Simulation of distinguishable particles

Diplomarbeit

der Philosophisch-naturwissenschaftlichen Fakultät

der Universität Bern

vorgelegt von

Reto von Allmen

Juni 2003

Leiter der Arbeit:

Prof. Dr. U.-J. Wiese
Institut für theoretische Physik, Universität Bern

Contents

Chapter 1

Introduction

Feynman's path integral formulation of quantum mechanics in imaginary time reveals a deep connection between classical statistical physics and quantum theory. Due to this connection, techniques used in one of those areas can often be applied in the other one. One of these techniques widely used in statistical physics is the numerical simulation of systems by means of Monte Carlo methods. In spite of the fact that computers have developed very fast in the past decades, it still takes a great deal of resources to simulate large systems with many degrees of freedom such as quantum many-body systems, and there is a need for efficient algorithms. A class of such efficient algorithms, the so called cluster algorithms, is known and extensively used in statistical mechanics. In this paper a similar but new algorithm is proposed. Namely we introduce so called Wolff planes that reflect parts of particle paths while respecting the symmetry of the external potential (in our case a spherically symmetric harmonic oscillator potential that plays the role of a experimental trap). The path clusters are built in a way that detailed balance is satisfied when the cluster is flipped (reflected). This cluster algorithm has the enjoyable feature that autocorrelation times are significantly reduced compared to pure Monte Carlo simulations. Our algorithm is able to simulate two component Bose systems, consisting of two types of distinguishable particles with a finite range, repulsive pair-interaction between the two different particle kinds (but no inner species interaction) in a spherically symmetric external trap. When extending the algorithm to handle indistinguishable particles, the possibility arises to simulate two component Bose condensations. This kind of phenomenon has recently been explored experimentally, analytically and numerically. Experimentally, it was possible to create a Bose condensate of 2×10^6 atoms of ^{87}Rb in either one of the $|2, 2\rangle$ or $|1, -1\rangle$ spin states [1]. They used an apparatus with a magneto-optic trap which has an effect comparable to a harmonic oscillator trap with separate strength for the two states. As they evaporatively cooled the cloud down to an overlapping Bose condensate, they found a repulsive interaction of the two components. This experimental result was then theoretically described by [2], using a Hartree-Fock calculation with a harmonic oscillator potential but magnetically displaced trap centers. They managed to describe the hemispherical separation of the two components for a certain interaction range. Another group [3] found in numerical simulations, that the ground state at low temperature and with large repulsive interaction between the two components is rather of an isotropic shape with one component (^{23}Na) situated at the trap center, surrouded by the other component (^{87}Rb). This configuration seemed to be metastable for after a while the two components changed place. In [4] an algorithm, based on the Gross-Pitaevskii equation, was developed to determine the density profiles of binary

mixtures of Bose condensates of alkali atoms in the Thomas-Fermi approximation. They were able to simulate the whole range of states from interpenetrating superfluids to separated phases and found a great variety of ground state and vortex structures. Our algorithm could represent a different approach to simulate the same phenomena. However, in this paper we don't attack the full problem but deal with the simplified version where all particles are distinguishable. The generalization to indistinguishable particles, which is needed to observe phenomena as Bose condensation, is quite straightforward and the techniques are known[1]. The prime modification consists in including exchanges of particles of the same type.
Since the algorithm can be applied to a single particle species (with no pair interaction) and is expected to be very efficient in this case too, another interesting task would be to simulate liquid ^{4}He which exposes superfluidity below 2.19K. Here one can rely on a huge amount of experimental data but also on theoretical and numerical studies.[2]
This paper is organized as follows. In chapter 2 we give a brief review of Feynman's path integral formalism in both real and imaginary time. The imaginary time formulation is used to show the deep analogy between simulations in statistical physics and quantum theory and we provide the formulas which are used later. Chapter 3 describes the numerical implementation of the path integral concept and explains the idea of importance sampling and the Metropolis algorithm while in chapter 4 the new path-cluster algorithm is introduced. In chapter 5 some numerical simulations of a two component system of distinguishable particles in a trap at a high and low temperature is performed and the results discussed. Finally a summary and an outlook for possible future studies are given in chapter 6. In appendix A we give a proof for the existence and uniqueness of an equilibrium state in a Markov chain with discrete state space. Several techniques for statistical data analysis, such as the bootstrap, blocking and jackknife method are given in appendix B. The last appendix C contains a short description of the simulation program. It describes the data structure being used as well as the implementation of the Metropolis and cluster algorithm and gives a short introduction to linked lists which are used to set up a spatial grid for implementing the short range pair interaction.

[1] For a good survey see e.g. D. M. Ceperley and M. H. Kalos, *Quantum Many-Body Problems* in [5] and references therein

[2] For an excellent review on simulating condensed helium see [6]

Chapter 2

Path integrals in real and imaginary time

Our tool to study quantum mechanical particle systems is Feynman's approach using the path integral formalism [7] in imaginary time. This formalism reveals a great similarity between quantum (field) theory and statistical physics and thus enables the application of numerical concepts such as Monte Carlo and cluster algorithms frequently used in the latter.

2.1 Path integrals in real time

Consider the quantum mechanical transition amplitude for a single particle at (real) time t with coordinate x (in d space dimensions) to a state with coordinate x' at time t':

$$U(x',t';x,t) = \langle x'|e^{-iH(t'-t)}|x\rangle \tag{2.1}$$

with $H = \frac{p^2}{2m} + V(x)$ being the Hamilton operator of the particle with mass m. Now the time interval is divided into N steps of equal size: $t' - t = N\epsilon$. At each time slice $t_k = t + k\epsilon$ a complete set of position eigenstates is inserted ($x_0 \equiv x$ and $x_N \equiv x'$)

$$U(x',t';x,t) = \int d^d x_1 \dots d^d x_{N-1} \langle x_0|e^{-iH\epsilon}|x_1\rangle \dots \langle x_{N-1}|e^{-iH\epsilon}|x_N\rangle. \tag{2.2}$$

For such a time-slice the operator relation $e^{A+B} = e^A e^B e^{\frac{1}{2}[A,B]+\dots}$ (Baker-Campbell-Hausdorff formula) yields the approximation (for small ϵ),

$$\begin{aligned} \langle x_k|e^{-i\epsilon H}|x_{k+1}\rangle &= \int \frac{d^d p_k}{(2\pi)^d} \langle x_k|e^{-i\epsilon V(x)} e^{-i\epsilon\frac{p^2}{2m}}|p_k\rangle\langle p_k|x_{k+1}\rangle + O(\epsilon^2) && (2.3)\\ &\approx \int \frac{d^d p_k}{(2\pi)^d} e^{i[p_k(x_{k+1}-x_k) - \epsilon H(x_k,p_k)]} && (2.4) \end{aligned}$$

where $H(x_k,p_k) = \frac{p_k^2}{2m} + V(x_k)$ is now an ordinary c-function. Plugging the approximation back into (2.2) gives

$$U(x',t';x,t) = \int \prod_{k=1}^{N-1} d^d x_k \int \prod_{k=1}^{N} \frac{d^d p_k}{(2\pi)^d} \exp\left\{ i \sum_{j=1}^{N} [p_j(x_{j-1} - x_j) - \epsilon H(x_j,p_j)] \right\}. \tag{2.5}$$

For our specific choice of the Hamiltonian, the momentum integrations can be carried out

$$U(x',t';x,t) = \mathcal{N} \int \prod_{k=1}^{N-1} d^d x_k \exp\left\{ i\epsilon \sum_{j=1}^{N} \left[\frac{m}{2} \left(\frac{x_{j-1}-x_j}{\epsilon} \right)^2 - V(x_j) \right] \right\} \tag{2.6}$$

($\mathcal{N}$ is a constant which can be determined by evaluating the free particle case). In order to take the continuum limit, one has to let $\epsilon \to 0$ and $N \to \infty$ but keeping $N\epsilon = t' - t$ fixed. Making the formal replacement $\lim_{N\to\infty} \prod_{k=1}^{N-1} d^d x_k \to Dx$ yields

$$U(x',t';x,t) = \mathcal{N} \int Dx\; e^{iS[x;t',t]} \tag{2.7}$$

with the classical action $S[x;t',t] = \int_t^{t'} d\tilde{t} \left(\frac{1}{2} m\dot{x}^2 - V(x) \right)$. With this formula, the transition amplitude can be interpreted as a sum over all paths, each of them weighted by i times the classical action of it. Therefore it is called the *path integral* formula.

2.2 Path integrals in imaginary time

A different way to write down the transition amplitude is to insert a complete set of energy eigenstates ($H|n\rangle = E_n|n\rangle$) in (2.1):

$$U(x',t';x,t) = \langle x'|e^{-iH(t'-t)}|x\rangle = \sum_n \langle x'|n\rangle\langle n|x\rangle e^{-iE_n(t'-t)}. \tag{2.8}$$

Since the energy spectrum is assumed to be bounded from below, the function above (i.e. the transition amplitude) can be analytically continued: $t \to -it$. With the same procedure as in the previous section one obtains instead of (2.6)

$$U(x',-it';x,-it) = \mathcal{N} \int \prod_{k=1}^{N-1} d^d x_k \exp\left\{ -\varepsilon \sum_{j=1}^{N} \left[\frac{m}{2} \left(\frac{x_{j-1}-x_j}{\varepsilon} \right)^2 + V(x_j) \right] \right\}. \tag{2.9}$$

Here (note the plus sign in front of the potential!) $\varepsilon = i\epsilon$ and $\mathcal{N}$ is a normalization constant. This formula will play a crucial role in the numerical calculations. The reason for taking the imaginary time instead of the real time path integral is because now each path has a real, positive weight, which is also bounded (since the potential is bounded from below and the kinetic term is positive semidefinite). Paths with a small action have a bigger weight and are thus more important, i.e. they contribute more to the transition amplitude than paths with larger actions.

Although the continuation to imaginary time seems somewhat artificial at first sight, it has some very pleasant features. When imposing periodic boundary conditions in time, $x_0 = x_N$, and taking the trace (integrating over the initial/final states), (2.8) becomes in imaginary time ($t' - t \to -i\tau, \quad \tau > 0$)

$$\int dx\, U(x,\tau) = \sum_n e^{-\frac{1}{\hbar}\tau E_n} \tag{2.10}$$

which, when making the formal replacement $\tau \to \hbar\beta = \hbar/kT$, looks the same as the partition function of statistical mechanics

$$Z = \sum_n e^{-\beta E_n} \tag{2.11}$$

that can also be written as a path integral

$$Z = \int Dx \, \exp\left\{-\int_0^{\beta\hbar} d\tau \, \left(\frac{m}{2}\dot{x}^2 + V(x)\right)\right\} \tag{2.12}$$

$$= \int Dx \, e^{-S_E[x]} \tag{2.13}$$

with $Dx = \lim_{N\to\infty}\prod_{k=1}^{N} d^d x_k$ and S_E the Euclidean action (from now on S denotes the Euclidean action, unless otherwise stated).
From all this, one sees that simulating a quantum mechanical system in periodic imaginary time is the same as a finite temperature simulation in statistical physics! Thereby large temperatures correspond to small β and thus to short time intervals τ ($\hbar = 1$).
Finally, the finite time lattice approximation for a system with M particles looks like:

$$Z \propto \int \prod_{l=1}^{M} Dx^{(l)} \, e^{-S[x]} \tag{2.14}$$

where $x^{(l)}$ denotes the path of the lth particle while $[x]$ stands for the whole configuration. The action now includes a potential $V[x]$ that can contain interaction terms

$$S[x] = \sum_{j=1}^{N} \varepsilon \left\{ \sum_{l=1}^{M} \frac{m_l}{2} \left(\frac{x_{j-1}^{(l)} - x_j^{(l)}}{\varepsilon} \right)^2 + V[x] \right\}. \tag{2.15}$$

Especially, the action of our simulation will contain two types of particles, N_A of type A and N_B of type B — with paths denoted by $x^{(A_k)}$ or $x^{(B_k)}$ respectively, in an external harmonic oscillator potential (with frequencies ω_A and ω_B). It looks like

$$S[x] = \sum_{j=1}^{N} \varepsilon \left\{ \sum_{Y=A,B} \sum_{l=Y_1}^{N_Y} \left[\frac{m_Y}{2} \left(\frac{x_{j-1}^{(l)} - x_j^{(l)}}{\varepsilon} \right)^2 + \frac{m_Y}{2}\omega_Y^2 \left(x_j^{(l)} \right)^2 \right] + V_j(x_j^{(A)}, x_k^{(B)}) \right\} \tag{2.16}$$

where $V_k(x_k^{(A)}, x_k^{(B)})$ is a short range, repulsive interaction only between pairs of particles of the two types at slice k (to be specified). Correspondingly, Z will be

$$Z \propto \int \prod_{l=A_1}^{N_A} Dx^{(l)} \prod_{m=B_1}^{N_B} Dx^{(m)} \, e^{-S[x]}. \tag{2.17}$$

2.3 Observables in the Path Integral Formalism

The observables of quantum mechanical systems are expectation values of the corresponding (hermitean) operator. The thermal expectation value of an operator O is defined as

$$\langle O \rangle = \frac{\int Dx \, O[x] e^{-S[x]}}{\int Dx \, e^{-S[x]}} = \frac{1}{Z} \int Dx \, O[x] e^{-S[x]} \tag{2.18}$$

where $O[x]$ is the observable for a given configuration $[x]$. This definition guarantees that the measured quantity of a path contributes with the correct weight (that of the path itself) to the expectation value.
For the future use, we provide here the formulas for the observables to be measured. Suppose we have two types of particles. A configuration $[x]$ consists then of N_A particle paths of type A, labeled with $x^{(A_l)}$ $(l = 1, \ldots, N_A)$, and N_B particle paths of type B, $x^{(B_l)}$ $(l = 1, \ldots, N_B)$. We measure the squared average distance from the origin (this will be the minimum of the external potential)

$$r_Y^2[x] = \frac{1}{N_Y N} \sum_{l=Y_1}^{N_Y} \sum_{k=1}^{N} (x_k^{(l)})^2 \tag{2.19}$$

(Y stands for either A or B, while lower indices denote the number of the time slice). Furthermore we measure the average principle moments of inertia (with reference to the center of mass) of each particle type. In order to do this we determine the inertia tensor at a time slice k

$$I^{(Y)}_{\alpha\beta;k} = \sum_{l=Y_1}^{N_Y} m_Y \left\{ \delta_{\alpha\beta}[(x^{(l)}_{1;k})^2 + (x^{(l)}_{2;k})^2 + (x^{(l)}_{3;k})^2] - x^{(l)}_{\alpha;k} x^{(l)}_{\beta;k} \right\} \tag{2.20}$$

(Again Y is either A or B and $x_{1;k}$ denotes the first center of mass coordinate component at slice k etc.), calculate its eigenvalues $\hat{I}^{(Y)}_{1;k}, \ldots, \hat{I}^{(Y)}_{3;k}$ before we average them over all time slices

$$I_1^{(Y)}[x] = \frac{1}{N} \sum_{k=1}^{N} \hat{I}^{(Y)}_{1;k}, \quad I_2^{(Y)}[x] = \frac{1}{N} \sum_{k=1}^{N} \hat{I}^{(Y)}_{2;k}, \quad I_3^{(Y)}[x] = \frac{1}{N} \sum_{k=1}^{N} \hat{I}^{(Y)}_{3;k}. \tag{2.21}$$

Chapter 3

Monte Carlo method

3.1 Importance sampling

Because there are infinitely many paths which enter in (2.18), it is impossible for a computer to generate all of them and calculate their contribution in a finite amount of time. Instead the idea is to produce a finite subset of all paths whose frequency of occurrence corresponds to their weight. That means that configurations with a large weight appear more often than those with smaller weight. But let us now be a bit more precise. Suppose that the probability density for a configuration $[x]$ is given by

$$p[x] = \frac{\exp(-S[x])}{\int Dx \, \exp(-S[x])}. \tag{3.1}$$

Then, when drawing M samples $[x_i]$ $(i = 1 \ldots, M)$ from the distribution above, one has an estimate for $\langle O \rangle$ given by

$$\bar{O} = \frac{1}{M} \sum_{i=1}^{M} O[x_i]. \tag{3.2}$$

In the limit $M \to \infty$ the law of large numbers tells us that the expectation value of the operator O is equal to the average of the sample values:

$$\langle O \rangle = \bar{O} \qquad (M \to \infty). \tag{3.3}$$

3.2 Markov chains and Metropolis algorithm

In the previous section we have seen that if the configurations have the proper probability, the calculation of expectation values reduces to a simple averaging of sample values. The question that now arises is how to generate the distribution (3.1). The answer is given by Markov chains and the Metropolis algorithm. A simple method would be to generate a configuration (i.e. the coordinate values at each time slice) completely at random and accept it as a sample path with a probability equal to its weight. This method though is very inefficient and would never be practical, because many of the generated paths would be rejected due to their small weight. This problem is overcome by Markov chains.

3.2.1 Markov processes

For our purposes, a Markov process is a mechanism which, given a configuration $[x] =: [x_i]$, produces a new configuration $[x'] =: [x_{i+1}]$ in a random fashion. The probability of generating a configuration $[x']$ given configuration $[x]$ is called transition probability $W[x \to x']$. For a true Markov process the transition probabilities for any transition should satisfy two conditions:

1. They should not depend on time
2. They should depend only on the corresponding configurations $[x]$ and $[x']$

And, of course, in order to be a probability they should satisfy the constraint

$$\int Dx'\, W[x \to x'] = 1. \tag{3.4}$$

Note that the transition probability $W[x \to x]$ does in general not have to be zero – the system may remain in the same state: $[x_{i+1}] = [x_i]$.
A *Markov chain* is a procedure that, given a configuration, generates other configurations using a Markov process sequentially. Thereby a generated configuration is the source for the next Markov process, thus a chain of configurations $[x_1] \to [x_2] \to \ldots \to [x_M]$ is produced.

3.2.2 Ergodicity

Since in the distribution (3.1) each path has a non-vanishing probability, we must ensure that every configuration can, in principle, be reached by a Markov chain: For any configuration $[x'] =: [x_j]$ there must exist a chain of states $[x_i] \to [x_{i+1}] \to \ldots \to [x_j]$ $(j > i)$ such that $W[x_i \to x_j] = W[x_i \to x_{i+1}] \cdot \ldots \cdot W[x_{j-1} \to x_j] > 0$, independent of the starting configuration $[x] =: [x_i]$. This necessary condition is called condition of ergodicity.

3.2.3 Detailed balance

The condition of ergodicity does not yet ensure that the correct probability distribution is achieved. Intuitively one feels that this can be done by setting the transition probability appropriately. But before we turn to this subject, let us first define the notion of *equilibrium*. For a moment, we assume that the probability for a system to be in a certain configuration depends on time, call it $p[x](t)$. The probability density for the (time independent) transition to the configuration $[x']$ in the time interval dt shall be denoted by $W[x \to x']\, dt$. Then the change in time of $p[x](t)$ is given by (Master equation)

$$\frac{dp[x](t)}{dt} = \int Dx' \, \left\{ p[x'](t) W[x' \to x] - p[x](t) W[x \to x'] \right\}. \tag{3.5}$$

A system now is called to be in equilibrium, if $\dot{p}[x] \equiv 0$ for all states $[x]$. This means that

$$\int Dx'\, p[x] W[x \to x'] = \int Dx'\, p[x'] W[x' \to x] \tag{3.6}$$

for any state $[x]$. Using (3.4) gives

$$p[x] = \int Dx'\, p[x'] W[x' \to x]. \tag{3.7}$$

So in order to be in equilibrium, the transition probabilities must satisfy this condition. If we enforce (3.7) to

$$p[x]W[x \to x'] = p[x']W[x' \to x] \tag{3.8}$$

then even the uniqueness of the equilibrium distribution is guaranteed (See Appendix A). The condition (3.8) is called condition of detailed balance. It implies (3.7) (integration over $[x']$). To get a feeling for this condition, let us translate it into words: The left hand side can be interpreted as the probability of the system being in the configuration $[x]$ and making a transition to $[x']$, thus it's the overall rate for the transition from the configuration $[x]$ to $[x']$. For the right hand side the analogue is true. So the condition of detailed balance says that on average the number of transitions from configuration $[x]$ to the configuration $[x']$ is equal to the number of transitions from $[x']$ to $[x]$.[1]
Applied to our situation ($p[x] \propto e^{-S[x]}$) the condition of detailed balance reads

$$\frac{W[x \to x']}{W[x' \to x]} = e^{-(S[x']-S[x])}. \tag{3.9}$$

This equation still does not uniquely determine the transition probabilities (since it tells only something about their ratios). E.g. a simple choice would be $W[x \to x'] \propto e^{-\frac{1}{2}(S[x']-S([x])}$. One can easily check that such transition probabilities satisfy (3.9). There are many ways to accomplish the condition of detailed balance. One has to check, which one is the most efficient for a given simulation. We will investigate especially the one called *Metropolis algorithm*, invented by N. Metropolis et al. in 1953 [8].

3.2.4 Acceptance ratios

Although we now know what condition the transition probabilities should satisfy, the way one could implement it in an algorithm is not yet clear: Given a configuration $[x_i]$, there are many possible configurations for the system to assume in the next Markov-step. If the algorithm was constructed such that it makes a given transition with a certain probability, it would have to be tuned such that finding its way back also happens with a suitable probability, so that detailed balance is obeyed. But luckily there is a solution: Expand the transition probability into two factors:

$$W[x_i \to x'] = g[x_i \to x']A[x_i \to x'] \tag{3.10}$$

and interpret them as follows: $g[x_i \to x']$ is the *selection probability*, the probability that the algorithm generates/proposes the configuration $[x']$ out of $[x_i]$, and $A[x_i \to x']$ the *acceptance ratio*, the probability that the new configuration is accepted, i.e. that the next state in the Markov process is the state $[x'] = [x_{i+1}]$ (else it remains in the previous state: $[x_{i+1}] = [x_i]$). If $A[x_i \to x'] = 0$ the new configuration is always rejected, the system remains in the state $[x_i]$. This is possible because for any value of $W[x \to x]$ (between zero and one) detailed balance is satisfied, we can always allow the system to remain in the same state (though choosing $A[x_i \to x'] = 0$ would be rather dumb because then ergodicity is violated). The advantage of (3.10) is, that the way we generate a new configuration is arbitrary, we only have to adjust the acceptance ratios to guarantee detailed balance. Of course, one is interested in

[1] Forget here for a moment the subtleties of a continuous configuration space where such a transition would happen with probability zero. Instead think of the configuration space as being countable.

making the acceptance ratios moderately large to achieve a large variety of states (in contrast to nearly always staying in the same state).
But let us finally apply these general considerations to a concrete algorithm, the one of Metropolis!

3.2.5 The Metropolis algorithm

This algorithm makes no restriction on how the new configuration is generated, except that the selection probability for a transition and its reverse should be the same. It only determines the acceptance ratio:

$$A[x \to x'] = \min\left[1, \quad e^{S[x]-S[x']}\right]. \tag{3.11}$$

That is: the new configuration is always accepted, if its action is smaller than the action of the old configuration. If it is larger, then the new configuration is accepted with a corresponding probability.
Let us now check that the resulting transition probability has the required properties:

- Detailed balance

$$\frac{W[x \to x']}{W[x' \to x]} = \frac{A[x \to x']}{A[x' \to x]} = e^{S[x]-S[x']} = \frac{p[x']}{p[x]}$$

- Positive semi-definiteness.

$$(0 < A[x \to x'] \leq 1) \Rightarrow (0 \leq W[x \to x'] \leq 1) \tag{3.12}$$

  if $g[x \to x']$ is chosen properly ($0 \leq g[x \to x'] \leq 1$).

3.2.6 Setting up the algorithm

Until now, no word has been lost about how to generate the configurations (and thus determining $g[x \to x']$). So this section will be on this topic.
In principle, the Markov process can start with any arbitrary configuration. But, of course, it is in general better to start with a configuration which is already "near" to equilibrium. Assume the starting configuration to be $[x]$. Its action shall be denoted by $S[x]$. One could now generate a new configuration $[x']$ at random (so that each configuration has the same selection probability), compute the difference in the actions and accept (or reject) the new configuration with a probability according to (3.11). This method however would not be very efficient – in practice even impossible, because many configurations would be generated and then rejected almost every time due to their large action. A better way is, that one only changes one coordinate of one particle at a single time slice and this only within a given interval $2\Delta x$ (for each component). This *Metropolis move* in general causes a change in the action (with respect to the path with the old coordinate values) and can be accepted or rejected as described above. Then one proceeds with the next time slice and/or particle until all coordinates of every particle at every time slice has once been processed. This is called a *Metropolis sweep*. As mentioned, the new coordinate $x_k'^{(i)}$ lies within the interval $x_k^{(i)} \pm \Delta x$ but is chosen at random (same probability for each new coordinate). This ensures that the selection probability for the reverse process $x_k'^{(i)} \to x_k^{(i)}$ is the same. At first sight this seems to violate ergodicity. Indeed one Metropolis sweep alone is not ergodic. But in a simulation

many sweeps are performed, so that there exists a non vanishing probability to reach (and accept!) an arbitrary configuration by means of many such small movements. The advantage of this method is, that on average the acceptance rate is higher since the two configurations $[x]$ and $[x']$ do not differ very much (see also the next passage).
The experimental part of the simulation consists now in finding good values of Δx and choosing a starting configuration such that equilibrium is reached as fast as possible. Too small Δx's have the consequence, that diffusion through configuration space is slow and a long simulation time is needed to generate the desired diversity of configuration samples. If the value of Δx is to large, many configurations that are proposed will be rejected due to the large difference in the actions and thus again the diversity of generated configurations is small. In practice, Δx should be chosen such that the Metropolis acceptance rate (i.e. the ratio of accepted proposals and total proposals) is about 70%. Another thing one can do to achieve more different configurations is to perform at a single time slice several Metropolis moves successively. In fact this costs more computer time, but on the other hand autocorrelation times (see B.1) are reduced.
In principle we have now an algorithm that is able to simulate a large class of quantum mechanical systems. But we now will suggest an extension, that is applicable to some specific models and yields there for some parameters a boost in efficiency.

Chapter 4

Cluster Algorithm

4.1 Path-cluster algorithm

The Metropolis algorithm as we implemented it has the disadvantage that it performs only local changes (i.e. at a single time slice and particle) in the configuration and this only within a given range $2\Delta x$ (the reasons for such an implementation were given in the previous section). It would be a great increase of efficiency if one was able to move several coordinates in one step over large distances but with the correct acceptance rate for the new configuration. In lattice spin models (where in comparison with our situation spin directions play the role of coordinates and the lattice points correspond to time slices) there are algorithms that flip (move) many spins in one step thereby generating an "acceptable" configuration. There exist mainly two so called *cluster algorithms*, originally invented by Swendsen and Wang [9] and Wolff [10], that proved to be very efficient in the important range of the parameters.
The general idea behind these algorithms, applied to our situation, may be roughly described like this: If one allows a particle coordinate to move over a large distance, one should also allow its temporal and spatial neighbors to move over a corresponding distance (i.e. take them into the same cluster), so that the cost in action (more precisely: in the kinetic and potential parts of the action) is not too big. Then one proceeds with the neighbors of the neighbors and so on until a kinetic or potential bond is broken and the cluster is complete. When doing this, care has to be taken, that the cluster cannot grow too much such that all coordinates are moved and one ends up with an "inverse" but physically equivalent configuration. This is done by means of so called *reference configurations*. Again, let us be more precise!

4.1.1 Wolff planes

Suppose that the action is invariant under some spatial reflections. For example, if the external potential is spherically symmetric ($V(\vec{x}) = V(r)$), the action does not change when *all* paths are rotated about an axis passing the point with $r = 0$ of the (external) potential minimum, or, reflected by a plane containing that point. Let us call such a plane a Wolff plane (note that the direction of its normal vector is arbitrary). To learn how to apply the cluster idea to such a situation, let us first consider only a one-particle system in an external spherically symmetric (harmonic) potential and periodic boundary conditions in (imaginary) time. Assume that a time slice has been chosen at random and the corresponding particle position $x_k^{(i)}$ ($k \in 1, \ldots, N$) (called the *seed*) is to be reflected at a Wolff plane. Now we ask

how large the probability for the next coordinate $x_{k+1}^{(i)}$ to be reflected too by the same plane should be, in order to satisfy detailed balance. This probability, call it $p(x_k^{(i)}, x_{k+1}^{(i)})$, shall be the one for adding $x_{k+1}^{(i)}$ to the cluster together with $x_k^{(i)}$. We further set up the constraint that only coordinates situated in the same half space (which is separated from the other by the Wolff plane) can be taken into the same cluster.

4.1.2 Temporal bonds

Before we deduce how one can chose $p(x_k^{(i)}, x_{k+1}^{(i)})$ let us first fix the notation: While x_k denotes the coordinates of the particle (for the rest of this section we omit the upper index because at the moment we only deal with one particle) at time slice k as usual, x'_k shall be the reflected coordinates in the other half space (at the same time slice). Furthermore $S[x]$ is the action of the configuration (path) $[x]$ and $s_{ij} = s(x_i, x_j)$ the *kinetic* action of the path consisting only of the two points x_i and x_j (no periodic boundary conditions here!) – similarly $s_{i'j} = s(x'_i, x_j)$ and so on.

Now consider the two configurations $[x]$ and $[x']$ (Fig. 4.1). The only difference between

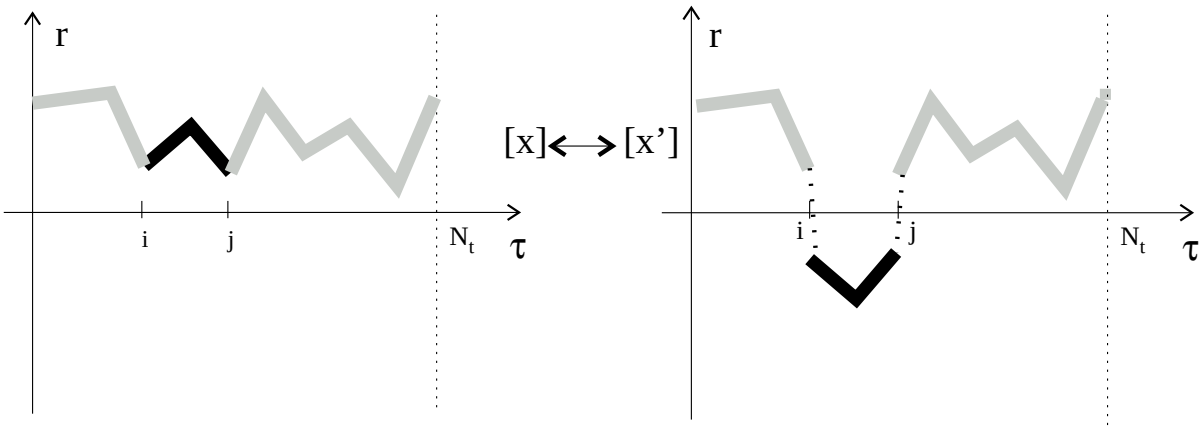

Figure 4.1: Breaking of temporal bonds

them is, that the black part is reflected at the Wolff plane represented by the horizontal axis. The probability for the transition $[x] \to [x']$ can be written as follows:

$$W[x \to x'] \quad \propto \quad [1 - p(x_{i-1}, x_i)][1 - p(x_j, x_{j+1})] \prod_{k=j+1}^{N_t+i-2} p(x_k, x_{k+1}) \times$$
$$\times \prod_{k=i+1}^{j-2} p(x_k, x_{k+1}).$$

(Remember periodic boundary conditions: $x_{k+N_t} \equiv x_k$) That is: all bonds except those between x_{i-1}, x_i and x_j, x_{j+1} ($j > i$) are set. Similarly one gets for the reverse transition

$$W[x' \to x] \quad \propto \quad [1 - p(x_{i-1}, x'_i)][1 - p(x'_j, x_{j+1)}] \prod_{k=j+1}^{N_t+i-2} p(x_k, x_{k+1}) \times$$
$$\times \prod_{k=i+1}^{j-2} p(x'_k, x'_{k+1}).$$

Intuitively it is clear, that the bond probabilities should only depend on the relative positions, i.e. $p(x_{k-1}, x_k) = p(x'_{k-1}, x'_k)$. On the other hand we demanded above that $p(x_{i-1}, x'_i) = 0$ because the two coordinates are in different half spaces (one is reflected). Furthermore, the proportionality factor for both directions is the same: it is the probability that the seed lies in the region to be reflected, which has the same size in both cases. Thus we have for the ratio

$$\frac{W[x \to x']}{W[x' \to x]} = [1 - p(x_{i-1}, x_i)][1 - p(x_j, x_{j+1)}]. \tag{4.1}$$

On the other hand we have the condition of detailed balance (3.9) which tells us that

$$\frac{W[x \to x']}{W[x' \to x]} = \frac{\exp(-S[x'])}{\exp(-(S[x']) + s_{i-1i'} + s_{j'j+1} - s_{i-1i} - s_{jj+1}))}. \tag{4.2}$$

One can easily check that for the choice

$$p(x_k, x_{k+1}) = \begin{cases} 1 - \frac{\exp(-s_{k'-1k})}{\exp(-s_{kk+1})} & : \text{ (if } x_k, x_{k+1} \text{ in same half space)} \\ 0 & : \text{ (otherwise)} \end{cases} \tag{4.3}$$

detailed balance is satisfied. It holds also for more complicated paths that cross the Wolff plane and the corresponding bond is broken because of the second part of (4.3). So this choice restricts the size of a cluster in those cases and assures that not always the whole path is reflected (that would not be very interesting).

4.1.3 Spatial bonds

Now it's time to consider several particles and their interactions. Suppose that we have two kinds of particles (A and B) that repel each other. There is no interaction between particles of the same kind. Imagine that a section of a path is contained in the cluster. This region will be reflected at the Wolff plane. If the reflected part comes near a path of a particle of the other kind, this costs a certain amount of action. This might be avoided if one allows parts of the other path to be taken in the same cluster and be reflected too (see Figure 4.2). But before we work out the probability of such spatial bonds, let us first again fix the notation.

Similar as before, $x_k^{(A_i)}$ shall denote the coordinates of particle i of kind A at time slice k. A prime indicates that the coordinates are reflected. $\tilde{s}_{A_iB_j}$ is the contribution of the pair interaction of $x_k^{(A_i)}$ and $x_k^{(B_j)}$ (same half space, same time indices!) to the total action.

Since we deal with a pairwise interaction, it's sufficient to consider only two particles of different type at the same time slice (see figure 4.3). Let us first take a look at configuration $[x']$ in figure 4.3. It would not make much sense if we attributed a non vanishing probability

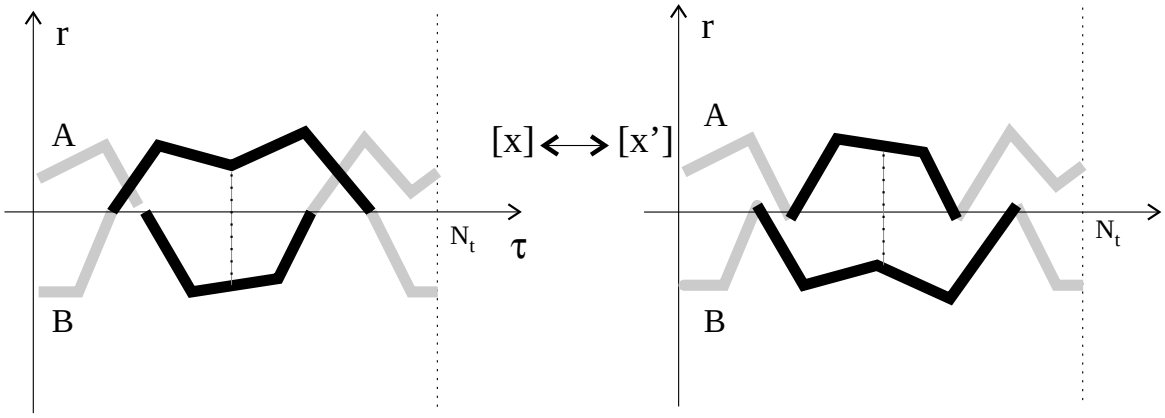

Figure 4.2: Cluster growth including other particle.

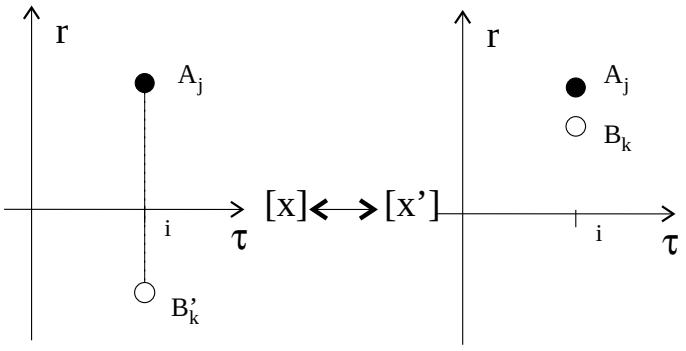

Figure 4.3: Spatial bond between two coordinates of different kind (Particle A_j is already in the cluster).

to this bond, because then both coordinates could be flipped and the particles would remain near each other and there wouldn't result a change in the interaction part: $\tilde{s}_{A_jB_k} = \tilde{s}_{A'_jB'_k}$. So we set $\tilde{p}(x_i^{(A_j)}, x_i^{(B_k)}) = 0$, i.e. particles of different kinds cannot be in the same cluster if they are located in the *same* half space. Thus we have for the transition probability

$$W[x' \to x] \propto 1 \tag{4.4}$$

and for the reverse transition

$$W[x \to x'] \propto 1 - \tilde{p}(x_i^{(A_j)}, x_i'^{(B_k)}). \tag{4.5}$$

Again detailed balance tells us that

$$\frac{W[x \to x']}{W[x' \to x]} = \exp(\tilde{s}_{A_jB'_k} - \tilde{s}_{A_jB_k}). \tag{4.6}$$

This condition is satisfied by taking

$$\tilde{p}(x_i^{(A_j)}, x_i'^{(B_k)}) = 1 - \frac{\exp(-\tilde{s}_{A_jB_k})}{\exp(-\tilde{s}_{A_jB'_k})} \quad : \quad (x_i^{(A_j)}, x_i'^{(B_k)} \text{ in different half spaces}). \tag{4.7}$$

When deducing this formula, we explicitly referred to a repulsive potential. The problem with attractive potentials consists in finding a reference configuration. That is to give a prescription when two neighboring particle coordinates cannot be in the same cluster (in the repulsive case this was the case, when the two coordinates were in the same half space) and thus preventing the cluster from growing too large. Now in the attractive case, one must allow particles that are in the same half space to be in a common cluster too (because they want to stay together). But if they are in the same cluster, the distance between them remains constant, instead of getting smaller as one wishes. If they are not in the same cluster, the distance would even grow and thus make the algorithm inefficient.

4.1.4 Summary and discussion of the algorithm

Our cluster algorithm can be summarized as follows:

1. Chose a particle and time slice at random and add the corresponding coordinate to the cluster. This is the *seed*.

2. Look at the temporal and spatial neighbors of the cluster member. Proceed if they are already in the cluster else add them with a probability (4.3) or (4.7), respectively.

3. Repeat the previous step until all neighbors of every cluster member have been checked. Note that a coordinate has several possibilities to become a member. E.g. first it is checked because it is a temporal neighbor of a cluster member. If it is not added, it has still the possibility to enter the cluster as a spatial neighbor of a coordinate in the cluster.

4. Perform the cluster move (reflection at the Wolff plane).

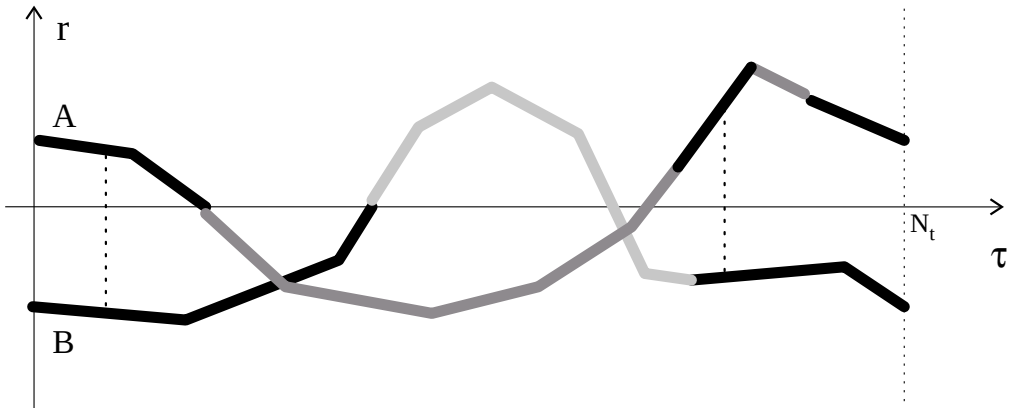

Figure 4.4: Cluster growth. Black regions are in the cluster, dotted lines indicate, where a spatial bond could have been accepted.

To illustrate a cluster where more than one particle is involved, consider figure (4.4). It shows, that the cluster can also grow beyond periodic time boundaries and that a cluster can be interrupted in the middle of a path and later continue to grow via a particle of the other kind.
As mentioned at the beginning, one great advantage of the cluster algorithm is that it changes configurations not only slightly as the Metropolis algorithm but can produce considerable modifications. How much a configuration after such a cluster flip resembles the configuration before, of course, depends on the size of the cluster. If it contained only a few coordinates, then the flipped configuration looks similar to the original. If, on the other hand, almost all coordinates are members of the cluster, then the new configuration also resembles the old one except for the fact that it is reflected. One concludes that the ideal cluster size is about half of the maximum size, containing as many different particles as possible, so that things really get mixed up.
One might ask why to have a Metropolis part in the program at all? The reason is that our cluster algorithm alone violates ergodicity! This is because coordinates are flipped with reference to the Wolff plane which contains the origin. Thus the distance of a coordinate to the origin remains the same – one needs something like a Metropolis algorithm to alter that distance!

4.1.5 Multi-cluster Algorithm

Our algorithm is a single cluster (Wolff-)algorithm [10]: One picks a seed, lets the cluster grow and then flips it. But there exists a variation of this algorithm, originally invented by Swendsen and Wang [9]. It is the so called multi-cluster algorithm. There one also picks a seed and lets the cluster grow as before (steps 1 to 3 in the previous algorithm summary). When finished, one picks another seed that is not yet in a cluster and does the same again – and so on, until every coordinate is member of a cluster. So one ends up with, say, n such clusters. In contrast to the single cluster algorithm, not each cluster is flipped but only with a probability of 1/2 independently. For a more detailed description see for instance [11].
The answer to the question, which one of the two algorithm is the better one, depends on the

observables one wants to measure. If it is a local quantity, the multi-cluster algorithm is more suitable, because it takes into account even the small clusters (the Wolff algorithm usually doesn't: the probability for the seed to lie in a certain cluster is proportional to its size). For global observables on the other hand, the Wolff algorithm is better, because it does not waste time on treating small clusters that hardly contribute to the measurement (observables can be determined clusterwise and then averaged with a weight proportional to the corresponding cluster size).

4.2 Testing the algorithm

Before we start using the algorithm and simulate, we first check the algorithm in situations where the solution is known.

4.2.1 Particle in an harmonic oscillator potential with finite lattice spacing

In many cases, analytical solutions to quantum mechanical problems are – if they exist at all – only known in the continuum limit, i.e. when the time lattice spacing limit $\epsilon \to 0$. So one usually has to extrapolate the results with finite lattice spacing (which is what one has in practice) to the continuum limit in order to say how large the influence of the finite spacing on the numerical results is. Luckily there is a (even physical) system where one can get some analytical results taking into account the finite lattice spacing! It is the system that describes a particle (or several particles but without interaction among themselves) in an external harmonic oscillator potential with periodic boundary conditions in time. At finite lattice spacing ϵ it is described by the Euclidean action

$$S([x]) = \epsilon \sum_{j=1}^{N} \left[\frac{m}{2} \left(\frac{x_{j+1} - x_j}{\epsilon} \right)^2 + \frac{\omega^2}{2} x_j^2 \right] \tag{4.8}$$

where N is the number of time slices and ω is the frequency of the external potential. From this one can derive (see e.g. [12] Appendix C), for example, the expectation value

$$\langle x^2 \rangle = \frac{d}{2\omega(1 + \epsilon^2\omega^2/4)^{1/2}} \left(\frac{1 + R^N}{1 - R^N} \right) \tag{4.9}$$

(the factor d accounts for the independent spatial dimensions) with

$$R = 1 + \frac{\epsilon^2\omega^2}{2} - \epsilon\omega \left(1 + \frac{\epsilon^2\omega^2}{4} \right)^{1/2} . \tag{4.10}$$

In (Table 4.1) the numerical results for $\langle x^2 \rangle$ in $d = 3$ dimensions are compared with the exact value for the parameters $m = 1$, $\omega = 2$, $\epsilon = 0.15$ and various N. To increase the statistics, we average over 10 particles (this has no effect on the result since there is no interaction). The Metropolis scale Δx was chosen such that the average acceptance rate was about 70% and for the 1-σ error estimation a bootstrap method with 1000 replications was used to determine the mean and error of the 2'000 sample values (see Appendix B.2). As one can make out, the exact numbers are all within the error range. So this is a strong hint that there is nothing wrong with the algorithm. But from that one cannot tell whether the spatial bond probabilities are chosen correctly, since we switched off the interaction between particles.

N	exact	numerical	$\pm$
5	1.17083	1.17039	0.00048
10	0.82034	0.82025	0.00025
20	0.74547	0.74552	0.00013

Table 4.1: Comparison of exact with numerical values of $\langle x^2 \rangle$ with 10^7 configurations

4.2.2 Algorithm applied to a discrete spatial lattice

The idea in this test is the following: Instead of continuous space coordinates one introduces also a spatial lattice so that the coordinates can assume only a few discrete values (no periodic boundary conditions in space but still in time). Then there is only a finite number of possible configurations L, i.e.

$$L = N_x^{dN_t N_p} \tag{4.11}$$

where N_x denotes the number of spatial lattice points (symmetric around the minimum of the external potential), N_t the number of time slices, N_p the number of (distinguishable) particles and d the number of space dimensions. When doing so, one has to modify the algorithm slightly. The main modification is, that one has no longer a Metropolis scale Δx but instead allows a coordinate to hop one lattice point in each direction. If the coordinate is situated at the edge of the lattice, it can only stay there or make a jump to the neighboring lattice point in the direction towards the center. Furthermore, the position of the Wolff plane in space is no longer arbitrary but has to take into account the symmetry of the spatial lattice too, i.e. it can only be any of the randomly chosen coordinate planes.
When the algorithm is adapted, one can generate again several configurations and compare their relative frequency of occurrence with the exact probability (the path integral now reduces to a finite sum that one can calculate easily, maybe with the aid of a computer). In this test one can of course include any kind of (repulsive) interaction and (symmetric) external potential! The disadvantage is, however, that one has not the general algorithm, so there may be flaws in the program that one cannot detect with this test (e.g. whether the random distribution of the Wolff vector is correct).

4.2.3 Comparison with pure Metropolis algorithm

Last but not least, there is the possibility to compare the whole algorithm with an independent program that uses only the Metropolis algorithm to generate the sample distribution. From this one gets the strongest hint whether the program is correct, because one doesn't have to alter the own algorithm but can nevertheless simulate a variety of systems. And one can directly compare the use of computer time and find out in which cases which algorithm is more efficient.

Chapter 5

Simulation of two types of particles in a trap

In this chapter we use the developed algorithm and do some numerical simulations. Since the system we simulate contains distinguishable particles, the results we get are not directly relevant for experimental results or the description of observed phenomena. The aim is rather to compare the new algorithm with common Monte Carlo simulation techniques.
In the first section, the parameters and properties of the system to be simulated are given. This is followed by a discussion of the equilibration period. In the third section we provide the results in a low and high temperature regime and compare these two cases. Finally some general remarks and considerations concerning the algorithm are given.

5.1 The physical system

The system we are going to simulate consists of distinguishable particles in periodic imaginary time. There are two kinds of particles A and B, N_A particles with mass m_A and N_B with mass m_B, which have a repulsive 2-particle interaction potential between particles of different types with a finite range

$$V_j(x_j^{(A)}, x_j^{(B)}) = \frac{1}{2} \sum_{l=A_1}^{N_A} \sum_{m=B_1}^{N_B} \begin{cases} V_0 & : \quad (\text{if } |x_j^{(l)} - x_j^{m)}| < R) \\ 0 & : \quad (\text{otherwise}) \end{cases} . \tag{5.1}$$

Both particle kinds are trapped in an external harmonic oscillator potential with frequencies ω_A and ω_B respectively (compare with the action (2.16)). The time lattice spacing ϵ is kept fixed but the number of time slices N varies according to the temperature $T \propto 1/\epsilon N$.
We intend to chose the parameters such that at sufficiently low temperature a nontrivial separation can be observed while at higher temperatures both particle types are mixed in a random fashion. With the word nontrivial we mean that the separation should take place in spite of symmetric parameters for the two particle kinds (On the other hand, if one would chose the mass of one type being significantly larger than the other mass while the external potential is the same for both, the separation at low enough temperatures would be trivial). For totally symmetric parameters for both particle types, our system has a $\mathbf{Z}(2)$ symmetry: We can make the transformation $x_k^{(A_i)} \leftrightarrow x_k^{(B_i)}$ $(k = 1, \ldots, N)$ without changing the action (2.16) and thus the path integrals and observables. But because of the repulsion, the two

particle types like to be separated from each other. A configuration at low temperature could look for instance such, that the particles of type A, centered around the external potential minumum, are surrounded by particles of type B. So this state would spontaneously break the $\mathbf{Z}(2)$ symmetry. Since the inverse situation, with particles B at the center is equivalent (it has the same action), it has the equal probability of being observed. So if one waits long enough, one should see transitions between the two states (and on average, the system should be in one or the other state for a equally long time).
On the other hand, if the separation at low temperatures had the shape of two hemispheres, the original isotropy of the action (2.16) is destroyed. Since that are very interesting situations, we make our simulations with symmetric parameters: We set both particle masses to unity – $m_A = m_B = 1$. Furthermore we wish that the different energy contributions to the action are of the same order of magnitude. This implies that the (identical) external potential frequencies, which have the dimension of an energy (in natural units), are of the order of one too, and so is the interaction strength V_0. The choice for the time lattice spacing is now guided by the following arguments: On one hand, it ought to be chosen as small as possible, so that one is near the continuum limit, on the other hand one desires to go to low temperatures ("low" compared to the natural energy scale), that is $N\epsilon \gg 1$. But increasing the number of time slices increases the simulation time, so the choice for ϵ is also a matter of patience.
Here are the model parameters for our simulation:

$$N_A = N_B = 20, \quad m_A = m_B = 1, \quad \omega_A = \omega_B = 0.3, \quad V_0 = 1, \quad R = 1, \quad \epsilon = 0.15. \tag{5.2}$$

The choices for the simulation parameters are a rather experimental task and are given below.

5.2 Equilibration

In section 3.2.3 and appendix A it was stated that whatever the initial state of the system is, the system comes to the unique equilibrium after an infinite Markov chain. But in practice one cannot wait for a very long time and is thus interested in a convergence as fast as possible. So if one wants to simulate the system at high temperatures, less equilibrium runs are needed if the starting state is a "hot" configuration with large kinetic energy while for low temperatures a starting configuration where the particles are at rest around the trap minimum is more suitable. However, for the simulation results, the starting configuration does not matter! But even for such well chosen starting points there are still needed some Markov steps for the system to have a distribution near equilibrium. So the question remains, when the system is in equilibrium and one can start the measurements (one could, of course, start measuring at once but then the results are more distorted by possibly improbable configurations at the beginning of the chain). In practice the answer is given by the following considerations. We first make some test runs where we make measurements of an observable right from the beginning and plot the produced data. Usually one observes that after some configurations the value of the observable flattens out and remains constant within statistical fluctuations and the actual error. Probably the system is then in equilibrium and one can read off the necessary number of equilibration configurations. However, there remains the possibility that the system is only in a meta-stable configuration from which the escape would take more computer time. To find out whether this is the case, one can repeat the whole procedure with a different starting configuration and check whether in this case the (same) observable converges to the same value or not. If it does so, it is even more probable that the corresponding flat region

is the equilibrium region. If not, one repeats the whole thing again with a third starting configuration and perhaps a larger number of generated configurations and so on.
At the same time, one can tune the Metropolis shift parameter Δx such that the average acceptance rate is about 70%. When all this is done, one can start with the actual simulations and enter the corresponding simulation parameters.

5.3 Results

In this section we present the simulation results at two different temperatures, given the model parameters (5.2) . The observables O we measured for each particle type were the average distance squared from the origin (trap minimum) (given in (2.19)) and the principle moments of inertia with reference to the center off mass (given in 2.20)). They are chosen in order to get an idea of the shape and the position of the particle clouds. We wondered whether the separation at low temperatures happened radially, i.e. whether one of the two kinds (both with equal mass) goes to the center of the trap and is surrounded by a spherical cloud consisting of the other particles (In this case, the two particle clouds should change places after some while because of the symmetric parameters and the fact, that both configurations have the same probability).
Another possibility would be, that the separation looks like two hemispheres, or even that some sort of speckles are produced. In those cases our observables would be not very well chosen and that's why in certain intervals we automatically generated snapshots of the radial and angular position of the particles.

5.3.1 High temperature regime

Here we made a simulation with $N_t = 5$ and $1.8 \cdot 10^7$ configurations (plus 1'000 equilibrium configurations). Δx was chosen to be 0.4 so that the average Metropolis acceptance rate was about 70%. The numbers for the average distance squared suggest, as it ought to be, that no radial separation has taken place. Also the principle moments of inertia and the snapshots (compare figure 5.1) hint at the fact, that there is a homogeneous cloud with the shape of an ellipsoid. This is due to the fact that the thermal energy scale is not significantly larger than the interaction scale (in this case, the clouds rather had the shape of a sphere) – but large enough to avoid a separation of the two particle types.

	Kind A	Kind B
$\langle r^2 \rangle$	99.99±0.30	100.49±0.29
I_1	953±3	958±3
I_2	1300±4	1306±4
I_3	1546±4	1554±5

From the symmetric results (for symmetric parameters) one sees, that the original $\mathbf{Z}(2)$ symmetry is not broken at high temperatures.

5.3.2 Low temperature regime – phase separation

This simulation was done with $N_t = 50$ and $1.8 \cdot 10^7$ configurations. Δx again being such, that a comparable acceptance rate of 70% was produced. Of course because of the lower

temperature both values for the average distance squared are smaller than in the previous case. Nevertheless there is no sign for a separation in the radial part and the principle moments of inertia tell us, that both particle clouds have a similar shape – again one of an ellipsoid:

	Kind A	Kind B
$\langle r^2 \rangle$	15.14±0.12	15.11±0.12
I_1	141±2	142±2
I_2	193±2	193±2
I_3	230±3	230±3

But from the snapshots (figure 5.2) can be seen, that there are lumps of particles of the same kind. So the separation happens in the angular parts. It looks as though there were indeed two hemispheres of different particle types, but there are too few particles to tell this for sure (it could also be lumps of particles of the same type). Nevertheless, this configuration is no longer homogeneous!

5.4 General remarks

To conclude this chapter, we give some general remarks about the efficiency of the algorithm.

- In the simulation we used an interaction with finite range. At first sight, this seems to be an unnecessary restriction – it would be more general to allow long range interactions. But as a consequence, one would have to consider all particles (of the other kind) at a given time slice and could not restrict oneself to take into account only the immediate neighbors (compare section C.3) and the used computer time would increase very fast with the number of particles.
  On the other hand, for just a few particles, the box concept from section C.3 is not worthwhile because it relies on a rather large apparatus.

- In comparison with a pure Metropolis simulation, the addition of the path cluster part proved, as expected, to be very efficient concerning the diffusion through configuration space. Also the autocorrelation time of the configurations, which can be made visible with the blocking method (see section B.3), was reduced significantly – depending on the parameters.[1] This means that one has to produce fewer configurations to get the same results!

- A low temperature simulation as we performed it, took about five days on a usual working station with a 2.4 GHz processor.

- An important issue in Monte Carlo simulations is the random number generator. Although there exist a huge number of them, their quality varies a lot. Many are quite "cheap" and have large autocorrelation times. A good one is "ranlux" from M. Lüscher[2]. The source is freeware and can be downloaded from the Internet.

[1] In lattice spin models there is the relation $\tau \propto \zeta^z$ (τ is the autocorrelation time, ζ the correlation length and d a positive real number). For Cluster algorithms d is significantly smaller than for pure Metropolis algorithms. In our simulation we found for some parameters, that the autocorrelation was reduced by roughly a factor 5.

[2] M. Lüscher, Computer Physics Communications 79 (1994) 100

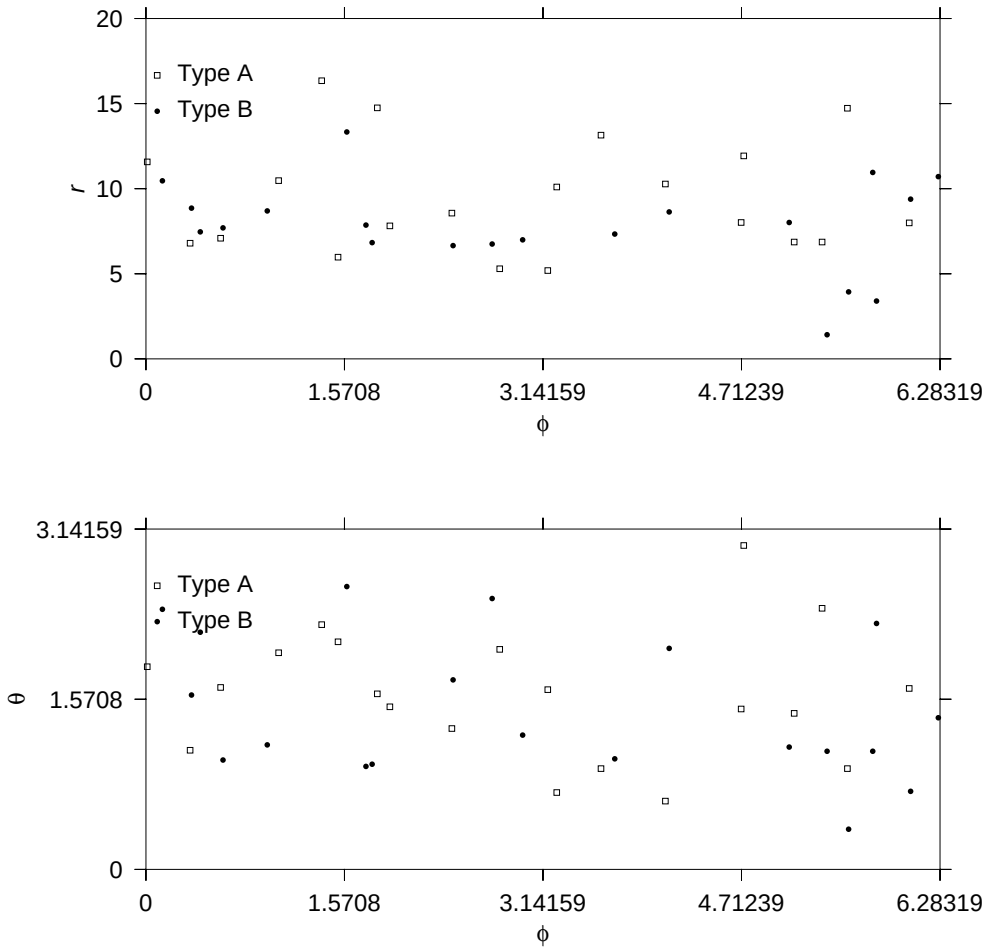

Figure 5.1: Snapshot at high temperature. The symbols indicate the position of the particles in spherical coordinates.

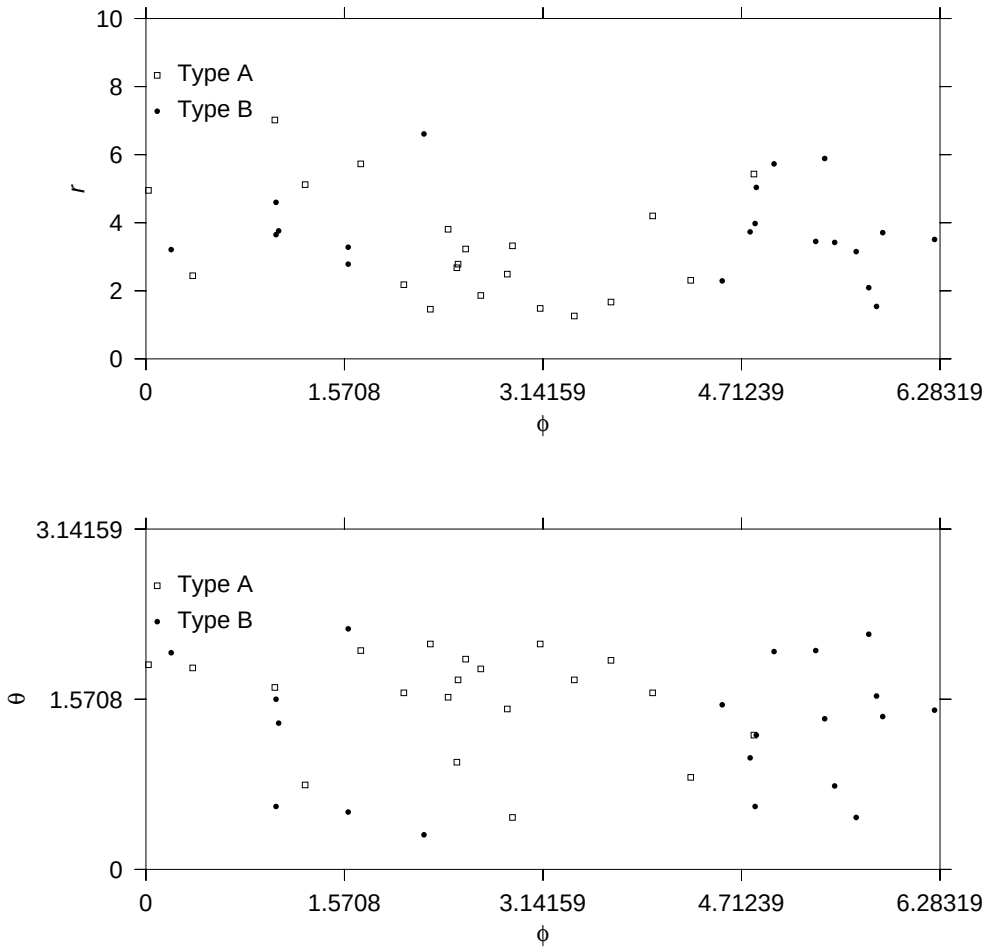

Figure 5.2: Snapshot at a low temperature. One can make out well the inhomogenity of the configuration.

Chapter 6

Summary and Outlook

In this paper we proposed a new algorithm of simulating a system of distinguishable particles of two different kinds with a short-range repulsive pair interaction in the framework of Feynman's path integral formalism. Up to now such simulations were done mostly using a Monte Carlo method only. Although this technique is also essential in our approach, we extend it with a cluster algorithm. After a short introduction to the theoretical background – the path integral formalism – we explained the idea of Markov chains and importance sampling. This knowledge was used to explain one of the probably most important sampling methods, the Metropolis algorithm. The inspiration for our cluster extension was obtained from the ideas of Wolff and Swendsen and Wang that invented this kind of algorithm in the framework of lattice spin models. We managed to adapt this technique and proposed a possibility to make non-local changes of particle paths under certain circumstances. We showed that for a (spherically) symmetric external harmonic oscillator potential which has the function of a trap, and for a repulsive pair interaction of two particle types, it is possible to reflect fractions of paths at a Wolff plane while maintaining the condition of detailed balance. We also explained the reason, why this path-cluster algorithm is inefficient when applied to the case of attractive pair interactions. Then we suggested several possibilities, how such an algorithm could be tested before we started actual simulations. We simulated a system with two types of distinguishable particles (20 of each type and with symmetric parameters) in a trap, once in a high and once in a low temperature regime. The latter exhibited features similar to those of two component Bose condensates which have already experimentally been observed and simulated with different algorithms and concepts. Namely we could show that there is no separation of the two components in the radial part, but that there is a separation in the spherical angles, similar to two hemispheres or local accumulations of particles of the same type. We further found that the new algorithm indeed reduced the autocorrelation time and yielded a boost in efficiency. So we have now a powerful tool to simulate many-particle systems as described. Furthermore we provided also several techniques, such as the jackknife, blocking or bootstrap method, for the analysis of (correlated) simulation data or concepts for implementing paths and short-range interactions by means of a spatial grid and linked lists.
As an extension of this work one could now investigate more precisely the differences of efficiency with respect to pure Metropolis simulations, especially in the condensed regime, or add more particles and modify the parameters. Also interesting would be, to examine more in detail the low temperature state for symmetric parameters and include new observables to describe better the separation of the particle types. Perhaps one could find, that the $\mathbf{Z}(2)$

symmetry is broken when more particles are added or the interaction strength is modified. But since our simulation are hardly of physical interest, one of the next steps would be to introduce particle permutations (particle permutations are of essential importance if one desires to study phenomena as Bose condensation and superfluidity) and thus make particles of the same type indistinguishable. This could be done by proposing the exchange of coordinates among particles of the same type in the Metropolis moves. There the spatial grid concept, which was used to implement the short-range pair interaction, would again be of great use, since those exchanges would most probably happen between neighboring particles (otherwise the cost in kinetic action would be too large). Another possibility would be to implement the so called multilevel metropolis method or other known techniques (see for example [6] p. 329ff.). Supplied with such algorithms, one could then simulate two component Bose condensates (see the introduction) and check, whether one gets correct results there. In the same time it could perhaps clarify the discrepancy of the different results of [2], where a hemispherical separation of the two components was proposed, and [3] which stated an radial separation.
Or one could attack the problem of simulating superfluid ^{4}He, where a bulk of experimental and simulation data exist. Although our algorithm allows to simulate only one particle species, there would be the problem, that the path-cluster algorithm is only applicable to repulsive (or no) particle interactions. Anyway, as described the algorithm can be extended to a large class of problems that can be handled efficiently with this new algorithm!

Appendix A

Equilibrium and Markov chains

The concept of Markov processes is widely used in many areas of physics, mathematics, economy etc. and the literature is accordingly huge. Since it is in addition a complex subject[1], the aim of this appendix is to provide only the basic notions and give the proof for the existence and uniqueness of the equilibrium state.
A Markov process is a stochastic (random) process, i.e. a set of random variables defined on a probability space, indexed by elements of a parameter set T (e.g. $\mathbf{R}$, $\mathbf{N}$ etc.). T can usually be thought of as Monte Carlo time and S, the range space of the random variables, as the configuration or state space (in our simulations, this is $\{1, \ldots, N_t\} \times \mathbf{R}^{3N}$).
Studying Markov processes, four main cases can be considered, depending on whether the parameter and the state space are discrete or not. Markov chains are Markov processes with discrete parameter space. We will concentrate on the case where both parameter and state space are discrete, because this is the easiest case and the results we derive here are similar to those of the other cases.[2]
Suppose now, that the state space contains N ($N \leq \infty$) elements (states). A Markov-chain matrix is now an $N \times N$ matrix W with entries $W_{ij} \geq 0$ satisfying $\sum_j W_{ij} = 1$ for all i. The interpretation of W is the following: If a system is in the state i, W_{ij} is the probability for the next state in the process being j. With this definition one can go on and define inductively the n-step transition matrix

$$W_{ij}^{(n)} = \sum_k W_{ik}^{(n-1)} W_{kj}. \tag{A.1}$$

We now want to prove the following statement: Let W be a finite Markov-chain matrix having the property that, for some positive m, $W_{ij}^{(m)} > 0$ for every i and j[3]. Then

$$\lim_{n \to \infty} W^{(n)} = \Pi \tag{A.2}$$

exists, where Π is a Markov-chain matrix with identical rows.
Proof: Let us first consider the case $m = 1$, where $W_{ij} \geq \epsilon > 0$ for every i and j. Let $m_j(n) = \min_i W_{ij}^{(n)} \doteq W_{i_0 j}^{(n)}$ denote the smallest element in the j'th column of $W^{(n)}$ and

[1] A good introduction is found in [13]. For a rigorous treatment see [14].
[2] But keep in mind, that in the path integral formulation in the continuum limit both spaces are continuous! However, the numerical implementation happens, in principle, in discrete parameter and state spaces.
[3] This corresponds to the ergodicity condition.

similarly $M_j(n) \doteq W^{(n)}_{i_1 j}$ be the largest element. It follows

$$\begin{aligned}
m_j(n) &= W^{(n)}_{i_0 j} \\
&= \sum_k W_{i_0 k} W^{(n-1)}_{kj} \\
&= \epsilon W^{(n-1)}_{i_1 j} + (W_{i_0 i_1} - \epsilon) W^{(n-1)}_{i_1 j} + \sum_{k \neq i_1} W_{i_0 k} W^{(n-1)}_{kj} \\
&\geq \epsilon M_j(n-1) + \left(W_{i_0 i_1} - \epsilon + \sum_{k \neq i_1} W_{i_0 k} \right) m_j(n-1) \\
m_j(n) &\geq \epsilon M_j(n-1) + (1-\epsilon) m_j(n-1) \qquad \text{(A.3)}
\end{aligned}$$

and also

$$\begin{aligned}
M_j(n) &= W^{(n)}_{i_1 j} \\
&= \sum_k W_{i_1 k} W^{(n-1)}_{kj} \\
&= \epsilon W^{(n-1)}_{i_0 j} + (W_{i_1 i_0} - \epsilon) W^{(n-1)}_{i_0 j} + \sum_{k \neq i_0} W_{i_1 k} W^{(n-1)}_{kj} \\
&\leq \epsilon m_j(n-1) + \left(W_{i_1 i_0} - \epsilon + \sum_{k \neq i_0} W_{i_1 k} \right) M_j(n-1) \\
M_j(n) &\leq \epsilon M_j(n-1) + (1-\epsilon) m_j(n-1). \qquad \text{(A.4)}
\end{aligned}$$

Subtracting (A.3) from (A.4) gives

$$M_j(n) - m_j(n) \leq (1 - 2\epsilon)[M_j(n-1) - m_j(n-1)] \qquad \text{(A.5)}$$

and thus

$$M_j(n) - m_j(n) \leq (1 - 2\epsilon)^n \to 0 \quad (n \to \infty). \qquad \text{(A.6)}$$

This means, that for every column j the maximum and the minimum of $\lim_{n\to\infty} W^{(n)} = \Pi$ are equal and thus all components of a row are the same.
Now suppose $m > 1$. With the result from above we know that

$$\lim_{n\to\infty} W^{(nm)} = \Pi \qquad \text{(A.7)}$$

and for any $k = 1, 2, \ldots, m-1$ we have

$$\lim_{n\to\infty} W^{(nm+k)} = \lim_{n\to\infty} W^{(k)} W^{(nm)} = W^{(k)} \Pi = \Pi \qquad \text{(A.8)}$$

where we used in the last step that $W^{(k)}$ has row-sums equal to one[4] while Π has constant columns. □
With this result, we conclude further, that the identical rows π of Π satisfy $\sum_i \pi_i W_{ij} = \pi_j$, $\pi_i > 0$ and $\sum_i \pi_i = 1$. And there is only one vector $v = \pi$ that satisfies these conditions.

[4]Proof: $\sum_j W^{(2)}_{ij} = \sum_{j,k} W_{ik} W_{kj} = \sum_k W_{ik} = 1$ and then by induction

Proof: Except for the uniqueness the proof is straightforward. Suppose that u is another vector that satisfies the conditions:

$$u = uW = uW^{(n)} = \lim_{n\to\infty} uW^{(n)} = u\Pi \tag{A.9}$$

But because $\sum_k u_k = 1$ and Π has identical columns, it is $u\Pi = v$. □

Let us finally apply these results to our simulation. Suppose we have an N dimensional state space. Furthermore the starting configuration is supposed to be state j, so that the "probability vector" is given by $v_0 = e_j$ where e_j denotes the j-th standard basis vector. This only says that we are in the state j for sure. Then we turn on the Metropolis machinery (this is indeed a Markov process, since the new configuration generated by the Metropolis algorithm only depends on the present one and not on the earlier ones). There are certain probabilities that the system makes transitions to other states – this may be represented by a matrix W whose elements W_{ij}[5] denote the probability for a transition from state i to state j. So already after one Metropolis step, we can only give probabilities for the system being in the different states. In other words: The vector v_0 has turned into a vector $v_1 = v_0 W$ which has more than one non-vanishing component, but whose sum is still equal to one (the probability to find the system in any state). The results from this chapter now tell us, that after an infinite number of such steps

$$v_0 \xrightarrow{W} v_1 \xrightarrow{W} \ldots \xrightarrow{W} v_\infty \tag{A.10}$$

the probabilities to find the system in a given state is unique and stationary – equilibrium has been reached.

[5]This corresponds to $W[x \to x']$ in the continuous state space Markov chain.

Appendix B

Statistical evaluation of the data

A crucial part of Monte Carlo and cluster algorithms is that they propose new configurations at "random" (the quotation marks indicate, that the random number generators one uses in numerical simulations are in fact entirely deterministic, so called pseudo random numbers). Thus the results for the observables in question are not always the same, but form a distribution. Since the sample is finite, one has only an empirical distribution for the observable values while the true distribution is unknown. Thus characteristical features of the distribution, such as its mean, standard deviation etc. which one gets from the simulation, are afflicted with an error. Unless this error is known (or rather: estimated) it is impossible to interpret any results. In this chapter we investigate the methods that allow us to estimate these errors. Once this estimate is performed and one can indicate a range for the value of the observable in question, the way to make this range smaller is easy: To produce more samples so that the empirical distribution is a better approximation to the true one!

B.1 Mean, standard deviation and autocorrelation

First we present some basic definitions of statistics. Suppose we have M samples O_i of an observable. Then the sample *mean* $\bar{O}$ is defined as

$$\bar{O} = \frac{1}{M} \sum_{i=1}^{M} O_i \tag{B.1}$$

i.e. the simple average. It can be proven that this is the best estimation for the expectation value of the (unknown) distribution of O. Essentially, the larger M, the better is the estimation: In the limit $M \to \infty$ one has

$$\lim_{M\to\infty} \frac{1}{M} \sum_{i=1}^{M} O_i = \langle O \rangle. \tag{B.2}$$

With the help of (B.1) one can estimate the *standard deviation of the mean* with

$$\sigma^2(O) = \frac{\sum_{i=1}^{m} (O_i - \bar{O})^2}{m(m-1)} = \frac{1}{m-1} (\overline{O^2} - \bar{O}^2). \tag{B.3}$$

This number can be interpreted as the *error* of a calculated value. If the distribution is Gaussian, σ has the property that in the limit $M \to \infty$ about two thirds of the sample values O_i

lie within the interval $[\bar{O} - \sigma, \bar{O} + \sigma]$. From (B.3) one sees that in order to make the error of a given observable smaller, one must increase the number of measurements.
The previous formulas are derived (by the method of maximum likelihood) with the assumption that the individual sample values O_i are independent of each other and that the true distribution of O is Gaussian. Presumably, both assumptions are not satisfied in our simulation: We implemented the Metropolis algorithm such that a new configuration is generated which differs not too much from the old one (the proposed coordinates lie within an interval of the size $2\Delta x$ around the old ones). One says that there is a non vanishing *autocorrelation* (for a precise definition see e.g. [15]). Furthermore, in general we simply do not know the true distribution of O! The question now is, whether it is still possible to make good estimates of the mean and its error. Indeed there are a few methods which can tackle these problems. One of them is the so called bootstrap method which will now be described in detail.

B.2 Bootstrap method

The bootstrap method was invented by Efron in the year 1979 (see [16] or, for a practical introduction, [17]). It is a widely applicable numerical method, based on a *resampling* process. The idea is strikingly simple: Assume that you have M measurements of an observable: $O_1, \ldots, O_M$ and you want to estimate the mean of the observable's underlying distribution F and its (estimation) error. To do this, proceed as follows:

1. Construct the empirical distribution $\hat{F}$, which is an estimation of F. In the empirical distribution each measurement has the same weight, namely $1/M$.

2. Draw a *bootstrap sample* by independent random sampling from $\hat{F}$, i.e. draw M values $O_1^*, \ldots, O_M^*$ from $O_1, \ldots, O_M$ with replacement (this means, that one value can be drawn several times).

3. Calculate the mean of the bootstrap sample. This is supposed to be the k-th *bootstrap replication* $\overline{O^*}_k$.

4. Do the steps 2 and 3 a large number B of times, yielding $\overline{O^*}_1, \ldots \overline{O^*}_B$. Then you obtain the estimate of the mean of F
$$\overline{O^*} = \frac{1}{B} \sum_{b=1}^{B} \overline{O^*}_b \tag{B.4}$$
and its error
$$\sigma(\overline{O^*})^2 = \frac{\sum_{b=1}^{B} (\overline{O^*}_b - \overline{O^*})^2}{B-1} \tag{B.5}$$

Actually the method is more general than it now seems to be. It can be used to estimate any statistical quantity (such as correlation coefficients etc.), regardless of its complexity. Although the mathematical correctness has not been proven except for some specific cases, it's validity is strongly supported by empirical studies. The question that remains is how large to choose B to get a reliable estimate. In the literature the answer is not given, but in concrete examples values up to $B = 1000$ are used (but, of course, this depends also on the number of samples). So one should experiment a bit with different values and observe the corresponding accuracy.
There are some more methods to make error estimations. We will shortly describe two of them: The blocking method and the jackknife method.

B.3 Blocking method

As explained above, the data obtained in Monte Carlo simulations may be strongly autocorrelated and a naïve estimation of the error by means of (B.3) might be affected by this. To reduce autocorrelation, one can divide the M measurements into b blocks of size m_b. Then one averages the observable in question over each block, thus obtaining b new values. Then one estimates the mean (B.1) and its error (B.3) of these values. The new estimation depends of course on the block size: If the data is correlated, a small block size yields similar results as the naïve estimation, but when increasing the size, the block values become less and less correlated (because in the ideal case several correlated measurements are averaged down to one, which is then no longer correlated with the previous/next average). To find out empirically the minimal block size for uncorrelated data, one can plot the mean and its error versus the block size and look for the region where both remain "constant" (within statistical fluctuations).

B.4 Jackknife method

Although this method is not used in our simulation, we give a short description of this method since it is quite common. It combines some ideas of the bootstrap and blocking methods: It builds blocks by resampling. Shortly, it works like this (also compare [17]):

1. Temporarily remove the first sample value from the set of M values and calculate the observable average $\bar{O}_1$ of the remaining $M-1$ values.
2. Do the same for the next (k-th) sample value, yielding $\bar{O}_k$.
3. Repeat step 2. until you have all $\bar{O}_1, \ldots, \bar{O}_M$.
4. The estimate of $\langle O \rangle$ is then given by
$$\bar{O} = \frac{1}{M} \sum_{k=1}^{M} \bar{O}_k \tag{B.6}$$
5. and its error by
$$\sigma^2 = \frac{M-1}{M} \sum_{k=1}^{M} (\bar{O}_k - \bar{O})^2 \tag{B.7}$$

.

Unlike the bootstrap method, the jackknife needs uncorrelated data. Its advantage consists in efficiency for few (about 100) measurements (see [11] p. 72).

Appendix C

The simulation program

In this appendix, some ascpects of the simulation program are described. It is thought as a guideline or inspiration how one could make simulations using the described techniques. The main program (section C.5) depends on four modules which we describe first.

C.1 Lattice module

This module describes how one can implement particle paths and handle them.

- There is a data type, called `coordinate`, which is used to store the coordinates of a particle at a specific time slice. It consists of an array of d real numbers, representing the coordinate components in d space dimensions.
  For this data type two operations are defined: the difference, which yields the difference vector of two coordinates, and a multiplication that calculates the standard scalar product of two vectors.
- Equipped with the `coordinate` data type, one can represent a path: Store the coordinates at each time slice in an array. This gives a new data type, called `path`. To do this, one has to specify the (integer) number of time slices – stored in the variable `slices`.
- Now the set of particles can be represented by an array which contains particle paths as components. Since we are dealing with two different particle kinds, one has to introduce a label to distinguish them.
- Also included in the `lattice` module are two subroutines, called `hotstart` and `coldstart` whose aim is to generate an initial configuration. In the `coldstart` routine the generated configuration has no kinetic energy: The coordinates of a path take the same random value (within an interval around the origin whose maximum value can be specified) at each time slice. On the other hand, in the `hotstart` routine the coordinates take independent values (again in an interval to be specified) at each time slice. This can result in a high kinetic energy, therefore the name hotstart.

C.2 Buffering module

This module provides data types and routines to implement a buffer (stack). The stack is used to store members (particle coordinates) of the cluster. Roughly it is a list of variable

length to which items can be added (or removed – but we don't need this feature in our simulation).
Such a module could be implemented as follows:

- As already mentioned, the aim of the buffer is to remember which particle coordinates are members of the cluster. Since it would be a waste of memory to store the particle number, the concerning time slice and the coordinates of each member, we attribute to each combination of a particle number and a time slice an integer number, so that the correspondence is one to one. For example, start with the first particle at the first time slice and attribute to this coordinate the label 1. Then enumerate the following time slices of the same particle with the labels 2 up to the number of time slices. Then the next label stands for the coordinates of the second particle at the first time slice and so on, until all coordinates are labeled. Thus the maximum size of the buffer is equal to the number of particles times the number of time slices.

- The labels are stored in an array, whose size is the maximal buffer size and has to be determined when the program has started.

- The initialization of the buffer happens by means of a subroutine. There the maximum size of the buffer is determined (the necessary values are taken from the lattice module) and the buffer size is initally set to zero (empty buffer).

- There is a subroutine which adds a new label to the buffer. It first calculates and then stores the label in the next unoccupied (`BufferSize`+1) array component; afterwards it increases the buffer size by one.

C.3 Module Boxes

In the calculation of the action (-difference), there is a contribution from the interaction energy. If the interaction range is finite and one has many particles, it would be very inefficient to calculate all pair interactions between any two particles (of different kind) before and after a move, because most of the values would be zero and thus have no influence on the action. Therefore one introduces a spatial grid – with the grid length chosen equal to the interaction range and covering a large enough area of space – and makes lists of which particles are contained in which box. When calculating the pair interaction of a specific particle, one then has to search for interaction partners only in a few neighboring boxes (the number depends on the ratio of the box length and the interaction range, but is at least 27 in three space dimensions). This is worthwhile for large particle numbers (and parameters, where not most of the particles are concentrated in a few boxes): For N interacting particles one usually would have to calculate of the order of N^2 pair interactions! Because such a contents list of a box is of variable size (one box can contain any number of particles from zero up to the total particle number), we use a technique called *linked list*, which can be implemented only in programming languages that support a pointer concept. Of course, one could also use normal arrays to store the box contents. But because they have a fixed size which is minimally equal to the particle number N, one would need at least as many arrays of size N as there are boxes. For small grid lengths (with respect to the grid range) and large particle numbers one soon gets memory problems! Linked lists, on the other hand, are more expensive concerning computational operations, but have the advantage of being very flexible

and modest in memory usage.
For example, the boxes module contains the following data types and subroutines:

- A linked list is built up by nodes. Each node has at least two components: A content (i.e. a list item) and a pointer that points to the next node. For the contents we define a special data type called `Contents_Type`; its sole component is an integer (the particle number). The pointer that points to the next node is stored as a component of the type `Box_Type`. Now the variables `Contents` of type `Contents_Type` and `Box` of type `Box_Type` are the components of the self defined type `Node_Type`.
  For a more thorough and perhaps more comprehensible description of linked lists see any good book on data types or structures.[1]

- Because later in the program the contents of several boxes are used and combined into a single list, an array of yet undetermined size, called `List`, is provided.

- A subroutine is used to initialize a linked list. It sets the pointer of the first node in the list to point to nil (no successor yet).

- Another subroutine inserts a content at the beginning of a linked list. Both contents and list are specified as arguments. It happens like this: A node with the contents to be inserted is generated and the pointer of that node points to the first list item. Then the pointer of the root node is altered so that it now points to the newly created node (instead of the old first node).

- In order to remove a given content from a given list, yet another subroutine is provided. It searches the list contents from the root node downwards for the item in question. If the content is found, it is removed (two different cases must be treated: The case where the item to be removed is the last in the list, or not).

- A subroutine is needed to get the number of contents in a given list, a subroutine. It simply counts the nodes while moving through the list from top to bottom.

- Finally, the output of the list contents is handled by an own subroutine. If the contents are to be written into an array, it needs to know the number of contents of the list in question, so one first has to run the subroutine described in the previous item.

Of course, one could also have implemented the buffer from section C.2 in this manner. But it would be less efficient because as already mentioned, pointer operations are more expensive than simple array operations.

C.4 Bootstrap module

In this module, the bootstrap method for error evaluation is implemented (see B.2). The needed parameters (such as the number of replications and the statistical quantity to be estimated) are specified in the main program. It contains

- A variable to store the number of bootstrap replications. It is initialized in the main program.

[1] A quick conceptual introduction can also be found in [11] p. 345ff.

- A subroutine that reads in the sample values, determines their number and stores them in an array from which the resamples are to be drawn.
- An array that contains the resampled values of the statistical parameter to be estimated.
- The actual resampling is done by a subroutine: It chooses at random as many sample numbers as there are samples and compiles the corresponding sample values to a new sample, which is stored in the variable handed to the subroutine as an argument.
- Functions to calculate the sample mean and its error.

C.5 The main program

Here comes now the actual main program that simulates a system with two kinds of distinguishable particles in an external harmonic oscillator potential. There is no interaction among particles of the same kind but a repulsive one between particles of different kind. It depends on the modules described in the previous sections.
The program structure can be seen in the flowchart (see figure C.1) and roughly works like this:

- In the initialisation phase, the model parameters are set and written to an output file. Then a starting configuration is generated. This is usually a hot start or a cold start, described in section C.1. And finally the spatial grid is initialized.
- During the equilibration period, new configurations are generated (via the subroutine `GenerateConfig`, see below) but no measurements are made. The number of equilibrium runs is entered manually.
- Now the actual measurements start: After each generated configuration, the observable values of some configurations are averaged into blocks (the block size is also a parameter entered manually) and form the corresponding block value (see B.3) which is at the same time a sample value. So the whole procedure is repeated until the desired sample number is achieved, each time writing the sample values to the file specified in `datafile`.
- Finally the mean and its error of each observable is computed using the bootstrap method and the results written to the output file, followed by simulation data as the average Metropolis acceptance rate, the number of samples etc.

Now let us take a closer look at the most important subroutines:

- The subroutine `GenerateConfig` is in charge of the generation of new configurations. First several Metropolis sweeps are performed on each particle. Then the cluster move is prepared: One generates a vector of unit length with a random direction (Wolff vector). This may happen as follows: Chose a random coordinate in the unit cube and calculate its norm. If it is smaller than one, normalize the corresponding position vector and accept it as a random direction vector, else generate a new coordinate. The reason that only vectors with a norm smaller than one are accepted is, that the distribution of the directions would else not be uniform: those towards the vertices of the cube would be favored! Once the Wolff vector is at disposition, the subroutine `GrowCluster` is called

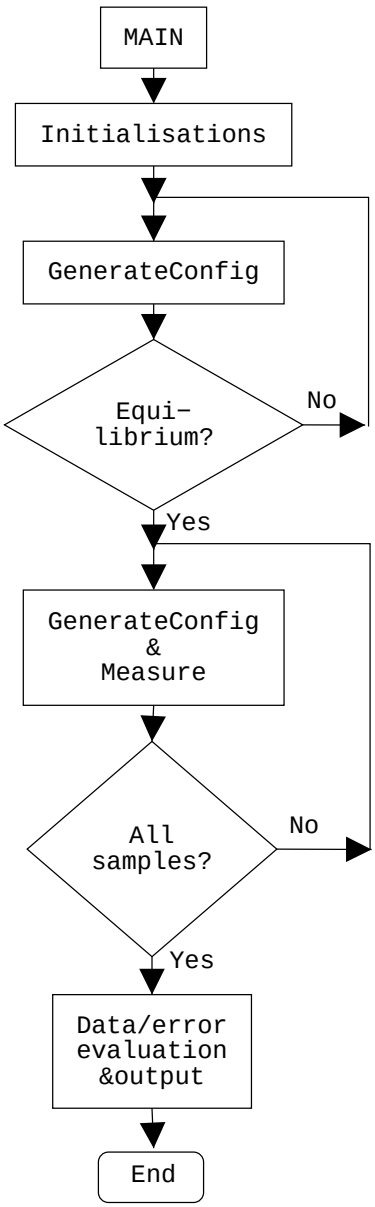

Figure C.1: Flowchart of the main program.

(description follows below) and finally, when the cluster is built, it is flipped in the subroutine `FlipCluster`.

- There is a subroutine `proposenew` which reads in a particle coordinate and proposes a new one. The components of the new coordinates lie with equal probability within an interval whose center is equal to the old component value and has the length $2\delta x$.

- The Metropolis subroutine modifies particle paths. It performs so called sweeps: Starting with the first time slice of the path to be updated, it makes a proposal by means of `proposenew` for the coordinates of that very time slice, leaving the coordinates of the neighboring time slices (aware of the periodic boundary conditions in time!) unchanged. Then the change in the kinetic part of the action with respect to the old three time slice coordinates is computed. Afterwards the same is done for the change in the external potential part of the action – here the neighboring coordinate values are totally unimportant. Finally the contribution of the interaction is calculated: The contents of neighboring boxes are listed (see section C.3) and the interaction energy is calculated (for the old and the new configuration). When the total action difference is computed, the new configuration is accepted with the adequate probability. This may be done as follows: Generate a uniformly distributed random number (between zero and one). If it is smaller than $\exp(-\Delta S)$ then the new configuration is accepted, otherwise rejected. If the configuration is rejected, the old coordinate is restored, else the box contents are updated.
  Then one advances to the next time slice, repeats the whole procedure and so on, until all time slices and particles are treated (one or more times).

- The cluster growth is treated in the subroutine `GrowCluster`. It first initializes the buffer and the member list and then picks a random seed (particle and time slice), adding the corresponding coordinate to the (empty) buffer. Now comes the growth-loop. First, the two time neighbors of the current buffer member are considered: Each time it is checked, whether it is not already in the cluster and the reference configuration (neighbor in the same half space) is satisfied. If this is the case, a bond between the two coordinates is set with the appropriate probability and, if accepted, the neighbor taken into the cluster (added to the buffer and marked as a member in the list). This procedure is repeated for the other time neighbor before the spatial neighbors are considered. Here only those spatial neighbors are taken into account, which could interact with the flipped coordinate (i.e. particles in the neighboring boxes of the flipped coordinate), because only spatial neighbors that are in the other half space (i.e. in the same as the flipped coordinate) can become cluster members due to the reference configuration. Furthermore it is checked whether these neighboring particles are of the other kind, if they are indeed in the other half space and if they are not already a cluster member (a coordinate can have several possibilities to become a cluster member – as a time neighbor or a space neighbor of any cluster member!). If all these conditions are satisfied, the corresponding bond is set with the respective probability and the neighbor added to the cluster or not. So when all the space neighbors in question are checked, the growth loop is repeated with the next buffer member and so on, until no more coordinates are added to the cluster.

- Flipping the cluster members is the task of the subroutine `FlipCluster`. It starts with

the first member in the buffer and reflects the corresponding coordinate with respect to the Wolff plane, updates the box contents and then proceeds with the next buffer member etc., until the end of the buffer is reached.

Acknowledgements

I would like to thank Prof. Uwe-Jens Wiese for suggesting this interesting problem and his excellent support during this work.
I would also like to thank the other institute members for the fruitful discussions and the pleasant atmosphere. Many thanks also to Markus Moser for handing me out his error analysis program. And, of course, I am very grateful to my parents who made this all possible.

Bibliography

[1] C. J. Myatt, E. A. Burt, R. W. Ghrist, E. A. Cornell, and C. E.Wiemann, Phys. Rev. Letters **78**, 586 (1997).

[2] B. D. Esry, C. H. Greene, J. P. Burke, and J. L. Bohm, Phys. Rev. Letters. **78**, 3594 (1997).

[3] R. Ejnisman, H. Pu, Y. E. Young, and N. P. Bigelow, Optics Express **2**, 330 (1998).

[4] T.-L. Ho and V. B. Shenoy, Phys. Rev. Letters. **77**, 3276 (1996).

[5] K. Binder, editor, *Monte Carlo Methods in Statistical Physics*, Springer-Verlag, 1979.

[6] D. M. Ceperley, Rev. Mod. Phys. **67,2**, 279 (1995).

[7] R. P. Feynman and A. R. Hibbs, *Quantum Mechanics and Path Integrals*, McGraw-Hill, 1965.

[8] N. Metropolis, A. W. Rosenbluth, M. N. Teller, and E. Teller, J. Chem. Phys. **21**, 1087ff. (1953).

[9] R. H. Swendsen and J.-S. Wang, Phys. Rev. Lett. **58**, 86 (1987).

[10] U. Wolff, Phys. Rev. Lett. **62**, 381 (1989).

[11] M. E. J. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics*, Claredon Press, Oxford, 1999.

[12] M. Creutz and B. Freedman, Ann. Phys. **132**, 427 (1981).

[13] J. Lamperti, *Stochastic processes*, Springer-Verlag, 1977.

[14] K. L. Chung, *Markov Chains with Stationary Transition Probabilities*, Springer-Verlag, 1960.

[15] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig, *Taschenbuch der Mathematik*, Harri Deutsch, 2001.

[16] B. Efron, Annals of Statistics **7**, 1 (1979).

[17] B. Efron and G. Gong, American Statistician **37**, 36 (1983).