Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

DIPLOMARBEIT

Monte Carlo Simulation Methods for Quantum Mechanical Systems

ausgeführt am

Institut für Allgemeine Physik (IAP)
der Technischen Universität Wien

unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Peter Mohn

durch

Robert Achleitner

Dürergasse 23, 1060 Wien

Wien, Mai 2008

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit Monte Carlo (MC) Simulationen mit Schwerpunkt auf quantenmechanischen Systemen. Zuerst wird ein kurzer Überblick über Monte-Carlo Methoden im Allgemeinen gegeben, danach wird eine bereits bekannte Anwendung, das klassische Ising Modell, behandelt. Der größte und wichtigste Teil dieser Diplomarbeit beschäftigt sich mit quantenmechanischen Systemen und deren Lösungsmethode auf Basis der stochastischen Reihenentwicklung (SSE). Es wird zuerst der theoretische Hintergrund dieser Methode dargestellt und dann der Algorithmus für das Heisenberg Modell auf endliche Spin-Ketten angewendet. Alle Simulationen wurden in FORTRAN programmiert.

Abstract

This thesis work deals with Monte Carlo (MC) simulations with the focus on quantum mechanical systems. I start with an introduction about Monte-Carlo methods in general and their application to the classical Ising model. The second, most important part deals with the problem in quantum mechanics, and presents a possible simulation method, the Stochastic Series Expansion (SSE). I give an introduction about the idea behind that method and give some results obtained for simulations for finite spin chains. All programming was performed in FORTRAN.

Contents

Part I
Introduction

1 Introduction

Computer Simulations are an important part of modern physics. Once a process can't be solved analytically anymore one needs numerical methods. These numerical calculations are of course performed on computers. The faster the CPUs become, the more complex a numerical calculation can be performed. Computer simulations in Physics are nowadays a broad field. Many of the numerical methods rely on the solution of analytically given equations (e.g. the Schrödinger equation) by means of deterministic methods. MC simulations go along a completely different path by applying statistical methods to obtain numerical results. The possibilities are huge as will be demonstrated during this diploma work.

2 Monte Carlo Simulations (MC)

2.1 Introduction

As mentioned above, Monte Carlo Simulations are simulations with random numbers. To give a better explanation, one can say MC Simulations use random numbers either to generate different possible states of the system or to weight possible changes to the system and use the results in statistics. To give an idea on how MC simulations work I will start with an easy example, the calculation of the irrational number π.

2.2 Example: Calculation of π

A good example as an introduction into the MC method is the calculation of the constant π. In figure 1 below you can see how this is done. There is a square with the length 1 and a quarter of a circle in it with the radius 1. Then points are drawn with random x and y coordinates.

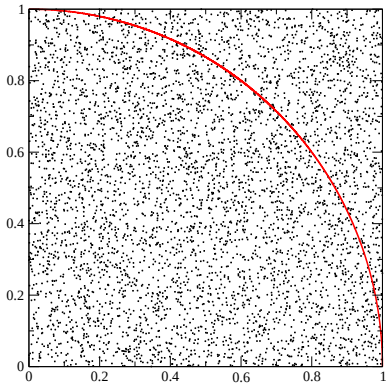

Figure 1: Graphical representation of how to calculate π with a MC-simulation

If the random number generator (RNG) is good enough, one should be able to fill the whole square equally if you just draw enough random points. So it's easy to see that the ratio of the areas of the square and the circle will have the same ratio as the points inside the circle and the total number of points. So all that has to be done is to draw a chosen amount of random points and count the ones that are inside our quarter circle. Then π should be given by $4\frac{M}{N}$ (M is the number of points counted inside the circle so that π is approximated by a rational fraction. N is the number of all random points. The factor 4 comes from working with a quarter circle). The simple algorithm is like following:

- Set your counter(M) to 0 and choose a number of random points(N)
- Start a loop
- Choose a random value for x (Between 0 and 1)
- Choose a random value for y (Between 0 and 1)
- Increase your counter by 1 if $x^2 + y^2$ is less or equal than 1
- End the loop if the chosen number of random points is reached
- Calculate π by $\pi = 4\frac{M}{N}$

Below are some sample results i obtained with this simple algorithm:

N	π
10^1	2.8000000
10^2	3.0800000
10^3	3.0680000
10^4	3.1268000
10^5	3.1406800
10^6	3.1391560
10^7	3.1415068

As you can see the value of the calculated π gets the close to the real value of π the more random points were used. This shows us an important property of typical MC simulations. They fulfill the 'Law of Large Numbers'. That means the more random points, or to be more general, the more random numbers are considered, the better will be the result you get. The re

Part II
Classical Ising

3 Theory

3.1 Model Description

The classical Ising model is a simple model for a spin system developed by Ernst Ising. In fact it is an approximation of the quantum mechanical heisenberg model which will be discussed later on [2]. The spins are aligned on a lattice and can have the values +1 or -1. They are coupled to each other with a coupling constant $J_{i,j}$. Usually there is also a Zeeman term which takes the influence of an applied external field into account. For simplicity we will neglect an external field for now. The hamiltonian, and thus also the total energy, is given by following:

$$H = \sum_{i,j} J_{i,j} S_i S_j \tag{1}$$

In this formula every spin S_i interacts with every other spin S_j. The $J_{i,j}$ are varying according to the interaction partners. For simplification we assume that our forces are short ranged and we can focus on next neighbor interaction only (That is not the case in real systems). Let's also consider our system got a lattice structure with a lattice constant J which means one has to deal with an isotropic system. The hamiltonian becomes

$$H = \sum_{i} J S_i S_{i+1} \tag{2}$$

for a one dimensional model and

$$H = \sum_{i,j} J_1 S_{i,j} S_{i+1,j} + J_2 S_{i,j} S_{i,j+1} \tag{3}$$

for a two dimensional model. Of course you could go further the same way for a three dimensional system.
The coupling constants J_i classify the system. For $J_i > 0$ the energy is lower when neighbor spins are aligned antiparallel. Thus the system is antiferromagnetic. For $J_i < 0$ the energy is lower when the neighbor spins are aligned parallel. Thus the system is ferromagnetic.

3.2 Weightfunction

Lets have a look at the statistics. According to the hamiltonian every state α got a total energy $E(\alpha)$. The statistical weight of each state is given by

the Boltzmann distribution. So the statistical weight of the state α is given by

$$W(\alpha) = e^{-\beta E(\alpha)} \tag{4}$$

With $\beta = \frac{1}{k_B T}$. The expectation value of a function $f(\alpha)$ is given by

$$\langle f \rangle = \frac{1}{Z} \sum_{\alpha} f(\alpha) e^{-\beta E(\alpha)} \tag{5}$$

with Z as the partition function

$$Z = \sum_{\alpha} e^{-\beta E(\alpha)} \tag{6}$$

With the help of equation (4) the transition probability from state i (initial) to state f (final) can be calculated:

$$P_{i \to f} = \frac{W_f}{W_i} = e^{-\beta (E_f - E_i)} \tag{7}$$

As one can see it is possible to get higher values for $P_{i \to f}$ than 1. This is nothing to worry about. A value for $P_{i \to f} \geq 1$ just means that the transition happens at 100%, because E_f is sufficiently lower than E_i. Since it makes no sense to have a probability higher than 1 the probability will be limited to 1. So we obtain

$$P_{i \to f} = min(e^{-\beta (E_f - E_i)}, 1) \tag{8}$$

Typical functions that are calculated in such models are response functions such as the susceptibility or the specific heat. Since this chapter has the aim of providing a better understanding for computer simulations and their reliability it is sufficient to focus a very simple model consisting only out of two spins. For that case the analytical solution can easily be obtained for the response function. It will be satisfying enough to just focus on the susceptibility χ for now. According to [7] χ is given by

$$\chi = \beta(\langle M^2 \rangle - \langle M \rangle^2) \tag{9}$$

with M as the magnetization. The derivation of the analytical function for χ out of equations (5) and (9) for a ferromagnetic system consisting out of two spins is given by following:

$$\langle M \rangle = \frac{0 \cdot e^{\beta J} + 2 \cdot e^{-\beta J}}{e^{\beta J} + e^{-\beta J}} = 2 \frac{e^{-\beta J}}{e^{\beta J} + e^{-\beta J}} \tag{10}$$

$$\langle M^2 \rangle = \frac{0^2 \cdot e^{\beta J} + 2^2 \cdot e^{-\beta J}}{e^{\beta J} + e^{-\beta J}} = 4 \frac{e^{-\beta J}}{e^{\beta J} + e^{-\beta J}} \tag{11}$$

$$\langle \chi \rangle = \beta(\langle M^2 \rangle - \langle M \rangle^2) \tag{12}$$

$$\langle\chi\rangle = \beta[4\frac{e^{-\beta J}}{e^{\beta J}+e^{-\beta J}} - (2\frac{e^{-\beta J}}{e^{\beta J}+e^{-\beta J}})^2] \tag{13}$$

$$\langle\chi\rangle = 4\beta[\frac{e^{-\beta J}}{e^{\beta J}+e^{-\beta J}} - \frac{e^{-2\beta J}}{(e^{\beta J}+e^{-\beta J})^2}] \tag{14}$$

$$\langle\chi\rangle = 4\beta[\frac{e^{-\beta J}(e^{\beta J}+e^{-\beta J}) - e^{-2\beta J}}{(e^{\beta J}+e^{-\beta J})^2} \tag{15}$$

$$\langle\chi\rangle = 4\beta[\frac{1+e^{-2\beta J}) - e^{-2\beta J}}{(e^{\beta J}+e^{-\beta J})^2} \tag{16}$$

$$\langle\chi\rangle = 4\beta\frac{1}{4\cosh^2(\beta J)} \tag{17}$$

$$\langle\chi\rangle = \frac{1}{k_B T}\frac{1}{\cosh^2(\frac{J}{k_B T})} \tag{18}$$

Since such a simplified model is just a toy model the constants can be chosen freely. For convenience let's set them to 1. The resulting function is plotted in figure 2. This analytical function can be used to compare whether a computer simulation gets close to this function. Actually this will be shown in the end of the next chapter.

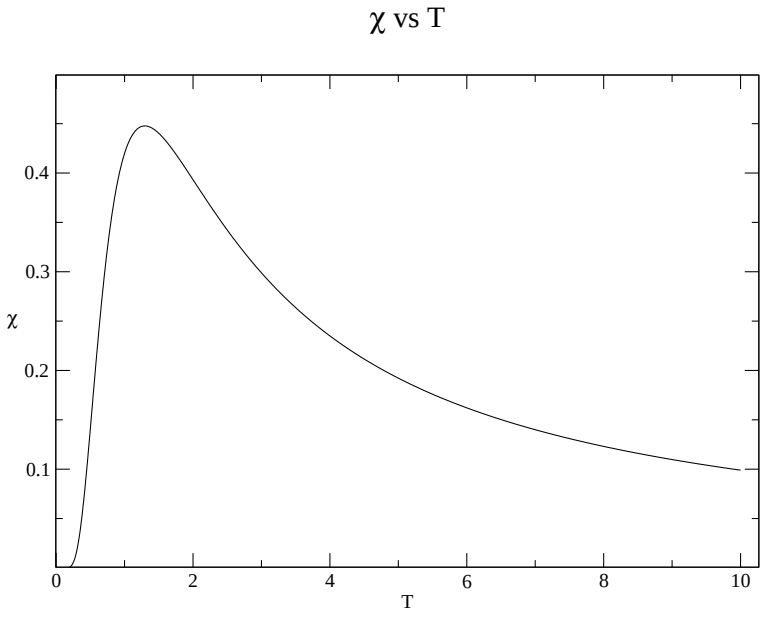

Figure 2: χ vs T - analytic solution for a ferromagnetic 2 spin system

4 Simulation

4.1 Algorithm

Since MC simulations are well known for the Ising model it is sufficient enough to give only a short summary about the algorithm with adding just a few more words to each point. If you would like to have more details i recommend [7]

1. Set up the model by setting different parameters (constants, starting temperature, number of spins, number of steps,...)
2. Determine a starting configuration. Either choose one manually or use random numbers to generate one.
3. Choose a spin; either one after the other or by randomly picking one and calculate its probability to flip according equation (8)
4. Draw a random number. If this number is greater than the transition probability then the spin does not flip. If it is less or equal than the transition probability then the spin flips. [3]
5. Save the important data (like magnetization, total energy) and repeat points 3-5 until the predetermined maximum number of steps is reached.
6. Calculate averages out of the saved data.

This is a pretty simple algorithm which is easy to program. There is one thing to add that I recommend, not just for this algorithm, but for MC simulations in general. Instead of doing one run with lots of steps i recommend to split up the steps into several runs. There are two reasons for doing this. The first reason is to neglect runs that end up in unlikely but possible configurations. The second reason is to neglect the limitation of the RNGs. That limitation comes from the periodicity of RNGs. Also to avoid runs starting with the same seed it is a good idea to generate some kind of list containing random numbers at first, from which a seed is drawn every time a new runs starts. That way you will start each run with a different seed for the RNG, which results in different starting values for the simulation. That way unlikely paths of a simulation, that can happen in one run, will average out by other runs. You can recognize that by a smoothing of your graphs.

4.2 Results

This part was intended to introduce the reader into the basic concepts of computational simulations of spin systems with the help of the Ising model. Because this thesis work is focused on the method of Stochastic Series Expansion I will settle here for just showing how accurate a MC simulation can be.

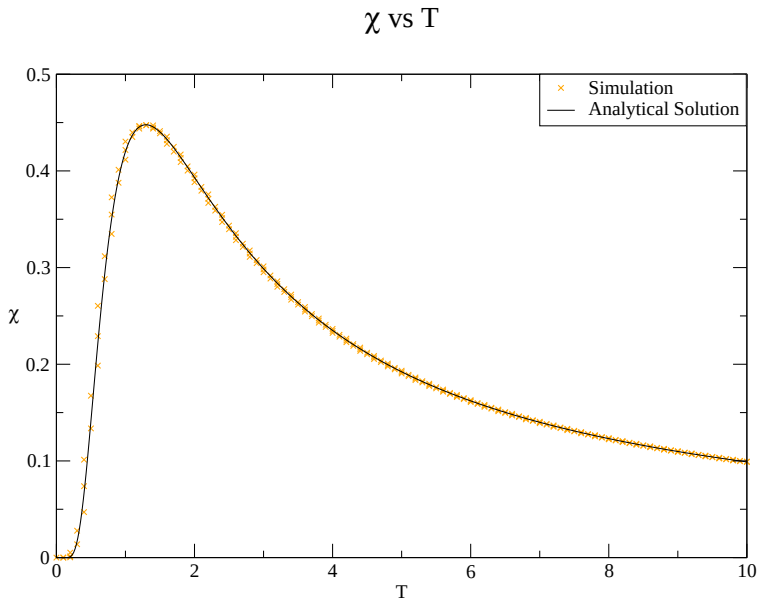

Figure 3: Analytical Solution and Simulated Results for χ with same parameters

In figure 3 you can see the analytical solution and the simulation result. For both graphs the same parameters ($k_B = 1$, $J = 1$, number of spins $N = 2$) were used to make them comparable. As you can see the numerical results fits perfectly onto the analytical curve. This shows quite well how accurate a numerical simulation can be.

Part III
Stochastic Series Expansion (SSE)

5 Classical Approach

5.1 Basics

Let's have a look at some statistical basics that were used in the Ising section. The statistical weight of a state $|\alpha\rangle$ is given by a Boltzmann factor

$$W(\alpha) = e^{-\beta E(\alpha)} \tag{19}$$

β stands for $\frac{1}{k_B T}$.
The partition function is given by

$$Z = \sum_{\alpha} W(\alpha) \tag{20}$$

Thus an expectation value is given by

$$\langle f \rangle = \frac{1}{Z} \sum_{\alpha} f(\alpha) W(\alpha) \tag{21}$$

The probabilities used in the Ising model are determined by the Boltzmann factor. Let's imagine the weightfactor (the exponential function) is not trivially solvable. What you can do now is to make a series expansion of this exponential function.

$$Z = \sum_{\alpha} W(\alpha) = \sum_{\alpha} e^{-\beta E(\alpha)} = \sum_{\alpha} \sum_{n} \frac{\beta^n (-E(\alpha))^n}{n!} = \sum_{\alpha} \sum_{n} W'(\alpha, n) \tag{22}$$

n is a new dimension of our configuration space. Thus a new weightfunction $W'(\alpha, n)$ was defined. Since this weightfunction depends on two variables (α, n) and can easily be distinguished from the old one depending only on α I will leave out the apostrophe from now on. If you have a look at $W(\alpha, n)$ you can see that for odd n it is possible to get a negative value. Since it makes no sense to have a negative statistical weight the zero point has to be shifted ($E \Rightarrow E - \varepsilon$) to make sure every shifted energy value is negative which causes all statistical weights to be positive.

$$W(\alpha, n) = \frac{\beta^n (\varepsilon - E)^n}{n!} \tag{23}$$

The fact that n ranges from 0 till ∞ and that the denominator is a factorial function calls for an approximation to make the series expansion a useful tool. The approximation will be an truncation of the series expansion. How to perform this truncation will be discussed later in chapter 6.1.

5.2 A mathematical trick to obtain energy expectation values and the specific heat more easily

First of all a new function H is defined

$$H = \epsilon - E \tag{24}$$

That way the weightfactor becomes

$$W(\alpha, n) = \frac{\beta^n H^n(\alpha)}{n!} \tag{25}$$

Let's calculate the expectation value of the hamiltonian H and transform it into a more convenient form

$$\langle H \rangle = \frac{1}{Z} \sum_{\alpha,n} H(\alpha) \frac{\beta^n H(\alpha)^n}{n!} \tag{26}$$

$$\langle H \rangle = \frac{1}{Z} \sum_{\alpha,n} \frac{\beta^n H(\alpha)^{n+1}}{n!} \tag{27}$$

$$\langle H \rangle = \frac{1}{Z} \sum_{\alpha,n} \frac{n+1}{\beta} \frac{\beta^{n+1} H(\alpha)^{n+1}}{(n+1)!} \tag{28}$$

$$\langle H \rangle = \frac{1}{Z} \sum_{\alpha,m} \frac{m}{\beta} \frac{\beta^m H(\alpha)^m}{(m)!} \tag{29}$$

with $m = n + 1$

$$\langle H \rangle = \frac{1}{\beta Z} \sum_{\alpha,m} m \frac{\beta^m H(\alpha)^m}{(m)!} \tag{30}$$

$$\langle H \rangle = \frac{1}{\beta Z} \sum_{\alpha,n} n \frac{\beta^n H(\alpha)^n}{(n)!} \tag{31}$$

with m renamed to n

$$\langle H \rangle = \frac{1}{\beta Z} \sum_{\alpha,n} n W(\alpha, n) \tag{32}$$

$$\langle H \rangle = \frac{\langle n \rangle}{\beta} \tag{33}$$

The expectation value of H is proportional to the expectation value of n. So it's only necessary to keep track of n. What exactly n is and how to calculate its expectation value will be shown later. From $H = \epsilon - E$ follows $E = \epsilon - H$. Thus the expectation value of E is given by

$$\langle E \rangle = \epsilon - \langle H \rangle = \epsilon - \frac{\langle n \rangle}{\beta} \tag{34}$$

Similar can be done for the expectation value of E^2 which leads to

$$\langle E^2 \rangle = \frac{1}{\beta^2}[\langle n^2 \rangle - \langle n \rangle^2] \tag{35}$$

If we insert those two equation into the equation for specific heat, which is

$$c_v = \beta(\langle E^2 \rangle - \langle E \rangle^2) \tag{36}$$

we get an equation for the specific heat depending only on n

$$c_v = \frac{1}{\beta}[\langle n^2 \rangle - \langle n \rangle^2 - \langle n \rangle] \tag{37}$$

These equations for the expectation values of the energies (34, 35) and the specific heat (37) also hold for quantum mechanics which can be easily profen by doing the same derivations as above for a quantum mechanical expectation value (the difference is the addition of Bra-Kets for the states, which corresponds to the trace in the expectation value).

6 Quantum Mechanical SSE

6.1 Basics

In quantum mechanics the expectation value of an observable A is defined by

$$\langle A\rangle = \frac{1}{Z} Tr(\hat{A} e^{-\beta \hat{H}}) \tag{38}$$

$\hat{\mathrm{H}}$ is the energy operator, the hamiltonian. Its eigenvalues are the different energy values of the different states. The partition function Z is defined by

$$Z = Tr(e^{-\beta \hat{H}}) \tag{39}$$

As mentioned in the last chapter, it is possible to express the expectation values for the energy and thus also the specific heat as functions depending on n only (34, 35, 37). But what exactly is this n and how to keep track of it during a numerical simulation?
First of all let's have a look on the weightfactors. Let's start with equation (38) and make a series expansion [4].

$$\langle A\rangle = \frac{1}{Z} \sum_{\alpha} \langle \alpha | \hat{A} e^{-\beta \hat{H}} | \alpha \rangle \tag{40}$$

$$\langle A\rangle = \frac{1}{Z} \sum_{\alpha} \langle \alpha | \hat{A} \sum_{n} \frac{\beta^n (-\hat{H})^n}{n!} | \alpha \rangle \tag{41}$$

$$\langle A\rangle = \frac{1}{Z} \sum_{\alpha,n} A(\alpha) \frac{\beta^n}{n!} \langle \alpha | (-\hat{H})^n | \alpha \rangle \tag{42}$$

$$\langle A\rangle = \frac{1}{Z} \sum_{\alpha,n} A(\alpha) W(\alpha, n) \tag{43}$$

The $|\alpha\rangle$ comes from the trace in (38) and are needed to get eigenvalues out of $\hat{H}$. As in the classical approach the energies, which are the eigenvalues of $\hat{H}$, have to be negative to make sure to not get a negative weight. To make sure this is fulfilled the zero point has to be shifted again and following ansatz is being made

$$\hat{H} = -\sum_{a,b} \hat{H}_{a,b} \tag{44}$$

with $\hat{H}_{a,b} > 0$

The hamiltonian is written as a sum of single bond operators, where b denotes the bondnumber and a the operator type. What exactly bondnumber

and operator types are will be discussed later on. For now it's only important to know that the $\hat{H}_{a,b}$ are a bunch of different operators. The zero-point shift of the main hamiltonian was put into one or more of the operators $\hat{H}_{a,b}$. Let's have a look at the factor $(-\hat{H})^n$ and how to solve it.

$$(-\hat{H})^n = (\sum_{a,b} \hat{H}_{a,b})^n \tag{45}$$

For the next step a new definition is introduced, the definition of an operatorstring $\{a,b\}$. Each string stands for a configuration of operators $\hat{H}_{a(1),b(1)}, \hat{H}_{a(2),b(2)}, ..., \hat{H}_{a(n),b(n)}$. The strings have the lengths of n. With the help of such operatorstrings the above equation can be rewritten as

$$(\sum_{a,b} \hat{H}_{a,b})^n = \sum_{\{a,b\}} \prod_{p=1}^{n} \hat{H}_{a(p),b(p)} \tag{46}$$

The result is a sum over different products of different configurations of n operators. So the number n corresponds to the number of operators in such a string. Let's put that into the formula for expectation values:

$$\langle A \rangle = \frac{1}{Z} \sum_{\alpha,n} A(\alpha) \frac{\beta^n}{n!} \sum_{\{a,b\}} \langle \alpha | \prod_{p=1}^{n} \hat{H}_{a(p),b(p)} | \alpha \rangle \tag{47}$$

Note that the strings $\{a,b\}$ in this equation have the length n and therefore the order of the sum over n and the sum over the strings $\{a,b\}$ can't be exchanged. However, if the strings are allowed to have all lengths from $n = 0$ till $n = \infty$ we can write the two sums as a sum of operator strings only whose lengths determine n as

$$\langle A \rangle = \frac{1}{Z} \sum_{\alpha,\{a,b\}} A(\alpha) \frac{\beta^n}{n!} \langle \alpha | \prod_{p=1}^{n} \hat{H}_{a(p),b(p)} | \alpha \rangle \tag{48}$$

This equation looks quite fine already except it still got infinite elements and the operator strings got different lengths. This is mathematically not a problem, but for an application in a computersimulation it is quite useless. To make this equation practically useful a truncation has to be made. Furthermore a way to unify the length of the strings has to be found
First assume that the truncation can be done at the number $M \, \epsilon \, \mathbb{N}$. Which value M should have will be described later on. Next a new operator type $\hat{H}_{0,0}$ is introduced. This operator type is just a unit operator and works as some kind of a space holder. With the help of this operator you can expand any string $\{a,b\}$ with the length $n \leq M$ to the length M by just adding $M - n$ unit operators $\hat{H}_{0,0}$. There are

$$l = \frac{M!}{(M-n)!n!} \tag{49}$$

ways of doing this. Because we transformed one $\alpha, \{a, b, \}$ configuration into l statistically equally distributed configurations we have to divide our former statistical weight by the factor l and obtain

$$\langle A \rangle = \frac{1}{Z} \sum_{\alpha,\{a,b\}} A(\alpha) \frac{\beta^n}{n!} \frac{(M-n)!n!}{M!} \langle \alpha | \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} | \alpha \rangle \tag{50}$$

$$\langle A \rangle = \frac{1}{Z} \sum_{\alpha,\{a,b\}} A(\alpha) \frac{\beta^n (M-n)!}{M!} \langle \alpha | \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} | \alpha \rangle \tag{51}$$

Thus the base of the configuration space are states α and strings $\{a, b\}$ with length M. n is now the number of non unit operators in those strings. The weightfactor of a configuration is given by

$$W(\alpha, \{a, b\}) = \frac{\beta^n (M-n)!}{M!} \langle \alpha | \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} | \alpha \rangle \tag{52}$$

This equation is highly important and will be used a couple of times in the QM-SSE algorithm later on.

6.2 Configuration limits of QM-SSE

As already discussed in former chapters, it is necessary to shift the zero point of the hamiltonian to make sure the statistical weight stays non-negative. This works pretty fine for the classical case, but does not work for all systems in QM-SSE. The problem is caused by the off-diagonal elements $\langle \alpha_1 | \hat{H}_{a,b} | \alpha_2 \rangle$ with $\alpha_1 \neq \alpha_2$, because the zero-point shift only effects diagonal elements. Thus some restrictions have to be made to the system to ensure a positive statistical weight. Systems with only positive off-diagonal elements don't need any restrictions to ensure positive weights. Therefore only systems with possible negative off-diagonal elements have to be restricted (for example anti-ferromagnetic systems).
How does this restriction look like? To make sure the weight is positive, an even number of negative operators must be used. From

$$\langle \alpha | \prod_{p=1}^{M} \hat{H}_{a,b} | \alpha \rangle \tag{53}$$

you can see that left and right from the product we got the same state, which comes from the trace. To get a non-zero weight, only operator strings are allowed that have that diagonal form. Therefore only systems are allowed that have only an even number of off-diagonal elements. For example for an one dimensional anti-ferromagnetic closed spin chain system only an even number of spins is allowed, because an odd number of spins would break this rule.

6.3 A graphical way to describe Operator strings

Before we proceed I want to introduce a graphical way to describe operator strings the so called world line presentation. To start this let's pick a one-dimensional lattice of spins which can either are up or down. For future uses within this thesis work, I will use blue circles as spin up and red circles as spin down. Let's pick 5 spins. With 5 spins you obtain 2^5 possible states $|\alpha\rangle$. For example with above definitions the state $|\alpha\rangle = |\uparrow,\uparrow,\downarrow,\uparrow,\downarrow\rangle$ can be displayed as following

Figure 4: State $|\alpha\rangle = |\uparrow,\uparrow,\downarrow,\uparrow,\downarrow\rangle$

Now let's draw a complete operator string $\{a, b\}$ at a particular state $|\alpha\rangle$. The length M of our string will be assumed as 8. Further we assume there are only 2 types of non unit operators and only next neighbor interaction is allowed. Another restriction is to allow only operators at bonds between two opposing spins. That way we get a really simple model for better understanding. How such a figure can look like is shown below.

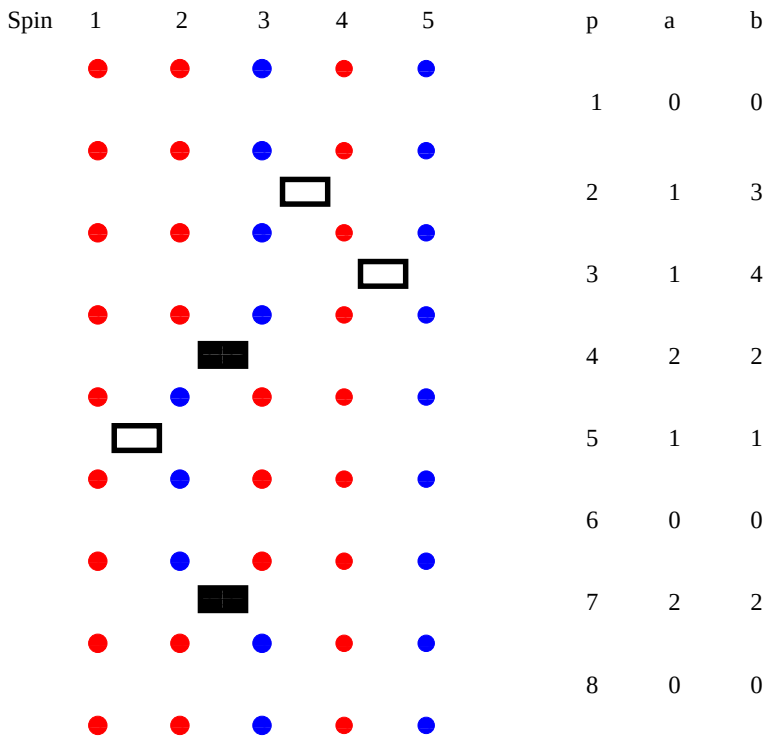

Figure 5: Complete Operatorstring at a defined state $|\alpha\rangle$

Let's analyze this figure step by step. First of all, the numbers 1 to 5 at the top of the colored dots label the different spins on our latice. This picture represents an open chain since spin 1 and spin 5 are not connected. Since only next neighbor interaction is allowed there exist only four different bonds (spin 1 - spin2, spin 2 - spin 3, spin 3 - spin 4, spin 4 - spin 5). The top row of dots represent the spin state $|\alpha\rangle = |\downarrow, \downarrow, \uparrow, \downarrow, \uparrow\rangle$. The space between the different rows represents each operator of the operator string. Empty space stands for a unit operator. The empty rectangles represent operator type 1. The full ones stand for operator type 2. The order of these operators is determined by going from the top till the bottom and the position is labeled by p. As you can see, p ranges from 1 till 8 which was the length of the operator string. The colum with a points out the operator type for each position p while the colum with b points out the bondnumber of those operators.
Now let's have a better look on those 2 operators. Type 1 will just give out a value without changing the state $|\alpha\rangle$ while type 2 exchanges the value of the spins it is acting on. This is easy to see by imaging the operators act on the above spin state and the result is the spin state below them. What has to be mentioned is, that the last spin state (the bottom row) has to be the same as the first one (the top row). This requirement is obvious if we have a look on the last part of the Weightfactor:

$$\langle\alpha| \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} |\alpha\rangle$$

The bra corresponds to the top row while the ket stands for the bottom row. In between is the product of operators ordered the same way as in the figure. Because we only have diagonal elements the top and the bottom state have to be identical.
Because it is inconvenient to draw all spins for every row even if they don't change only the top and bottom states are fully drawn. Other spins only show up at the corners of operators. Spins in the same column which don't change their value are represented by the edge-spins connected by a line. Thus figure 5 can be displayed as figure 6

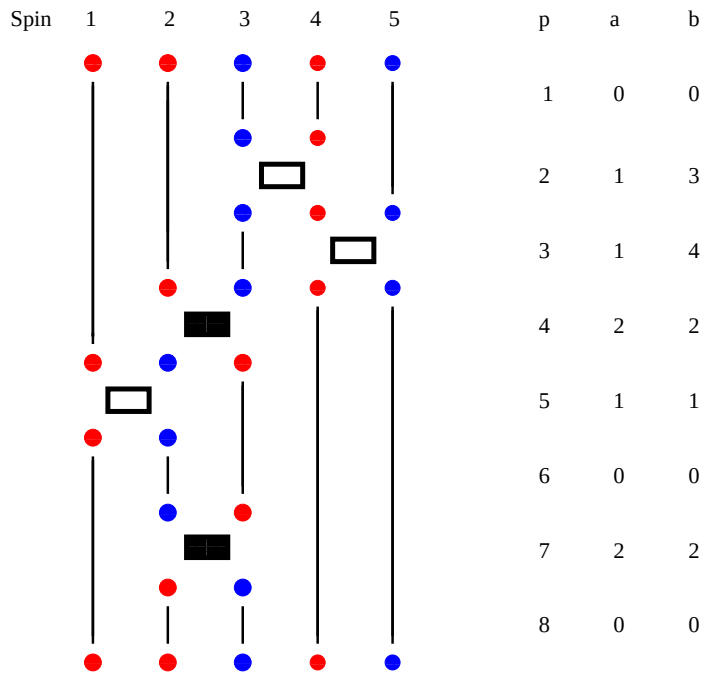

Figure 6: Optimized graphical Interpretation of an operator string

In future figures with this kind of schema only the graphical display will be shown, while the numerical columns (p,a,b) will be left out. In more complex models with more operator types new icons for those operators might appear. Also arrows connecting two edges of one operator will appear later on for the off-diagonal updates in the algorithm.

7 Simplified Heisenberg System

7.1 Theory

While the Ising model works with spins that can only have the values $+1$ or -1 the Heisenberg model works with whole quantummechanical observables. The hamiltonian (without a Zeeman term) looks as following:

$$\hat{H} = \sum_{i \neq j} J_{ij} \vec{S}_i \vec{S}_j \tag{54}$$

This can be rewritten as

$$\hat{H} = \sum_{i \neq j} J_{ij} (S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z) \tag{55}$$

I have to mention that in the literature there are two versions of this hamiltonian. This two versions are equivalent except the sign in front of the sum which comes from the opposite sign of the coupling constants $J_{i,j}$.
Let's choose the z-components as our base. ($|\alpha\rangle = |S_1^z, S_2^z, ..., S_N^z\rangle$ with $S_i^z|\alpha\rangle = \pm\frac{1}{2}$). Then with

$$S_i^x S_j^x + S_i^y S_j^y = \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+) \tag{56}$$

the hamiltonian becomes to

$$\hat{H} = \sum_{i \neq j} J_{ij} [S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)] \tag{57}$$

with S^+,S^- as the ladder operators.
Up to now, it was a general description of any Heisenberg system, but for an easier understanding of the upcoming algorithm let's simplify this system. While the the summation in the hamiltonian of the heisenberg model usually has to be done over all two-particle interactions, which have different coupling constants, we focus now on a simplified system that only takes next neighbor interaction into account. Furthermore every next neighbor pair will have the same distance and thus the same coupling constant J. The result is a spin lattice only with next neighbor bonds. As you can see in the figure below, such a system is easy to manage in an algorithm as we label each spin and each bond with a number:

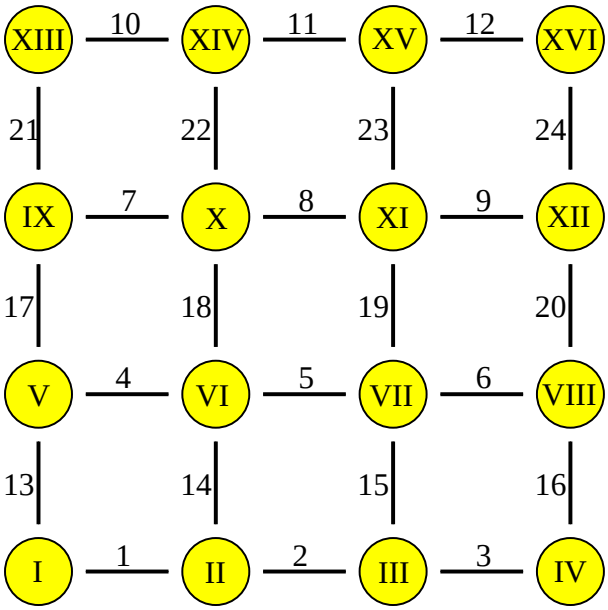

Figure 7: Two dimensional spin-lattice

In Figure 7 you can see a two dimensional 4×4 spin lattice. The spins are labeled in Roman numerals, while the bonds are labeled in Arabic numerals. There is a meaning why the bonds were labeled in in that way. By counting first through the horizontal and then through the vertical bonds it is easier in the algorithm to determine which kind of bond the program is treating at the moment. In the case of a isotropic lattice it would not matter, but if you want to simulate an anisotropic lattice you will have two different coupling constants which makes it necessary to determine which bond and thus which direction has to be considered. What also has to be mentioned is that for now we will only deal with finite systems as you can see from Figure 7, too. At the edges of our lattice there are no bonds. The reason for this is to avoid frustrated systems which could occur in periodic systems with an odd spin number in any direction (e.g. 3×3, 3×2, 2×3,...). For *even* systems it is possible to use periodic boundary conditions by just adding the necessary bonds (e.g. in the spin row *I-IV* an additional horizontal bond would be added right of spin *IV* labeled with number 4, thus the first horizontal bond in the second row would start with 5 and so on).
Now let's have a look at the hamiltonian itself. We have to define which types of bonds can occur and have to separate diagonal and off-diagonal elements. First of all let's separate the diagonal and off-diagonal terms and then have a look at the operator types. We can rewrite (57) as following:

$$\hat{H} = \sum_{i \neq j} J_{ij} S_i^z S_j^z + \sum_{i \neq j} \frac{J_{ij}}{2} (S_i^+ S_j^- + S_i^- S_j^+) \tag{58}$$

Let's focus on isotropic materials only. Thus we can replace the different coupling constants $J_{i,j}$ by one coupling constant J. Further we limit our possible systems to anti-ferromagnetic systems. Thus the coupling constant J has to be positive. As discussed already in chapter 6.1. the hamiltonian has to be separated into single bond operators

$$\hat{H} = -\sum_{a,b} \hat{H}_{a,b} \tag{59}$$

Let's discuss which operator types there are and what their eigenvalues are. The operators S_i^z and S_j^z don't change the state and their eigenvalues are $\pm\frac{1}{2}$. The ladder operators S_i^+,S_j^-,S_i^+andS_j^- change the state by raising(S_i^+) or lowering(S_i^-) the z-component of a spin. The combinations $S_i^+ S_j^-$ and $S_i^- S_j^+$ then change the z-component of two neighbor spins, by exchanging their values. Thus the first term in (58) corresponds to the diagonal part while the second term represents the off-diagonal part. The table below lists all possible single bond operators and their eigenvalues:

state before	state after	operator	eigenvalue
↑,↓	↑,↓	$\hat{H}_1$	$\frac{J}{4}$
↑,↓	↓,↑	$\hat{H}_2$	-$\frac{J}{2}$
↑,↑	↑,↑	$\hat{H}_3$	-$\frac{J}{4}$

Without an external field one cannot distinguish ↑ spin and ↓ spin. That's why there are only 3 different operator types left (e.g. the operator for ↑,↓ and ↓,↑ is the same). As you can see the eigenvalues of $\hat{H}_2$ and $\hat{H}_3$ are negative. For the chosen system the off-diagonal operator $\hat{H}_2$ will cause no trouble as explained in chapter 6.2. and will be treated as $+\frac{J}{2}$ from now. But because of the diagonal operator $\hat{H}_3$ the zero-point of the energy has to be shifted to fulfill the single bond operator condition

$$\hat{H}_{diagonal,b} \geqslant 0 \tag{60}$$

Thus $\hat{H} \longrightarrow \hat{H} - \frac{J}{4}$ and therefore $\hat{H}_1 \longrightarrow \frac{J}{2}$, $\hat{H}_2 \longrightarrow -\frac{J}{2}$ and $\hat{H}_3 \longrightarrow 0$. Due zero-point shift only two possible operators are left, because the statistical weight for a configuration that contains a $\hat{H}_3$ operator is zero and therefore can be left out. The eigenvalues of the two remaining operators $\hat{H}_1$ and $\hat{H}_2$ are equal which will make the algorithm a lot easier. Additionally the coupling constant J will be set to 1 which makes the eigenvalues to $\frac{1}{2}$.
Before I start to explain the algorithm I am going talk about the probabilities applied on the current model. The statistical weight of a configuration is given by equation (52). Since there are only operators left with the eigenvalues $\frac{1}{2}$ (n $\hat{H}_1$ and $\hat{H}_2$ operators) and 1 ($(M-n)$ unit-operators $\hat{H}_0$) the weight can be written as

$$W(\alpha, \{a, b\}) = \frac{\beta^n (M-n)!}{2^n M!} \tag{61}$$

For a constant temperature β and M are constants, thus only the number of non unit-operators n changes the weight. This means that different configurations with the same n have the same weight. This will be used in the off-diagonal update later on which can only transform $\hat{H}_1$ operators into $\hat{H}_2$ operators and vice versa. This leads to a transition probability of $\frac{1}{2}$. The number n will be updated in the diagonal update by using

$$P_{accept}(A \to B) = \frac{W(B) P_{select}(B \to A)}{W(A) P_{select}(A \to B))} \tag{62}$$

The ratio $\frac{P_{select}(B \to A)}{P_{select}(A \to B)}$ stands for the difference of the transition directions. There are B (free bonds) possibilities of adding an operator while there is only 1 way to remove an operator. Thus the transition probabilities for adding/removing an operator are given by

$$P_{accept}(n \to n+1) = min(\frac{B\beta}{2(M-n)}, 1) \tag{63}$$

$$P_{accept}(n \to n-1) = min(\frac{2(M-n+1)}{B\beta}, 1) \tag{64}$$

Equations (63) and (64) will be used as transition probabilities for the diagonal updates. The way the probabilities are written cause a cap at 1 to avoid a probability higher than 1 which would be unreasonable.

7.2 Algorithm

7.2.1 Overview

How does the algorithm look like? In the itemization below is a short step by step overview.

- **Generate a starting configuration**
- **Start the main program loop**
- **Perform diagonal updates:** Exchanging unit-operators $\hat{H}_{0,0}$ and diagonal operators $\hat{H}_{1,b}$
- **Construct the linked-vertex-list**
- **Perform off-diagonal updates:** Exchange diagonal operators $\hat{H}_{1,b}$ and off-diagonal operators $\hat{H}_{2,b}$ in loops with the help of the linked-vertex-list
- **Check if M needs to be updated**
- **Obtain data out of the simulation**
- **End the main program loop**
- **Calculate averages out of the collected data**

One more point has to be mentioned. For high temperatures there will hardly be any non unit operator as we can see from (63) and (64). This refers to a classical region and one might have to add a classical spin flip at high temperatures. But since with this method the quantum mechanical effects are simulated and one usually is more interested in quantum mechanical temperature regions it is not strictly necessary to implement classical spin flips.

7.2.2 Diagonal Updates

As the name tells, this update step is just about diagonal operators. There are two diagonal operators, the one 'real' diagonal operator $\hat{H}_{1,b}$ with the eigenvalue $+\frac{1}{2}$ and the unit operator $\hat{H}_{0,0}$, which is nothing more than a placeholder in the operator strings. What is done, is a full cycle (M) through all positions (p) in the current operatorstring. For each position p a check is performed to determine the current operator at this position and, if not the unit operator, to also determine the bond number. To do this the operatorstring's components have to be translated into numbers which is represented by the array *opstring[p]*. It's value for the operator $\hat{H}_{a(p),b(p)}$is given by

$$opstring[p] = 2(b(p) - 1) + a(p) \tag{65}$$

For example the operator $\hat{H}_{1,4}$ leads to $opstring[p] = 7$. The non unit operators got a value greater or equal than 1. Hence we can define the unit operator to have a value smaller or equal than 0, which makes it easy to check, weather the current operator string position is empty (filled with the unit operator as a spaceholder) or occupied by another operator. As you can see from (65) each possible combination of operator type and bond number leads to one value for the *opstring[p]* and vice versa (Except for the unit operator which could have any value lower than 1). Even if the unit operator could have any value below one, it is common to assign the unit operator the value 0

Now let's have a look at the different cases that can occur during our diagonal update cycle. The first case is a zero value of *opstring[p]*. In the graphical representation this means an empty operator row. Since this could be filled at any bond number, the bond number is randomly drawn from all possible bonds. Then one has to check weather the two spins connected with that bond are parallel or antiparallel aligned. Since the diagonal operator only acts on antiparallel spins, parallel ones are neglected and the cycle moves on to the next operator string position. In the case of antiparallel spins the transition probability according equation (63) is calculated. Then an random number in the range [0,1] is being generated and the insertion of the diagonal operator $\hat{H}_{1,b(p)}$ at the former chosen bond number b is accepted if the random number is lower than the calculated transition probability. If the random number is higher than the transition probability nothing happens and the cycle moves on to the next position p. Below is a diagram made for better demonstration of the possible insertion of a diagonal operator:

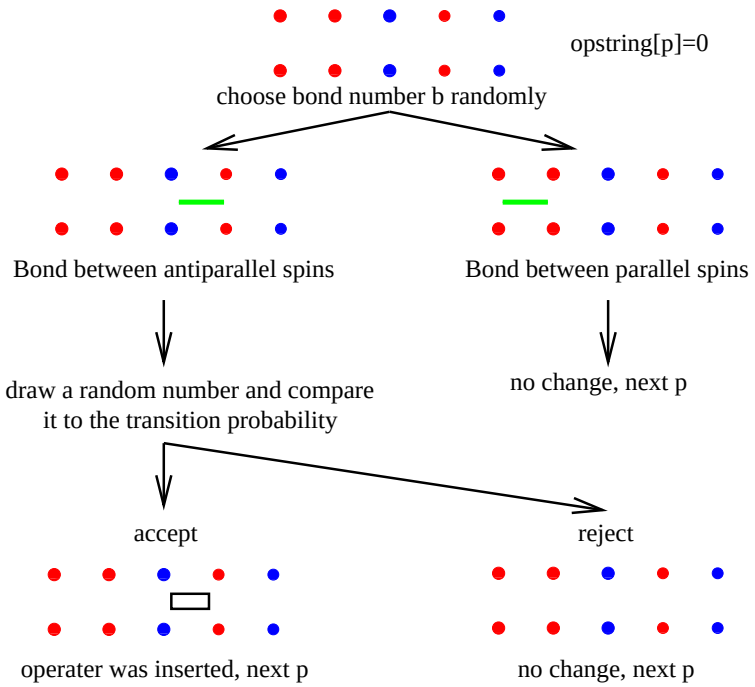

Figure 8: Diagonal Update Diagram of inserting an operator

The other possible case is to have already an operator at the current operator string position p. One has then to determine the bond number and whether it is a diagonal operator $\hat{H}_{1,b(p)}$ or an off-diagonal operator $\hat{H}_{2,b(p)}$. Both can easily be gained out of the current opstring[p] value. If one has to deal with an off-diagonal operator, nothing happens and the cycle moves on to the next position p. If the operator is the diagonal type, then equation (64) is used to calculate the transition probability to remove that operator. Then again a random number is drawn to check if the transition really happens or is neglected. The diagram below illustrates this case:

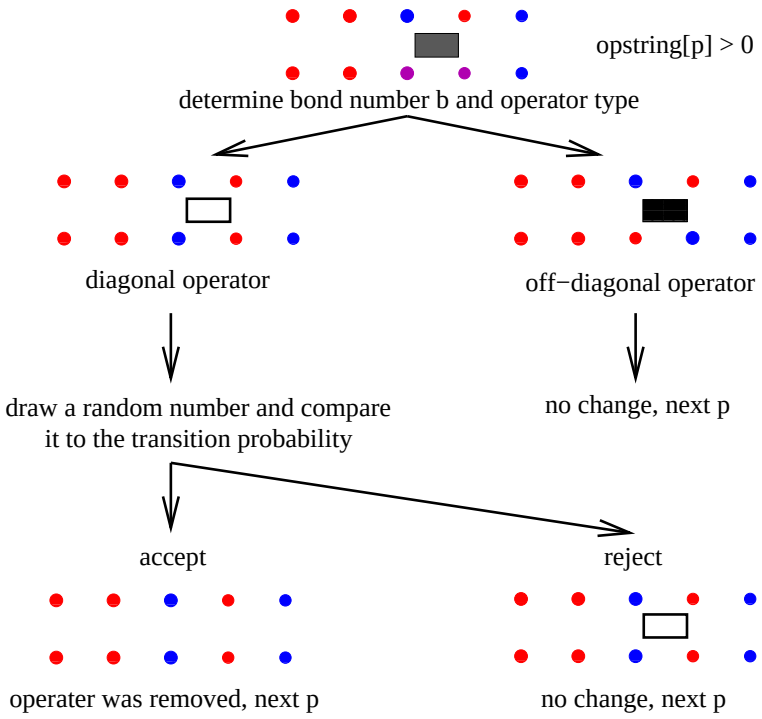

Figure 9: Diagonal update diagram of removing an operator

Once all operator string positions were cycled through (last p was equal to M) the diagonal update is finished and the next step, the constructing of the linked-vertex-list, starts.

7.2.3 Vertices and Linked-Vertex-List

A vertex is defined as an operator and spins before and after this operator has acted. It consists of an operator and different entrance and exit legs. In this simplified heisenberg model the vertices got 4 legs in total. All possible vertices are shown in following figure:

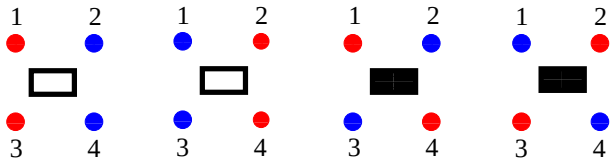

Figure 10: Vertices with their legs

These vertices look pretty much as the graphical view of an operator string described in chapter 6.3. Actually a diagram with vertices is an enhancement of the ordinary graphical representation of operator strings. The enhancement is found in the treatment of the legs. In the ordinary graphical view

the vertex like objects consist out of one operator and 4 spins, two before and two after the operator has acted. The spins are exactly between two operator positions which makes an overlap possible. In the vertex representation the spin rows are duplicated to avoid such an overlap which is illustrated in figure 11:

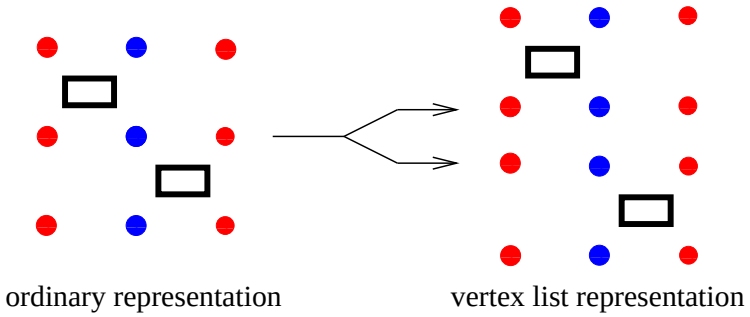

Figure 11: Enhancement of the graphical representation through vertices

The main aim of the vertex representation is to avoid that possible overlap of two consecutive positions in the operator string. This is important for an algorithm, because such overlaps could cause big troubles in the off-diagonal updates. Because this enhancement was mainly done for avoiding troubles in an algorithm and it would be unpracticable to draw those double spin rows in the graphical representation, the diagrams are drawn as the ordinary ones, but someone has to keep in mind that vertices are being drawn and an spin overlap is treated as two separated legs.
The linked-vertex-list is a list which connects every leg of each vertex in the current operator string to another leg. To do this a lattice of vertex-legs has to be created. Since there is one vertex possible for each position p in the operator string and each vertex got 4 legs a $4 \times M$ lattice has to be created, which i call vertex-leg lattice. For each lattice point a number is assigned by

$$v = 4(p-1) + l \tag{66}$$

with l being the leg number, which can have the value 1,2,3 or 4 according to figure 10. Then leg-pairs are created which correspond to lines in the graphical representation. In the algorithm this means you start from one leg in the graphical representation and move vertically until you reach a leg of another (or even the same) operator. If the top spin row is reached without a hit, then one has to continue from the bottom row and vice versa. A list is being created, which links the concerning vertex-leg lattice points. All other vertex-leg lattice points (empty operator string positions) will be set to zero. Thus a list was created, the so called linked-vertex-list. An example on how such a list looks like is given in figure 12.
In this diagram the form *vlist[v]=linked* v is used and this is a full vertex-leg

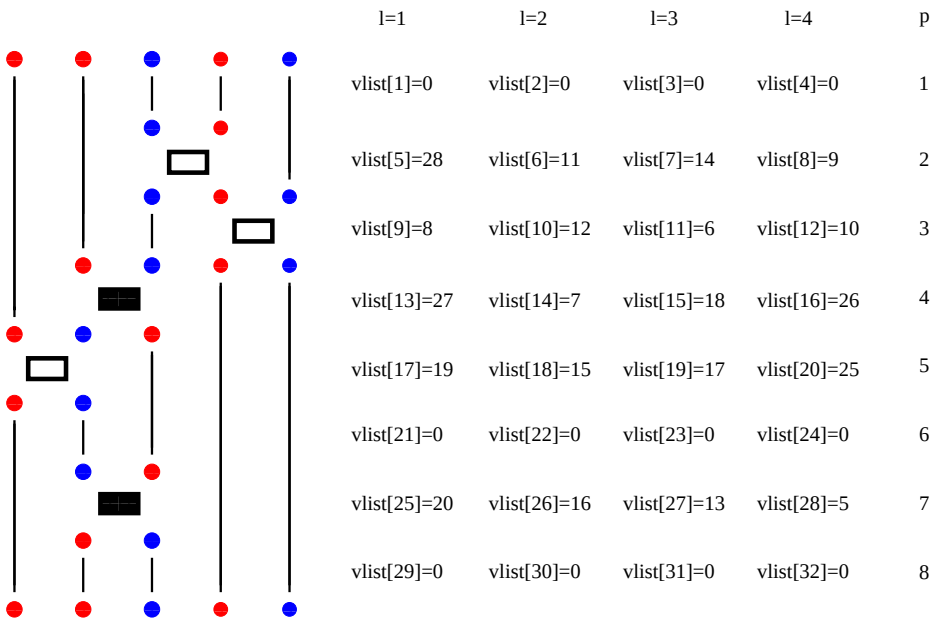

Figure 12: Example of an operatorstring and its linked-vertex-list

lattice with v being constructed by (66). With the help of such a list loops will be created as explained in the off-diagonal update section 7.2.4.

7.2.4 Off-Diagonal Updates

Up to now only diagonal operators were taken into account. As one can tell from the name this section considers also the off-diagonal operators. One way how this is done is the so called local off-diagonal update. In this method the type of two operators on the same bond (but of course different position p) is changed and thus also the spins at that particular bond. The problems that occur with this method is that there can be constraining operators between the concerning operators and the slowing down at critical points, which is always a problem for local update algorithm. Because this method is obsolete i won't give more detailed information about it.
A better method is the Loop update. In this method loops are created with the help of the linked-vertex-list. In short one starts at a leg at one vertex and moves according to the linked-vertex-list to the paired leg of the next vertex. There with some selection rule another leg on this vertex is chosen and again one moves the the paired leg and so on. Once the starting leg has been reached the loop closes itself Every visited spin along this path is flipped and also every visited operator is changed as will be explained below. Then the next loop is being constructed beginning at a not visited vertex leg. But first let's have a look at the selection rules at an operator as mentioned above.
Since all visited spins in a loop have to be flipped one can draw all possible transformations and sort out the ones that are forbidden. In figure 13 the possible transformations are drawn.

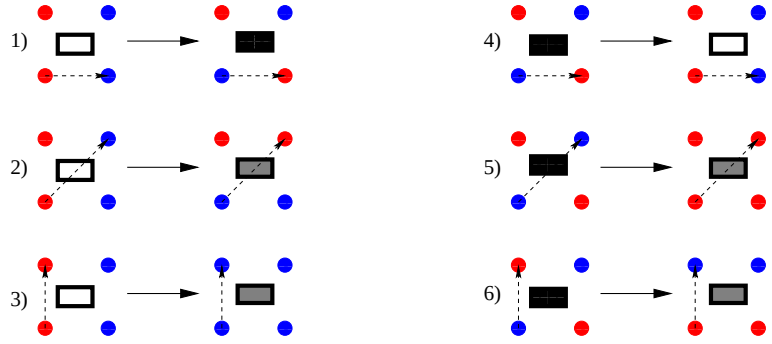

Figure 13: Possible operator transformations

1. Transformation of an operator $\hat{H}_{1,b}$ into an operator $\hat{H}_{2,b}$. Both operators exist in our model $\longrightarrow$ allowed

2. Transformation of an operator $\hat{H}_{1,b}$ into an operator that would look like $\frac{J}{2}(\hat{S}_i^+ \hat{S}_j^+ \hat{S}_i^- \hat{S}_j^-)$. Second operator does no exist in our model $\longrightarrow$ not allowed

3. Transformation of an operator $\hat{H}_{1,b}$ into an diagonal operator between to parallel spins. Second operator does in principle exist in our model, but got the eigenvalue 0 which leads to a statistical weight of 0. $\longrightarrow$ not allowed

4. Transformation of an operator $\hat{H}_{2,b}$ into an operator $\hat{H}_{1,b}$. Both operators exist in our model $\longrightarrow$ allowed

5. Transformation of an operator $\hat{H}_{1,b}$ into an diagonal operator between to parallel spins. Similar as in point 3) $\longrightarrow$ not allowed

6. Transformation of an operator $\hat{H}_{1,b}$ into the same non existing operater as in point 2) $\longrightarrow$ not allowed

As can be seen from this enumerations only operator transformations are allowed where you move along an operator horizontally. This is the selection rule on how loops have to be constructed. Hence to construct a loop one has to start at a vertex leg (that has not been visited by a previous loop already) and moves vertically to the paired leg of the next vertex and flip every spin passed this way (use periodic boundary conditions for the top and bottom spin rows). There you move horizontally to the neighbor leg on the same vertex and change the operator type. Then move again to the paired leg and so on. Once the loop is closed accept the loop-flip with the probability $\frac{1}{2}$ according to Swendsen-Wang [6],[5] or restore the state before the loop if rejected. Note that loops can't intersect each other. In figure 14 such a loop construction is shown.

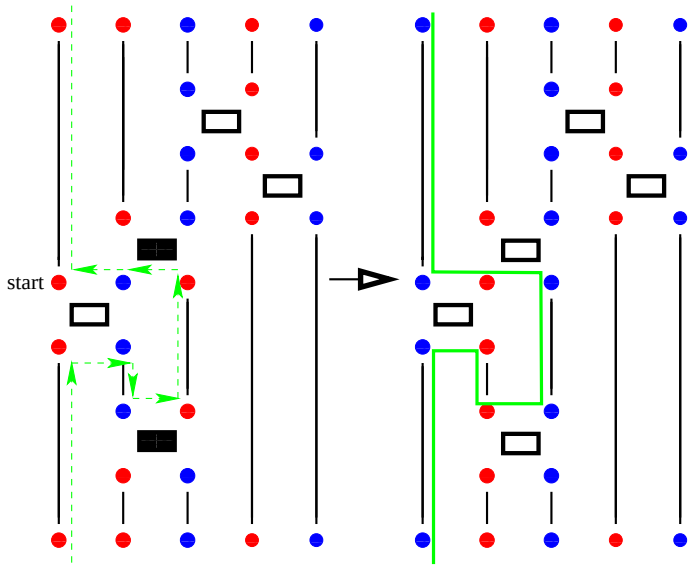

Figure 14: Construction of a loop and its flip

As you can see in figure 14 all spins along the loop have been flipped. Also the two off-diagonal operators $\hat{H}_{2,b}$ on the right side of the loop have been transformed into diagonal operators $\hat{H}_{1,b}$. The diagonal operator $\hat{H}_{1,b}$ on the left side seems to have been unchanged which would be a fallacy. In fact this operator has been visited twice by the loop which equivalent to the transformation $\hat{H}_{1,b} \rightarrow \hat{H}_{2,b} \rightarrow \hat{H}_{1,b}$ which means this operator has been transformed twice and thus it ended up as the same operator as before the loop. If you have a look at the top and bottom spin rows, which corresponds to $\langle\alpha|$ and $|\alpha\rangle$ you can see a change in the state $|\alpha\rangle$. Thus the off-diagonal update not only considers the off-diagonal operators, but also moves through different states $|\alpha\rangle$. That way, for long enough runs, all (or at least a representative amount) possible configurations were sampled according to the statistical weights and thus the interesting averages can be calculated by just saving the data after each full step (diagonal and off diagonal updates) and dividing it by the number of steps. The off-diagonal update is complete when all vertex legs had been used once in a loop. Now only the value of the cutoff M has to be determined which is done in the next section.

7.2.5 Determining M

This chapter will deal in detail with the cutoff M. In general for a truncation of a series any cutoff can be taken which is sufficiently big to lead to a neglectable deviation from the full series. On the other hand this truncation is used in a computer simulation and should be as low a possible to accelerate the simulation. Therefore a compromise for a lower limit has to be found which leads to a small enough error and guarantees at the same time a satisfying simulation performance. Before it will be explained how to apply this in an algorithm let's have a deeper look onto M itself.
M is the length of the operator strings. At the very beginning this operator string is filled with unit operators only. The probabilities of adding and removing a diagonal operator $\hat{H}_{1,b}$ is given by (63) and (64). It is obvious to see that the number of non unit operators n depends on the temperature and one can imagine that there will be some distribution around an average value for a particular temperature. In figure 15 three histograms were plotted over each other to compare them. They result from three simulations with identical parameters except M.

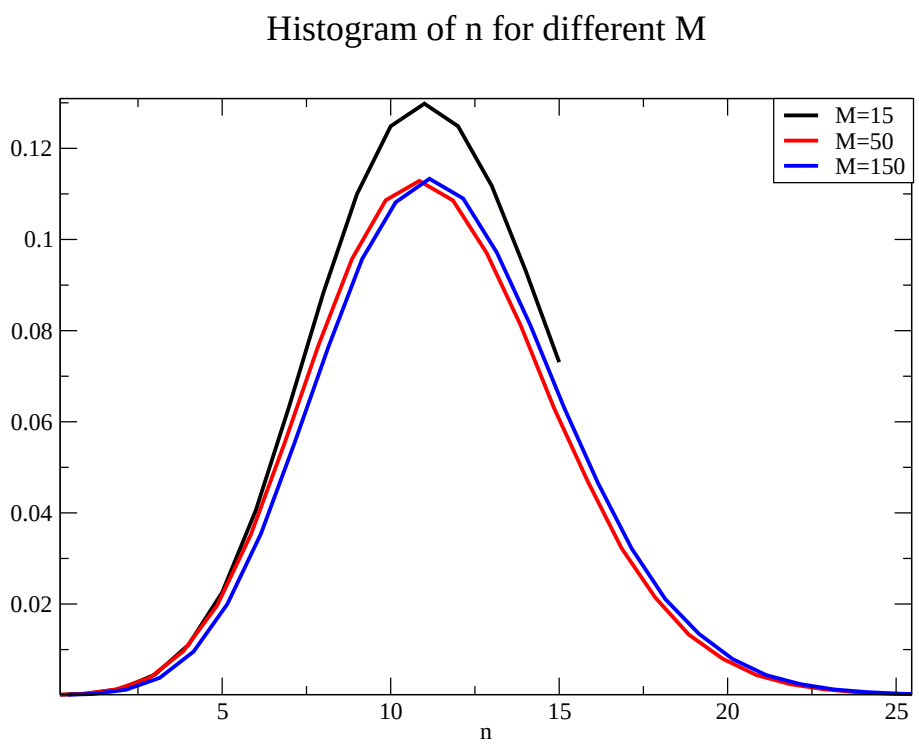

Figure 15: Histogram of n for different M=15,50,150

For better a better overview the curves for $M = 50$ and $M = 150$ have been shifted, else they would have been congruent. As you can see from figure 15 if a too low cutoff ($M = 15$) is chosen the distribution for n is not fully considered and this would obviously lead to wrong results. On the other hand once M is higher than the maximum occurring n it makes no difference for the distribution how large M is (no difference between $M = 50$ and $M = 150$). Hence the aim is to find a cutoff large enough to contain all occurring n, but small enough to avoid unnecessary computation time in the simulations. This is done by following condition:

$$M - n_{max} > \frac{n_{max}}{a} \tag{67}$$

As long as this condition holds, M doesn't need to be changed. a is a constant smaller or equal than the starting value of M. (67) leads to a cutoff M which is a fraction (depends on a) higher than the maximum n. In principle the condition $M = n_{max} + 1$ would be the best if someone would know the maximum n from the beginning. Using the smallest possible cutoff would lead to the best computational performance, but because the maximum n is usually not known from the beginning it would even slow down the simulation. The reason for this slowing down can be understood with following diagram of the algorithm:

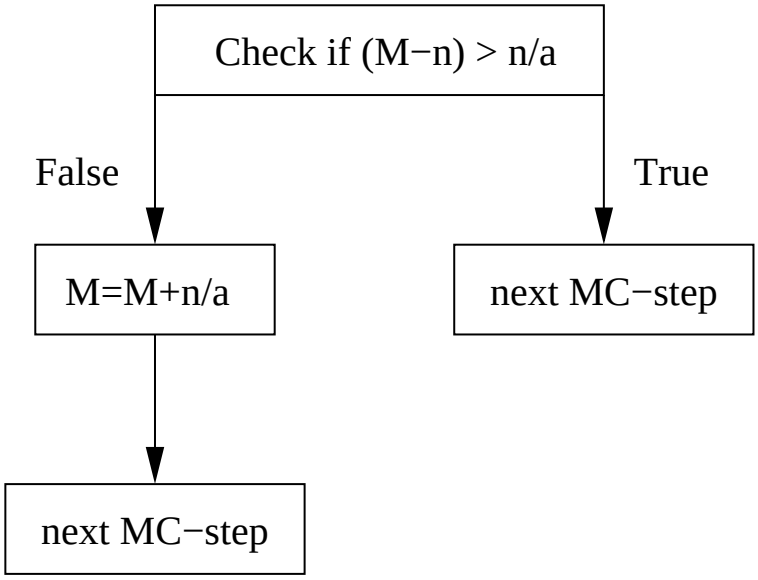

Figure 16: Control of M in the algorithm

Notice that this time n was used instead of n_{max}. That means, that after every MC update step a check of condition (67) is performed. This is the reason for the slowing down by trying to find the lowest possible cutoff. In

the algorithm M is determined dynamically, because usually n_{max} is not known. Because the statistical weights and probabilities change with M the stored data should usually be reseted after each M update until a stable cutoff has been found. If someone would search the ideal cutoff by increasing M by 1 lots of restarts will be the consequence while with condition (67) the increasing steps are higher and therefore a satisfying cutoff is found more quickly. Thus even if the M found that way is higher than the ideal one ($M = n_{max} + 1$) the increased computational time is often compensated by the faster convergence of M. Figures 17-20 show an example of how M develops.

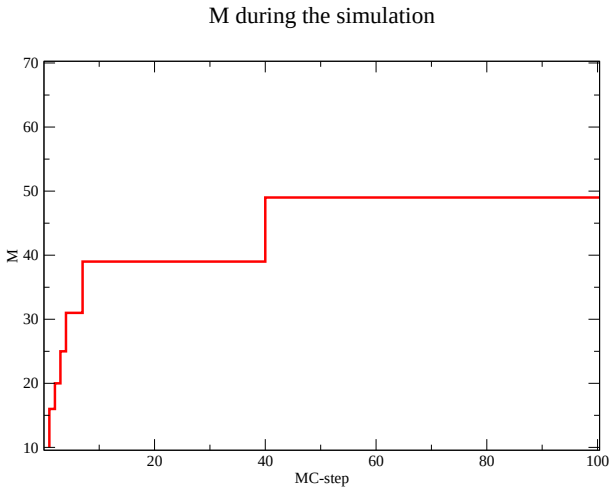

Figure 17: Development of M

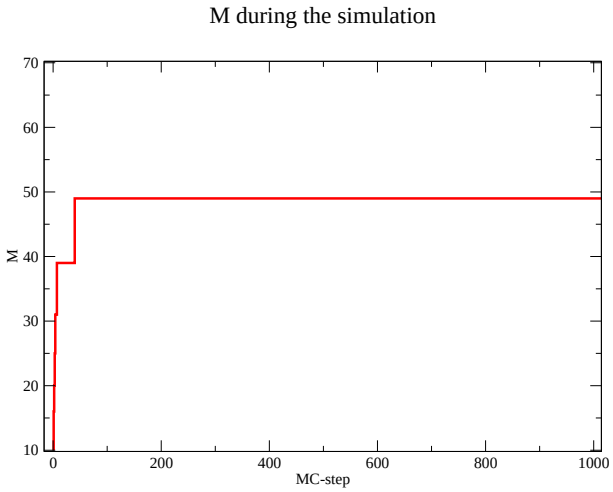

Figure 18: Development of M

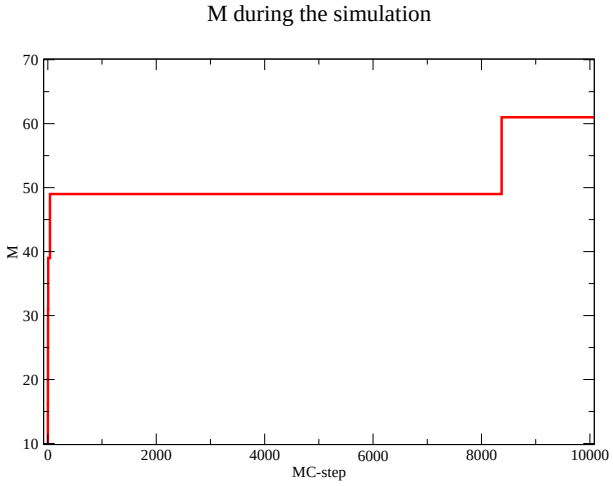

Figure 19: Development of M

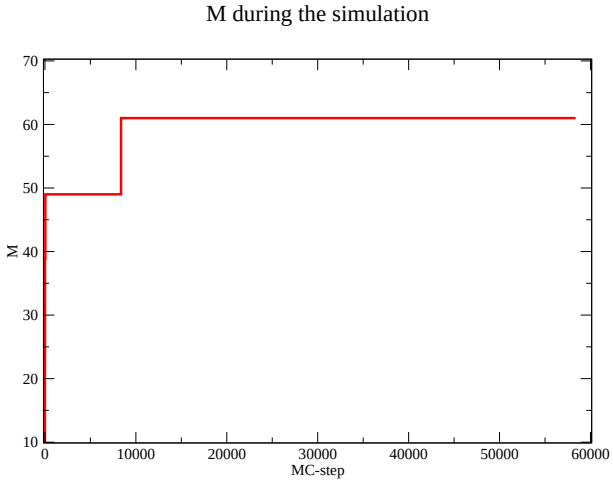

Figure 20: Development of M

As you can see M raises quickly in the beginning and slows down the closer it gets to the cap of n. Because of this behavior it's even better to understand that a slow step by step increase of M would take very long until it reaches a stable value which would cause a huge slowdown. A value of 4 for the constant a is quite common.
Knowing the algorithm how to update the cutoff M is just half the way. Raising means means increasing the length of the operator strings by just appending $M_{new} - M_{old}$ unit operators. From the graphical representation one can see it will not only be enough to just add those *empty* operator string positions, because one also needs to keep track of the spin rows between them. Because only unit operators were added this is simply done by just

carrying forward the spin state $|\alpha\rangle$ at M_{old} to the $M_{new} - M_{old}$ new spinrows. Then only the data needs to be saved before the next MC-step starts.

7.3 Simulation and Results

With the above algorithm I simulated a system out of 10 spins. A plot of the specific heat can be seen in figure 21.

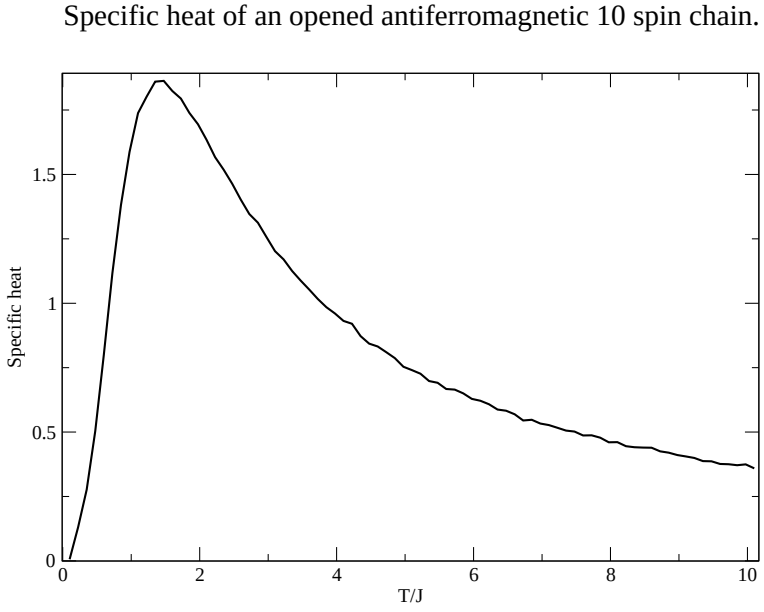

Figure 21: Specific heat of an opened antiferromagnetic 10 spin chain

The same way how the algorithm was developed for the antiferromagnetic system it can be done for the ferromagnetic system. In fact if only the sign of the coupling constant is changed and the absolute value remains the same the only difference resides in concerning only parallel instead of antiparallel spins for adding diagonal operators and in using the selection rule 5) (in both directions) in figure 13. Also closed chains are easy to realize by just adding the bond between the first and the last spin. Notice that these changes are only a few changes in algorithm while all probabilities of adding or lowering the non unit operator number n and for the loop updates remain the same. I simulated with this, still simplified, model different chain lengths for opened and closed chains to check when they start to be identical. In figure 22 the specific heat for the different chains are plotted. The full lines represent the opened chains and the dotted lines the closed ones. One can see for longer chains that the closed and open ones seem to be shifted by the length of 1 spin. The reason for this is the fact that not the number of spins, but the number of bonds is the important factor. If N is the number

of spins then there are $N-1$ bonds in open chains while there are N bonds for closed chains. Thus if you label the plots by their bond number you see opened and closed turn out to be the same at larger N.

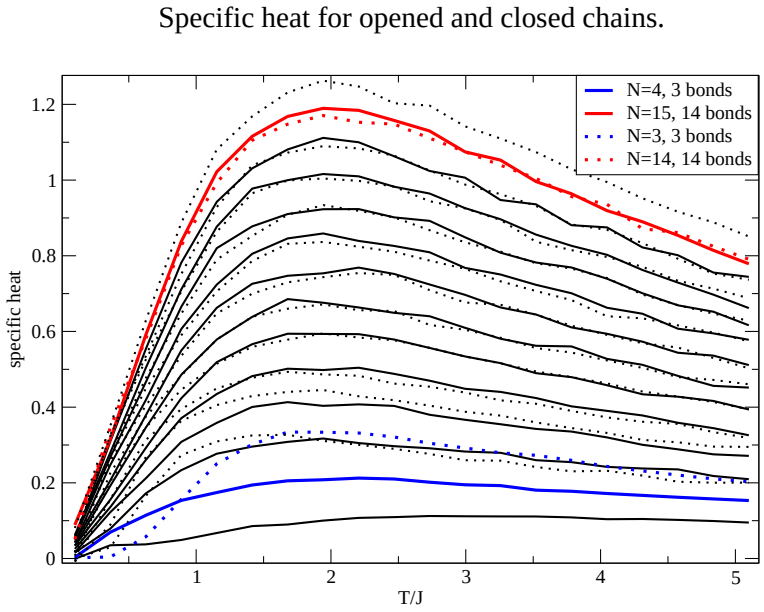

Figure 22: Specific heat of opened and closed chains with different length

8 QMSSE in more general Heisenberg systems

8.1 Theory

The purpose of Chapter 7 was to give an idea about the algorithm and the theory behind it. Now with the basic understanding of the QMSSE method, more complex Heisenberg systems can be discussed.

8.1.1 Statistical weight and probabilities

Let's have a look at the statistical weight before we simplified the model which was given by equation (52):

$$W(\alpha, \{a, b\}) = \frac{\beta^n (M-n)!}{M!} \langle \alpha | \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} | \alpha \rangle$$

The notations for n as the number of non unit operators and M as the cutoff hold for all systems and therefore the fraction term remains the same as in the simplified model. However the product term needs some more detailed treatment. In a more general system the eigenvalues of the single bond operators differ and therefore it is not enough anymore to keep track of the number of non unit operators only, but also to keep track of their composition. For that purpose a new occupation number n_a is introduced for each Operator type $\hat{H}_{a,b}$. n is then given by

$$n = \sum_{a=1}^{a_{max}} n_a \tag{68}$$

With the help of these occupation numbers the product in (52) can be rewritten as (69). Note that H_a are the eigen values of the operators $\hat{H}_{a,b}$.

$$\langle \alpha | \prod_{p=1}^{M} \hat{H}_{a(p),b(p)} | \alpha \rangle = \prod_{a=1}^{a_{max}} H_a^{n_a} \tag{69}$$

By using equation (69) the general statistical weight can be written in terms of numbers only (70) which will be the base to calculate the transition probabilities.

$$W(\alpha, \{a, b\}) = \frac{\beta^n (M-n)!}{M!} \prod_{a=1}^{a_{max}} H_a^{n_a} \tag{70}$$

With this modified weight and (62) the transition probabilities for raising or lowering a particular n_a ($n_a \longrightarrow n_a + 1$ and $n_a \longrightarrow n_a - 1$) are given below by (71) and (72)

$$P_{accept}(n_a \to n_a + 1) = min(\frac{B\beta H_a}{M-n}, 1) \tag{71}$$

$$P_{accept}(n_a \to n_a - 1) = min(\frac{M-n+1}{B\beta H_a}, 1) \tag{72}$$

In chapter 7 the off-diagonal updates had been performed by using the only possible selection rule to transform the operators during a loop update and accept those loop flips with a probability of $\frac{1}{2}$. In more general systems there are more selection rules and therefore more than just one possible operator transition is possible while the loop flip probability remains $\frac{1}{2}$. More about these selection rules is written in the next section.

8.1.2 Vertices and selection rules

In chapter 7 only two non unit operators occurred. Thus only the two vertex-transformations explained in figure 13 (Chapter 7.2.4) existed. In a more general heisenberg model with non vanishing operators between parallel spins it is obvious that there are more than just two different vertex transformations and hence there is more than just one possible transition at an operator in the loop updates (From figure 13 transitions 1, 3, 4 and 5 are allowed in a more general heisenberg model). This leads to a new problem on how to determine which transitions will be chosen at each operator passed during a loop update. The solution is to weight those transitions by their statistical values which leads to a transition probability only depending on the concerning eigenvalues:

$$P_{H_i \longrightarrow H_f} = \frac{W(H_f)}{\sum_k W(H_k)} \tag{73}$$

The $W(H_i)$ are the statistical weights of the configuration after the transition where f denotes the chosen state and k all possible states according to the selection rules. Inserting (70) results in

$$P_{H_i \longrightarrow H_f} = \frac{\frac{\beta^n (M-n)!}{M!} \prod_{a=1}^{a_{max}} H_a^{n_a(f)}}{\sum_k \frac{\beta^n (M-n)!}{M!} \prod_{a=1}^{a_{max}} H_a^{n_a(k)}} \tag{74}$$

Since M,n and β are not changed the terms with the factorials cancel each other out what leads to

$$P_{H_i \longrightarrow H_f} = \frac{\prod_{a=1}^{a_{max}} H_a^{n_a(f)}}{\sum_k \prod_{a=1}^{a_{max}} H_a^{n_a(k)}} \tag{75}$$

Because an operator transition just changes two occupation numbers by increasing one by 1 and decreasing the other one by 1 the final probability of choosing a particular transition is given by

$$P_{H_i \longrightarrow H_f} = \frac{H_f}{\sum_k H_k} \tag{76}$$

8.1.3 Implementation of an external field h

To implement an external field is more tricky than it looks at first. The general heisenberg hamiltonian with an external field h is given by

$$\hat{H} = \sum_{i \neq j} J_{ij}[S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)] - h \sum_i S_i^z \tag{77}$$

Because we work with single bond operators the external field has to be carried over on those bonds. This is done by dividing the energy resulting from the external field at a lattice site through the number of bonds that lattice side has got (not a border site) [4]. Thus the hamiltonian is given by

$$\hat{H} = \sum_{i \neq j} J_{ij}[S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)] - \frac{h}{2d} \sum_i S_i^z \tag{78}$$

The Zeeman term does not change a state $|\alpha\rangle$ and hence it is a diagonal term. Notice that the Zeeman term will not lead to an own single bond operator, but rather add up to the other diagonal operators which will be shown in section 8.1.4. Additionally note that first with the use of an external field the quantization axis is defined and thus the directions for spin up and spin down are determined which can clearly be seen in the energy splitting of parallel spin up and spin down neighbors.

8.1.4 Example of a more general heisenberg system

For a better understanding on how to build up the single bond operators and its eigenvalues an example is given as an anisotropic Heisenberg system with a Zeeman term. Further this system is chosen to be ferromagnetic ($J_{i,j} < 0$) and to make it more manageable to be in two dimensions. This leads to two different coupling constants (it is assumed that coupling constants $J_{i,j}$ are constant along an axis) which are denoted by J_1 and J_2. The hamiltonian is then given by

$$\hat{H} = J_1 \sum_{i \neq j}^{x} [S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)] +$$

$$J_2 \sum_{i \neq j}^{y} [S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)] - \frac{h}{4} \sum_{i \neq j} (S_i^z + S_j^z)$$

The x and y above the sums stands for the different bond directions of neighbor spins. Again the hamiltonian must be split up into single bond

operators. In the table below you can see all possible operators and what happens to the states they are acting on.

operator	acts on	results in	eigenvalue
$\hat{H}_{1,b} = -J_1 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z)$	$\|\uparrow,\uparrow\rangle$	$\|\uparrow,\uparrow\rangle$	$H_1 = -\frac{J_1}{4} + \frac{h}{4}$
$\hat{H}_{2,b} = -J_1 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z)$	$\|\downarrow,\downarrow\rangle$	$\|\downarrow,\downarrow\rangle$	$H_2 = -\frac{J_1}{4} - \frac{h}{4}$
$\hat{H}_{3,b} = -J_1 S_i^z S_j^z$	$\|\uparrow,\downarrow\rangle$	$\|\uparrow,\downarrow\rangle$	$H_3 = \frac{J_1}{4}$
$\hat{H}_{4,b} = -\frac{J_1}{2}(S_i^+ S_j^- + S_i^- S_j^+)$	$\|\uparrow,\downarrow\rangle$	$\|\downarrow,\uparrow\rangle$	$H_4 = -\frac{J_1}{2}$
$\hat{H}_{5,b} = -J_2 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z)$	$\|\uparrow,\uparrow\rangle$	$\|\uparrow,\uparrow\rangle$	$H_5 = -\frac{J_2}{4} + \frac{h}{4}$
$\hat{H}_{6,b} = -J_2 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z)$	$\|\downarrow,\downarrow\rangle$	$\|\downarrow,\downarrow\rangle$	$H_6 = -\frac{J_2}{4} - \frac{h}{4}$
$\hat{H}_{7,b} = -J_2 S_i^z S_j^z$	$\|\uparrow,\downarrow\rangle$	$\|\uparrow,\downarrow\rangle$	$H_7 = \frac{J_2}{4}$
$\hat{H}_{8,b} = -\frac{J_2}{2}(S_i^+ S_j^- + S_i^- S_j^+)$	$\|\uparrow,\downarrow\rangle$	$\|\downarrow,\uparrow\rangle$	$H_8 = -\frac{J_2}{2}$

The operators have been labeled the same way as their eigenvalues which makes it more obvious to see that an operator type also depends on the spins that it acts on (there is a difference between parallel up and down spins now). Remember the condition $\hat{H}_{a,b} > 0$ for diagonal operators. To fulfill this condition the zero point of the hamiltonian has to be shifted accordingly. Remember also that for a ferromagnetic system the coupling constants J_1 and J_2 are positive. The minimum shift to assure positive eigenvalues is $C = -\frac{J_1}{4} - \frac{J_2}{4} + \frac{h}{4}$ and thus the final usable operators and their eigenvalues are

operator	acts on	results in	eigenvalue
$\hat{H}_{1,b} = -J_1 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z) + C$	$\|\uparrow,\uparrow\rangle$	$\|\uparrow,\uparrow\rangle$	$-\frac{J_1}{2} - \frac{J_2}{4} + \frac{h}{2}$
$\hat{H}_{2,b} = -J_1 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z) + C$	$\|\downarrow,\downarrow\rangle$	$\|\downarrow,\downarrow\rangle$	$-\frac{J_1}{2} - \frac{J_2}{4}$
$\hat{H}_{3,b} = -J_1 S_i^z S_j^z + C$	$\|\uparrow,\downarrow\rangle$	$\|\uparrow,\downarrow\rangle$	$-\frac{J_2}{4} + \frac{h}{4}$
$\hat{H}_{4,b} = -\frac{J_1}{2}(S_i^+ S_j^- + S_i^- S_j^+)$	$\|\uparrow,\downarrow\rangle$	$\|\downarrow,\uparrow\rangle$	$\frac{J_1}{2}$
$\hat{H}_{5,b} = -J_2 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z) + C$	$\|\uparrow,\uparrow\rangle$	$\|\uparrow,\uparrow\rangle$	$-\frac{J_2}{2} - \frac{J_1}{4} + \frac{h}{2}$
$\hat{H}_{6,b} = -J_2 S_i^z S_j^z + \frac{h}{4}(S_i^z + S_j^z) + C$	$\|\downarrow,\downarrow\rangle$	$\|\downarrow,\downarrow\rangle$	$-\frac{J_2}{2} - \frac{J_1}{4}$
$\hat{H}_{7,b} = -J_2 S_i^z S_j^z + C$	$\|\uparrow,\downarrow\rangle$	$\|\uparrow,\downarrow\rangle$	$-\frac{J_1}{4} + \frac{h}{4}$
$\hat{H}_{8,b} = -\frac{J_2}{2}(S_i^+ S_j^- + S_i^- S_j^+)$	$\|\uparrow,\downarrow\rangle$	$\|\downarrow,\uparrow\rangle$	$\frac{J_2}{2}$

8.2 Differences in the algorithm

This section will point out the differences in the algorithm compared to the simplified Heisenberg system from chapter 7. The differences will not be huge, but be the kind of that can slow down the simulation a lot (IF statements; drawing a random number).

8.2.1 Diagonal updates

The basis of the algorithm remains the same except one addition. In the simple heisenberg system only one type of diagonal operator occurred. In more general systems more diagonal operator types occur which can clearly be seen from 8.1.4. Therefore at each diagonal update step additionally to checking whether there is already an operator and how the spins at the selected bond are aligned one has to determine which kind of diagonal operator can be inserted or removed. This additional check can be done either by IF statements which would lead to a slowing down of the simulation, or better by a quite complex mathematical formulation.

8.2.2 Off-diagonal updates

The beginning of the off-diagonal updates is the construction of the linked vertex list. The list is constructed by just linking vertex legs horizontally as in figure 12. It doesn't matter which nun unit operator type is at what vertex and therefore there is no difference in the algorithm compared to the simple model of creating the linked vertex list. However in the loop updates a change has to be made resulting from the change of the selection rules discussed in 8.1.2. Each time during the loop construction when an operator is passed the transformed operator has to be sampled according to (76). This is done by assigning each possible sampled operator a numerical interval between $[0; 1]$ with respect to its ratio to the other possible operators thus the sum over all those newly created intervals yields the full interval $[0; 1]$. Then a random number in $[0; 1]$ is generated and compared to the ranges of those generated intervals. The operator connected to the interval the random number is located in is chosen and the loop proceeds to the next operator. One can imagine that drawing that many random numbers are causing a huge slow down compared to the simple model where just one transition was allowed.
Also note that with the loop algorithm according to [6],[5] each vertex leg can only be visited once and thus it might be possible that even if in general there might be three possible operator transitions there might only be one allowed left. Therefore an additional check is necessary which also causes longer run times.

8.3 Simulation and Results

I have done a view different simulations for an anisotropic two dimensional Heisenberg system with different spin configurations and different external fields. The simulated systems are toy models and therefore the constants are seen as dimensionless numbers and the results are only useful to recognize some trends and are not really comparable to any experiments. For that purpose one has to extend the algorithm in a way to take also more than just next neighbor interaction into account Additionally constants or at least the right dimensions (one can normalize results by plotting ratios) have to be considered.

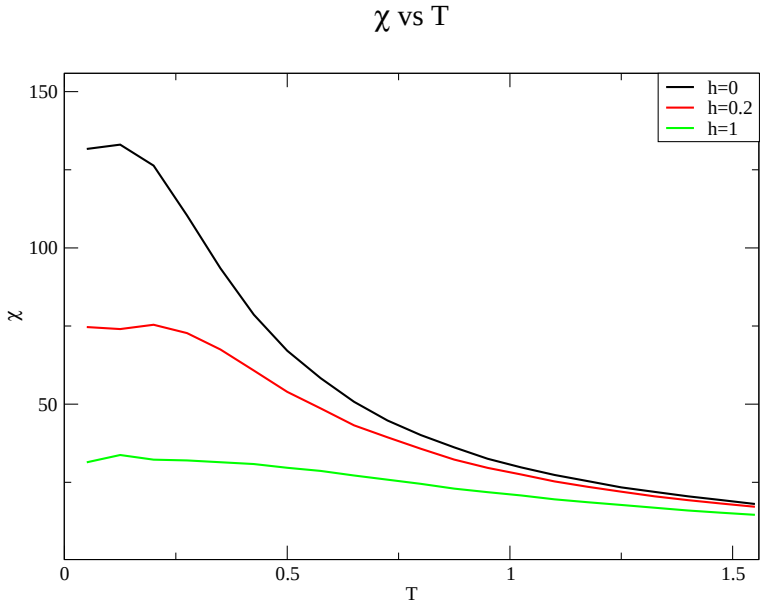

Figure 23: Simulation of two spin chains each with 5 spins with different external fields.

In figure 23 three simulations of a ferromagnetic 5×2 system have been performed with different external fields. The coupling constant between the spins inside each chain is ten times higher than the coupling constant between the chains. The trend is that for higher external fields the susceptibility drops. This makes sense since the susceptibility is the response of the system on a change of the external field. The higher the external field the more arranged is the system already to it. Then a small change of the field will result in a small response of the system and hence to a low susceptibility. Also one can see that for high temperatures the system behaves classically.

9 Lookout

9.1 Vertices

In general all vertex paths are possible (depending on the model). What I haven't mentioned yet is the possibility of a bouncing. Bouncing means that the entrance leg of a vertex is identical with the exit leg. Thus the operator at the concerning vertex remains the same and so does the spin at the entrance/exit leg (two spin flips lead to the same spin). This possible vertex path usually has to be considered, but can be neglected in some special cases, like in the simplified model [4]. Nevertheless I want to give an overview of all possible vertex paths we had so far (fig. 24).

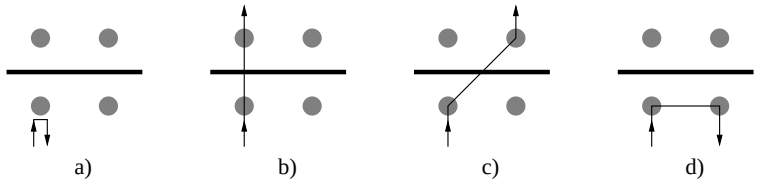

Figure 24: Different processes for single bond operators in a vertex representation. a) is a bouncing path that leaves the spin and the operator unchanged which refers to a 'bounce' process. b) 'continue-straight' c) 'switch-and-continue' d) 'switch-and-reverse'

By exchanging the gray circles with possible spin configurations you receive the possible operator transitions. Those transitions correspond to the ones seen in former chapters already and only allow operator transitions between operators with the difference of two spins. There are also algorithms that consider transitions with more than just two spin flips which leads to the multi-branch-cluster updates. This has to interpreted as entering at one leg and exiting out of the other 3 legs and so the loop generates branches (therefore the name).

9.2 Plaquettes

The QMSSE method does not only work for single bonds, but it also holds for models where interactions between more than just two lattice sites are allowed. One then is talking about plaquettes instead of bonds. For example plaquettes with four spins are given in figure 25.

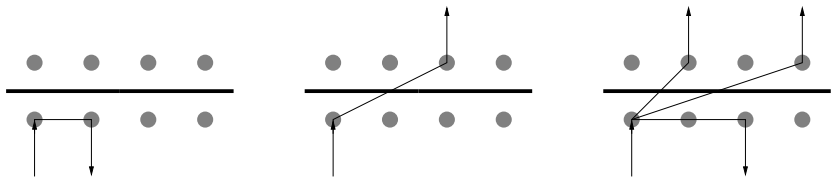

Figure 25: Examples for vertices for plaquette operators. The one on the right side corresponds to a multi-branch process

For more insight in plaquette operators [1] gives a nice introduction.

10 Conclusion

QMSSE is a powerful tool in todays numerical methods of simulating physical systems, because its possibilities are huge. Any system and any model can be simulated where a lattice can be constructed and controlled, where bond or plaquette operators can be defined and the statistical weights of possible space configurations are positive. Problems arise for long calculation times simulating systems at very low temperatures which leads to large numbers of non unit operaters, but this is compensated with the rising power of computers. On the other hand known problems that occur in other numerical methods, like the critical slowdown, are avoided in QMSSE. Altogether QMSSE is the numerical method currently used in science to simulatequantummechanicall systems and their properties.

A Abbreviations and definitions

RNG:

RNG(Random Number Generator) is an algorithm which produces so called pseudo-random numbers. By a chosen starting value, called Seed, this algorithm gives a sequence of varying numbers that produces a huge amount of different numbers until the sequence starts from the beginning again. Those numbers should fill the chosen interval ergodically. Every Seed starts this sequence from a different point and lead therefore to different runs of simulations. Since numbers are varying like they were random you call them pseudo-random numbers. For long simulations you should use a good RNG which runs through a long sequence of numbers to avoid the sequence to restart within your simulation.

MC:

MC (Monte Carlo) is a simulation method based random numbers. The name really comes from the city and it's casinos in which the importance of random numbers should be reflected.

Metropolis-Algorithm:

Unlike the Monte-Carlo Method, the name doesn't derive from the meaning of a huge city even if it would be a nice way to explain this method, by transferring the skyline of a metropolis(High in the center, low at the edges) into histogram. Instead it is named after the mathematician Nicolas Metropolis, one of the first people who developed that algorithm. In this algorithm the random numbers are being sampled by considering statistical weights. For example if you try to hit the bulls-eye on a dart-board, throw lots of darts and draw a histogram with sampling the number of hits dependent on the distance from the bulls-eye you will end up with a higher density near the middle and with dying away densities the further away from the middle. So if you would make a computer simulation for such dart throws you would need to sample your random numbers according to that histogram. Same holds for Boltzmann distributed systems as will be used later on in this work.

SSE:

SSE (Stochastic Series Expansion) is a method to do numerical simulations for exponential functions which are not easy to solve or not solveable at all. This method is used for Quantum-Mechanical systems where you need to consider all states and have to calculate a trace over an exponent. This leads to QM-SSE.

Update:

If you talk about an *update* in simulations a step is meant in that simulation that can change (but doesn't need to change) the configuration. For example this can be the possible flipping of a spin.

B FORTRAN code - Open spin chain in the simplified Heisenberg system

```
      PROGRAM quchain

      INTEGER bond,D1,D2,n,
     1p3stroke,vspin,opcounter,initcounter,initcap
     2M,p,a_type,seed,spin,spin_old,loopcounter,
     3opstring,opstring_old,bondnumber,average,
     4spin_i,spin_j,vertexlist,vcheck,flip,v,
     5counter,pstroke,p2stroke,step,steps,averages,
     6cutconstant,magn1,magn2,j,magn,M_counter,
     7bond_old,a_type_old,vsum,spinopcheck,nold
     8v_step,vstroke,v2stroke,v_start,vertexpos

      REAL RND

      DOUBLE PRECISION temp,beta,kb,check,P_accept,
     1T_begin,T_end,T_step,Z,avmagn,Weight,avn,
     2avsqun,cv,cv_array,T_array,S,Jx,Jy,h,H1,H2,
     3H3,H4,H5,H6,H7,H8,Typecheck,
     4avmagn_array,Weightfactor,
     5avsqumagn,Chi,Chi_array

      DIMENSION bond(1:10000),a_type(1:10000),
     1spin(1:100,0:10000),spin_old(1:100,0:10000),
     2opstring(1:10000),vertexlist(1:40000),
     3vertexpos(1:40000),vcheck(1:40000),
     4opstring_old(1:10000),cv_array(1:100000),
     5T_array(1:100000),bond_old(1:10000),
     6a_type_old(1:10000),spinopcheck(1:100),
     7avmagn_array(1:100000),Chi_array(1:100000)

c---------- PARAMETERS ------------------------

      WRITE(*,*) START
      WRITE(*,*) SEED=
      READ(*,*),seed

      DO D1=6,20

      cv_array=0
```

```
      T_array=0
      avmagn_array=0
      Chi_array=0

      T_begin=0.1
      T_end=5.1
      T_range=20
      M=10
      cutconstant=4
      steps=50000
      averages=20
      initcap=1000
      kb=0.5
      Jx=1

      H1=Jx/2
      H8=Jx/2
      H3=0
      H6=Jx/2
c---------------------------------

      OPEN (UNIT=1,STATUS=UNKNOWN,FILE=chi.dat)
      OPEN (UNIT=2,STATUS=UNKNOWN,FILE=c.dat)
      OPEN (UNIT=3,STATUS=UNKNOWN,FILE=S.dat)

 10   FORMAT(1X,2I8)
 20   FORMAT(1X,2F20.10)
 30   FORMAT(1X,1F20.10)

c-------------------------------------------------

      T_step=(T_end-(T_begin*1.001))/(T_range-1)
      bondnumber=D1-1
      average=1

c--------------- START AVERAGES-----------------
      DO WHILE(average.le.averages)
      counter=1
      temp=T_end
      M=10
      bond=0
      a_type=0
      opstring=0
      spin=0
      vertexlist=0
```

```
      vcheck=0
      vertexpos=0
      spinopcheck=0
      n=0
c------------ starting config ----------------------

 50   CONTINUE

      DO i=1,D1
        check = RND(seed)
        IF (check.lt.0.5) THEN
          spin(i,0)=1
        ELSE
          spin(i,0)=-1
        ENDIF
        DO p=1,M
          spin(i,p)=spin(i,0)
        ENDDO
      ENDDO

c***************** Start MC steps ********************

      DO WHILE(temp.ge.T_begin)

      beta=1/(temp*kb)
      Z=0
      avmagn=0
      avsqumagn=0
      avn=0
      avsqun=0
      step=1
      initcounter=0

      DO WHILE(step.le.steps)

      IF (MOD(step,10000).eq.0) THEN
        WRITE(*,*) Step: ,step,/,steps,          Temp: ,
     1T_end,/,temp,/,T_begin
        WRITE(*,*)Run: ,average,/,averages,
     1Chain lenght: ,D1
      ENDIF

c-------------- DIAGONAL UPDATES ---------------------

      DO p=1,M
```

```
   IF (opstring(p).eq.0) THEN
        check=RND(seed)*bondnumber
        bond(p)=check-MOD(check,1.0)+1
        spin_i=bond(p)
        spin_j=spin_i+1

        IF (spin(spin_i,p)+spin(spin_j,p).eq.0) THEN
          GOTO 100
        ENDIF

        P_accept=bondnumber*beta/(M-n)*(
1H1*(1+spin(spin_i,p))/2+
3H8*(1-spin(spin_i,p))/2)
        P_accept=MIN(P_accept,1.0)
        check=RND(seed)

        IF (P_accept.lt.check) THEN
          bond(p)=0
          GOTO 100
        ENDIF

        opstring(p)=8*(bond(p)-1)+(0.5*(spin(spin_i,p)+1)
1-4*(spin(spin_i,p)-1))
        a_type(p)=opstring(p)-8*(bond(p)-1)
        n=n+1

   ELSE IF (a_type(p).ne.6) THEN

          spin_i=bond(p)
          spin_j=spin_i+1
          P_accept=(M-n+1)/beta/bondnumber*(
11/H1*(1+spin(spin_i,p))/2+
31/H8*(1-spin(spin_i,p))/2)
          P_accept=MIN(P_accept,1.0)
          check=RND(seed)

          IF (P_accept.ge.check) THEN

              n=n-1

              bond(p)=0
              opstring(p)=0
              a_type(p)=0

          ENDIF
```

```
        ENDIF

 100    CONTINUE

      ENDDO
c-------------------------------------------------

      DO j=1,4*M
        vertexpos(j)=0
        vcheck(j)=0
        vertexlist(j)=0
      ENDDO

c------------ OFF-DIAGONAL UPDATES -------------------

c - - - - - - - - Constructing vertexlist - - - - - - -

      DO p=1,M

        IF (opstring(p).eq.0) THEN
          GOTO 150
        ENDIF

        spin_i=bond(p)
        spin_j=spin_i+1
        vertexpos(4*(p-1)+1)=spin_i
        vertexpos(4*(p-1)+3)=spin_i
        vertexpos(4*(p-1)+2)=spin_j
        vertexpos(4*(p-1)+4)=spin_j

 150    CONTINUE

      ENDDO

      DO p=1,M
        v=4*(p-1)+1
        int=-1

        IF (vertexpos(v).eq.0) THEN
          GOTO 300
        ENDIF

 200    CONTINUE

        IF (vcheck(v).eq.0) THEN
          vcheck(v)=1
```

```
          i=v+int

          IF (i.eq.0) THEN
            i=i+4*M
          ENDIF
          IF (i.gt.4*M) THEN
            i=i-4*M
          ENDIF

          DO WHILE(vertexpos(i).ne.vertexpos(v))
            i=i+int

            IF (i.le.0) THEN
              i=i+4*M
            ENDIF
            IF (i.gt.4*M) THEN
              i=i-4*M
            ENDIF
          ENDDO

          vertexlist(v)=i
          vertexlist(i)=v
          vcheck(i)=1
          int=-int
          IF (MOD(i,2).eq.0) THEN
            v=i-1
          ELSE
            v=i+1
          ENDIF
          GOTO 200
        ENDIF

 300    CONTINUE
      ENDDO

c - - - - - - - - - - - - - - - - - - - - - - - - - - -

c - - - Safe Path, undo if P-Accept-check failed - - - -

      DO j=1,4*M
        vcheck(j)=0
      ENDDO

      v_step=1
      vstroke=1
```

```
 DO WHILE (vstroke.le.4*M)

    DO j=1,M
    opstring_old(j)=opstring(j)
    DO k=1,D1
    spin_old(k,j)=spin(k,j)
    ENDDO
    bond_old(j)=bond(j)
    a_type_old(j)=a_type(j)
    ENDDO
    DO k=1,D1
       spin_old(k,0)=spin(k,0)
    ENDDO

    v=vstroke
    v_start=v
    opcounter=0

    IF (vertexlist(v).eq.0.OR.
1vcheck(v).eq.1) THEN
       GOTO 500
    ENDIF

    p2stroke=(v-1)/4+1
    v=vertexlist(v)
    vcheck(v)=1
    pstroke=(v-1)/4+1
    spin_i=bond(pstroke)
    spin_j=spin_i+1
    p3stroke=pstroke

    IF (MOD(v-1,4).lt.2) THEN
       i=pstroke-1
       p2stroke=p2stroke-1
       j=-1
    ELSE
       i=pstroke
       j=1
    ENDIF

    IF (p2stroke.eq.M) THEN
       p2stroke=0
    ENDIF
```

```
IF (i.lt.0) THEN
  i=i+M
ELSE IF (i.ge.M) THEN
  i=i-M
ENDIF

IF (i.eq.p2stroke) THEN
  IF (MOD(v-1,4).eq.0.OR.MOD(v-1,4).eq.2) THEN
    spin(spin_i,i)=-spin(spin_i,i)
    spin(spin_i,M)=spin(spin_i,0)
  ELSE
    spin(spin_j,i)=-spin(spin_j,i)
    spin(spin_j,M)=spin(spin_j,0)
  ENDIF
   i=i+j
ENDIF

IF (p2stroke.eq.M) THEN
  p2stroke=0
ENDIF

IF (i.lt.0) THEN
  i=i+M
ELSE IF (i.ge.M) THEN
  i=i-M
ENDIF

DO WHILE(i.ne.p2stroke)

  IF (MOD(v-1,4).eq.0.OR.MOD(v-1,4).eq.2) THEN
    spin(spin_i,i)=-spin(spin_i,i)
    spin(spin_i,M)=spin(spin_i,0)
  ELSE
    spin(spin_j,i)=-spin(spin_j,i)
    spin(spin_j,M)=spin(spin_j,0)
  ENDIF
  i=i+j
  IF (i.lt.0) THEN
    i=i+M
  ELSE IF (i.ge.M) THEN
    i=i-M
  ENDIF

ENDDO
```

```
    pstroke=p3stroke

    vspin=spin(vertexpos(v),pstroke)

    a_type(pstroke)=6*((a_type(pstroke)-6)*
1(a_type(pstroke)-1)/14-
2(a_type(pstroke)-6)*(8-a_type(pstroke))/35)+
3(a_type(pstroke)-1)*(8-a_type(pstroke))/10*
4((vspin+1)/2+(1-vspin)*4)

    opcounter=opcounter+1

    v=5-v+8*(pstroke-1)

    opstring(pstroke)=8*(bond(pstroke)-1)+a_type(pstroke)

    DO WHILE (v.ne.v_start)

    vcheck(v)=1
    p2stroke=(v-1)/4+1

    v=vertexlist(v)
    vcheck(v)=1
    pstroke=(v-1)/4+1

    spin_i=bond(pstroke)
    spin_j=spin_i+1

    p3stroke=pstroke

    IF (MOD(v-1,4).lt.2) THEN
      i=pstroke-1
      p2stroke=p2stroke-1
      j=-1
    ELSE
      i=pstroke
      j=1
    ENDIF

    IF (p2stroke.eq.M) THEN
      p2stroke=0
    ENDIF

    IF (i.lt.0) THEN
      i=i+M
    ELSE IF (i.ge.M) THEN
```

```
   i=i-M
ENDIF

IF (i.eq.p2stroke) THEN
   IF (MOD(v-1,4).eq.0.OR.MOD(v-1,4).eq.2) THEN
      spin(spin_i,i)=-spin(spin_i,i)
      spin(spin_i,M)=spin(spin_i,0)
   ELSE
      spin(spin_j,i)=-spin(spin_j,i)
      spin(spin_j,M)=spin(spin_j,0)
   ENDIF
    i=i+j
ENDIF

IF (p2stroke.eq.M) THEN
   p2stroke=0
ENDIF

IF (i.lt.0) THEN
   i=i+M
ELSE IF (i.ge.M) THEN
   i=i-M
ENDIF

DO WHILE(i.ne.p2stroke)

   IF (MOD(v-1,4).eq.0.OR.MOD(v-1,4).eq.2) THEN
      spin(spin_i,i)=-spin(spin_i,i)
      spin(spin_i,M)=spin(spin_i,0)
   ELSE
      spin(spin_j,i)=-spin(spin_j,i)
      spin(spin_j,M)=spin(spin_j,0)
   ENDIF
   i=i+j
   IF (i.lt.0) THEN
      i=i+M
   ELSE IF (i.ge.M) THEN
      i=i-M
   ENDIF

ENDDO

pstroke=p3stroke

vspin=spin(vertexpos(v),pstroke)
```

```
      a_type(pstroke)=6*((a_type(pstroke)-6)*
     1(a_type(pstroke)-1)/14-
     2(a_type(pstroke)-6)*(8-a_type(pstroke))/35)+
     3(a_type(pstroke)-1)*(8-a_type(pstroke))/10*
     4((vspin+1)/2+(1-vspin)*4)

      opcounter=opcounter+1
      v=5-v+8*(pstroke-1)
      opstring(pstroke)=8*(bond(pstroke)-1)+a_type(pstroke)

       ENDDO

      vcheck(v_start)=1

      P_accept=0.5

      check=RND(seed)

      IF(P_accept.lt.check) THEN

        DO j=1,M
        opstring(j)=opstring_old(j)
        DO k=1,D1
        spin(k,j)=spin_old(k,j)
        ENDDO
        bond(j)=bond_old(j)
        a_type(j)=a_type_old(j)
        ENDDO
        DO k=1,D1
          spin(k,0)=spin_old(k,0)
        ENDDO

      ENDIF

500    CONTINUE
      vstroke=vstroke+v_step
      v_step=4-v_step
    ENDDO


c- - - - - - - - - - - - - - - - - - - - - - - - - - - -

    magn=0
    DO i=1,D1
      magn=magn+spin(i,0)
```

```
      ENDDO

      avmagn=avmagn+ABS(magn)
      avsqumagn=avsqumagn+magn**2


      avn=avn+n
      avsqun=avsqun+n**2


c-.-.-.-.-.-.-.- DETERMINE NEW M-.-.-.-.-.-.-.-.-.-.-

      step=step+1

      initcounter=initcounter+1

      IF (initcounter.eq.initcap) THEN
        avn=0
        avsqun=0
        avmagn=0
        avsqumagn=0
        Z=0
        step=1
      ENDIF

      IF ((M-n).le.(n/cutconstant)) THEN
        DO i=1,n/cutconstant
          DO j=1,D1
            spin(j,M+i)=spin(j,M)
          ENDDO
        ENDDO
        avn=0
        avsqun=0
        avmagn=0
        avsqumagn=0
        Z=0
initcounter=0
        M=M+n/cutconstant
step=1
        WRITE(*,*)UPDATE - M/n ,M,n
      ENDIF

      IF (MOD(step,10000).eq.0) THEN
        WRITE(*,*)M/n: ,M,n
      ENDIF
```

```
      ENDDO

      avn=avn/steps
      avsqun=avsqun/steps
      cv=(1/beta)*(avsqun-avn**2-avn)
      cv_array(counter)=cv_array(counter)+cv/averages
      T_array(counter)=temp
      avmagn=avmagn/steps
      avsqumagn=avsqumagn/steps
      Chi=beta*(avsqumagn-avmagn**2)
      avmagn_array(counter)=avmagn_array(counter)+avmagn/averages
      Chi_array(counter)=Chi_array(counter)+Chi/averages
      counter=counter+1
      temp=temp-T_step

      ENDDO
c-----------end averages-----------

      average=average+1

      ENDDO
c---------------------------------

c************************************************

      S=0
      DO i=1,counter-1
         S=S+cv_array(counter-i)/T_array(counter-i)*T_step
         WRITE(UNIT=1,FMT=20),T_array(counter-i),
     1Chi_array(counter-i)
         WRITE(UNIT=3,FMT=20),T_array(counter-i),S
         WRITE(UNIT=2,FMT=20),T_array(counter-i),
     1cv_array(counter-i)
      ENDDO

 900  CONTINUE
c------ loop end for different N
      ENDDO
c-------------------------------
      END

c---------------------- RNG ----------------------------

      REAL FUNCTION RND(seed)
      IMPLICIT NONE
```

```
      INTEGER a,m,q,p,n,ndiv,j,k,seed
      REAL rm,rmax
c     m=2**31-1 and m=a*q+p
      PARAMETER (a=16807, m=2147483647, rm=1.0/m)
      PARAMETER (q=127773, p=2836, n=32, ndiv=1+(m-1)/n)
      PARAMETER (rmax=1.0-1.2e-7)
      INTEGER r(n),r0,r1
      SAVE r,r0,r1
      LOGICAL first
      DATA r/n*0/,first/.true./
c     initialize table of random numbers
      IF (first) THEN
        first=.false.
        r1=abs(seed)
        DO j=n+8,1,-1
          k=r1/q
          r1=a*(r1-k*q)-p*k
          if (r1.lt.0.) r1=r1+m
          if (j.le.n) r(j)=r1
        ENDDO
        r0=r(1)
      END IF
c     beginning when not initializing
c     compute r1=mod(a*r1,m) without overflows
      k=r1/q
      r1=a*(r1-k*q)-p*k
      if (r1.lt.0) r1=r1+m
      j=1+r0/ndiv
      r0=r(j)
      r(j)=r1
      rnd=min(rm*r0,rmax)
      END
```

References

[1] Roger G. Melko and Anders W. Sandvik. Stochastic series expansion algorithm for the s=1/2 xy model with four-site ring exchange. *Physical Review E*, 72:026702, 2005.

[2] P. Mohn. *Magnetism in the Solid State*. Springer Series in Solid-State Sciences , Volume 134, 2003.

[3] M.N. Rosenbluth A.H. Teller N. Metropolis, A.W. Rosenbluth and E. Teller. Metropolis et al. *J. Chem. Phys. 21*, 1953.

[4] Anders W. Sandvik and Juhani Kurkijärvi. Quantum monte carlo simulation method for spin systems. *Phys. Rev. B*, 43(7):5950–5961, Mar 1991.

[5] Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Phys. Rev. Lett.*, 58(2):86–88, Jan 1987.

[6] Olav F. Syljuåsen and Anders W. Sandvik. Quantum monte carlo with directed loops. *Phys. Rev. E*, 66(4):046701, Oct 2002.

[7] J.M. Yeomans. *Statistical Mechanics of Phase Transitions*. Oxford University Press, Oxford, 1992.

Acknowledgements

It is time now to give acknowledgement to those people that accompanied me during my study and this thesis work. I want to thank my family for providing me with a great background and especially my parents for supporting me also financially which made it possible for me to spend two terms abroad and to focus completely on this thesis work in the final stages of my study. I also want to thank Julia, my companion in life, who gave me the strength and vigor I needed.
Special thanks go to my supervising tutor Prof. Peter Mohn. Not only that he was always helpful in giving me important input for my thesis work, but he also shared a lot of his time to discuss various problems that occurred along this work which was the foundation for me of proceeding so well in understanding the theory and its implementation in the algorithm. I also want to point out his extraordinary patience which can't be taken for granted if one considers how frequently I 'invaded' his office with tons of questions.