Research Project in Computational Physics

Sky-MoCa
Introduction to Monte Carlo Methods by Example

by

Niki Kilbertus

supervised by

Prof. Dr. Christian Back

Faculty for Physics
University of Regensburg

September 2016

Abstract

We give a general introduction to Monte Carlo methods with a special focus on simulated annealing and the Metropolis algorithm. This report is conceived as an ab-initio guide to implement and properly analyse simulated annealing by the example of a three-dimensional lattice spin model. This system can be used to simulate different phases of chiral magnets such as MnSi at finite temperature in an external magnetic field. A thorough theoretical understanding of the requirements, applicability and errors is developed and illustrated by reference to the specific example. All code is publicly available for reproducability. Common pitfalls are identified and we provide clear and practical guidelines on how to avoid them.

Contents

1 Theoretical background

1.1 Introduction

Monte Carlo methods comprise many high level concepts as well as manifold specific algorithms. They all have in common that they make use of randomness to draw conclusions about statistical quantities. Generally, Monte Carlo methods are applicable to almost any problem with statistically defined properties. Therefore it does not come as a surprise that they are commonly used in physics to describe complex interacting systems with many degrees of freedom. Although generally applicable and relatively simple to understand and implement, Monte Carlo methods must not be applied blindly. They are almost always inferior to more specialized methods and might fail altogether in certain situations. Hence, careful analysis is absolutely vital for any Monte Carlo method. In theory there are accurate measures for the applicability of the methods, but in practice it frequently proves quite delicate to apply those measures correctly. Unfortunately, pitfalls are often overlooked or even ignored in the literature. This report focuses on an ab-initio introduction to Monte Carlo methods by reference to a specific real world example. After developing the theoretical concepts we actually apply them in great detail to illustrate some common difficulties.

The concrete physical model deals with a rather recently discovered magnetic phenomenon, called *skyrmions*. Since the theoretical prediction of skyrmions in the eighties by Bogdanov [2] based on previous work in particle physics by Skyrme [17] and the subsequent experimental discovery, this configuration has been extensively investigated both by theorists and experimentalists [10, 11, 14, 23, 22, 16, 9, 18, 12, 15]. Skyrmions have caused a lot of excitement mostly due to their promising properties that might qualify them as fast and efficient memory [7]. Three-dimensional non-perturbative classical Monte Carlo methods developed recently [3, 9], allow us to compute the full finite temperature phase diagram and explore phase transitions. The goal of this project was to implement a Monte Carlo code with the discretized lattice Hamiltonian introduced in [3] and basically reproduce the results discussed there. In contrast to most work on Monte Carlo methods for spin lattices, we found it worthwhile to provide a detailed exposition of the algorithms and numerical aspects.

As a consequence there will be two separate reports. This one focuses on the algorithmic aspects and Monte Carlo methods in general, while the other one will go into more detail about the model and discuss physical results. The present text can serve as a step-by-step introduction on how to write Monte Carlo codes and discusses associated pitfalls in detail based on a concrete example. All code is publicly available on GitHub under https://github.com/nikikilbertus/Sky-MoCa and we hope that it will

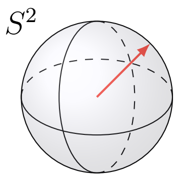

Figure 1.1: We consider three-dimensional spins, i. e. elements of the unit sphere S^2.

be useful to others.

The report is structured as follows. We briefly introduce the model Hamiltonian in section 1.2. In section 1.3 we provide a first high level overview of Monte Carlo integration and optimization and state some important results mostly without proofs. Section 1.4 is the core part of this chapter. We derive and discuss properties of the so called *Metropolis algorithm* in quite some detail. Furthermore, practical verification strategies for Monte Carlo codes in general and the Metropolis algorithm in particular are treated in section 1.6. Chapter 2 introduces our implementation of the algorithm, Sky-MoCa, in section 2.1. In section 2.2, we go through the theoretic verification strategies discussed earlier in a specific sample simulation using Sky-MoCa. This should give the reader a feeling for what to expect when writing their own Monte Carlo code. Since the main goal of this report is to illuminate the methods, we move physical results to the second paper. Finally, we conclude in section 2.3.

1.2 The Spin Lattice Model

A common high-level way to think about magnetism in condensed matter is in terms of complex collective behavior of spins, each of which is associated with a magnetic moment. Astoundingly, this figuratively simple model is quite powerful and allows for a thorough explanation of a wide range of phenomena. Let us consider a three-dimensional lattice with equidistant spacing in each direction. To each lattice site we attach a spin, represented by an element of the unit sphere S^2, see Figure 1.1. In the following we will often resort to the two-dimensional model for illustration purposes, because it is easier to draw on a two-dimensional surface as illustrated in Figure 1.2. However, all computations solely concern the three-dimensional model. Each vertex of the lattice could for example represent a atom in a solid with a rigid crystal like structure. Hence the whole lattice can be interpreted as the regular atomic structure of a cubical piece of solid material.

We work with a cubic lattice

$$\Sigma := \{1, \dots, N_x\} \times \{1, \dots, N_y\} \times \{1, \dots, N_z\} \subset \mathbb{N}^3 \subset \mathbb{R}^3 \,, \tag{1.1}$$

where we interpret $(i, j, k) \in \Sigma$ as $i\hat{\mathbf{x}}+j\hat{\mathbf{y}}+k\hat{\mathbf{z}} \in \mathbb{R}^3$. Here, $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ are the standard basis vectors of $\mathbb{R}^3$. Note that we can translate the whole lattice by arbitrary integer linear combinations of the standard basis vectors, thus starting at $(1, 1, 1)$ does not have any physical meaning. It merely corresponds nicely to the numerical implementation in

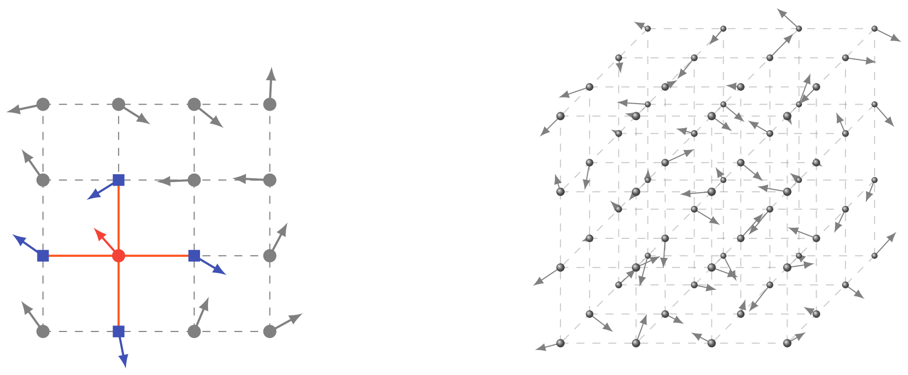

Figure 1.2: On the left side we show a two-dimensional spin lattice. Each lattice site carries a magnetic moment or spin, represented by an arrow of unit length, i. e., in the two-dimensional picture, by an element of S^1. The neighbors of a lattice site are the ones above, below, left and right in the two-dimensional case. The blue squares are the neighbors of the red circle. The three-dimensional picture on the right side becomes unclear in a two-dimensional drawing rather quickly. Note that the magnetic moments are now also three-dimensional, i. e. elements of the two-dimensional unit sphere S^2. In three dimensions each vertex has up to six neighbors.

any 1-indexed programming language. At each point $\mathbf{r} \in \Sigma$ we attach a spin $\mathbf{S_r} \in S^2$, resulting in the overall configuration space

$$\Pi := \prod_{\mathbf{r}\in\Sigma} S^2 \,. \tag{1.2}$$

Each element of Π consists of $|\Sigma| = N_x N_y N_z$ spins and thus describes one possible configuration of the whole system. We refer to the specific spin at position $\mathbf{r} \in \Sigma$ by $\mathbf{S_r} \in S^2$. Note that only the positions of the spins are discretized, but not explicitly their directions. In any real implementation there is always a fine grained discretization caused by the finite number of representable floating point numbers. Discrete vertices and continuous three-dimensional spins mirror nicely the natural crystal structure of solids.

In a typical physical simulation one might want to find the ground state of this system, i. e. the lowest energy state, for some given external and internal parameters. Alternatively, we could be interested in macroscopic properties such as the specific heat or magnetization. Necessarily, we need to define some notion of energy and also identify some possible parameters for the spin lattice. Apparently any physical measure of energy must be based on interactions between the spins within the system or with external fields. Let us elaborate on some possible interactions. Each of them comes with a constant that can be interpreted as a weight, i. e. how much the respective interaction contributes to the energy with respect to the others. Those material constants can be treated as free parameters in the simulation.

1.2.1 Interactions

Ferromagnetic/direct exchange

The most obvious interaction is the *ferromagnetic* or *direct* exchange. Pictorially speaking, it favors constellations where spins that are close to each other point into the same direction. A system only interacting this way will end up in a state where all spins are parallel to each other. The measure for parallelism of two neighboring spins $\mathbf{S}_{\mathbf{r}_1}, \mathbf{S}_{\mathbf{r}_2} \in S^2$ can be expressed as

$$-\mathbf{S}_{\mathbf{r}_1} \cdot \mathbf{S}_{\mathbf{r}_2} = -\|\mathbf{S}_{\mathbf{r}_1}\|\|\mathbf{S}_{\mathbf{r}_2}\| \cos(\alpha) \,, \tag{1.3}$$

where α is the angle between $\mathbf{S}_{\mathbf{r}_1}$ and $\mathbf{S}_{\mathbf{r}_2}$. The minus sign ensures that the energy of two parallel spins is actually smaller than the energy of two perpendicular or even antiparallel ones. We will always consider the ferromagnetic exchange to be *local* or *short ranged* in the sense that it only contributes to the energy for adjacent lattice sites, as shown in Figure 1.2.

Interaction with an external field

Another important and absolutely standard exchange term describes the interaction of the system with an *external magnetic field.* Clearly, every spin tries to align with an external field $\mathbf{B}$, which we express mathematically via $-\mathbf{B} \cdot \mathbf{S}$ for every spin $\mathbf{S}$ on the lattice. This contribution is also called the *Zeeman energy.* It can be misleading to use terms such as *non-local* or *long ranged* for this exchange, since it is not an interaction between two or more spins within the system, but affects each lattice site independently in the same fashion. Naturally, for the external field, $\mathbf{B}$ itself is the weight or parameter of the interaction.

Dipole-dipole interaction

The *dipole-dipole interaction* is somewhat more complex, but also much weaker than the two previous ones. For two spins $\mathbf{S}_{\mathbf{r}_1}, \mathbf{S}_{\mathbf{r}_2} \in S^2$, it is given by

$$-\|\mathbf{r}\|^{-3}(3(\mathbf{S}_{\mathbf{r}_1} \cdot \hat{\mathbf{r}})(\mathbf{S}_{\mathbf{r}_2} \cdot \hat{\mathbf{r}}) - \mathbf{S}_{\mathbf{r}_1} \cdot \mathbf{S}_{\mathbf{r}_2}) \,, \tag{1.4}$$

where $\mathbf{r} = \mathbf{r}_2 - \mathbf{r}_1$ and $\hat{\mathbf{r}} = \mathbf{r}/\|\mathbf{r}\|$ points from the location of the first spin to the location of the second one. The dipole-dipole interaction depends on the distance between the two lattice sites as well as the orientation of the two spins not only relative to each other, but also to the line connecting them. Moreover, the explicit dependence on the relative position already indicates that the dipole-dipole interaction is relevant for each pair of magnetic moments in the system, it is a long ranged interaction. For a lattice with N^3 vertices the number of pairs scales like N^6. Due to limited computational resources and its relative weakness, most simulations do not take the dipole-dipole exchange into account. We will also disregard it completely in our implementation and further discussion in this report.

Dzyaloshinskii-Moriya exchange

In this work we are interested in crystals that lack inversion symmetry, e. g. MnSi, and thus exhibit chiral magnets. The missing inversion symmetry gives rise to the so called *weak Dzyaloshinskii-Moriya* (DM) coupling. Like the FM interaction, the DM exchange only contributes for adjacent lattice sites. The discretized version for two spins $\mathbf{S}_{\mathbf{r}_1}, \mathbf{S}_{\mathbf{r}_2} \in S^2$ at neighboring positions $\mathbf{r}_1, \mathbf{r}_2 \in \Sigma$ reads

$$-(\mathbf{S}_{\mathbf{r}_1} \times \mathbf{S}_{\mathbf{r}_2}) \cdot (\mathbf{r}_2 - \mathbf{r}_1)\,. \tag{1.5}$$

Note that $\mathbf{r}_2 - \mathbf{r}_1$ is normalized by definition for neighboring vertices. Since the cross product is zero for parallel vectors and maximal for perpendicular ones, the DM coupling acts against the ferromagnetic interaction and favors constellations where adjacent spins are perpendicular to each other in the plane normal to their connecting line.

1.2.2 The Hamiltonian

Summing up the interactions

Let us now combine the FM and DM interaction as well as an external magnetic field to compute the energy for the lattice Σ. To this end we add up the contributions from each spin and for each interaction. For instance, the external field exchange term simply becomes

$$-\sum_{\mathbf{r}\in\Sigma} \mathbf{S}_\mathbf{r} \cdot \mathbf{B}\,. \tag{1.6}$$

Adding up the terms for the FM interaction with all direct neighbors naively as in

$$-\sum_{\mathbf{r}\in\Sigma} \mathbf{S}_\mathbf{r} \cdot (\mathbf{S}_{\mathbf{r}+\hat{\mathbf{x}}} + \mathbf{S}_{\mathbf{r}-\hat{\mathbf{x}}} + \mathbf{S}_{\mathbf{r}+\hat{\mathbf{y}}} + \mathbf{S}_{\mathbf{r}-\hat{\mathbf{x}}} + \mathbf{S}_{\mathbf{r}+\hat{\mathbf{z}}} + \mathbf{S}_{\mathbf{r}-\hat{\mathbf{z}}})\,, \tag{1.7}$$

poses two problems. First, apparently in this fashion we count the interaction between some pairs of spins multiple times. If Σ was an infinite grid, i. e. $\Sigma = \mathbb{Z}^3$, we would count every pair exactly twice and could simply divide (1.7) by 2. However, Σ is finite which leads to the second more subtle issue. Not every lattice site has a neighbor in each direction $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$. Consider the spin in the lower right corner of the lattice in Figure 1.2. The sum in our first naive expression for the FM interaction (1.7) suggests that we need its right and lower neighbor, which apparently do not exist. As a consequence, we need to define some behavior at the boundaries. There are several ways to do this and a plausible treatment of boundary conditions is a central aspect of many different areas in scientific computing. It is important to realize that this is not simply an implementation nuisance that we can get rid off by any means that do not break the code. Various boundary conditions represent different physical systems and can significantly alter the results of a simulation.

Boundary conditions

In many cases the desired simulation volume is an infinite space, or at least so large compared to all internal length scales that it is practically infinite. On physical computing machines we are always limited to finite grids. The most obvious way for finite systems is to set all spins outside of Σ to zero, i. e. $\mathbf{S_r} = 0$ for all $\mathbf{r} \in \mathbb{Z}^3 \setminus \Sigma$. Consequently the lattice sites on the boundary have less neighbors to interact with. This is called an *open boundary*. Open boundaries are often undesirable, because they heavily impact the physical behavior. In our example, a magnetic moment at the boundary is less effected by the FM and DM interaction, simply because it has less neighbors. However, the external field still has the same impact on a boundary spin as on any other vertex. This imbalance leads to polarized magnetic moments at the boundaries, i. e. they tend to all point in the direction of the external magnetic field $\mathbf{B}$ to maximize their Zeeman energy.

One common way to get out of this dilemma is to implement *periodic boundary conditions*. We think of our lattice as a finite box of volume L^3 for $L \in \mathbb{R}_{\geq 0}$ and simply replicate that box infinitely many times to fill the whole space, see Figure 1.3 for a two-dimensional illustration. In practice, we will access the lattice sites by indexing a three-dimensional array with indices $(i, j, k) \in \Sigma$. Whenever the algorithm tries to index out of bounds, e. g. requests $j = N_y + 1$, we simply use $j = 1$ again. For $i = 0$ we use $i = N_x$, $k = N_z + 2$ is replaced by $k = 2$ and so on. In general, this behavior can be achieved by always working with the indices

$$((i-1 \mod N_x)+1, (j-1 \mod N_y)+1, (k-1 \mod N_z)+1) \in \Sigma \tag{1.8}$$

whenever the algorithm requests the spin at position $(i, j, k) \in \mathbb{Z}^3$. By construction, those new indices always lie in Σ. We can now unproblematically include spins at officially non-existing positions in our mathematical expressions, implicitly assuming that those are being wrapped around periodically to positions within the original lattice. This way every vertex has the same number of neighbors and we do not observe spurious boundary effects. In this report, we will only employ periodic boundary conditions in all three directions using (1.8). For other scenarios we kept the possibility of opening up the boundaries in one direction in our implementation. This solves the issue of missing neighbors, which we encountered in (1.7).

Combining all interactions

Now that we have established a well defined behavior at the boundary, we can also tackle the problem of counting interactions multiple times. For periodic boundary conditions we could divide expression (1.7) by 2. In the implementation we do not want to actually carry out redundant computations, thus we work with

$$-\sum_{\mathbf{r}\in\Sigma} \mathbf{S_r} \cdot \left(\mathbf{S_{r+\hat{x}}} + \mathbf{S_{r+\hat{y}}} + \mathbf{S_{r+\hat{z}}}\right) . \tag{1.9}$$

It is worth convincing oneself that in this expression, assuming open or periodic boundary conditions, every pair of neighboring lattice sites is included exactly once.

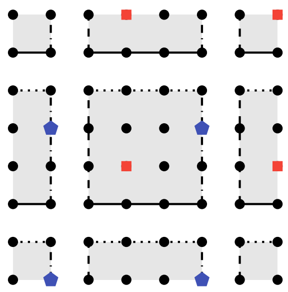

Figure 1.3: We illustrate periodic boundary conditions for a two-dimensional 4×4 grid. The simulation volume stored in memory is given by the shaded square in the middle. This square is copied over and over again in all directions as indicated by the surrounding lattice sites. Note how the four boundaries of the square keep their orientation throughout that copying procedure. While it appears like we have filled an infinite space, all red squares are identified, because they are represented internally by only one vertex in the central square. The same holds true for the green pentagons of course. If the code tries to fetch the spin at the right neighbor of the green pentagon in the central square, we can instead pick the right neighbor of *any* green pentagon. One of those will always lie in the center square and is thus available in memory.

Finally, the overall DM energy is analogously

$$-\sum_{\mathbf{r}\in\Sigma}\left((\mathbf{S_r} \times \mathbf{S_{r+\hat{x}}}) \cdot \hat{\mathbf{x}} + (\mathbf{S_r} \times \mathbf{S_{r+\hat{y}}}) \cdot \hat{\mathbf{y}} + (\mathbf{S_r} \times \mathbf{S_{r+\hat{z}}}) \cdot \hat{\mathbf{z}}\right) . \tag{1.10}$$

Everything we discussed about boundary conditions for the FM exchange also holds true for the DM interaction. Adding up all interactions (1.6), (1.9) and (1.10), we obtain the Hamiltonian, i. e. the energy of the system in a given configuration $\pi \in \Pi$

$$\begin{aligned} H(\pi) = -\sum_{\mathbf{r}\in\Sigma}\Big(&\mathbf{S_r} \cdot \mathbf{B} + J\, \mathbf{S_r} \cdot (\mathbf{S_{r+\hat{x}}} + \mathbf{S_{r+\hat{y}}} + \mathbf{S_{r+\hat{z}}}) \\ &+ K\, (\mathbf{S_r} \times \mathbf{S_{r+\hat{x}}} \cdot \hat{\mathbf{x}} + \mathbf{S_r} \times \mathbf{S_{r+\hat{y}}} \cdot \hat{\mathbf{y}} + \mathbf{S_r} \times \mathbf{S_{r+\hat{z}}} \cdot \hat{\mathbf{z}})\Big) . \end{aligned} \tag{1.11}$$

The parameters J, K and $\mathbf{B}$ are sometimes used synonymously for the ferromagnetic, the Dzyaloshinskii-Moriya and the external field interaction terms respectively. The physical behavior of the system strongly depends on these freely adjustable parameters. For a given set of parameters and at zero temperature we expect the system to settle in the ground state, which can be defined as the lowest energy configuration

$$\arg\min_{\pi\in\Pi} H(\pi) . \tag{1.12}$$

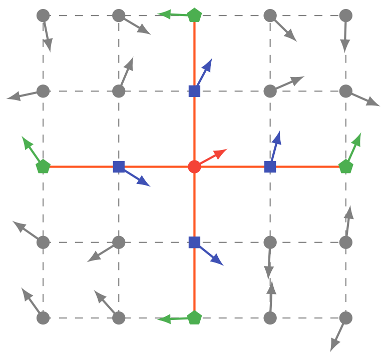

Figure 1.4: We select a spin at position $\mathbf{r} \in \Sigma$ (red circle) and show its nearest neighbors as blue squares and the next-nearest neighbors as green pentagons. Note that although the diagonally adjacent vertices are closer to $\mathbf{r}$ than the next-nearest neighbors, they are not included in any exchange terms of the Hamiltonian.

Anisotropy

In (1.11) we have collected all interactions on the lattice. Those originally stem from a continuous model. Therefore we have to expect discretization errors. In particular, the transition to a finite cubical lattice introduces anisotropies that are absent in the original continuum model. Buhrandt and Fritz show in [3] that we can partly compensate for those disruptive anisotropies by adding next-nearest neighbor interactions to the Hamiltonian. The optimal choice to render higher order deviations from the continuous model as small as possible is

$$\begin{aligned} H'(\pi) = \sum_{\mathbf{r}\in\Sigma} \Bigg(& \frac{J}{16} \mathbf{S}_{\mathbf{r}} \cdot (\mathbf{S}_{\mathbf{r}+2\hat{\mathbf{x}}} + \mathbf{S}_{\mathbf{r}+2\hat{\mathbf{y}}} + \mathbf{S}_{\mathbf{r}+2\hat{\mathbf{z}}}) \\ & + \frac{K}{8} (\mathbf{S}_{\mathbf{r}} \times \mathbf{S}_{\mathbf{r}+2\hat{\mathbf{x}}} \cdot \hat{\mathbf{x}} + \mathbf{S}_{\mathbf{r}} \times \mathbf{S}_{\mathbf{r}+2\hat{\mathbf{y}}} \cdot \hat{\mathbf{y}} + \mathbf{S}_{\mathbf{r}} \times \mathbf{S}_{\mathbf{r}+2\hat{\mathbf{z}}} \cdot \hat{\mathbf{z}}) \Bigg) . \end{aligned} \tag{1.13}$$

The Hamiltonian used in the simulation is then $H + H'$. Note that H' has the same structure of the FM and DM term of H, but enters with smaller constants and opposite sign. Thus it complements the nearest neighbor exchange by a directly related next-nearest neighbor interaction term. Adding next-nearest neighbor interactions makes a big difference computationally, but is handled analogously to the nearest neighbor interactions, see Figure 1.4. Periodic and open boundaries still work precisely the same way and are already taken care of by (1.8).

1.2.3 A note on parameter values

Recall that there is a great conflict of interest between the FM and the DM interactions. While the FM term in the Hamiltonian is minimal when all spins are parallel to each other, the DM term favors a configuration where neighboring spins are orthogonal.

The specific compromise between those effects depends mostly on their respective strengths J and K. In reality the direct interaction J is much stronger and one typically encounters helical or conical structures with long modulation periods for zero or small external fields respectively. For example in MnSi the spins wind around a given modulation axis only once per roughly 40 lattice spacings. By tuning the ratio J/K we can set the periodicity. For a period of N lattice sites, we have to set J/K to $1/\tan(2\pi/N)$. In all our simulations we chose $J = 1$ and $K = \tan(2\pi/10)$, i. e. one full modulation every 10 vertices. Because $K = \tan(2\pi/10) \approx 0.73$ is not much smaller than $J = 1$ we cannot directly model known chiral magnets such as MnSi in our simulation. However, larger periodicities would require much larger grids and thus be computationally infeasible. Moreover, we let the external magnetic field point in the $\hat{\mathbf{z}}$ direction $\mathbf{B} = (0, 0, B)$ and only keep B as a free scalar parameter.

We can fix one of the parameters, in this case J, arbitrarily and then work in abstract lattice units. A computer can only deal with dimensionless numbers and in numerical simulations one often makes convenient, but arbitrary choices. It is then sometimes quite difficult to translate the results back to physically meaningful values.

The alert reader might have realized that so far our model does not contain any thermal energy. When talking about magnetism, temperature typically has a strong influence on the physical properties of various materials. However, thermal energy cannot be accounted for by an additional term in the Hamiltonian. Our intuition tells us that thermal energy is intrinsically of dynamic nature, thus cannot be captured sensibly in a single static spin configuration. Thermodynamic observables are statistical properties of an ever evolving underlying system. In fact we will account for temperature not in the Hamiltonian but via the Metropolis algorithm, which we describe in section 1.4, where the dynamic and statistical nature of thermodynamics will become apparent.

1.2.4 Summary

In this section we have set up our model, which we want to summarize briefly. We work on the lattice

$$\Sigma := \{1, \dots, N_x\} \times \{1, \dots, N_y\} \times \{1, \dots, N_z\} \subset \mathbb{N}^3 \subset \mathbb{R}^3 \tag{1.14}$$

and attach a spin $\mathbf{S_r} \in S^2$ to each lattice site $\mathbf{r} \in \Sigma$. This yields the configuration space

$$\Pi := \prod_{\mathbf{r}\in\Sigma} S^2 \,. \tag{1.15}$$

On Π we define the Hamiltonian $H + H'$ from (1.11) and (1.13) where we implement periodic (or partially open) boundary conditions. The parameters $J = 1$ and $K = \tan(2\pi/10)$ of the FM and DM interaction are fixed, which leaves us only with the magnetic field in the $\hat{\mathbf{z}}$ direction B as a free parameter. In the next section we will introduce the temperature as a second adjustable quantity. Instead of finding the energy minimizing configuration as stated in (1.12) at zero temperature, we now seek to evaluate prominent observables such as the magnetization or specific heat of the system for given values of B and T. As mentioned before, we cannot expect to compute such macroscopic properties from a single static configuration $\pi \in \Pi$ due to thermal effects. Before we go into statistical physics, let us present a first primer on general Monte Carlo methods.

1.3 General Monte Carlo methods

Now that we have clearly stated the model, let us start going into algorithmic techniques. Monte Carlo methods stand for a broad class of numerical algorithms that are based on some sort of repeated random sampling. The term was coined by the casinos in Monte Carlo, where gamblers bet on the outcome of random processes. It is hard to formulate a more specific feature that all Monte Carlo algorithms have in common. Most often, Monte Carlo techniques are used for optimization, integration or sampling probability distributions. Our specific problem contains a mixture of all three fields, arguably with the main focus on integration.

Before we describe the Metropolis algorithm in detail, we have to issue a warning. While Monte Carlo methods are generally easy to implement and are sometimes even applied blindly to virtually any problem, they are almost always hopelessly inferior to specialized techniques in terms of accuracy and efficiency. They should be the last resort after everything else has failed. We will reason about this in the following subsections. Let us start out with a general description of Monte Carlo integration and optimization.

The remainder of this chapter is based on a collection of different overview articles [8, 19, 20] as well as comprehensive books on Monte Carlo methods in statistical physics [13, 6].

1.3.1 Integration

Overview and comparison of integration methods

The oldest and most naive numerical integration methods make use of interpolating functions. They sample the integrand at a fixed finite number of base points in the domain and approximate the integrand by an interpolation that is easy to integrate. Consider for example the trapezoidal rule. It samples the integrand on a homogeneous grid and approximates it by a piecewise linear function between the grid vertices, see Figure 1.5. The integral of a piecewise linear function is easily computed by adding up the areas of the trapezoids underneath the curve. The larger the number of base points, i. e. the smaller the grid spacing, the more accurate is the result for sufficiently well behaved integrands. The important aspect here is that for a given integration rule and a given number of base points the choice of nodal points in the domain is completely deterministic.

More sophisticated routines refine the grid of base points iteratively in regions, where the error does not meet certain accuracy and precision goals. But even such *adaptive* methods have fixed rules how the new samples have to be chosen. Typically that determinism brings great advantage, because one can prove optimality results and certain error bounds for wise choices of base points.

In contrast, Monte Carlo techniques draw the samples randomly from some probability distribution. The general idea is illustrated by the one-dimensional example of a real-valued function f on the interval $[0,1] \subset \mathbb{R}$. From the definition of the expectation value we have

$$\int_0^1 f(x)\,dx = E_{\mathcal{U}_{[0,1]}}(f)\,, \tag{1.16}$$

where $E_{\mathcal{U}_{[0,1]}}$ is simply the expectation value over the uniform distribution $\mathcal{U}$ on the interval $[0,1]$. The expectation value can be approximated by averaging f over $N \in \mathbb{N}$ independent samples x_i of a $\mathcal{U}_{[0,1]}$ distributed random variable

$$E_{\mathcal{U}_{[0,1]}}(f) \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{1.17}$$

and thus

$$\int_0^1 f(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)\,. \tag{1.18}$$

Both paradigms, the trapezoidal rule as well as the Monte Carlo sampling, are illustrated for a simple one-dimensional function in the top row of Figure 1.5. In this one-dimensional scenario the Monte Carlo integration is absolutely not competitive. Even the crude trapezoidal rule will easily outperform it in terms of accuracy, precision and also efficiency for any given number of base points N. It turns out that the error of the trapezoidal rule scales like $1/N^2$. Piecewise quadratic approximation, called Simpon's rule or Kepler rule, already brings the error down to $\sim 1/N^4$. Hence these methods converge extremely fast for suitable integrands. Keep in mind that we are talking about the simplest rules imaginable which are hardly ever used in practice. State of the art integration methods are far more sophisticated with still better convergence properties for a certain classes of functions.

However, to exploit the full potential of specialized integration routines one still has to have some idea of what the integrand looks like. Periodic functions call for other routines than slightly singular functions with strongly localized features for instance. Furthermore, even adaptive rules require to at least crudely sample the whole integration domain. In high-dimensional integrals they easily fall victim to the curse of dimension. For example, imagine we want to compute a ten-dimensional integral. Even a coarse first grid of ten base points in each direction requires 10^{10} function evaluations. For a complex integrand that requires hundreds or thousands of floating point operations per evaluation, even that initial sampling exceeds the capacities of a typical privately owned computer. Once we try to refine the grid by say a factor of 2 in each dimension we get into real trouble, since the number of base points consequently increases by a factor of thousand, rendering most direct integration routines infeasible even for the fastest computers available. Let us contrast these insights with the workings and errors of Monte Carlo integration.

Error of Monte Carlo integration

First of all, the law of large numbers ensures that the approximation (1.17) converges for large N

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(x_i) = E_{\mathcal{U}_{[0,1]}}(f)\,. \tag{1.19}$$

However, it is important to know how fast it converges, i. e. how large the error is. Let us henceforth use I and $\hat{I}$ for the exact and approximate values of the integral

$$I := \int_0^1 f(x)\,dx \qquad \text{and} \qquad \hat{I} := \frac{1}{N}\sum_{i=1}^{N} f(x_i)\,. \tag{1.20}$$

We define the variance of f by

$$\sigma^2(f) = \int_0^1 (f(x) - I)^2\,dx\,. \tag{1.21}$$

To measure the error in $\hat{I}$ we consider the x_i independently identically distributed random variables with distribution $\mathcal{U}_{[0,1]}$. One can then show that the variance of the Monte Carlo estimate is

$$E_{x_1,\dots,x_N}\big((\hat{I} - I)^2\big) = \int_0^1 dx_1 \cdots \int_0^1 dx_N \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i) - I\right)^2 = \frac{\sigma^2(f)}{N}\,. \tag{1.22}$$

What this tells us is that the average error we make in the Monte Carlo integration is $\sigma(f)/\sqrt{N}$. It is important to realize that with truly random numbers, each separate run of the same Monte Carlo integration will generally yield different results. For any single one of them the error could be much larger than $\sigma(f)/\sqrt{N}$, but for infinitely many runs the average of all observed errors will always average to the standard deviation of f divided by $\sqrt{N}$. By increasing the number of samples we can only bound the probability of being far off. In stark contrast, interpolation methods like the trapezoidal rule actually allow for strict absolute upper bounds of the error for any function meeting certain differentiability criteria.

The single most important result of Monte Carlo integration is that the factor of $\sqrt{N}$ is independent of the dimension of the integration domain. This tells us that even in a 10^3 dimensional integration, where interpolation and adaptive methods would fail straightaway, we can be sure that the average error scales like $1/\sqrt{N}$, where N is the number of samples. This is the key justification for the use of Monte Carlo methods. Keep in mind, however, that a convergence rate of $1/\sqrt{N}$ is tantalizingly slow compared to what we are used from other numerical algorithms. For example, consider the integral

$$I := \int_0^1 e^{-x}\,dx\,. \tag{1.23}$$

Assume we want the error $\sigma(e^{-x})/\sqrt{N}$ to be smaller than $5 \cdot 10^{-4}$. We compute

$$\sigma(e^{-x}) = \sqrt{E_{\mathcal{U}_{[0,1]}}(e^{-2x}) - E_{\mathcal{U}_{[0,1]}}(e^{-x})^2} = 0.18098\ldots\ , \tag{1.24}$$

which leads to a minimal N of roughly $2 \cdot 10^5$ to achieve our accuracy goal. We need a couple hundred thousand function evaluations for a moderate error in a simple and well behaved integral. For comparison, the trapezoidal rule reaches the same error for about 15 base points, see the second row of Figure 1.5, where we show only 7 base points for illustration purposes.

Importance sampling

While the N dependence in the error of Monte Carlo integration is certainly more important, because it is the one we have full control over, it pays off to also elaborate on the standard deviation of the integrand $\sigma(f)$. Although the integrand is usually fixed by the problem, there are some tricks to improve the error by decreasing the variance of the integrand. Let us start from a generic integral

$$\int_{\Omega\subset\mathbb{R}^n} f(x)\, d\lambda^n(x)\ , \tag{1.25}$$

where λ^n is the ordinary Lebesgue measure on $\mathbb{R}^n$. Next, let us rewrite this as

$$\int_{\Omega\subset\mathbb{R}^n} f(x)\, d\lambda^n(x) = \int_{\Omega\subset\mathbb{R}^n} \frac{f(x)}{p(x)} p(x)\, d\lambda^n(x) = \int_{\Omega\subset\mathbb{R}^n} \frac{f(x)}{p(x)}\, d\mu(x) = E_p\left(\frac{f(x)}{p(x)}\right)\ , \tag{1.26}$$

where p is the probability density function of some measure μ on Ω with respect to the Lebesgue measure. The expectation value E_p is with respect to this measure. Since we have changed the integration measure, we can now approximate the integral by

$$\int_{\Omega\subset\mathbb{R}^n} f(x)\, d\lambda^n(x) \approx \frac{\lambda^n(\Omega)}{N} \sum_{i=1}^{N} \frac{f(x_i^p)}{p(x_i^p)}\ , \tag{1.27}$$

where the x_i^p are now independent samples from the probability distribution induced by p. The key point is that the error estimate is unaffected by the distribution we draw the samples from and is thus given by

$$\frac{\sigma(f/p)}{\sqrt{N}}\ . \tag{1.28}$$

While we cannot change the standard deviation of f, we can now try to minimize the variance of the ratio f/p by choosing p wisely. An optimal choice would mean $f/p =$ const. This shows that p should be large/small where f is large/small, which makes sense intuitively. We concentrate the samples in regions where the contributions to the integral are large. Regions where f is insignificantly small will only be sampled sparsely. Therefore we call this procedure *importance sampling.*

For this method to work in practice, one needs to be able to efficiently sample the distribution induced by p. In general this is a difficult task and a lot of research has gone into the development of good pseudo random number generators following arbitrary probability distributions. Moreover, Monte Carlo integration is typically used precisely when we do not know much about f, hence cannot choose p adequately upfront.

There are algorithms based on importance sampling, for example VEGAS, that update p in real time during the integration to concentrate on important regions. However, it turns out that many integrals that come up in physics, particularly in statistical physics as we will see later, already come in the form

$$\int_{\Omega\subset\mathbb{R}^n} f(x)p(x)\,d\lambda^n(x)\,, \tag{1.29}$$

where p is a probability density function of some measure μ on Ω. In this case we can directly rewrite the integral as

$$\int_{\Omega\subset\mathbb{R}^n} f(x)\,d\mu(x) \approx \frac{\lambda^n(\Omega)}{N}\sum_{i=1}^{N} f(x_i^p)\,. \tag{1.30}$$

Usually $\sigma(f)$ is smaller than $\sigma(f\cdot p)$ and as long as we can sample p efficiently, we can improve the error for fixed N.

Let us illustrate importance sampling with an example. On the right side of the second row of Figure 1.5 we plot the function e^{-x} and some uniformly distributed random sample points for ordinary Monte Carlo integration. The bulk of the area under the curve is on the left, while the long exponential tail can practically be neglected. However, many of our uniformly chosen samples fall into the right half, where the function is basically zero. At the same time we scarcely sample the important region. The rather large variation of e^{-x} leads to big errors as we have discovered in (1.24). Fortunately, with the right normalization e^{-x} is a probability density function that can be sampled efficiently on a computer. The integral $\int_0^1 e^{-x}\,dx$ is of the form (1.29) with

$$f(x) = \frac{e}{e-1} \qquad \text{and} \qquad p(x) = \frac{e^{1-x}}{e-1}\,. \tag{1.31}$$

In this case the Monte Carlo approximation becomes trivial

$$\frac{1}{N}\sum_{i=1}^{N} f(x_i^p) = \frac{1}{N}\sum_{i=1}^{N}\frac{e}{e-1} = \frac{e}{e-1}\,, \tag{1.32}$$

which is the exact value of the integral, see last row of Figure 1.5. In this simplified case the integrand itself is up to a normalization constant a probability density function on $[0,1]$. The function f that is left as an integrand is constant, thus has vanishing variance and a single sample is enough to get the error down to zero. Hence we already computed the integral exactly by finding the normalization constant for p. In real use cases the function f would not be constant and we would have to collect more than just one sample for a reasonable estimate.

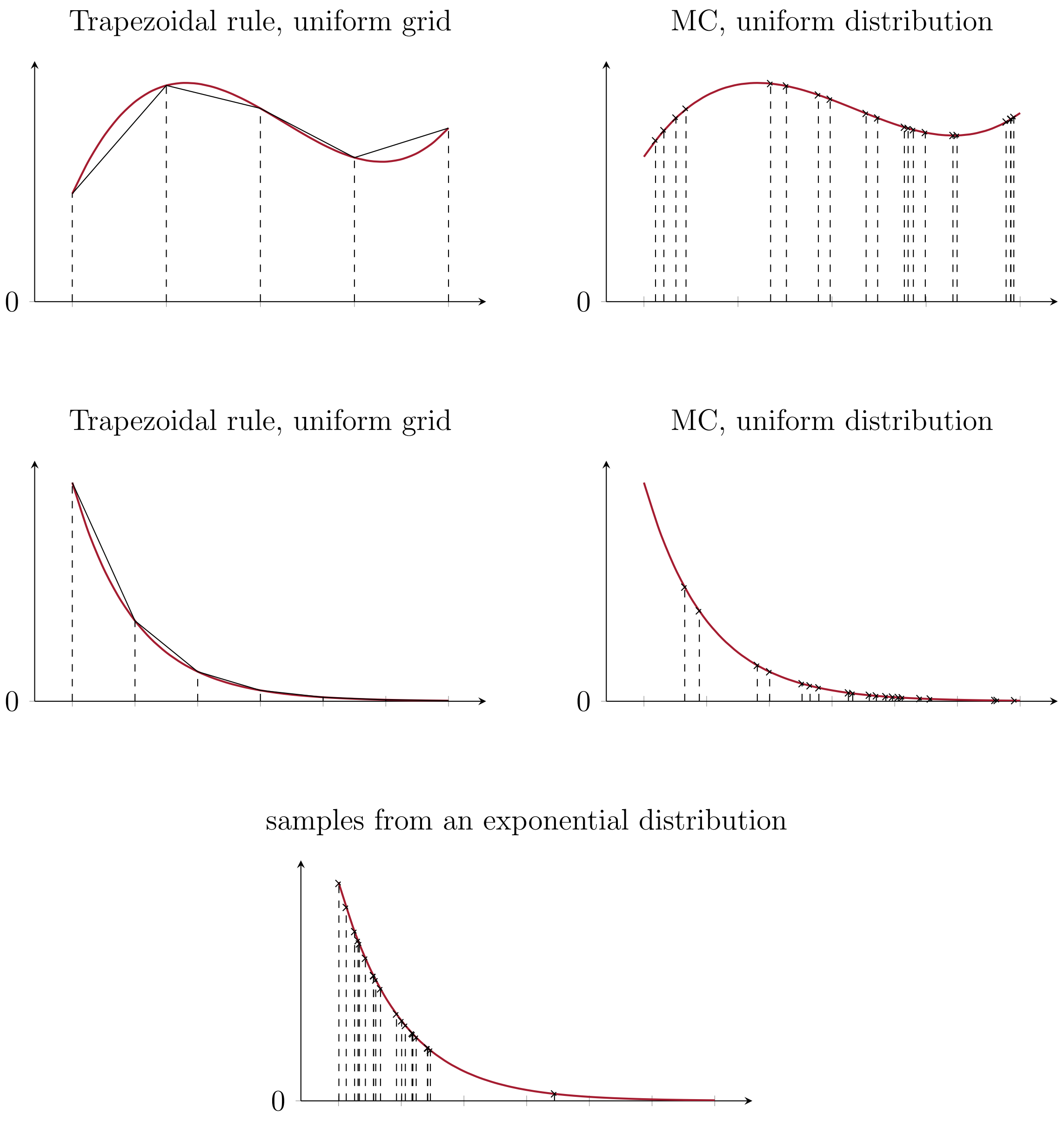

Figure 1.5: We show the basic principle of the trapezoidal rule in the left column and Monte Carlo integration in the right column. In the top row we consider some arbitrary function and draw from a uniform distribution in the Monte Carlo integration. In the second row we integrate the function e^{-x}. While sampling uniformly yields a big error due to the large variation of the exponential function, we can instead draw our samples according to an exponential distribution as shown in the bottom center plot.

Jackknife

While the theoretic prediction for the error in a Monte Carlo integration of the function f with N random samples is given by $\sigma(f)/\sqrt{N}$, we cannot compute this quantity in practice. Usually one does not have access to $\sigma(f)$. Instead, one typically employs a procedure called *jackknife* to estimate the error. Given the samples f_i for $i \in \{1, \ldots, N\}$, we compute the mean

$$\overline{f} := \frac{1}{N} \sum_{i=1}^{N} f_i \tag{1.33}$$

and define

$$\overline{f}_i := \frac{1}{N-1} \sum_{j=1, j \neq i}^{N} f_j \, , \tag{1.34}$$

for all $i \in \{1, \ldots, N\}$. The quantity $\overline{f}_i$ is the mean of all samples except the i-th one. An estimate of the variance of the original estimator is then given by

$$\sigma_{\mathrm{jk}}^2(f) = \frac{N-1}{N} \sum_{i=1}^{N} (\overline{f}_i - \overline{f})^2 \, . \tag{1.35}$$

More specifically, this is the so called *leave one out* jackknife method. There are many more sophisticated resampling methods like *bootstrapping* [5]. The jackknife is a comparatively rough tool [4]. However, because it is easy to use and implement, it is still widely used for error estimates in Monte Carlo methods.

1.3.2 Optimization

Overview

In a typical optimization problem we seek to find the minimum of a real-valued function f over $\Omega \subset \mathbb{R}^n$. Depending on f and Ω this can be anything from trivial to practically infeasible. As a consequence there are numerous possible classification schemes. Let us briefly mention two important properties. First, we distinguish optimization techniques by whether they use Hessians, gradients or only function values. Apparently the degree of differentiability of the objective function f plays a crucial role. If f is sufficiently smooth and we have access to gradients (and Hessians) either analytically or numerically, we can build a first order (second order) approximation to the function at a given point $x \in \Omega$. This leads to so called local iterative methods, where we have to specify a start point, approximate the function around this point, proceed a certain distance in the downward direction of the model and repeat the procedure from there. Numerous gradient-descent and (Quasi-)Newton methods fall into that category of iterative methods.

The convergence rate and also the complexity depend on how much information we have about the derivatives of the function, but most iterative methods are guaranteed to converge to a minimum for sufficiently well behaved objectives and wisely

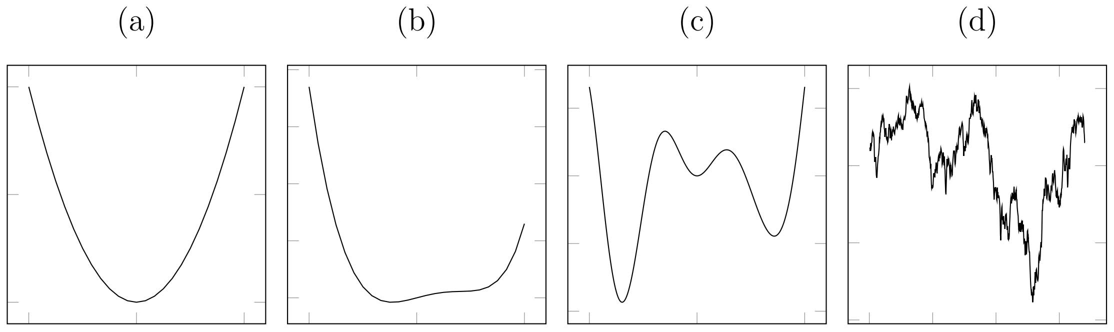

Figure 1.6: Convex functions like the one shown in (a) can be minimized extremely efficiently even over high-dimensional domains. While the second example (b) is not convex anymore, it is smooth and has a unique minimum. Therefore local iterative methods will do an excellent job and converge quickly to the minimum. Function (c) is still smooth, but has several minima. Depending on where we position the starting point, iterative gradient methods might get stuck in one of the two local minima on the right and not find the global minimum on the left. The fourth function (d) is not even once differentiable and has numerous minima. We can only use heuristics and hope to get close to the true minimum.

chosen parameters. The major trouble with local methods is precisely that they find *a* minimum, but potentially not *the* minimum. If a function has several local minima, depending on the start point, we might end up in any of them and might overlook a global minimum somewhere else hidden to our descent method by a barrier. In this case we could either still employ iterative methods with multiple starting points and hope that one of them guides us to the global minimum, or switch to heuristics.

Heuristics are not mathematically guaranteed to find the solution. In fact, little can be proven about how well heuristics will perform in general. However, they often turn out to be useful, especially if we cannot apply a finitely terminating or at least provably convergent iterative algorithm, see Figure 1.6. Interestingly, many heuristics have been inspired by natural processes the genetic algorithms, bee or ant colony optimization, evolutionary algorithms or simulated annealing. We will come back to simulated annealing in a slightly different setting in section 1.5. Although we will not try to minimize the energy of our model in the strict sense, simulated annealing will enable us to reach configurations we might have missed otherwise. Therefore it is worthwhile to discuss Monte Carlo optimization for our application.

The domain of our spin lattice optimization problem is a $2N_xN_yN_z$-dimensional space. Even for relatively small lattices optimization in such a high-dimensional space is generically hard. Unfortunately, it appears completely hopeless to efficiently implement the gradient or even Hessian of the Hamiltonian (1.11). Moreover, it is hard to develop intuition about how the energy landscape looks like, not only because we can barely visualize anything in more than three dimensions. It appears certain, however, that it will include a vast number of local minima with rough hills and valleys. It is likely to be a higher-dimensional equivalent of plot (d) in Figure 1.6. All of this is bad news for efficient iterative optimization methods and we have to fall back to heuristics.

Our only option is to sample the search space preferably in a smart and structured way and hope to capture the minimum within reasonable computation time. This is precisely what simulated annealing is about. It merely consist of a sampling strategy like the Metropolis algorithm that involves some randomness and tries to gradually settle in states of lower and lower energy. We will discover the Metropolis algorithm in the next section. As was the case for integration, this is horrifically inefficient. By now we understand why Monte Carlo techniques should only be considered once everything else failed or proved intractable.

1.4 The Metropolis algorithm

The name *simulated annealing* already hints towards thermodynamic processes, which we identified to be a key feature of our system, but were not yet able to bake it into the model. Clearly any real system at finite temperature is not completely static, i. e. stuck in a fixed state $\pi \in \Pi$, but it is constantly subject to thermal fluctuations. The Metropolis algorithm, which is the core part of the simulated annealing algorithm we will describe later, is designed to mimic real world systems at finite temperature and will take care of thermal fluctuations. That implies that we will not seek to find a single best state $\pi \in \Pi$, but only consider thermal averages of observables over many configurations. To fully understand and appreciate the Metropolis algorithm and its ties to physics, we need some basic concepts from statistical physics.

1.4.1 Statistical physics in a nutshell

In macroscopic many body systems such as gases or condensed matter at finite temperature, we are often not interested in every single detail of that system at a given moment in time. For example, we do not want to know the position and velocity as well as rotational and vibrational modes of each single molecule in a piston. Sensible observables are typically some sort of macroscopic averages like the volume, temperature or pressure of the gas. For a solid we might be interested in its specific heat, conductivity or susceptibility rather than the joint many body wave function of all electrons. Statistical physics provides a sound and rigorous way to define those averages.

Instead of working with the complete microstates, in our case the elements of Π, directly, we only talk about the probability $P_T(\pi)$ that the system is in a certain state $\pi \in \Pi$ at temperature T. To this end we imagine a large ensemble of exact replicas of the system and observe them all at the same time figuratively counting how often each microstate occurs. Starting from the assumption that in a closed system in equilibrium each microstate π is equally likely, we can derive the so called *canonical ensemble*. It states that the probability of a non-closed system in equilibrium at finite temperature T, e. g. via contact to a large heat bath, to be found in state π is

$$P_T(\pi) = \frac{\exp\left(-\frac{H(\pi)}{k_B T}\right)}{\sum_{\pi\in\Pi}\exp\left(-\frac{H(\pi)}{k_B T}\right)}\,. \tag{1.36}$$

This is also referred to as the *equilibrium Boltzmann distribution* with k_B being the Boltzmann constant. For a finite configuration space $P_T(\pi)$ are discrete probabilities that add up to one by construction. The normalizing factor

$$Z := \sum_{\pi\in\Pi}\exp(-\beta H(\pi)) \tag{1.37}$$

in (1.36) is called *partition function.* Here we introduced the common shortcut $\beta := (k_B T)^{-1}$. In our case, Π consists of infinitely many states and the summation in the partition function has to be replaced by an integral

$$Z := \int_\Pi \exp(-\beta H(\pi))\, d\mu_\Pi(\pi)\,, \tag{1.38}$$

where the measure μ_Π depends on the configuration space. In our system it is a product measure of the standard measure on S^2, but the mathematical details are not important for the further treatment in this report. Note that we will always consider the equilibrium Boltzmann distribution for a fixed temperature T and sometimes drop the subscript T and simply denote it by P.

The equilibrium Boltzmann distribution (1.36) tells us that configurations with lower energy are exponentially more likely than configurations with higher energy. Furthermore, this effect is most pronounced at low temperatures. If T is large, states with higher energy still occur with a decent probability compared to the minimizing state. On the other hand, for $T \to 0$ basically only the state with minimal energy has a non-zero probability. Both facts match our intuition. At high temperatures, thermal fluctuations are large and the system can visit many different microstates. At zero temperature we expect the system to settle in the ground state.

In a probabilistic description, the energy or magnetization of one single specific configuration looses its significance. Instead we measure expectation values of observables. For an observable $\mathcal{O} : \Pi \to \mathbb{R}$ we compute the so called *thermal average*

$$\langle\mathcal{O}\rangle := \frac{1}{Z}\sum_{\pi\in\Pi}\mathcal{O}(\pi)\exp(-\beta H(\pi))\,, \tag{1.39}$$

hence we average over all states weighted by their probability at a given temperature. For a continuous state space this becomes

$$\langle\mathcal{O}\rangle := \frac{1}{Z}\int_\Pi \mathcal{O}(\pi)\exp(-\beta H(\pi))\, d\mu_\Pi(\pi)\,. \tag{1.40}$$

In our case the observables we are primarily interested in are the energy $\langle H\rangle$ and the three components of the magnetization $\langle\mathbf{M}\rangle = \langle\sum_{\mathbf{r}\in\Sigma}\mathbf{S}_\mathbf{r}/|\Sigma|\rangle$.

One question immediately poses itself: How could we possibly compute the high-dimensional integral in (1.40) for our specific situation? Even for finite configuration spaces the sum in (1.39) would contain extraordinarily many terms. If each lattice site could only be in two states, up or down, already on a small 10^3 lattice we would have to sum up $2^{1000} \approx 10^{300}$ configurations. This is far beyond the possibilities of any machine imaginable today. The large dimensionality of the problem renders it a good fit for Monte Carlo methods. Indeed (1.40) bears striking resemblance to (1.29). The standard measure on S^2 can be derived from the Lebesgue measure λ^3 on $\mathbb{R}^3$ and (1.36) by construction can be interpreted as a probability density function p on Π. It assigns almost no weight to configurations with high energy. As a consequence they will not contribute much to the integral. The thermal average obviously calls for importance sampling

$$\langle \mathcal{O} \rangle \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}(\pi_i) \,, \tag{1.41}$$

where π_i are independent samples drawn from the Boltzmann distribution P_T. Finally we see how Monte Carlo integration enters the picture.

1.4.2 Generating random configurations

The challenge is to generate independent random samples following the Boltzmann distribution (1.36). Generating a uniformly random configuration is relatively easy, although there is one major pitfall one has to look out for. The strategy is of course to pick a uniformly random spin at each vertex of the grid. We assume that the programming language of choice has efficient built in routines for pseudo random numbers following common distributions such as the uniform, normal or exponential distribution.

Some programmers claim to generate uniformly random elements of S^2 in the following way. They generate three random numbers $x, y, z \sim \mathcal{U}_{[-1,1]}$ and then normalize the resulting vector $(x, y, z)/\sqrt{x^2 + y^2 + z^2}$. While (x, y, z) in this case is uniformly distributed on the cube $[-1, 1]^3$, by simply projecting all points onto the sphere $S^2 \subset [-1, 1]^2$, the points that were originally generated outside the sphere, i. e. in $[-1, 1]^3 \setminus B^3$, are unevenly distributed. Here the unit ball B^3 is the interior of S^2. The corners of the cube host more points than the areas where the sphere actually touches the faces of the cube and thus we expect more points in a small neighborhood around $(1, 1, 1)/\sqrt{3}$ than around $(0, 0, 1)$.

There are several ways to mitigate this effect. One is to discard all points (x, y, z) with length greater than one and redraw new points in the cube until they fall inside the sphere. We do not recommend this method, because of its inefficiency. In three dimensions we have $\lambda^3([-1, 1]^3) = 8$, $\lambda^3(B^3) = 4\pi/3$ and thus

$$\frac{\lambda^3([-1, 1]^3 \setminus B^3)}{\lambda^3([-1, 1]^3)} \approx 0.476 \,, \tag{1.42}$$

i. e. one would have to discard almost every second candidate. The ratio becomes ever larger in higher dimensions. A slightly better way is make use of the fact that S^2 is a two-dimensional manifold, thus can be parameterized by only two real numbers. One can independently draw $x, y \sim \mathcal{U}_{[-1,1]}$, discard and redraw them if $x^2 + y^2 > 1$ and otherwise return

$$\begin{pmatrix} 2x\sqrt{1-x^2-y^2} \\ 2y\sqrt{1-x^2-y^2} \\ 1-2(x^2+y^2) \end{pmatrix}, \tag{1.43}$$

which results in a uniform distribution on the sphere. This is a slight improvement, because we only discard the fraction

$$\frac{\lambda^2([-1,1]^2 \setminus B^2)}{\lambda^2([-1,1]^2)} \approx 0.215\,, \tag{1.44}$$

but still not optimal. Interestingly, the original method $(x, y, z)/\|(x, y, z)\|$ works if we draw x, y, z independently from a normal distribution with zero mean. The downside of this approach is that the generation of normally distributed random numbers is slightly slower than uniformly distributed ones.

Finally, let us describe the method we used. In the simulation itself we prefer the redundant representation as three-dimensional vectors, because it simplifies the computation of dot and cross products as well as projections onto coordinate planes. Nevertheless for the generation of uniformly random spins on S^2 it is advantageous to work in spherical coordinates $(\phi, \theta) \in [0, 2\pi) \times [0, \pi)$. The next pitfall is already waiting for us. Uniformly chosen ϕ and θ will also *not* result in a uniform distribution on S^2. The density of points would be larger near the north and south pole, i. e. for $\theta \approx 0$ and $\theta \approx \pi$, than near the equator $\theta \approx \pi/2$. The correct procedure is the following:

1. Draw $z \sim \mathcal{U}_{[-1,1]}$ and $\phi \sim \mathcal{U}_{[0,2\pi)}$.
2. Compute $r_{xy} = \sqrt{1 - z^2}$.
3. The vector $(r_{xy}\cos(\phi), r_{xy}\sin(\phi), z)$ is a sample from $\mathcal{U}_{S^2}$.

Here we make use of the fact that the z component, i. e. $\cos(\theta)$, of uniformly distributed points on S^2 is uniformly distributed on $[-1, 1]$. The azimuthal angle ϕ is also uniformly distributed on $[0, 2\pi)$. The x and y components are then computed such that the resulting vector has unit length. Note that for all intervals above it does not matter whether we include the end points or not, because they constitute a null set with respect to the Lebesgue measure. In our benchmarks the latter method was almost a factor two faster than all other presented options.

By now we have at least clarified how to generate a uniformly random configuration $\pi \sim \mathcal{U}_\Pi$, which we will need later. The Metropolis algorithm is a recipe on how to generate configurations drawn from the equilibrium Boltzmann distribution if we have access to uniformly random samples. The mathematical tool for this recipe to succeed is the theory of Markov chains.

1.4.3 Markov chains

The crucial idea is to not create each random sample of the equilibrium Boltzmann distribution completely from scratch, but to slowly work towards more likely configurations. If we simply pick a uniform random configuration, we can only evaluate its energy once it has been generated. It would take a huge number of samples to figure out which energies are relatively high and therefore unimportant, or relatively low and therefore contribute a lot. Even if we knew the energy range up front, we could not generate configurations in a certain energy range from scratch, but only decide after we have generated them.

The cure for this problem are so called *Markov chains.* In a Markov chain we start at any configuration π_0 and modify it to transition to the next one π_1. Then we continue from there, which yields the following sequence or chain

$$\pi_0 \xrightarrow{\phi} \pi_1 \xrightarrow{\phi} \pi_2 \xrightarrow{\phi} \cdots \tag{1.45}$$

The transition procedure, which we denoted by ϕ, has to fulfill two requirements. At some point we want configuration π to occur in the chain with probability $P_T(\pi)$, i. e. it has to actually sample the equilibrium Boltzmann distribution. The second requirement stems from the fact that for Monte Carlo integration to work we need independent samples. Since π_i now clearly depends on π_j for $j < i$, we have to make sure that ϕ modifies each state enough to render π_{i+1} practically independent of π_i. We can characterize ϕ by the probability with which it transforms state π into state π'. For each pair of states $\pi, \pi' \in \Pi$ we denote the transition probability by $\mathcal{T}(\pi \to \pi')$. Proper transition probabilities have to fulfill

$$\sum_{\pi' \in \Pi} \mathcal{T}(\pi' \to \pi) = 1 \qquad \text{or} \qquad \int_\Pi \mathcal{T}(\pi' \to \pi)\, d\mu_\Pi(\pi') = 1 \tag{1.46}$$

meaning that starting from π the system has to go to *some* other state in the configuration space with probability 1. Furthermore we require $\mathcal{T}(\pi \to \pi') \geq 0$ for all $\pi, \pi' \in \Pi$.

State transitions of the three-dimensional spin lattice

For illustration purposes let us explain what the transition process ϕ will look like in our specific example. The simplest way to modify a state $\pi \in \Pi$ is to pick one of the spins $\mathbf{S_r} \in S^2$ and change it. Since we have no indicators yet as to which spin we should change and how to modify it to bring the new state close to a sample from the Boltzmann distribution, we follow the Monte Carlo scheme and select the position $\mathbf{r} \in \Sigma$ as well as the new spin $\mathbf{S}'_\mathbf{r} \in S^2$ uniformly at random, respectively. Obviously, flipping one spin out of many does not yet result in an independent state. In fact the correlation between the two configurations would be almost one. Thus ϕ has to consist of a little more than just changing one spin. Henceforth let us call the action of changing one spin an *update.*

Instead of a single update, we change many spins and we change them often. Successively updating $|\Sigma| = N_x N_y N_z$ random vertices is called one *lattice sweep* or just *sweep*. The transition ϕ from one configuration to the next will consist of $N_{\text{sweep}} \in \mathbb{N}$ many lattice sweeps. Thus, between two successive configuration each lattice site has been updated N_{sweep} many times on average, see Figure 1.7. The exact number N_{sweep} has to be determined by careful analysis such that π_{i+1} is independent of π_i. We will come to this delicate business in section 1.6. In our simulation we will measure time by the number of lattice sweeps. This measure is called Monte Carlo time and one sweep is referred to as a Monte Carlo step. One has to be careful not to confuse consecutive configurations in the Markov chain, which are separated by multiple sweeps and intermediate configurations that are separated by a single sweep. We will hardly ever consider different configurations within one sweep, i. e. separated by less than $|\Sigma|$ updates.

Apparently, if we simply keep updating spins in that fashion, each transition will have the same probability and we only generate one uniformly random sample of Π after the other instead of drawing from the Boltzmann distribution. By carefully selecting which updates actually are allowed to happen and which we reject without any changes to the current state, we can tune the transition probabilities to our liking.

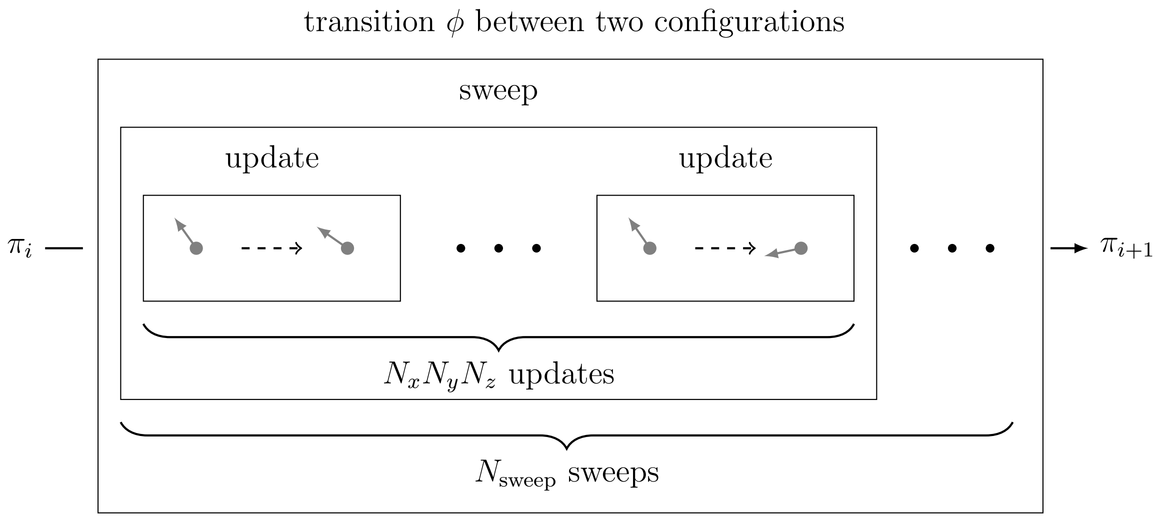
transition ϕ between two configurations
sweep
update
update
π_i
π_{i+1}
$N_xN_yN_z$ updates
N_{sweep} sweeps

Figure 1.7: To render state π_{i+1} independent of state π, we perform N_{sweep} sweeps, each consisting of $|\Sigma| = N_xN_yN_z$ updates.

Transition probabilities

Let $P_i(\pi)$ be the probability that at the i-th step of the chain we are in state π. This probability can be computed recursively. What is the probability that at step $i+1$ we are in state π? There are basically two ways we could end up there. Either we were in a different state π' at the i-th step and hit the transition probability $\mathcal{T}(\pi' \to \pi)$, or we have already been in π and stayed there, which happens with probability $\mathcal{T}(\pi \to \pi)$.

Those can be combined into

$$P_{i+1}(\pi) = \sum_{\pi' \in \Pi} P_i(\pi')\mathcal{T}(\pi' \to \pi) \qquad \text{or} \qquad P_{i+1}(\pi) = \int_\Pi P_i(\pi')\mathcal{T}(\pi' \to \pi)\, d\mu_\Pi(\pi')\,. \tag{1.47}$$

Let us refactor this expression, using only the summation notation, because it is simpler and can trivially be transformed into the continuous version. With (1.46), we find

$$\begin{aligned} P_{i+1}(\pi) &= \sum_{\pi' \in \Pi} P_i(\pi')\mathcal{T}(\pi' \to \pi) + P_i(\pi)\Big(1 - \sum_{\pi' \in \Pi} \mathcal{T}(\pi \to \pi')\Big) \\ &= P_i(\pi) + \sum_{\pi' \in \Pi} \left(P_i(\pi')\mathcal{T}(\pi' \to \pi) - P_i(\pi)\mathcal{T}(\pi \to \pi')\right). \end{aligned} \tag{1.48}$$

In this form we can interpret the first term of the sum as all the ways we can enter state π at step $i+1$ and the second term as all the ways we can leave state π in step $i+1$. This form will prove helpful in a moment.

Now we aim at tuning the transition probabilities such that $\lim_{i\to\infty} P_i = P$, i. e. the probabilities in (1.47) converge to a stationary distribution that agrees with the Boltzmann distribution (1.36). In a stationary distribution, the probability of entering a certain state has to be equal to the probability of leaving that state. If we entered a state more often than we left it or vice versa, the distribution would clearly not be stationary. Mathematically this signifies that we can tune the transition probabilities such that the terms of the sum in (1.48) cancel each other out. In the limit $i \to \infty$, where $P_i(\pi) = P(\pi)$ for all states $\pi \in \Pi$, we thus find

$$P(\pi')\mathcal{T}(\pi' \to \pi) = P(\pi)\mathcal{T}(\pi \to \pi')\,. \tag{1.49}$$

This condition is called *detailed balance* and is a key feature of Markov chains. We can interpret it as reversibility of the transition process ϕ. Going from π to π' is equally likely as the other way round.

For completeness, let us mention that the second key requirement for Markov chains to work properly is *ergodicity*. Ergodicity says that each state must be reachable from every other state in a finite number of steps. We are not going to elaborate on the theoretical importance of this property, but only point out that ergodicity is clearly fulfilled in our specific model, because we have to change a maximum of $N_xN_yN_z$ spins and each possible spin flip has non-zero probability at finite temperature. One can show that if detailed balance and ergodicity are fulfilled, the Markov chain is guaranteed to converge to a stationary distribution.

The Metropolis algorithm

The ratio of the transition probabilities in the stationary distribution reveal another interesting aspect

$$\frac{\mathcal{T}(\pi \to \pi')}{\mathcal{T}(\pi' \to \pi)} = \frac{P(\pi')}{P(\pi)} = e^{-\beta(H(\pi')-H(\pi))} =: e^{-\beta\Delta H}\,. \tag{1.50}$$

Thus the transition probabilities depend only on the energy difference of the two states in question. In particular there is no memory of previous states. At each step we can decide the transition probability purely from the current and the proposed successor configuration. This property is our main guidance in engineering the transition probabilities. One rather obvious choice is given by

$$\mathcal{T}(\pi \to \pi') = \min(1, e^{-\beta(H(\pi')-H(\pi))}) \,. \tag{1.51}$$

Those are the so called *Metropolis* (or *Metropolis-Hastings*) transition probabilities. The whole Markov chain Monte Carlo integration for thermodynamic systems like ours using the Metropolis transition probabilities is simply called *Metropolis algorithm* or *Metropolis-Hastings algorithm.* There are also other plausible choices for the transition probabilities, for example the so called *Heatbath*, *Glauber*, or *overrelaxation* algorithms, but we will only discuss and implement the Metropolis algorithm.

Now that we know the transition probabilities, we can bake them into our state transition procedure ϕ. Recall that ϕ consists of N_{sweep} lattice sweeps, each of which performs $|\Sigma|$ random updates of single spins. Each time we have selected a spin $\mathbf{S_r}$ from the current configuration π for an update, we will first generate its potential replacement $\mathbf{S}'_{\mathbf{r}} \sim \mathcal{U}_{S^2}$, which would lead to state π'. But now we only actually perform the update, if

$$\mathcal{T}(\pi \to \pi') \geq p, \quad \text{where } p \text{ is a sample from } \mathcal{U}_{[0,1)} \,. \tag{1.52}$$

Note that this is the only time we consider configurations that are separated by just one update. We do not accept or reject full lattice sweeps or even applications of ϕ but every single update within every sweep. Thereby we abide to the desired transition probabilities in each update, thus in each sweep and eventually also in each complete transition ϕ.

Thermalization and the start configuration

Theoretically a Markov chain goes on forever, traversing the configuration space, each time choosing the successor state at random according to the transition probabilities. Given detailed balance and ergodicity, after sufficiently many steps, we have converged to a stationary distribution and each new state can be considered an independent sample from the equilibrium Boltzmann distribution. Those initial steps before the distribution becomes stationary are called *thermalization.* The close ties to thermodynamics are obvious, where a system can only be treated statistically, if it is in thermal equilibrium, i. e. after it has thermalized. Hence, before we can actually use the states π_i as independent samples of the Boltzmann distribution to estimate thermal averages via (1.41), we have to perform N_{therm} lattice sweeps to equilibrate the system. Henceforth the time t is measured by the number of lattice sweeps we have performed. While detailed balance and ergodicity guarantee that there is some finite $N_{\text{therm}} \in \mathbb{N}$, the precise value depends on a number of factors.

First of all it depends on the start configuration. In lattice models like ours the two most commonly used options are a *hot start* or a *cold start.* In a hot start we initialize the first state uniformly at random $\pi_0 \sim \mathcal{U}_\Pi$. Practically that means that we initialize each spin $\mathbf{S}_\mathbf{r}$ for $\mathbf{r} \in \Sigma$ independently uniformly at random $\mathbf{S}_\mathbf{r} \sim \mathcal{U}_{S^2}$ as discussed in subsection 1.4.2. It is called hot start, because in the limit $T \to \infty$ the Boltzmann distribution becomes a uniform distribution and each state is equally likely. Hence at infinite temperature a hot start already follows the Boltzmann distribution and the system is in thermal equilibrium from the very beginning. In a cold start all spins are chosen in the same direction, possibly parallel to the external magnetic field, which corresponds to a global minimum of the Hamiltonian including only direct exchange and the Zeeman energy at zero temperature.

Secondly, the thermalization time varies with temperature. We have already seen that at high temperatures almost all transitions have a decent probability and we can explore many states in a short time, guiding the system into equilibrium faster. At very low temperatures the Boltzmann distribution is highly localized, peaked at low energy states and we will have to reject many proposed successors before we discover the probable configurations. Generally this yields a long thermalization time for low temperatures. We will experience this tendency quite forcefully in chapter 2, when we analyze our specific model.

Thirdly, the system size plays a crucial role. The equilibration time unsurprisingly grows directly with the system size. This can be understood easily simply from the fact that in a large system it takes longer for all parts of it to communicate with each other and find a joint equilibrium state. We will discuss in the next section, how one can detect when the system is fully equilibrated. From here on we will consider π_0 to be the initial configuration and π_1 to be the first configuration after thermalization, i. e. the first one that can actually be used to compute thermal averages. Afterwards all following configurations are separated by N_sweep sweeps.

1.4.4 Summary and pseudocode

The ultimate goal of the Metropolis algorithm is to estimate observables such as the energy or the magnetization at fixed temperature T and external field B. Therefore it has to output the collection of desired observables, which we denote generically by $\mathcal{O}$, for N independent samples of the Boltzmann distribution. Let us now put all the pieces together and lay out a pseudo code for the Metropolis algorithm at fixed temperature T and external field B in Algorithm 1. The full metropolis algorithm can be depicted schematically as

$$\begin{tikzcd}
\text{initialize} \arrow[d] & \text{save } \mathcal{O}(\pi_1) & \text{save } \mathcal{O}(\pi_2) & & \text{save } \mathcal{O}(\pi_N) \\
\pi_0 \arrow[r, "N_\text{therm}", "\text{sweeps}"'] & \pi_1 \arrow[u] \arrow[r, "N_\text{sweep}", "\text{sweeps}"'] & \pi_2 \arrow[u] \arrow[r, "N_\text{sweep}", "\text{sweeps}"'] & \cdots \arrow[r, "N_\text{sweep}", "\text{sweeps}"'] & \pi_N \arrow[u] .
\end{tikzcd} \tag{1.53}$$

The starting configurations π_0 can technically be chosen as one pleases, we always use a hot start $\pi_0 \sim \mathcal{U}_\Pi$. The initial state π_0 is not used in computing thermal averages.

We have to choose N_{therm} large enough for the system to equilibrate, i. e. reach a stationary distribution, and N_{sweep} large enough to render subsequent configurations of the Markov chain uncorrelated. The number of configurations N determines the error $\sim 1/\sqrt{N}$ of the final estimate

$$\langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}(\pi_i) \,. \tag{1.54}$$

In practice the error estimate for this value $\langle \mathcal{O} \rangle$ is computed by the jackknife method as described earlier in subsection 1.3.1.

Algorithm 1 Metropolis algorithm

Require: Initialize the spin lattice, e. g. with a hot start $\pi_0 \sim \mathcal{U}_\Pi$.

```
 1: procedure METROPOLIS($\pi_0$, $T$, $B$)                    ▷ $T, B$ are fixed
 2:     for $i$ from 1 to $N_{\text{therm}}$ do                 ▷ thermalization
 3:         SWEEP($\pi_0$)
 4:     end for
 5:     set $\pi_1 := \pi_0$
 6:     save $\mathcal{O}(\pi_1)$
 7:     for $i$ from 1 to $N - 1$ do
 8:         set $\pi_{i+1} :=$ NEXT_CONFIG($\pi_i$)
 9:         save $\mathcal{O}(\pi_{i+1})$
10:     end for
11: end procedure

12: function NEXT_CONFIG($\pi$)
13:     for $j$ from 1 to $N_{\text{sweep}}$ do
14:         SWEEP($\pi$)
15:     end for
16:     return $\pi$
17: end function

18: procedure SWEEP($\pi$)                                    ▷ SWEEP modifies $\pi$
19:     for $i$ from 1 to $N_x N_y N_z$ do
20:         draw $\mathbf{r} \sim \mathcal{U}_\Sigma$
21:         draw $\mathbf{S}' \sim \mathcal{U}_{S^2}$
22:         if $\min\big(1, \exp(-\beta \Delta H)\big) > \text{rand}(0,1)$ then
23:             replace $\mathbf{S}_\mathbf{r} \in \pi$ by $\mathbf{S}'_\mathbf{r}$
24:         end if
25:     end for
26: end procedure
```

1.5 Simulated annealing

In section 1.2 we formulated our initial goal as an optimization problem. However, in everything that followed the general introduction of Monte Carlo methods in section 1.3 was concerned with integration only. We discovered that a truly thermodynamic treatment does not allow us to distinguish a single energetically favored state, but necessitates in thermal averages which we approximate by Monte Carlo integration using importance sampling. Proper samples are provided by a Markov chain with Metropolis transition probabilities. Hence, we can now observe thermodynamic quantities such as the energy or magnetization of the spin lattice in equilibrium at fixed temperature T and magnetization B. A certain notion of minimizing the energy is implicit, insofar as we are always more likely to accept transitions towards lower energy states. While we are now able to scan the whole phase space spanned by B and T and partition it into regions with similar properties, we cannot dynamically observe phase transitions.

Imagine a rough energy landscape similar to the one depicted in plot (d) of Figure 1.6. If we fix T and B from the very beginning, even the Markov chain might get stuck in a certain region of the configuration space or is extremely unlikely to find specific parts of it. In physical systems such rare configurations that are hard to reach do indeed emerge. If we start at a high temperature, however, the Markov chain will definitely sample most of the configuration space with a decent probability. If we then slowly decrease the temperature, it has a substantially better chance of annealing into such an interesting portion of Π. This is the whole idea behind simulated annealing. Instead of fixing T and B once and for all, we might start at at fixed point in the phase space, wait until the system has thermalized and compute N configurations of the Markov chain for the starting T and B like before. But instead of stopping right there and initializing a whole new run, one changes T or B by a tiny amount, discards some sweeps for thermalization and records another N configurations of the Markov chain. It is essentially a wrapper around the Metropolis algorithm that keeps building up the Markov chain with intermediate changes of the parameters. This method allows us to trace out paths within the phase space. It can be used to observe phase transitions or reach configurations that are hard to find by fixing the position in phase space from the very beginning.

For completeness let us explain why simulated annealing was introduced as an optimization method. Imagine our objective was actually to minimize the Hamiltonian. While there are no iterative or local methods to minimize it directly, we have found that the Metropolis algorithm efficiently samples the configuration space with a favor for low energy states, especially for low temperatures. Thus if we start at a high temperature and slowly decrease it, the Markov chain will tend to concentrate around lower and lower energies. If we are lucky and the energy landscape is not too rough, there is a good chance it will eventually settle in the absolute minimum for $T \to 0$. Apparently we need to *anneal* the system, because starting with $T = 0$, the Metropolis algorithm will never accept any changes towards higher energies at all and we are bound to get stuck in the nearest local minimum. Moreover, we cannot bridge or tunnel large barriers, because any single update can only change the energy by a fixed amount.

1.6 Analysis and statistics

In the previous sections we discussed the Metropolis algorithm and provided a pseudocode which should make it fairly easy to implement. However, we issued quite a few warnings without yet giving specific instructions on how to stay on the safe side of things. How should one choose the thermalization time N_{therm} and how do we know whether the system is in equilibrium? How large does N_{sweep} have to be such that consecutive configurations of the Markov chain can be considered independent? In this section we try to answer these questions or at least provide some practical guidelines.

1.6.1 Thermalization

Equilibration is a sensitive and often underestimated topic. It is by far not sufficient to let "a few sweeps" go by after a hot start, even at high temperatures. In the literature one can find guidelines that suggest that the thermalization time should be at least $15-20\%$ of the total runtime. The total runtime, as always measured in Monte Carlo time, i. e. the number of lattice sweeps, is given by $N_{\text{therm}} + N \cdot N_{\text{sweep}}$. Such estimates, while helpful for getting an idea of the order of magnitude, often lure scientists into blindly allocating 20% of the overall runtime for equilibration and feel safe. This is plain wrong and indefensible. Despite all the factors that go into the thermalization time, the number of sweeps in between configurations and the overall number of configurations one chooses to compute are not among them. How could one expect the thermalization time to decrease only because we stop the Markov chain after a few configurations? Those are just the most obvious flaws in such recommendations. Besides the system size, the temperature and the start configuration, the equilibration time also differs for various observables and of course for different systems and models. We cannot expect to find a solution without closely analyzing the specific final implementation of the model at hand.

That being said, even then there is no 100% foolproof recipe to determine N_{therm} in a fully satisfying rigorous fashion. The most common approach is to simply plot the observables one is interested in. After the implementation is finished and before we actually start to record states of the Markov chain, we write out *all* observables we are interested in after every single sweep beginning with the initial state. If one is interested in the energy and the magnetization of the system, one would plot the energy and all three components of the magnetization over Monte Carlo time t, i. e. over the number of sweeps. In section 2.2 we will do precisely that. We discover that all observables level out over time and then fluctuate around a constant value. This happens much later for the magnetization than for the energy. Thus it is vital to look at such plots for all observables. The point from which onwards all observables are basically constant, which is typically measured by eye, is the minimal equilibration time for this specific system at a fixed point of the phase space.

Keep in mind that in principle one has to repeat this analysis whenever one changes the temperature, the system size, the magnetic field or any other parameters of the model. Since hot start conditions are chosen at random, even the exact same simulation with a different start configuration could have a longer or shorter thermalization time. Therefore a generous additional cushion is mandatory. Analyzing and determining N_{therm} for the whole range of parameters one intends to simulate is the very first step after the Metropolis algorithm has been implemented and tested for correctness.

1.6.2 Autocorrelation and errors

Another pitfall we have emphasized several times already is the necessity of *independent* samples for Monte Carlo integration to work properly. Our samples are the states of a Markov chain with the generic characteristic that each state is generated by modifying its predecessor. By definition the resulting samples are not independent. However, by separating subsequent configurations by sufficiently many Monte Carlo steps we can bring down their correlation until they can be considered independent for all practical purposes. We will now quantify how N_{sweep} has to be chosen to guarantee such independence. Again it does not seem too harmful to simply let a “good amount of” sweeps go by in between consecutive configurations of the Markov chain, but there is actually analytic evidence that one has to take correlations seriously.

Autocorrelation is *the* central tool to quantify correlations of individual states within a time series. Given an observable $\mathcal{O}$ at Monte Carlo times $\{1, \dots, N\}$, where we start counting once the system has thermalized, the autocorrelation function can be estimated

$$R_{\mathcal{O}}(t) := \frac{1}{\sigma^2(N-t)} \sum_{i=1}^{N-t} \mathcal{O}(i)\mathcal{O}(i+t) - \mu^2 \tag{1.55}$$

for $t \in \{0, \dots, N-1\}$, where σ^2 is the variance and μ the mean of the observable $\mathcal{O}$. Note that it only makes sense to talk about the mean and variance once we draw from a stationary distribution, i. e. after the system has thermalized. Since we do not have access to the true mean and variance of observables, we have to estimate those quantities too. We use the standard sample mean and sample variance of $\{\mathcal{O}(1), \dots, \mathcal{O}(N)\}$

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}(i) \qquad \text{and} \qquad \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (\mathcal{O}(i) - \mu)^2 \tag{1.56}$$

which makes $R_{\mathcal{O}}$ a biased estimate. A few remarks are in order. First, the observations are taken at subsequent Monte Carlo steps right after thermalization, *not* at subsequent configurations of a Markov chain.

Furthermore, note that the sum in (1.55) at $t = 0$ with the normalizing factor is just the variance σ^2, which results in $R_{\mathcal{O}}(0) = 1$. The autocorrelation can be understood as the covariance of the sequence with a shifted version of itself. Since we want to measure the correlation of states at different times in the same chain this is entirely plausible.

Like the covariance, the autocorrelation can become negative. Negative autocorrelation at time t means that large negative values of $\mathcal{O}$ at a certain time correlate with large positive values t steps later in the chain or vice versa. The variance in the denominator normalizes the autocorrelation to lie in $[-1, 1]$.

One major issue with the estimator $R_{\mathcal{O}}$ is due to the fact that we are dealing with a finite sequence of measurements. The sum in (1.55) runs over fewer and fewer terms the larger t. However, we need a certain number of terms in the sum to average out random fluctuations and keep the variance of our estimate for the autocorrelation small enough. We will see in section 2.2 that autocorrelations indeed become noisy and statistically unreliable quickly. Therefore it only makes sense to look at $R_{\mathcal{O}}$ for $t \ll N$. Our goal is to find the time after which the new states are not correlated with the starting configuration anymore, i. e. the time after which the autocorrelation has dropped to zero or at least beneath some threshold close to zero.

It can be shown that the autocorrelation function in Markov chain Monte Carlo simulations generically decays like $\exp(-t/\tau_{\mathcal{O}})$, where $\tau_{\mathcal{O}}$ is called *autocorrelation time* for observable $\mathcal{O}$. Technically this exponential behavior only holds for $N \to \infty$ and $t \to \infty$ with $t/N \ll 1$. However, for the random walk of the Markov chain in the configuration space this asymptotic behavior is also approximated for small t such that we assume $R_{\mathcal{O}}(t) \approx \exp(-t/\tau_{\mathcal{O}})$. The simplest but also an inaccurate way to determine $\tau_{\mathcal{O}}$ is to plot the autocorrelation function and find the time after which it has dropped to $1/e$. Alternatively one can use the integrated autocorrelation time

$$\tau_{\mathcal{O}}^{\text{int}} = \frac{1}{2} + \sum_{t=1}^{N-1} R_{\mathcal{O}}(t) \tag{1.57}$$

for error analysis which is typically easier to compute. As the name suggests this definition stems from the continuous case, where for a purely exponential autocorrelation function $R_{\mathcal{O}}(t) = \exp(t/\tau_{\mathcal{O}})$ one finds

$$\int_0^\infty R_{\mathcal{O}}(t)\, dt = \tau_{\mathcal{O}} \,. \tag{1.58}$$

The discrete version (1.57) roots in the approximation

$$\frac{1}{2} + \sum_{t=1}^{\infty} \exp\left(-\frac{t}{\tau}\right) = \tau + \frac{1}{12\tau} - \frac{1}{720\tau^3} + \mathcal{O}\left(\frac{1}{\tau^5}\right) . \tag{1.59}$$

Already for $\tau = 10$ the relative error drops below 0.1% and approaches zero quickly. For all practical purposes $\tau_{\mathcal{O}} \approx \tau_{\mathcal{O}}^{\text{int}}$ if N is sufficiently large. However, due to statistical fluctuations and the finite number of measurements they might disagree quite significantly in real applications.

The importance of the autocorrelation time becomes obvious when one re-derives the error of Monte Carlo integration without the assumption that the samples are independent. Recall that for independent measurements we found that the error is given by $\sigma/\sqrt{N}$, with the variance σ from (1.56). If we set N_{sweep} to one, i. e. if we used every configuration after each sweep to estimate the thermal average, the error with correlations becomes

$$\sqrt{\frac{\sigma^2}{N}(1 + 2\tau_{\mathcal{O}}^{\text{int}})} \,. \tag{1.60}$$

Thus we have to perform sufficiently many sweeps in between two usable configurations such that each two samples are uncorrelated and the autocorrelation function as well as the integrated autocorrelation time go to zero. The number of sweeps in between two configurations N_{sweep} should be at least $5\tau_{\mathcal{O}}^{\text{int}}$ to confidently use the uncorrelated error estimate. Note that all this delicate business of choosing independent configurations only affects the errors and not the expectation values of observables.

Let us spend a few more words on practical matters. The finite number of measurements limits both the computation of the autocorrelation function as well as the sum in (1.57) to compute the integrated autocorrelation time. This issue could in principle be overcome by using more data points and extrapolating to $N \to \infty$. However, there is another even bigger practical problem. The values of $R_{\mathcal{O}}$ become statistically unreliable even for large N and $t \ll N$. This typically spoils exponential fits to determine $\tau_{\mathcal{O}}$ directly from $R_{\mathcal{O}}$. One has to find a sensitive balance between taking into account enough values to capture the asymptotic decay of the exponential, but not too many to avoid statistically unreliable values spoiling the fit. The same holds true for the sum in the integrated autocorrelation time.

In theory the sum needs to run over infinitely many values, but due to the exponential decay one can truncate it with clear conscience. Alternatively one can use values at small t to extrapolate to subsequent values by an exponential fit. However, their contribution is typically small enough to neglect them. Again, one has to choose the truncation carefully to leave out noisy values at late times t. Usually one tries to truncate the series self-consistently such that

$$\tau_{\mathcal{O}}^{\text{int}} \approx \frac{1}{2} + \sum_{t=1}^{m\cdot\tau_{\mathcal{O}}^{\text{int}}} R_{\mathcal{O}}(t) \,, \tag{1.61}$$

where m should be roughly between 5 and 6. Even then the values for the autocorrelation time $\tau_{\mathcal{O}}$ from an exponential fit and the integrated autocorrelation time $\tau_{\mathcal{O}}^{\text{int}}$, while estimating the same quantity, can differ significantly. One should always use the more conservative of the two.

Let us summarize how to set N_{sweep}. We initialize the system and discard the first N_{therm} sweeps completely. Then we record every observable of interest after every single sweep for as many sweeps as we can within reasonable computation times. A rule of thumb says that one should use at least $1000\tau_{\mathcal{O}}$ values to properly estimate $\tau_{\mathcal{O}}$. Keep in mind that every rule of thumb should be met with skepticism. With these values we compute the autocorrelation function (1.55) for each observable and plot them for small values of t. One will usually see a close to exponential decay for small t which directly transitions to noisy fluctuations, maybe even before the autocorrelation has approached zero at all. In this case increasing N might help. For each observable independently one then finds a consistent truncation (1.61) typically by a systematic search of m and should also fit an exponential to the data on the chosen range as a cross check. Finally, we use

$$\tau := \max_{\mathcal{O}} \tau_{\mathcal{O}}^{\text{int}} \tag{1.62}$$

to determine $N_{\text{sweep}} \in [5\tau, 6\tau]$.

1.6.3 Additional remarks

It is important to realize that for Monte Carlo methods in general and the Metropolis algorithm in particular one has to spend quite some time on analyzing the system before we even start the first simulation runs. Only if N_{therm} and N_{sweep} are chosen properly, can we be sure that our results make sense. One has to carefully monitor autocorrelations starting with small systems and high statistic and slowly transition to larger systems constantly supervising thermalization and correlations between states.

The Metropolis algorithm is typically rather inefficient. First of all, Monte Carlo methods are implicitly slow. Secondly, repeatedly accessing random lattice sites, their neighbors and next-nearest neighbors prohibits efficient caching of data. One can render the algorithm more cache friendly by updating one lattice site after the other as they are stored in memory instead of choosing them at random. However, for some systems this destroys ergodicity, i. e. some states might be unreachable from certain configurations. Despite the long computation times we consistently used random updates.

Low acceptance rates are another crucial performance drawback. Imagine our lattice model at low temperature and a high magnetic field. We expect the system to settle in a fully polarized state with little thermal fluctuations. Let's say we have reached a state where most spins are already parallel to the external field, but just a few are still misaligned. Whenever an aligned spin is selected the update is almost certainly rejected, because the proposal is likely to be misaligned again, would therefore increase the energy and due to the small temperature such moves are rejected. We might need several sweeps until one of the still misaligned spins is selected for an update and even then the probability that the random proposal is parallel to the magnetic field is tiny. Thus we will have to perform many sweeps to properly align just a few lattice sites. This will also have an impact on the autocorrelation time.

There are several alternatives and improvements to the Metropolis algorithm. Besides choosing different acceptance probabilities like in the *heatbath*, *Glauber* or *overrelaxation* algorithms, there are also fundamentally different approaches such as cluster algorithms or parallel tempering Monte Carlo (PTMC) methods. We only implemented the most primitive version of simulated annealing possible. However, even with such a simple setup one can observe interesting physical results. It should be a good starting point for more sophisticated algorithms.

Furthermore, there is a phenomenon called *critical slowing*, which implies that at certain parameter values, for example close to phase transitions or at very low temperatures, the acceptance rate will plummet, the autocorrelation time diverges and the evolution of the system becomes excruciatingly slow, basically comes to a halt.

This phenomenon is directly linked to interesting physical properties of the system and hence often leads to difficulties precisely where we want to observe it. We will not go into more detail about how to efficiently deal with critical slowing in this report.

It is best practice to monitor the acceptance rate, i. e. the number of accepted updates divided by the number of proposed updates, throughout the simulation. While the acceptance rate directly correlates with autocorrelation times in the initial analysis, one should also monitor it it throughout the actual simulations, for example separately for every configuration in the Markov chain. It can be used to detect critical slowing and when error estimates might break down.

2 Sky-MoCa – A Specific Application

2.1 The Code

We implemented *Sky-MoCa*, a simulated annealing algorithm based on Markov chain Monte Carlo integration with Metropolis transition probabilities for a three-dimensional spin lattice in Julia [1]. Julia is a relatively young language which is advertised on the official Julia homepage as a *high-level, high-performance dynamic programming language for technical computing.* It is a perfect fit for rapid prototyping of numerical simulations, where we care most about developer time, but speed still matters. Its syntax will be familiar to users of technical computing environments such as Matlab, Octave or also Python. Another nice feature of Julia is that it can be used in Jupyter notebooks which work precisely like IPython notebooks.

This allows for nice presentation and even faster prototyping and testing. Of course one can always export a regular Julia .jl file that can subsequently be run on a remote machine. The code is available at GitHub under https://github.com/nikikilbertus/Sky-MoCa. It has been written with readability in mind, which sometimes comes at the cost of performance, and is fully documented such that we will not go into implementation details in this report. Each simulation run produces a single HDF5 file containing detailed information down to the full configuration after every lattice sweep if desired. While there are specific post-processing and analysis sections in the Julia notebook, the repository also contains a Mathematica notebook with several visualization capabilities [21].

2.2 Results

In this section we perform the analysis described theoretically in section 1.6 for the spin lattice model introduced in section 1.2. We show that determining N_{therm} and N_{sweep} is a delicate task and that the Metropolis algorithm, while theoretically quite universal, might become infeasible in practice not only for pathological systems. One has to take great care in ensuring equilibrium and uncorrelated configurations, otherwise the simple error estimate $\sigma/\sqrt{N}$ might significantly underpredict the true error.

Recall that the first order of business is to determine the thermalization time. Let us start with a relatively small system of $N_x = N_y = N_z = 10$, i. e. a total of 10^3 vertices. In this first stage we want to determine the thermalization and autocorrelation times for three different temperatures $T \in \{1.5, 0.7, 0.1\}$ at a fixed magnetic field $B = 0.3$. Recall that $J = 1$ and $K = \tan(2\pi/10)$, thus all parameters are fixed. We initialize the system with a hot start and then perform $N = 10^4$ lattice sweeps. Those run on a MacBook Pro with a 2.2 GHz Intel Core i7 and 16 GB of 1600 MHz DDR3 memory in less than a minute.

2.2.1 Reasonable analysis at high temperature

We expect the system to thermalize fastest at high temperatures. In Figure 2.1, we simply plot the energy and all three components of the magnetization as a function of Monte Carlo time. We do not show the whole region up to 10^4 steps, because apparently the system has thermalized after just a few hundred sweeps. As expected the energy drops rapidly and levels out at a constant value. The magnetization in $\hat{\mathbf{z}}$ direction grows rapidly and levels out at a positive value, thereby following the external magnetic field, which also points in the $\hat{\mathbf{z}}$ direction. The $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ components stay roughly at zero. The fluctuations are pretty large, which is expected at high temperatures. We decided in this case that $N_{\text{therm}} = 500$ is a reasonably long thermalization time. In the second row of Figure 2.1 we show the autocorrelation functions for all four observables. Note that the maximal time shift we are showing here is $t = 500$, thus much smaller than the number of configurations we have used for the autocorrelation, namely $N - N_{\text{therm}} = 10^4 - 500 = 9500$.

For all observables the autocorrelations drop quickly and then transition into noisy behavior already at relatively small t. Apparently already around $t \approx 50$, the autocorrelation functions are completely unreliable. In the last two rows of Figure 2.1, we plot each of the four autocorrelation functions separately with an exponential fit. In the least square one parameter fits to $\exp(-t/\tau)$ we have only used the values of the autocorrelation functions shown in the plots, i. e. $t \leq 80$. At the same time we compute the integrated autocorrelation time τ^{int} according to (1.61) with m such that we sum over all values shown in the plot. The fit parameter for τ as well as τ^{int} are shown in the plots.

We immediately realize that they do not agree perfectly, but are reasonably close, given the statistical fluctuations in the autocorrelation functions. Whenever the autocorrelation lies above the exponential fit for a significant portion of the range as for the energy H, the integrated autocorrelation is typically larger than the fit value. However, when it mostly runs below the fit as for M_z it is not necessarily smaller. Indeed we have altered (1.61) slightly to add up absolute values of the autocorrelation functions. Otherwise we could compensate large correlation by anticorrelation and thereby reach a smaller autocorrelation time. This is highly risky of course, since negative correlation is correlation too and cannot cancel positive correlation.

In the end we take the maximum over all autocorrelation times and settle with a value of $T = 17$. Had we just looked at the energy, we could have reasoned $\tau \approx 8$, less than half. We realize that with 9500 configurations we used less than 600τ sweeps to estimate τ from the autocorrelation functions which is already at the edge of best practice. We should have chosen $N \gtrsim 2 \cdot 10^4$ for better statistics. At this temperature fluctuations are large and the exact choice of N_{therm} is still a matter of taste. Since the thermalization is only necessary once in the beginning, it is not a big computational setback to choose conservatively. The large temperature is also reflected in the acceptance rate. It qualitatively shows a similar behavior like the energy, dropping quickly and then flattening out at around 40%. Thereby we are confident that the system is indeed evolving, sampling many different configurations. Eventually, for this situation we would recommend to skip $N_{\text{sweep}} = 6 \cdot \tau \approx 100$ sweeps between two consecutive configurations of the Markov chain.

2.2.2 Faulty analysis at lower temperatures

To also give an example of faulty analysis, we repeat the same procedure for $T = 0.7$ and $T = 0.1$. All other parameters, in particular $N = 10^4$, have not been changed. In Figure 2.2 and Figure 2.3 we show the best results achievable by varying the interval on which to fit autocorrelations. First of all, the amplitude of the fluctuations in the observables clearly decrease with decreasing temperature. To some extent this makes it easier to determine N_{therm}. However, with the theoretical preparation from the previous chapter the issues become quite obvious. While we can still observe thermalization reasonably well and deduced $N_{\text{therm}} = 1500$ and $N_{\text{therm}} = 5000$ for $T = 0.7$ and $T = 0.1$ respectively, the autocorrelation functions become statistically unreliable even above 0.2. Thus the autocorrelation times inferred from the exponential fit and the integrated autocorrelation time differ by a factor of up to 1.5. The fits are a disaster and due to the large fluctuations in the autocorrelation functions we cannot expect the integrated autocorrelation time to give reliable results.

The cure of course lies in improving the statistics. For $N = 10^4$ and the chosen thermalization times there are only 8500 and 5000 measurements left to compute the autocorrelation functions. Because τ seems to be on the order of a few hundreds, we have only used roughly 10τ sweeps to determine τ. We need at least one or two orders of magnitude more Monte Carlo steps for a reliable estimate of the autocorrelation time. The acceptance rate drops to approximately 15% and only 2% for $T = 0.7$ and $T = 0.1$ respectively, which is a strong indicator for slow evolution and results in long autocorrelation times. Varying the temperature clearly shows that every point in parameter space requires its own analysis. Both the thermalization time $N_{\text{therm}} = 500$ and decorrelation length $N_{\text{sweep}} = 100$ from the first example at $T = 1.5$ are far too small for lower temperatures.

For $T = 0.7$ and $B = 0.3$ we are also close to a phase transition and thus might experience critical slowing. Redoing the analysis for a slightly different value of B might completely change the outcome and actually yield reasonable results. Of course, when inspecting a new system, there is no way of knowing where critical slowing might occur. Simulated annealing can be a valuable tool in the sense that we can start at a well understood, high temperature point in phase space and gradually change the magnetic field and/or the temperature. Note that technically the decorrelation length also has to be adjusted throughout the different stages of simulated annealing. Monitoring the acceptance rate can help to detect critical slowing.

Finally, let us correct the faulty analysis by increasing N to $2 \cdot 10^5$ for $T = 0.6$ and $B = 0.2$, see Figure 2.4. The autocorrelation functions are much smoother for larger values of t and the fits seem to be reliable and agree better with the integrated autocorrelation time. We can now more confidently deduce $\tau \approx 182$ and thus $N_{\text{sweep}} \approx 1000$. This shows that the previous analysis would have underestimated the autocorrelation length of M_z and also shows that with such a large decorrelation length a statistically proper simulation already becomes computationally challenging. These $2 \cdot 10^5$ sweeps already took about half an hour on a laptop computer. For $N = 1000$ configurations the runtime for this tiny system on a regular personal computer already goes into the hours. In practice one might settle with less sweeps in between configurations regardless. A thorough understanding of the autocorrelation at least tells us how this spoils the naive error estimates.

We delay the actual results of full simulation runs for the second report, where we go into more details about the physics of the system. We specifically investigate the transitions into and out of the skyrmion phase.

2.3 Conclusion

Monte Carlo methods provide powerful tools for situations where specialized rapidly convergent numerical methods fail or become infeasible. However, their general applicability and relative simplicity often hide the complexity of how to properly apply them and facilitate premature conclusions. Especially the preliminary analysis of equilibration and autocorrelation times is a tedious, but mandatory task.

We motivated and introduced Monte Carlo methods with a special focus on a combination of simulated annealing and the Metropolis algorithm. In addition to pointing out common pitfalls in theory, we illustrated proper and improper analysis by the example of a three-dimensional spin lattice. While this report focuses purely on the algorithmic aspects, in an accompanying paper we also discuss physical results of simulations run with Sky-MoCa.

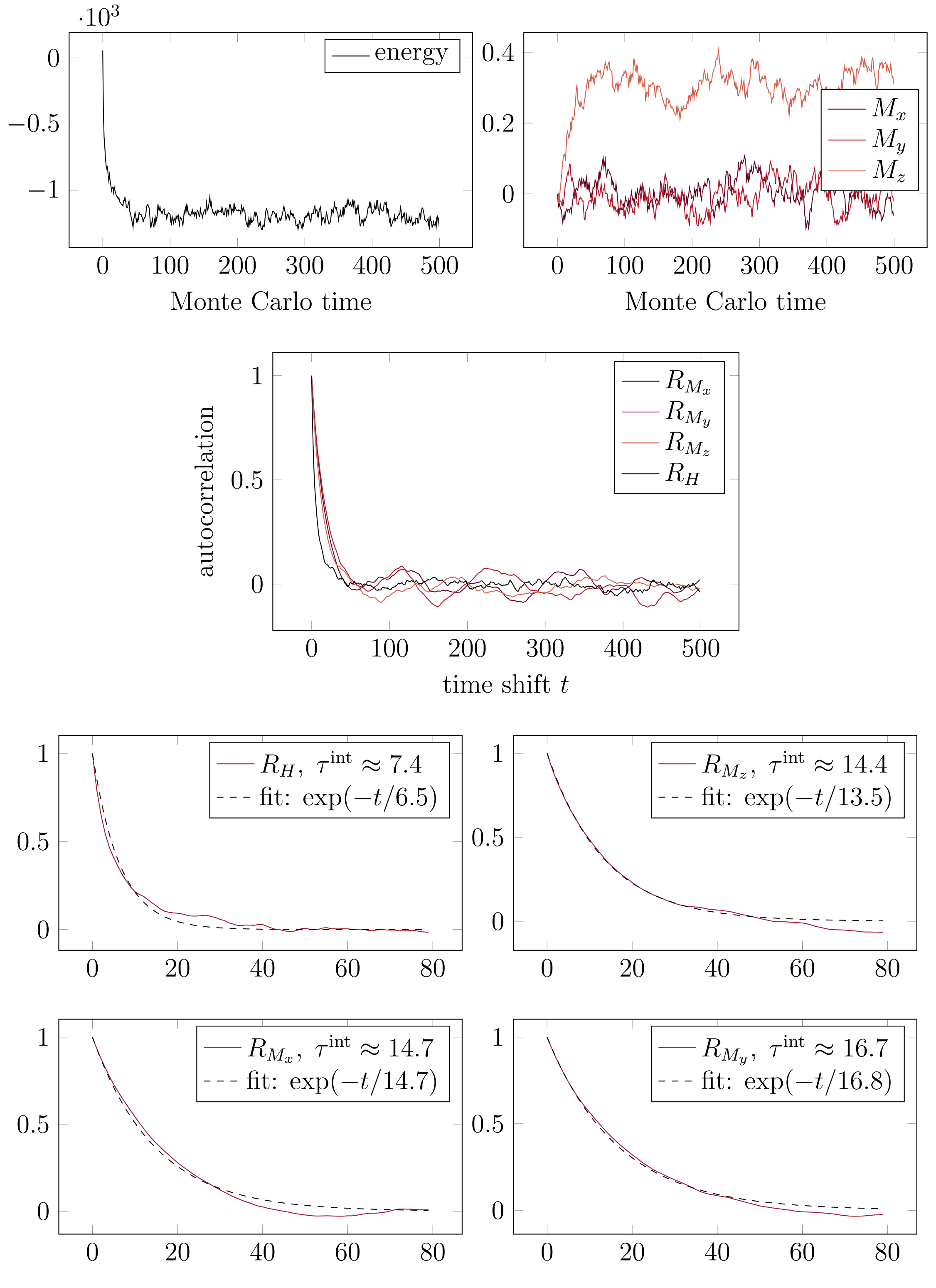

Figure 2.1: We show the thermalization and autocorrelation behavior for a system with 10^3 lattice sites, periodic boundary conditions and a uniformly random start configuration at $T = 1.5$ and $B = 0.3$. In this case we chose $N_{\text{therm}} = 500$. The acceptance rate settles around 40%. A detailed interpretation of the plots is discussed in the text.

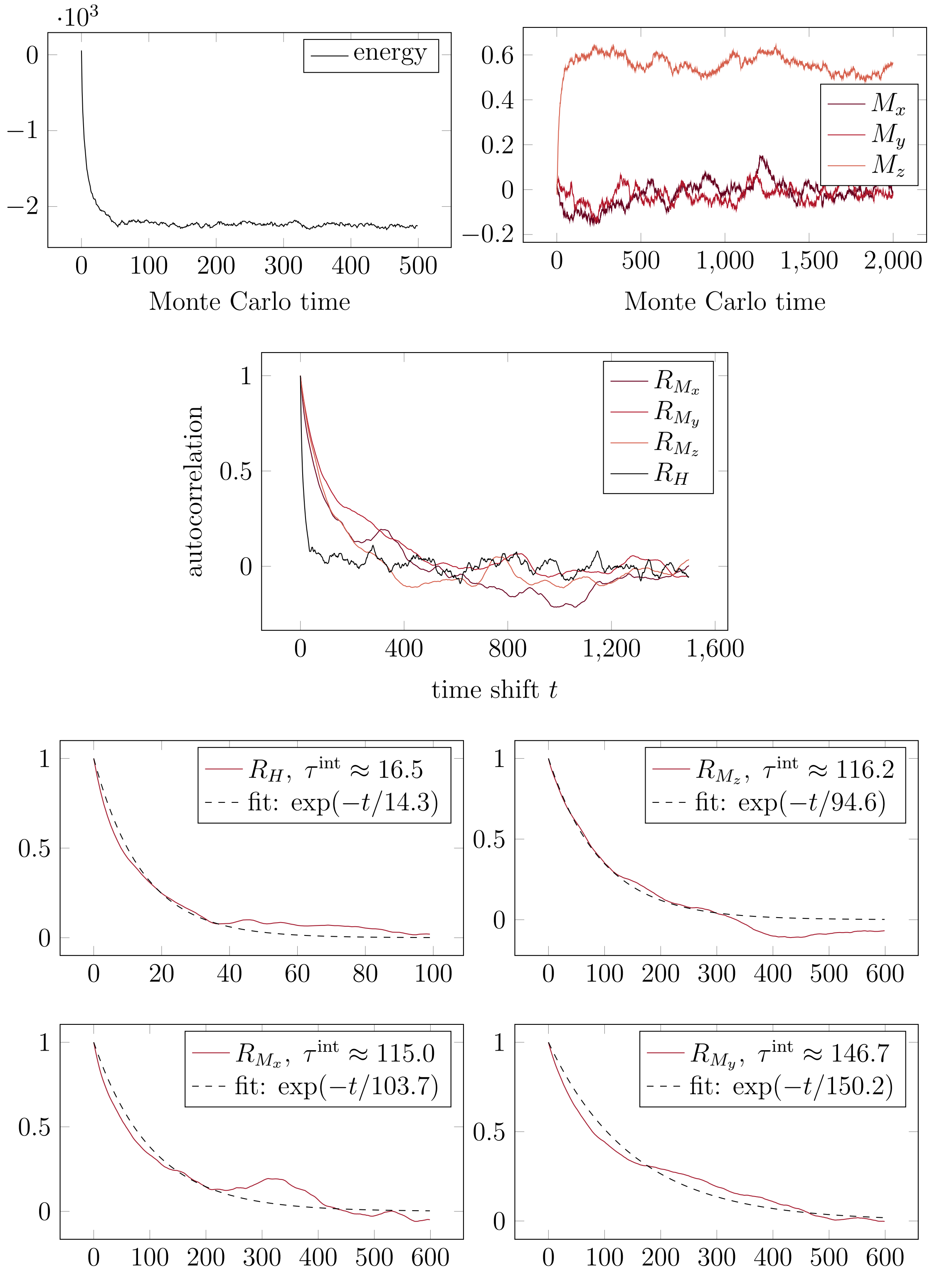

Figure 2.2: We show the thermalization and autocorrelation behavior for a system with 10^3 lattice sites, periodic boundary conditions and a uniformly random start configuration at $T = 0.7$ and $B = 0.3$. In this case we chose $N_{\text{therm}} = 1500$. The acceptance rate settles around 15%. A detailed interpretation of the plots is discussed in the text.

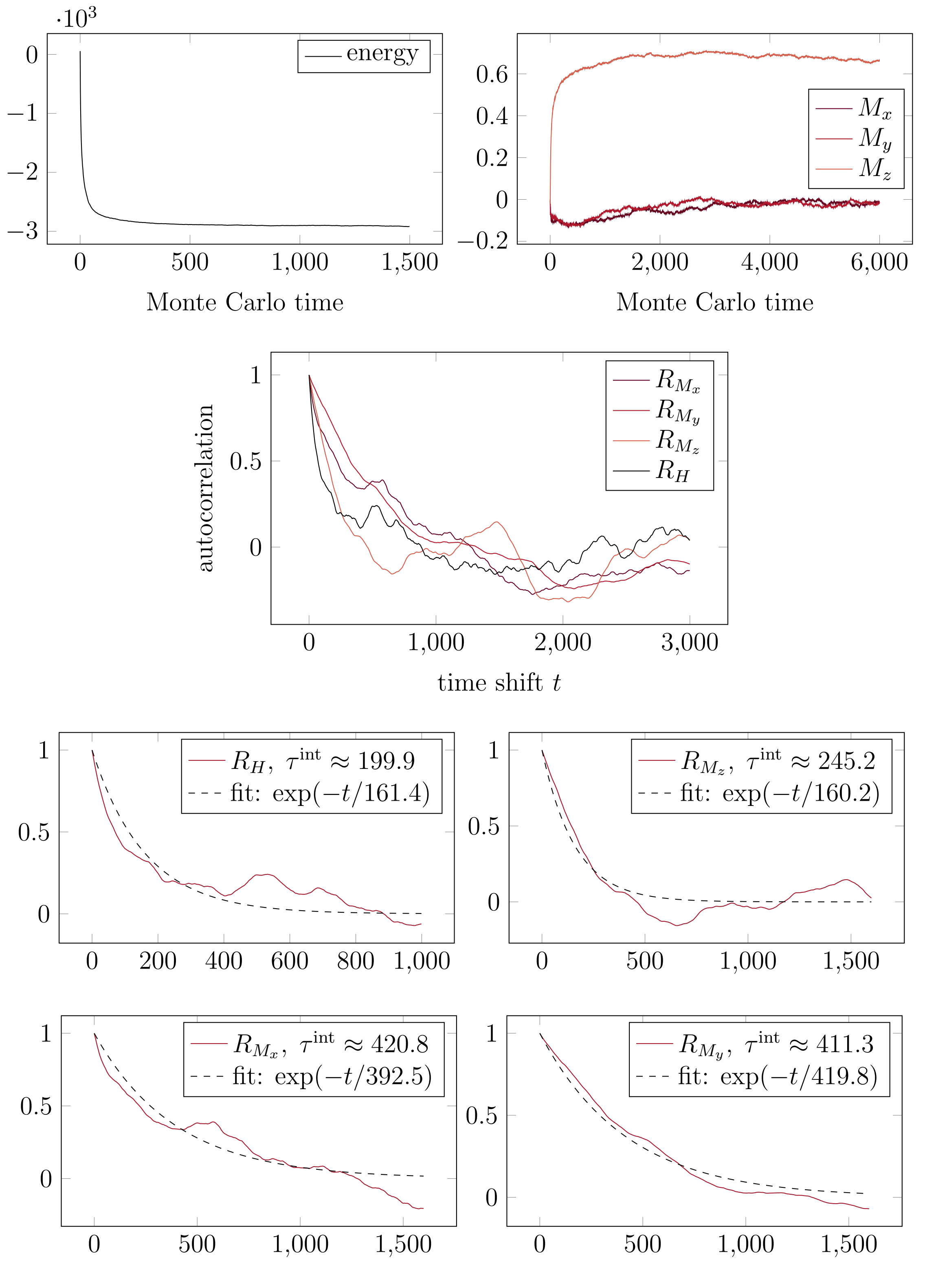

Figure 2.3: We show the thermalization and autocorrelation behavior for a system with 10^3 lattice sites, periodic boundary conditions and a uniformly random start configuration at $T = 0.1$ and $B = 0.3$. In this case we chose $N_{\text{therm}} = 5000$. The acceptance rate settles around 2%. A detailed interpretation of the plots is discussed in the text.

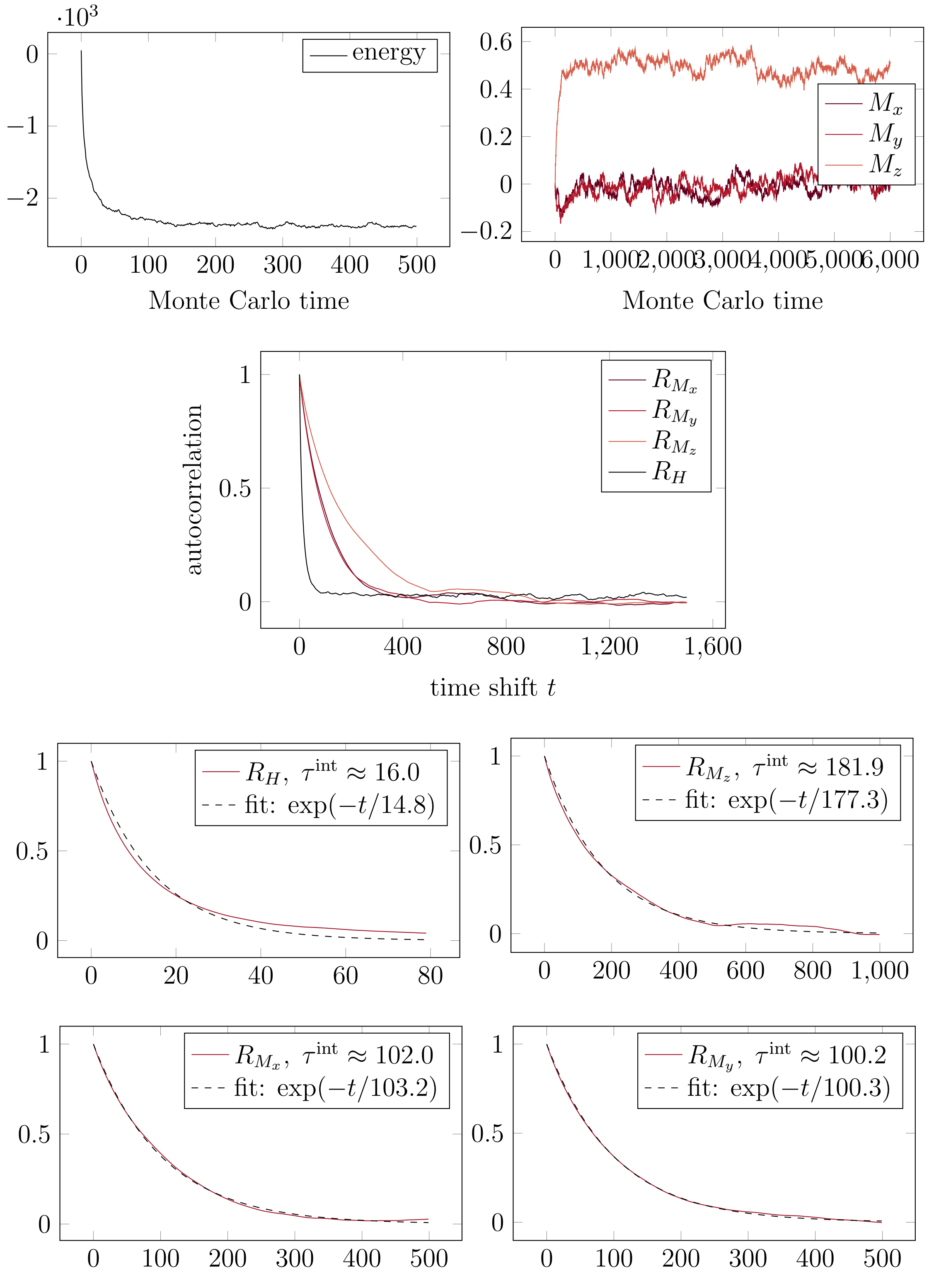

Figure 2.4: We show the thermalization and autocorrelation behavior for a system with 10^3 lattice sites, periodic boundary conditions and a uniformly random start configuration at $T = 0.6$ and $B = 0.2$. In this case we chose $N_{\text{therm}} = 5000$. The acceptance rate settles around 12%. A detailed interpretation of the plots is discussed in the text.

References

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah, *Julia: A fresh approach to numerical computing*, (2014).

[2] A.N. Bogdanov and Yablonskii D.A., *Thermodynamically stable 'vortices' in magnetically ordered crystals. the mixed state of magnets.*, Sov. Phys. JETP **68** (1989), no. 1, 101–103.

[3] S. Buhrandt and L. Fritz, *Skyrmion lattice phase in three-dimensional chiral magnets from Monte Carlo simulations*, Phys. Rev. B **88** (2013), no. 19, 195137.

[4] Bradley Efron, *The jackknife, the bootstrap and other resampling plans*, 1982.

[5] Bradley Efron and Robert J Tibshirani, *An introduction to the bootstrap*, 1994.

[6] Alexander K Hartmann and Heiko Rieger, *Optimization algorithms in physics*, vol. 1, Wiley, 2001.

[7] C. Heo, N. S. Kiselev, A. K. Nandy, S. Blügel, and T. Rasing, *Switching of chiral magnetic skyrmions by picosecond magnetic field pulses via transient topological states*, Scientific Reports **6** (2016), 27146.

[8] H. G. Katzgraber, *Introduction to Monte Carlo Methods*, ArXiv e-prints (2009).

[9] P. Milde, D. Köhler, J. Seidel, L. M. Eng, A. Bauer, A. Chacon, J. Kindervater, S. Mühlbauer, C. Pfleiderer, S. Buhrandt, C. Schütte, and A. Rosch, *Unwinding of a skyrmion lattice by magnetic monopoles*, Science **340** (2013), no. 6136, 1076–1080.

[10] S. Mühlbauer, B. Binz, F. Jonietz, C. Pfleiderer, A. Rosch, A. Neubauer, R. Georgii, and P. Böni, *Skyrmion lattice in a chiral magnet*, Science **323** (2009), no. 5916, 915–919.

[11] W. Münzer, A. Neubauer, T. Adams, S. Mühlbauer, C. Franz, F. Jonietz, R. Georgii, P. Böni, B. Pedersen, M. Schmidt, A. Rosch, and C. Pfleiderer, *Skyrmion lattice in the doped semiconductor* $fe_{1-x}co_xSi$, Phys. Rev. B **81** (2010), 041203.

[12] Naoto Nagaosa and Yoshinori Tokura, *Topological properties and dynamics of magnetic skyrmions*, Nat Nano **8** (2013), no. 12, 899–911, Review.

[13] M.E.J. Newman and G.T. Barkema, *Monte carlo methods in statistical physics*, Clarendon Press, 1999.

[14] C Pfleiderer, T Adams, A Bauer, W Biberacher, B Binz, F Birkelbach, P Böni, C Franz, R Georgii, M Janoschek, F Jonietz, T Keller, R Ritz, S Mühlbauer, W Münzer, A Neubauer, B Pedersen, and A Rosch, *Skyrmion lattices in metallic and semiconducting b20 transition metal compounds*, Journal of Physics: Condensed Matter **22** (2010), no. 16, 164207.

[15] Jan Seidel, *Topological structures in ferroic materials*, Springer, 2016.

[16] S. Seki, X. Z. Yu, S. Ishiwata, and Y. Tokura, *Observation of skyrmions in a multiferroic material*, Science **336** (2012), no. 6078, 198–201.

[17] T. H. R. Skyrme, *A non-linear field theory*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **260** (1961), no. 1300, 127–138.

[18] Y. Tokunaga, X. Z. Yu, J. S. White, H. M. Ronnow, D. Morikawa, Y. Taguchi, and Y. Tokura, *A new class of chiral materials hosting magnetic skyrmions beyond room temperature*, Nat Commun **6** (2015), Article.

[19] J.-C. Walter and G. T. Barkema, *An introduction to Monte Carlo methods*, Physica A Statistical Mechanics and its Applications **418** (2015), 78–87.

[20] Stefan Weinzierl, *Introduction to Monte Carlo methods*, (2000).

[21] Inc. Wolfram Research, *Mathematica*, Champaign, Illinois.

[22] X. Z. Yu, N. Kanazawa, Y. Onose, K. Kimoto, W. Z. Zhang, S. Ishiwata, Y. Matsui, and Y. Tokura, *Near room-temperature formation of a skyrmion crystal in thin-films of the helimagnet fege*, Nat Mater **10** (2011), no. 2, 106–109.

[23] X. Z. Yu, Y. Onose, N. Kanazawa, J. H. Park, J. H. Han, Y. Matsui, N. Nagaosa, and Y. Tokura, *Real-space observation of a two-dimensional skyrmion crystal*, Nature **465** (2010), no. 7300, 901–904.