.
293 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2011.
      All Rights Reserved.
294 ///
295 /// @section Information
296 /// @author  Richard Evans, richard.evans@york.ac.uk
297 /// @version 1.0
298 /// @date    05/02/2011
299 ///
300 /// @return EXIT_SUCCESS
301 ///
302 /// @internal
303 /// Created:     05/02/2011
304 /// Revision:    ---
305 ///=====================================================================
306 ///
307 int LLG_Heun_cuda(){
308
309
310   return EXIT_SUCCESS;
311 }
312
313 }
```

B.2 Midpoint method

```
1 //------------------------------------------------------------------------
2 //
3 //   Vampire - A code for atomistic simulation of magnetic materials
4 //
5 //   Copyright (C) 2009-2012 R.F.L.Evans
6 //
7 //   Email:richard.evans@york.ac.uk
```

```
8  //
9  //  This program is free software; you can redistribute it and/or
       modify
10 //  it under the terms of the GNU General Public License as published
       by
11 //  the Free Software Foundation; either version 2 of the License, or
12 //  (at your option) any later version.
13 //
14 //  This program is distributed in the hope that it will be useful, but
15 //  WITHOUT ANY WARRANTY; without even the implied warranty of
16 //  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
17 //  General Public License for more details.
18 //
19 //  You should have received a copy of the GNU General Public License
20 //  along with this program; if not, write to the Free Software
       Foundation,
21 //  Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
22 //
23 // ------------------------------------------------------------------
24 //
25 ///
26 /// @file
27 /// @brief Contains the LLG (Midpoint) integrator
28 ///
29 /// @details The LLG integrator...
30 ///
31 /// @section notes Implementation Notes
32 /// This is a list of other notes, not related to functionality but
       rather to implementation.
33 /// Also include references, formulae and other notes here.
34 ///
35 /// @section License
36 /// Use of this code, either in source or compiled form, is subject to
       license from the authors.
37 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2010.
       All Rights Reserved.
38 ///
39 /// @section info File Information
40 /// @author  Richard Evans, richard.evans@york.ac.uk
41 /// @version 1.0
42 /// @date    07/02/2011
43 /// @internal
44 /// Created:    05/02/2011
45 /// Revision:   ---
46 ///=====================================================================
47 ///
48
49 // Standard Libraries
50 #include <cmath>
51 #include <cstdlib>
52 #include <iostream>
53
54 // Vampire Header files
55 #include "atoms.hpp"
56 #include "errors.hpp"
57 #include "LLG.hpp"
```

```
58 #include "material.hpp"
59 #include "sim.hpp"
60
61 //Function prototypes
62 int calculate_spin_fields(const int,const int);
63 int calculate_external_fields(const int,const int);
64
65 namespace sim{
66
67 /// @brief LLG Midpoint Integrator
68 ///
69 /// @callgraph
70 /// @callergraph
71 ///
72 /// @details Integrates the system using the LLG and Heun solver
73 ///
74 /// @section License
75 /// Use of this code, either in source or compiled form, is subject to
    license from the authors.
76 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2011.
    All Rights Reserved.
77 ///
78 /// @section Information
79 /// @author  Richard Evans, richard.evans@york.ac.uk
80 /// @version 1.0
81 /// @date    14/03/2011
82 ///
83 /// @return EXIT_SUCCESS
84 ///
85 /// @internal
86 /// Created:    05/02/2011
87 /// Revision:   ---
88 ///=====================================================================
89 ///
90 int LLG_Midpoint(){
91
92   // check calling of routine if error checking is activated
93   if(err::check==true){std::cout << "sim::LLG_Midpoint has been called"
       << std::endl;}
94
95   using namespace LLG_arrays;
96
97   // Check for initialisation of LLG integration arrays
98   if(LLG_set==false) sim::LLGinit();
99
100  // Local variables for system integration
101  const int num_atoms=atoms::num_atoms;
102
103  // Store initial spin positions
104  for(int atom=0;atom<num_atoms;atom++){
105    x_initial_spin_array[atom] = atoms::x_spin_array[atom];
106    y_initial_spin_array[atom] = atoms::y_spin_array[atom];
107    z_initial_spin_array[atom] = atoms::z_spin_array[atom];
108  }
109
110  // Calculate fields
```

```
111   calculate_spin_fields(0,num_atoms);
112   calculate_external_fields(0,num_atoms);
113
114   // Calculate Predictor Step
115   for(int atom=0;atom<num_atoms;atom++){
116
117     const int imaterial=atoms::type_array[atom];
118     const double alpha = mp::material[imaterial].alpha;
119     const double beta  = -1.0*mp::dt*mp::material[imaterial].
      one_oneplusalpha_sq*0.5;
120     const double beta2 = beta*beta;
121
122     // Store local spin in S and local field in H
123     const double S[3] = {atoms::x_spin_array[atom],atoms::y_spin_array[
      atom],atoms::z_spin_array[atom]};
124     const double H[3] = {atoms::x_total_spin_field_array[atom]+atoms::
      x_total_external_field_array[atom],
125                          atoms::y_total_spin_field_array[atom]+atoms::
      y_total_external_field_array[atom],
126                          atoms::z_total_spin_field_array[atom]+atoms::
      z_total_external_field_array[atom]};
127
128     // Calculate F = [H + alpha* (S x H)]
129     const double F[3] = {H[0] + alpha*(S[1]*H[2]-S[2]*H[1]),
130                          H[1] + alpha*(S[2]*H[0]-S[0]*H[2]),
131                          H[2] + alpha*(S[0]*H[1]-S[1]*H[0])};
132
133     const double FdotF = F[0]*F[0] + F[1]*F[1] + F[2]*F[2];
134     const double beta2FdotS = beta2*(F[0]*S[0] + F[1]*S[1] + F[2]*S[2])
      ;
135     const double one_o_one_plus_beta2FdotF = 1.0/(1.0 + beta2*FdotF);
136     const double one_minus_beta2FdotF = 1.0 - beta2*FdotF;
137
138     // Calculate intermediate spin position (S + S')/2
139     x_spin_storage_array[atom] = (S[0] + one_o_one_plus_beta2FdotF*(S
      [0]*one_minus_beta2FdotF + 2.0*(beta*(F[1]*S[2]-F[2]*S[1]) + F[0]*
      beta2FdotS)))*0.5;
140     y_spin_storage_array[atom] = (S[1] + one_o_one_plus_beta2FdotF*(S
      [1]*one_minus_beta2FdotF + 2.0*(beta*(F[2]*S[0]-F[0]*S[2]) + F[1]*
      beta2FdotS)))*0.5;
141     z_spin_storage_array[atom] = (S[2] + one_o_one_plus_beta2FdotF*(S
      [2]*one_minus_beta2FdotF + 2.0*(beta*(F[0]*S[1]-F[1]*S[0]) + F[2]*
      beta2FdotS)))*0.5;
142
143   }
144
145   // Store Midpoint to spin array
146   for(int atom=0;atom<num_atoms;atom++){
147     atoms::x_spin_array[atom]=x_spin_storage_array[atom];
148     atoms::y_spin_array[atom]=y_spin_storage_array[atom];
149     atoms::z_spin_array[atom]=z_spin_storage_array[atom];
150   }
151
152   // Recalculate spin dependent fields
153   calculate_spin_fields(0,num_atoms);
154
```

```
155   // Calculate Corrector Step
156   for(int atom=0;atom<num_atoms;atom++){
157
158     const int imaterial=atoms::type_array[atom];
159     const double alpha = mp::material[imaterial].alpha;
160     const double beta  = -1.0*mp::dt*mp::material[imaterial].
      one_oneplusalpha_sq*0.5;
161     const double beta2 = beta*beta;
162
163     // Store local spin in S and local field in H
164     const double M[3] = {atoms::x_spin_array[atom],atoms::y_spin_array[
      atom],atoms::z_spin_array[atom]};
165     const double S[3] = {x_initial_spin_array[atom],
      y_initial_spin_array[atom],z_initial_spin_array[atom]};
166     const double H[3] = {atoms::x_total_spin_field_array[atom]+atoms::
      x_total_external_field_array[atom],
167                          atoms::y_total_spin_field_array[atom]+atoms::
      y_total_external_field_array[atom],
168                          atoms::z_total_spin_field_array[atom]+atoms::
      z_total_external_field_array[atom]};
169
170     // Calculate F = [H + alpha* (M x H)]
171     const double F[3] = {H[0] + alpha*(M[1]*H[2]-M[2]*H[1]),
172                          H[1] + alpha*(M[2]*H[0]-M[0]*H[2]),
173                          H[2] + alpha*(M[0]*H[1]-M[1]*H[0])};
174
175     const double FdotF = F[0]*F[0] + F[1]*F[1] + F[2]*F[2];
176     const double beta2FdotS = beta2*(F[0]*S[0] + F[1]*S[1] + F[2]*S[2])
      ;
177     const double one_o_one_plus_beta2FdotF = 1.0/(1.0 + beta2*FdotF);
178     const double one_minus_beta2FdotF = 1.0 - beta2*FdotF;
179
180     // Calculate final spin position
181     atoms::x_spin_array[atom] = one_o_one_plus_beta2FdotF*(S[0]*
      one_minus_beta2FdotF + 2.0*(beta*(F[1]*S[2]-F[2]*S[1]) + F[0]*
      beta2FdotS));
182     atoms::y_spin_array[atom] = one_o_one_plus_beta2FdotF*(S[1]*
      one_minus_beta2FdotF + 2.0*(beta*(F[2]*S[0]-F[0]*S[2]) + F[1]*
      beta2FdotS));
183     atoms::z_spin_array[atom] = one_o_one_plus_beta2FdotF*(S[2]*
      one_minus_beta2FdotF + 2.0*(beta*(F[0]*S[1]-F[1]*S[0]) + F[2]*
      beta2FdotS));
184   }
185
186   return EXIT_SUCCESS;
187 }
188
189 /// @brief LLG Heun Integrator (CUDA)
190 ///
191 /// @callgraph
192 /// @callergraph
193 ///
194 /// @details Integrates the system using the LLG and Heun solver
195 ///
196 /// @section License
197 /// Use of this code, either in source or compiled form, is subject to
```

```
    license from the authors.
198 /// Copyright \htmlonly &copy \endhtmlonly Richard Evans, 2009-2011.
    All Rights Reserved.
199 ///
200 /// @section Information
201 /// @author  Richard Evans, richard.evans@york.ac.uk
202 /// @version 1.0
203 /// @date    05/02/2011
204 ///
205 /// @return EXIT_SUCCESS
206 ///
207 /// @internal
208 /// Created:     05/02/2011
209 /// Revision:    ---
210 ///=====================================================================
211 ///
212 int LLG_Midpoint_cuda(){
213
214
215   return EXIT_SUCCESS;
216 }
217
218 }
```

Appendix C

Implementation for the read-back curve

In this and the following Appendix we present the source code of the main programs, which are used to receive the results in Chapter 6. It can be found on the GTO-server in */home/florians/diplomarbeit*. The goal is to compute the read-back signal curves as in Fig. 6.8 with the following external input:

- granular media in form of vtu-files (see Sec. C.2, line 12 and Sec. C.6, line 60) and ini-files (see Sec. C.3, lines 15-17)
- switching probability phase diagrams as in Chapter 5 for different grains sizes to account the grain size distribution in the granular media (see Sec. C.4, line 53)
- a reader module in form of a sensitivity function (see Fig. 6.7 for the function and Sec. C.6, line 101)

The file "run14.py" in Sec. C.1 starts the simulation with the following required parameters:

- a scalar valued shell-input ranks the seed of the granular medium and might be useful in a parallel computing process on different media, because the results of all seeds will be saved in a common directory (line 16)
- x- and y-dimensions of the rectangular cross section of the granular media in nm (see lines 24 & 25)
- mean grain size and its standard deviation in nm to choose the right medium (lines 28 & 30)
- vector that determines the requested bit pattern in the form $[v_1, v_2, v_3, \ldots]$ where $v_i \in \mathbb{Z}^*$ means v_i consecutive $(+1)$-bits for $v_i > 0$ and $|v_i|$ consecutive (-1)-bits for $v_i < 0$ (line 36)
- total amount of different granular media, which is only important for the designation of the written output files (line 39)

- time period of the applied magnetic field (with field rise and decay time) in ns per bit (line 42)
- designation of the files with the used phase plots and numerical values of their corresponding grains sizes (lines 46 – 62)
- velocity of the writing head in m/s (line 75)
- initial and maximum temperature of the applied Gaussian heat pulse in K (lines 81 & 82)
- designation of the written output files (lines 90 – 93)

Note that "laser_simulation_on_grains6.py" in Sec. C.4 also uses the offtrack-variance of the Gaussian heat pulse as an internal parameter, which might be varied (see line 29).

C.1 run14.py

```
1  #import makegrains3
2  import sys
3  import grains2circles2
4  import prepare_ini2
5  import laser_simulation_on_grains6
6  import probabilities2ini
7  import randy_reader
8  #import auswertung3
9  from os import *
10 import numpy as np
11
12 #Version 14: Randys Reader (sonst wie Version 11)
13 #im Unterschied zu 12 siehe 93 (Version 6 statt 7 --> keine perfekten
       bits!)
14
15
16 shellinput=int(sys.argv[1]) #Shellinput des Skripts!!!
17
18 ##################################################
19 ##################################################
20 #Eingabeparameter
21
22
23 #Dimensionen des Grids
24 dimX = 500
25 dimY = 60
26
27 #mittlere Korngroesse
28 loc=[4.0]
29 #Sigma
30 sigma=0.01
31
32 #Richtung der Bits
```

```
#bits=[-1.0, 1.0, 1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, 1.0, 1.0,
    -1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, 1.0, 1.0,
    1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0,
    -1.0, -1.0]
#bits=[-1,2,-5,3,-2,1,-3,1,-1,1,-1,4,-1,2,-1,1,-2,2,-3]
#bits=[-1.0 for i in range(18)] + [1.0 for i in range(19)]
bits=[-18,19]

#Anzahl der Kornverteilungen
N=50

#Dauer des Magnetfelds[ns]
magt=1.2


#Hier werden alle Phasenplots eingelesen - unabhaengig davon, ob sie
    gebraucht werden oder nicht
file8=[['HAMR_LLB_FePt_3p5x3p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_4x4x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_4p5x4p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_4p75x4p75x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_5x5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_5p25x5p25x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_5p5x5p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_6x6x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_6p5x6p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_7x7x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_7p5x7p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_8x8x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_8p5x8p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_9x9x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_9p5x9p5x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt', '
    HAMR_LLB_FePt_10x10x10_phasePlots_peakTemp_vs\
_phaseShift_h0p8_FWHM20nm_vel10mps.txt'], [3.5, 4.0, 4.5, 4.75, 5.0,
    5.25, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0]]
assert len(file8[0])==len(file8[1])
#Curie-Temperaturen
TC=[527.540385235,  528.190384748, 534.172286487, 535.04891787,
    536.565286497, 538.555170869, 539.122235808, 540.527385274,
    542.161636173, 543.299429463, 544.814870786, 545.648990743,
    545.67909223, 546.098435376, 546.498110758, 546.658816218]
```

```
66 assert len(file8[0])==len(TC)
67
68
69 #Datenfiles
70 #filename=sum([[file6,file7,file5] for i in range(600,730,20)],[])
71 filename=[file8 for i in range(len(loc))]
72
73 #Schreibgeschwindigkeiten
74 #v=sum([[7.5,10,20] for i in range(600,730,20)],[]) #Geschwindigkeit
       des Laserpulses in m/s
75 v=[10 for i in range(len(loc))]
76 assert len(filename)==len(v)
77
78 #Tmin/Tmax des Hitzepulses
79 #Tmin=sum([[270.,270.,270.] for i in range(600,730,20)],[]) #Temperatur
       -Minimum des gaussfoermigen Peaks
80 #Tmax=sum([[i,i,i] for i in range(600,730,20)],[])  #Temperatur-Maximum
        am Peak
81 Tmin=[270. for i in range(len(loc))]
82 Tmax=[715. for i in range(len(loc))]
83 assert len(filename)==len(Tmin)
84 assert len(filename)==len(Tmax)
85
86
87 #Name der Ordner, in dem die Ergebnisse gespeichert werden
88 outputname=[0 for j in range(len(loc))]
89 for j in range(len(loc)):
90   if v[j]==7.5:
91     outputname[j]='simulation_fertig_vel7p5mps_Tmax%
      dK_verschiedenegrains_N=%d_loc=%1.1f_sigma=%1.6
      f_mehrbits_groessen_randy_calib' % (Tmax[j],N,loc[j],sigma)
92   else:
93     outputname[j]='simulation_fertig_vel%dmps_Tmax%
      dK_verschiedenegrains_N=%d_loc=%1.1f_sigma=%1.6
      f_mehrbits_groessen_randy_calib' % (v[j],Tmax[j],N,loc[j],sigma)
94 assert len(filename)==len(outputname)
95
96
97 ################################################
98 ################################################
99
100 def run12(seed,j,dimX,dimY,loc,bits,sigma):
101
102   if not path.exists("%s/seed%d" % (outputname[j],seed)):
103     system("mkdir %s/seed%d" % (outputname[j],seed))
104
105 # #makegrains3.makegrains3(seed,dimX,dimY,loc)  #auskommentieren, falls
       Mesh-Daten schon vorhanden
106   grains2circles2.grains2circles2(seed,dimX,dimY,loc,sigma,outputname[j
      ])
107   print 'grains2circles2 done'
108   prepare_ini2.prepare_ini2(seed,dimX,dimY,loc,sigma,outputname[j])
109   print 'prepare_ini2 done'
110   for i in range(len(bits)):
111     laser_simulation_on_grains6.laser_simulation_on_grains6(seed,
      filename[j],v[j],Tmin[j],Tmax[j],-dimX/2+50+sum([abs(bits[k]) for k
```

```
    in range(i)])*magt*v[j],abs(bits[i]),magt,TC,outputname[j])
112     probabilities2ini.probabilities2ini(seed,dimX,dimY,loc,'%1.6f' % np
    .sign(bits[i]),len(bits),i,outputname[j])
113   print 'laser_simulation_on_grains probabilities2ini done'
114
115     randy_reader.randy_reader(seed,dimX,dimY,loc,sigma,outputname[j])
116   #Wichtigster Output der letzten beiden Simulationsschritte werden in
      Ordner kopiert. Dazu gehoert "headfiled.log", welches u.a. die
      zeitliche Magnetisierung des Readers beinhaltet und die Bilddatei
      des magnetisierten Mediums.
117     system("cp %s/seed%d/curve.txt %s/curve%d.txt" % (outputname[j],
    seed,outputname[j],seed))
118   #system("cp %s/seed%d/model.0001.inp %s/model_meshseed_spinseed%d.inp
      " % (outputname[j],seed,outputname[j],seed))
119
120 for j in range(len(loc)):
121   if not path.exists("%s" % outputname[j]):
122     system("mkdir %s" % outputname[j])
123
124   for i in range(shellinput,shellinput+1):  #Shellinput des Skripts!!!
125     run12(i,j,dimX,dimY,loc[j],bits,sigma)
126
127 # auswertung3.auswertung3(N,outputname[j],j)
```

C.2 grains2circles2.py

```
1 from math import *
2 #from Tkinter import *
3 from os import *
4
5 #Version 2: modelname geaendert, zusaetzlicher Parameter "sigma"
6
7 def grains2circles2(seed,dimX,dimY,loc,sigma,outputname):
8     """Programm wandelt die Koerner in flaechengleiche Kreise um.
    Mittelpunkt ist der Schwerpunkt. Benoetigt werden nur die Eckpunkte
     der Polygone und die Connectivity. Input ist Filename (und
    Verzeichnis) des vtu-Files und Laenge (x) und Breite (y) der Platte
    ."""
9
10     modelname = 'isolatedGrains%dx%d_seed%d_anisotropy_random_D%1.1
    f_sigma%1.6f' % (dimX,dimY,seed,loc,sigma)
11
12     datei=file('/home/florians/ULTRA/vtk/%s.vtu' % modelname,'r')
13     inhalt=datei.readlines()
14
15     #Eckpunkte der Polygone auslesen und formatieren:
16     rawpoints=inhalt[5]
17     rawpoints=rawpoints.replace('<DataArray NumberOfComponents="3"
    format="ascii" type="Float64">','')
18     rawpoints=rawpoints.replace('</DataArray>','')
19     rawpoints=rawpoints.split()
20     points=[[0,0] for i in range(len(rawpoints)/3)]
21     for i in range(len(points)):
22         points[i][0]=float(rawpoints[3*i])
23         points[i][1]=float(rawpoints[3*i+1])
```

```
24
25      #Indizes der Eckpunkte auslesen, die Polygon ergeben
26      connect=inhalt[8]
27      connect=connect.replace('<DataArray Name="connectivity" format="
        ascii" type="Int32">','')
28      connect=connect.replace('</DataArray>','')
29      connect=connect.split('  ')
30      for i in range(len(connect)):
31          connect[i]=connect[i].split()
32      for i in range(len(connect)):
33          for j in range(len(connect[i])):
34              connect[i][j]=int(connect[i][j])
35
36      datei.close()
37
38      #Mittelpunkte der Kreise (Schwerpunkte der Polygone) berechnen
39      midpoints=[[0,0] for i in range(len(connect))]
40      for i in connect:
41          midpoints[connect.index(i)][0]=sum([points[j][0] for j in i])/
        len(i)
42          midpoints[connect.index(i)][1]=sum([points[j][1] for j in i])/
        len(i)
43
44      #Radien der Kreise berechnen mit Gaussscher Trapezformel
45      radii=[0 for i in range(len(connect))]
46      for i in connect:
47          radii[connect.index(i)]=sqrt(sum([(points[i[j%len(i)]][1]+
        points[i[(j+1)%len(i)]][1])*(points[i[j%len(i)]][0]-points[i[(j+1)%
        len(i)]][0]) for j in range(len(i))])/2/pi)
48
49
50      #Ausgabe von Koordinaten der Mittelpunkte und Radien
51
52  #    if not path.exists('circles/%s' % modelname):
53  #    mkdir('circles/%s' % modelname)
54  #
55  #    thefile=file('circles/%s/radii.txt' % modelname,'w')
56  #    for item in radii:
57  #        print>>thefile, item
58  #
59  #    thefile2=file('circles/%s/pointsX.txt' % modelname,'w')
60  #    for item in midpoints:
61  #        print>>thefile2, item[0]
62
63  #    thefile3=file('circles/%s/pointsY.txt' % modelname,'w')
64  #    for item in midpoints:
65  #        print>>thefile3, item[1]
66
67      if not path.exists("%s/seed%d/circles" % (outputname,seed)):
68        system("mkdir %s/seed%d/circles" % (outputname,seed))
69
70      thefile=file('%s/seed%d/circles/radii.txt' % (outputname,seed),'w')
71      for item in radii:
72          print>>thefile, item
73      thefile.close()
74
```

```
75      thefile2=file('%s/seed%d/circles/pointsX.txt' % (outputname,seed),'
        w')
76      for item in midpoints:
77          print>>thefile2, item[0]
78      thefile2.close()
79
80      thefile3=file('%s/seed%d/circles/pointsY.txt' % (outputname,seed),'
        w')
81      for item in midpoints:
82          print>>thefile3, item[1]
83      thefile3.close()
84
85
86
87
88  #    #Graphische Darstellung
89
90  # skalierungsfaktor=600/dimX
91  #
92  #    canvas_width = skalierungsfaktor*dimX
93  #    canvas_height =skalierungsfaktor*dimY
94
95  #    master = Tk()
96
97  #    w = Canvas(
98  #            width=canvas_width,
99  #            height=canvas_height)
100 #    w.pack()
101
102 #    for i in range(len(radii)):
103 #        w.create_oval(skalierungsfaktor*(x/2.+midpoints[i][0]-radii[i
        ]),skalierungsfaktor*y-skalierungsfaktor*(y/2.+midpoints[i][1]-
        radii[i]),skalierungsfaktor*(x/2.+midpoints[i][0]+radii[i]),
        skalierungsfaktor*y-skalierungsfaktor*(y/2.+midpoints[i][1]+radii[i
        ]))
104
105 #    mainloop()
```

C.3 prepare_ini2.py

```
1  from os import *
2  #from Tkinter import *
3  import random
4  import numpy
5
6  # File kopiert passende inp/ini/krn-Files aus Unterordner und
       initialisiert die letzten Spalte im ini-File mit zufaelligen -1 und
        1
7
8  #Version 2 -> geaenderte Name, zusaetzlicher Parameter "sigma"
9
10 def prepare_ini2(seed,dimX,dimY,loc,sigma,outputname):
11   """Programm erzeugt aus Liste mit Switching-Wahrscheinlichkeiten (im
       Verzeichnis /circles) und den Modell-Files (im Verzeichnis /inp)
       eine den Wahrscheinlichkeiten entsprechend zufaellige
```

```
    Magnetisierung der einzelnen Grains. Anschliessend wird die FEMME-
    Simulation zur Magnetfeldauslesung gestartet. Im aktuellen
    Verzeichnis muss sich dazu unter anderem das File "headdynamics.par
    " mit den entsprechenden Parametern befinden"""
12
13  #Verschieben der noetigen Modell-Files in das aktuelle Verzeichnis
14  modellname='isolatedGrains%dx%d_seed%d_anisotropy_random_D%1.1f_sigma
    %1.6f' % (int(dimX),int(dimY),seed,loc,sigma)
15  system("cp /home/florians/ULTRA/inp/%s.inp %s/seed%d/model.inp" % (
    modellname,outputname,seed))
16  system("cp /home/florians/ULTRA/inp/%s.ini %s/seed%d/model.ini" % (
    modellname,outputname,seed))
17  system("cp /home/florians/ULTRA/inp/%s.krn %s/seed%d/model.krn" % (
    modellname,outputname,seed))
18
19  datei=file('%s/seed%d/model.ini' % (outputname,seed),'r')
20  inhalt=datei.readlines()
21  datei.close()
22
23  #Seed-Initialisierung und Erzeugung der Zufallszahlen -> Ergebnisse
    +/-1 werden in letzte Spalte von ini-File eingetragen
24  random.seed(seed)
25  for i in range(len(inhalt)):
26    inhalt[i]=inhalt[i].split()
27    if random.random()<0.5:
28      inhalt[i][3]=1.0
29    else:
30      inhalt[i][3]=-1.0
31
32  thefile=file('%s/seed%d/model.ini' % (outputname,seed),'w')
33  for item in inhalt:
34    print>>thefile, "%s %s %s %f" % (item[0],item[1],item[2],item[3])
35  thefile.close()
```

C.4 laser_simulation_on_grains6.py

```
1  from os import *
2  from math import *
3  import scipy.interpolate
4  import numpy as np
5  import bisect
6
7  #Version 6 fuer lange Platte und wiederholten Schreibprozessen und
       Berueecksichtigung der Graingroessen.
8  #Reverse direction!!!
9  #"xstart" ist bezeichnet die Stelle der Platte, wo der Nullpunkt des
       Phasenplots hinkommen soll!
10 #"anz" ist die Anzahl an Bits, die in Serie geschrieben werden sollen
11 #"filename" ist hier ein 2x16-array, in welchem die filenames und die
       Durchmesser aus
       [3.5,4.0,4.5,4.75,5.0,5.25,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0]
        stehen
12
13 #Dieses Skript soll fuer gegebene 2D-Kornverteilung (Ordner "circles"
       mit Mittelpunkten und Radien) "Heat assisted magnetic recording"
```

```
    simulieren. Die Switch-Wahrscheinlichkeiten werden aus dem
    Phasendiagramm in Fig.2 in [12] eingelesen. Details zum Model
    finden sich in [12], Kapitel 2.
14
15
16 #Fitparameter gemaess [13], Gleichung (11) -> [T_C^\inf , d_0 , \Lambda
     ]
17 fit=[5.54453734e+02,3.71840245e-01,1.32505732e+00]
18 def fitfunc(fit,d):
19     return fit[0]-fit[0]*np.power(fit[1]/d,fit[2])
20
21
22
23 def laser_simulation_on_grains6(seed,filename,v,Tmin,Tmax,xstart,anz,
    magt,TC,outputname):
24
25   def gaussian1d(y,y0,Tmin,Tmax,sigmasq):
26     """Funktion liefert maximale Temperatur offtrack bei gaussfoermigem
      Hitzepuls senkrecht zur y-Achse. y0 ist die Zentrumskoordinate und
      y die Stelle, an dem die maximale Temperatur berechnet wird. (
    siehe Paper [3], Formel (1))"""
27     return (Tmax-Tmin)*exp((-(y-y0)*(y-y0))/2/sigmasq)+Tmin
28
29   sigmasq=1600./8./log(2.)     #Varianz der oertlichen Gaussverteilung
30
31
32   #Einlesen der Mittelpunkte und Radien
33   datei=file('%s/seed%d/circles/radii.txt' % (outputname,seed),'r')
34   r=datei.readlines()
35   datei.close()
36
37   datei=file('%s/seed%d/circles/pointsX.txt' % (outputname,seed),'r')
38   x=datei.readlines()
39   datei.close()
40
41   datei=file('%s/seed%d/circles/pointsY.txt' % (outputname,seed),'r')
42   y=datei.readlines()
43   datei.close()
44
45
46   #Einlesen aller Datenfiles
47   downtrackpos=[[] for k in range(len(filename[0]))]
48   Tpeak=[[] for k in range(len(filename[0]))]
49   probability=[[] for k in range(len(filename[0]))]
50   probabilityext=[[] for k in range(len(filename[0]))]
51
52   for k in range(len(filename[0])):
53     datei=file('/home/florians/ULTRA/phasenplots/%s' % filename[0][k],'
    r')
54     dat=datei.readlines()
55     datei.close()
56
57
58     dat.pop(0)  #Loeschen der Header-Zeile
59     for i in range(len(dat)):
60       dat[i]=dat[i].split()
```

```
61
62
63      #Auslesen und loeschen aller Leerzeilen -> Indizes geben an, wann
        neuer Listenabschnitt beginnt
64      indexemptylists = [0] * dat.count([])
65      for i in range(len(indexemptylists)):
66        indexemptylists[i]=dat.index([])
67        dat.remove([])
68
69
70      #Anlegen der Arrays fuer Downtrackposition (0.Spalte), Peak-
        Temperatur (1.Spalte), Wahrscheinlichkeiten (2. Spalte)
71      #Beachte: len(downtrackpos)*len(Tpeak)=len(probability)
72      m=len(indexemptylists)
73      n=indexemptylists[0]
74      downtrackpos[k]=[0.0 for i in range(m)]
75      Tpeak[k]=[0.0 for j in range(n)]
76      probability[k]=[[0.0 for j in range(n)] for i in range(m)]
77
78      for i in range(m):
79        downtrackpos[k][-1-i]=float(dat[indexemptylists[i]-1][0])*(-v)
80      deltadowntrack=downtrackpos[k][1]-downtrackpos[k][0]
81      downtrackpos[k]=downtrackpos[k]+list(np.arange(downtrackpos[k][-1]+
        deltadowntrack,downtrackpos[k][-1]+deltadowntrack+v*magt*(anz-1),
        deltadowntrack))
82      addentries=len(downtrackpos[k])-m
83
84      for i in range(n):
85        Tpeak[k][i]=float(dat[i][1])
86
87      #Wahrscheinlichkeitsvektor wird umgedreht, da *(-v)
88      for i in range(m):
89        for j in range(n):
90          probability[k][i][j]=float(dat[i*n+j][2])
91      probability[k].reverse()
92      probability[k]=np.array(probability[k])
93      probabilityext[k]=np.array([[0.0 for j in range(n)] for i in range(
        m+addentries)])
94
95      for j in range(n):
96        tmp=list(probability[k][:,j])
97        maxi=max(tmp)
98        maxind1=tmp.index(maxi)
99        tmp.reverse()
100       maxind2=len(tmp)-1-tmp.index(maxi)
101       tmp.reverse()
102       maxi=np.mean(tmp[maxind1:(maxind2+1)])     #zwischen den Maxima
          wird gemittelt
103       probabilityext[k][:,j]=list(probability[k][0:int((maxind1+maxind2
          )/2),j])+[maxi for i in range(addentries)]+list(probability[k][int
          ((maxind1+maxind2)/2):,j])
104
105
106
107 ############################
108 #Kleine und grosse Wahrscheinlichkeiten werden herausgefiltert!!!!!!!!
```

```
109 ############################
110
111
112 #      if probability[i][j]<0.05:
113 #        probability[i][j]=0.0
114 #      elif probability[i][j]>0.95:
115 #        probability[i][j]=1.0
116
117 ############################
118 ############################
119
120   #Wahrscheinlichkeit fuer jede Kreisscheibe wird berechnet
121   circleprobability=[0]*len(x)
122
123
124   for i in range(len(x)):
125     d=2*float(r[i])
126     if d<3.5:
127       d=3.5
128     elif d>10.0:
129       d=10.0
130
131     indexgeq=bisect.bisect_left(filename[1],d)
132
133     #Index wird so angepasst, dass der dazugehoerige Eintrag immer am
      naechsten bei d liegt
134     if indexgeq>0:
135       if indexgeq==len(filename[1]) or (d-filename[1][indexgeq-1] <
      filename[1][indexgeq]-d):
136         indexgeq=indexgeq-1
137
138     T=gaussian1d(float(y[i]),0,Tmin,Tmax,sigmasq)*fitfunc(fit,filename
      [1][indexgeq])/fitfunc(fit,d)
139
140     if (float(x[i])-xstart)>=downtrackpos[indexgeq][0] and (float(x[i])
      -xstart)<=downtrackpos[indexgeq][-1] and T>=Tpeak[indexgeq][0] and
      T<=Tpeak[indexgeq][-1]:
141       circleprobability[i]=float(scipy.interpolate.interpn([
      downtrackpos[indexgeq],Tpeak[indexgeq]],probabilityext[indexgeq],[(
      float(x[i])-xstart),T]))
142     else:
143       circleprobability[i]=0
144
145
146
147   #Schreibt ein File "probabilities.txt" in den Ordner "circles",
      welches ein Array mit den Wahrscheinlichkeiten enthaelt.
148   datei=file('%s/seed%d/circles/probabilities.txt' % (outputname,seed),
      'w')
149   for j in circleprobability:
150     datei.write("%f\n" % j)
151   datei.close()
```

C.5 probabilities2ini.py

```
from os import *
import random
import numpy

# File aendert die initialisierten Werte im ini-File (das muss schon im
    Home-Verzeichnis sein -> "prepare_ini.py") gemaess den
    Wahrscheinlichekeiten und updown = '-1.0'/'+1.0'


def probabilities2ini(seed,dimX,dimY,loc,updown,anzbits,i,outputname):
  """Programm erzeugt aus Liste mit Switching-Wahrscheinlichkeiten (im
    Verzeichnis /circles) und den Modell-Files (im Verzeichnis /inp)
    eine den Wahrscheinlichkeiten entsprechend zufaellige
    Magnetisierung der einzelnen Grains. Anschliessend wird die FEMME-
    Simulation zur Magnetfeldauslesung gestartet. Im aktuellen
    Verzeichnis muss sich dazu unter anderem das File "headdynamics.par
    " mit den entsprechenden Parametern befinden"""


  #Auslesen der Wahrscheinlichkeiten
  datei=file('%s/seed%d/circles/probabilities.txt' % (outputname,seed),
    'r')
  probabilities=datei.readlines()
  datei.close()
  for i in range(len(probabilities)):
    probabilities[i]=float(probabilities[i])

  datei=file('%s/seed%d/model.ini' % (outputname,seed),'r')
  inhalt=datei.readlines()
  datei.close()

  #Seed-Initialisierung und Erzeugung der Zufallszahlen -> Ergebnisse
    +/-1 werden in letzte Spalte von ini-File eingetragen
  random.seed((seed-1)*anzbits+1+i) #die Initialisierung ist so, dass
    der seed immer von einer Bitstelle zur naechsen um 1 erhoeht wird.
  for i in range(len(inhalt)):
    inhalt[i]=inhalt[i].split()
    if random.random()<probabilities[i]:
      inhalt[i][3]=updown

  thefile=file('%s/seed%d/model.ini' % (outputname,seed),'w')
  for item in inhalt:
    print>>thefile, "%s %s %s %s" % (item[0],item[1],item[2],item[3])
  thefile.close()
```

C.6 randy_reader.py

```
import numpy as np
#import matplotlib.pyplot as plt

#Jordan-Algorithmus (https://de.wikipedia.org/wiki/Punkt-in-Polygon-
    Test_nach_Jordan):
def kreuz_prod_test(A,B,C):
    if (A[1]==B[1] and B[1]==C[1]):
        if ((B[0]<=A[0] and A[0]<=C[0]) or (C[0]<=A[0] and A[0]<=B[0]))
```

```
    :
8               return 0
9           else:
10              return 1
11      if (A[1]==B[1] and A[0]==B[0]):
12          return 0
13      if (B[1]>C[1]):
14          B,C=C,B
15      if (A[1]<=B[1] or A[1]>C[1]):
16          return 1
17      delta=(B[0]-A[0])*(C[1]-A[1])-(B[1]-A[1])*(C[0]-A[0])
18      if delta>0:
19          return -1
20      elif delta<0:
21          return 1
22      else:
23          return 0
24
25
26
27  def punkt_in_polygon(P,Q):
28      t=-1
29      for i in range(len(P)-1):
30          t=t*kreuz_prod_test(Q,P[i],P[i+1])
31      t=t*kreuz_prod_test(Q,P[-1],P[0])
32      return t
33
34
35
36
37
38  #Bestimmt, in welchem Vieleck der Punkt (x,y) liegt. Zurueckgegeben
        wird der entsprechende Index in "connect". Zur Bestimmung wird der
        Jordan-Algorithmus verwendet (https://de.wikipedia.org/wiki/Punkt-
        in-Polygon-Test_nach_Jordan). startindex gibt den gewuenschten
        startindex an.
39  def check_index(x,y,points,connect,startidx):
40      for i in range(startidx,len(connect)):
41          erg=punkt_in_polygon(points[connect[i],:],(x,y))
42          if erg==1:
43              return i
44      for i in range(startidx):
45          erg=punkt_in_polygon(points[connect[i],:],(x,y))
46          if erg==1:
47              return i
48      return -1
49
50
51
52
53  def randy_reader(seed,dimX,dimY,loc,sigma,outputname):
54
55      modelname = 'isolatedGrains%dx%d_seed%d_anisotropy_random_D%1.1
        f_sigma%1.6f' % (dimX,dimY,seed,loc,sigma)
56      #modelname = 'isolatedGrains500x60_seed1_anisotropy_random_D5.0
        _sigma0.010000.vtu'    #TEST!!!!!!
```

```
57      #dimX=500      #TEST!!!!!
58      #dimY=60      #TEST!!!!!
59
60      datei=file('/home/florians/ULTRA/vtk/%s.vtu' % modelname,'r')
61      #datei=file(modelname,'r')    #TEST!!!!!
62      inhalt=datei.readlines()
63      datei.close()
64
65      #Eckpunkte der Polygone auslesen und formatieren:
66      rawpoints=inhalt[5]
67      rawpoints=rawpoints.replace('<DataArray NumberOfComponents="3"
        format="ascii" type="Float64">','')
68      rawpoints=rawpoints.replace('</DataArray>','')
69      rawpoints=rawpoints.split()
70      points=np.zeros([len(rawpoints)/3,2])
71      for i in range(len(points)):
72          points[i,0]=float(rawpoints[3*i])
73          points[i,1]=float(rawpoints[3*i+1])
74
75
76      #Indizes der Eckpunkte auslesen, die Polygon ergeben
77      connect=inhalt[8]
78      connect=connect.replace('<DataArray Name="connectivity" format="
        ascii" type="Int32">','')
79      connect=connect.replace('</DataArray>','')
80      connect=connect.split('  ')
81      for i in range(len(connect)):
82          connect[i]=connect[i].split()
83      for i in range(len(connect)):
84          for j in range(len(connect[i])):
85              connect[i][j]=int(connect[i][j])
86      for i in range(len(connect)):
87          connect[i]=np.array(connect[i])
88
89
90      #Einlesen der ini-Daten (Magnetisierungsrichtung)
91      datei=file('%s/seed%d/model.ini' % (outputname,seed),'r')
92      #datei=file('model.ini','r') #TEST!!!!!!
93      ini=datei.readlines()
94      datei.close()
95      for i in range(len(ini)):
96          ini[i]=ini[i].split()
97          ini[i]=float(ini[i][-1])
98
99
100     #Einlesen der Reader-Sensitivity-Function
101     sensitivity = np.loadtxt("randy_reader.txt", delimiter=",")
102     dim=sensitivity.shape
103     deltax=0.5  #Zellengroesse der Sesitivity-Function in downtrack
104     deltay=0.5  #Zellengroesse der Sesitivity-Function in offtrack
105
106     x_distance=dim[0]*deltax
107     y_distance=dim[1]*deltay
108
109     x=np.linspace(-(dim[0]-1)*deltax/2,(dim[0]-1)*deltax/2,dim[0])
110     y=np.linspace(-(dim[1]-1)*deltay/2,(dim[1]-1)*deltay/2,dim[1])
```

```
111
112
113     grid_deltax=deltax  #Punkteabstand auf dem Grid, ueber das der
    Reader bewegt wird
114
115     assert y_distance<=dimY
116     grid_x=np.arange(-dimX/2+grid_deltax/2,dimX/2,grid_deltax)
117     grid_y=np.arange(-y_distance/2+deltay/2,y_distance/2,deltay)
118
119     mag=np.zeros([len(grid_x),len(grid_y)])
120
121     check_erg=0
122     for i in range(len(grid_x)):
123         for j in range(len(grid_y)):
124             #print (i,j)
125             check_erg=check_index(grid_x[i],grid_y[j],points,connect,
    check_erg)
126             if check_erg>=0:
127                 mag[i,j]=ini[check_erg]
128
129     res=np.zeros(len(grid_x))
130
131     for t in range(len(grid_x)):
132         #print t
133         res[t]=sum(sum(sensitivity[max(int(dim[0]/2)-t,0):min(dim[0],
    len(grid_x)+int(dim[0]/2)-t),:]*mag[max(t-int(dim[0]/2),0):min(dim
    [0]-int(dim[0]/2)+t,len(grid_x)),:]))
134
135     np.savetxt('%s/seed%d/curve.txt' % (outputname,seed),res)
```

Appendix D

Preparation for the SNR calculator

The signal curves of the programs in the previous Appendix shall now be compared for the SNR calculator of SEAGATE that allows us to determine the SNR of 50 signal curves from different granular media (see also Subsec. 6.4.1). Note that the SNR calculator requires a certain bit pattern given in Sec. 6.3. The following source code can be found on the GTO-server in */home/florians/diplomarbeit*. The file "auswertung7.py" in Sec. D.1 has the following input parameters:

- *seedmax*: total amount of different seeds those results should be summarized for the SNR calculator (For the calculator exactly 50 seeds are required, so it will be necessary to do the simulation of the previous Appendix at least for seeds 1–50 and choose $seedmax := 50$.)
- *outputname*: path/name of the directory where the results of *seedmax* seeds are provided
- v, *magt*: velocity of the writing head in m/s and time period of the applied magnetic field (with field rise and decay time) per bit in ns ($v \cdot magt =$ bit-length in nm)
- *zeropoint*: down-track coordinate of the zero crossing between the 18th and 19th bit in nm (This value is taken as a reference value to determine the position of the bits by the bit-length. It can be either calculated somehow analytically or determined experimentally by the file in Sec. D.2 which works analogously as "auswertung7.py" but uses a *seedmax* times repeated simulation of 18 consecutive (-1)- and 19 (+1)-bits to calculate the mean zero crossing of their signals and returns it.)
- d: integer that is added to the name of the output file (see line 49)
- The variables in lines 39 & 41 could also be varied, but the resulting file for the SNR calculator should contain 37 bits with a resolution of 7 or 9 samples per bit.

The program reads the *seedmax* signal vectors, shortens them via *zeropoint* and the bit length to finally contain precisely the signal of the 37 bits and changes the resolution to 7 or 9 equidistant samples per bit via interpolation. Afterwards the manipulated *seedmax* vectors are stringed together to a long array with $seedmax \cdot 37 \cdot 7$ or $seedmax \cdot 37 \cdot 9$ elements and saved in a txt-file (see lines 49–53). The final step is to convert that txt-array into a binary format (*.bin) via the MATLAB-file "conv2bin.m" of Sec. D.3, where the input- and output-filenames (as strings) are the only parameters. The received file can then finally used in the SNR calculator, which can be started via *ewsnranalsim.exe "path to files" "*.bin" "options.txt"* from the windows console and requires the MATLAB Compiler Runtime "MCR 8.1 (Matlab 2013a)".

D.1 auswertung7.py

```
1  from os import *
2  import matplotlib.pyplot as plt
3  import bisect
4  import numpy as np
5  import math
6
7  #Fuer Daten aus Randys Reader!
8
9  def auswertung7(seedmax,outputname,v,magt,zeropoint,d):
10   """v bezeichnet die Schreibgeschwindigkeit [m/s], magt die Dauer des
       Magnetfeldes [ns] (z.B. 1.2ns). zeropoint entspricht Nullpunkt
       zwischen Bit 18 und 19 - berechnet durch *_calib.py-Files"""
11
12   #Deklaration der Arrays fuer die Daten (5000 ist die maximale Laenge
       der Spalten im log-File)
13   x=np.zeros([seedmax,1000])
14   mz=np.zeros([seedmax,1000])
15   minimum=[0 for i in range(seedmax)]
16   maximum=[0 for i in range(seedmax)]
17   argminimum=[0 for i in range(seedmax)]
18   argmaximum=[0 for i in range(seedmax)]
19   startx=[0 for i in range(seedmax)]    #Integralsgrenzen
20   endx=[0 for i in range(seedmax)]    #
21   lengthintegrallist=[0 for i in range(seedmax)]        #Laenge der
       Liste im Integrationsbereich
22   root=[0 for i in range(seedmax)]
23   startindex=[0 for i in range(seedmax)]
24   endindex=[1000 for i in range(seedmax)]
25
26   #Einlesen der Daten
27   for i in range(seedmax):
28     print i
29     mz[i,:]=np.loadtxt('%s/curve%d.txt' % (outputname,i+1))
30
31     grid_deltax=0.5
32     dimX=500
33     x[i,:]=np.arange(grid_deltax/2,dimX,grid_deltax)
34
```

```
35
36
37
38    #Samples pro Bit (7 oder 9)
39    samplesperbit=9
40    #Anzahl der betrachteten Bits
41    numberofbits=37
42    #Array fuer Ergebnisse
43    samples=[[0 for j in range(samplesperbit*numberofbits)] for i in
        range(seedmax)]
44    xvalues=[zeropoint-v*magt*18+v*magt/samplesperbit/2+j*v*magt/
        samplesperbit for j in range(samplesperbit*numberofbits)]
45    for i in range(seedmax):
46      samples[i]=np.interp(xvalues,x[i][:endindex[i]],mz[i][:endindex[i
        ]])
47
48
49    datei=file('%s/curves%d.dat' % (outputname,d),'w')
50    for i in range(seedmax):
51      for j in samples[i]:
52        datei.write("%f\n" % j)
53    datei.close()
54
55
56
57    plt.figure()
58    #Graphen fuer alle Seeds:
59    for i in range(seedmax):
60      plt.plot(xvalues, samples[i])
61      #plt.plot(samples[i])
62
63  # #Graph fuer nur einen Seed:
64  # plt.plot(x[7][:endindex[7]], mz[7][:endindex[7]])
65  ##  plt.plot(intarea, intfctvalues[0])
66
67  # plt.plot([x[0][0],x[0][endindex[0]-1]],[zeroline,zeroline],'--',color
        ='k')
68    plt.axis([0, 500, -200, 200])
69    plt.rc('text', usetex=True)
70    plt.xlabel('down-track position [$nm$]')
71    plt.ylabel('$M_z$')
72    plt.savefig('%s/figure_2.png' % outputname)
73  # plt.show()
74
75    plt.close()
```

D.2 auswertung7_calib.py

```
1  from os import *
2  import matplotlib.pyplot as plt
3  import bisect
4  import numpy as np
5  import math
6
7  def auswertung7_calib(seedmax,outputname,bitlength):
```

```
8
9
10  #Deklaration der Arrays fuer die Daten (800 ist die maximale Laenge
      der Spalten im log-File)
11  x=np.zeros([seedmax,1000])
12  mz=np.zeros([seedmax,1000])
13  minimum=[0 for i in range(seedmax)]
14  maximum=[0 for i in range(seedmax)]
15  argminimum=[0 for i in range(seedmax)]
16  argmaximum=[0 for i in range(seedmax)]
17  startx=[0 for i in range(seedmax)]    #Integralsgrenzen
18  endx=[0 for i in range(seedmax)]    #
19  lengthintegrallist=[0 for i in range(seedmax)]        #Laenge der
      Liste im Integrationsbereich
20  root=[0 for i in range(seedmax)]
21  startindex=[0 for i in range(seedmax)]
22  endindex=[1000 for i in range(seedmax)]
23
24  #Einlesen der Daten
25  for i in range(seedmax):
26    mz[i,:]=np.loadtxt('%s/curve%d.txt' % (outputname,i+1))
27
28    grid_deltax=0.5
29    dimX=500
30    x[i,:]=np.arange(grid_deltax/2,dimX,grid_deltax)
31
32    maxindex=bisect.bisect_left(x[i][:endindex[i]],np.floor(15*
      bitlength))
33    minindex=bisect.bisect_left(x[i][:endindex[i]],np.floor(30*
      bitlength))
34    mztmp=mz[i][maxindex:minindex]
35    jumpindex=bisect.bisect_left(mztmp,0)
36    root[i]=np.interp(0,[mztmp[jumpindex],mztmp[jumpindex+1]],[x[i][
      maxindex+jumpindex],x[i][maxindex+jumpindex+1]])
37
38  #Nullpunkt fuer Bits
39  zeropoint=np.mean(root)
40
41
42
43
44  plt.figure()
45  #Graphen fuer alle Seeds:
46  for i in range(seedmax):
47    plt.plot(x[i][:endindex[i]], mz[i][:endindex[i]])
48
49 ##  #Graph fuer nur einen Seed:
50 ##  plt.plot(x[0][:endindex[0]], mz[0][:endindex[0]])
51 ### plt.plot(intarea, intfctvalues[0])
52
53 ##  plt.plot([x[0][0],x[0][endindex[0]-1]],[zeroline,zeroline],'--',
      color='k')
54  plt.axis([0, 500,-1000,1000])
55  plt.rc('text', usetex=True)
56  plt.xlabel('down-track position [$nm$]')
57  plt.ylabel('$M_z$')
```

```
58    plt.savefig('%s/figure_1.png' % outputname)
59 # plt.show()
60
61    plt.close()
62
63    return zeropoint
```

D.3 conv2bin.m

```
1 function conv2bin(input,output)
2 data=textread(input);
3 fileID=fopen(output,'w');
4 fwrite(fileID,data,'double')
5 fclose(fileID)
```

Bibliography

[1] R. F. L. Evans, W. J. Fan, P. Chureemart, T. A. Ostler, M. O. A. Ellis, and R. W. Chantrell. Atomistic spin model simulations of magnetic nanomaterials. *Journal of Physics: Condensed Matter*, 26(10):103202, 2014.

[2] R. F. L. Evans, D. Hinzke, U. Atxitia, U. Nowak, R. W. Chantrell, and O. Chubykalo-Fesenko. Stochastic form of the Landau-Lifshitz-Bloch equation. *Physical Review B: Condensed Matter and Materials Physics*, 85(1):014433, Jan. 2012.

[3] S. Hernández, P. Lu, S. Granz, P. Krivosik, P. Huang, W. Eppler, T. Rausch, and E. Gage. Using ensemble waveform analysis to compare heat assisted magnetic recording characteristics of modeled and measured signals. *IEEE Transactions on Magnetics*, 53(2):1–6, Feb 2017.

[4] Y. Kubota, Y. Peng, and Y. e. Ding. Heat-assisted magnetic recording's extensibility to high linear and areal density. *IEEE Transactions on Magnetics*, PP:1–6, 08 2018.

[5] D. Lewis and N. Nigam. Geometric integration on spheres and some interesting applications. *Journal of Computational and Applied Mathematics*, 151:141–170, Feb. 2003.

[6] D. Mortari. On the rigid rotation concept in n-dimensional spaces. 49, 07 2001.

[7] O. Muthsam, C. Vogler, and D. Suess. Noise reduction in heat-assisted magnetic recording of bit-patterned media by optimizing a high/low Tc bilayer structure. *Journal of Applied Physics*, 122(21):213903, 2017.

[8] K. Takano. Readback spatial sensitivity function by reciprocity principle and media readback flux. *IEEE Transactions on Magnetics*, 49(7):3818–3821, July 2013.

[9] G. Varvaro and F. Casoli. Ultrahigh-desity magnetic recording. *CRC Press*, 2016.

[10] C. Vogler, C. Abert, F. Bruckner, and D. Suess. Landau-Lifshitz-Bloch equation for exchange-coupled grains. *Phys. Rev. B*, 90:214431, Dec 2014.

[11] C. Vogler, C. Abert, F. Bruckner, D. Suess, and D. Praetorius. Heat-assisted magnetic recording of bit-patterned media beyond 10 Tb/in2. *Applied Physics Letters*, 108(10), 2016. cited By 0.

[12] C. Vogler, C. Abert, F. Bruckner, D. Suess, and D. Praetorius. Influence of grain size and exchange interaction on the LLB modeling procedure. *Journal of Applied Physics*, 120(22):223903, 2016.

[13] X. Wang, B. Valcu, and N. Yeh. Transition width limit in magnetic recording. *Applied Physics Letters*, 94(20):202508, 2009.

[14] D. Weller and A. Moser. Thermal effect limits in ultrahigh-density magnetic recording. *IEEE Transactions on Magnetics*, 35(6):4423–4439, Nov 1999.

[15] J.-G. Zhu and H. Li. Understanding signal and noise in heat assisted magnetic recording (invited). *IEEE Transactions on Magnetics*, 49(2):765–772, 2013.