GOETHE UNIVERSITÄT
FRANKFURT AM MAIN

Master's thesis

Implementation of the Variational Monte Carlo method for the Hubbard model

Robert Rüger

Institut für Theoretische Physik
Johann Wolfgang Goethe-Universität
Frankfurt am Main

August 2013

Supervisor: Prof. Dr. Roser Valentí
Second Examiner: Prof. Dr. Eberhard Engel

Contents

1 Introduction

Band theory based upon the independent electron approach has been successfully used to explain many physical properties of solids, such as electrical conductance and optical absorption, and can be seen as the foundation upon which the entire field of solid state theory is built. Perhaps the most prominent feature of band theory is that one can immediately distinguish between metals and insulators by looking at the position of the Fermi level with respect to the bands: If the Fermi level falls within a band (or in other words if at least one band is only partially filled) there are low-lying excitable states and the system is conducting. If the Fermi level falls between bands (or in other words all of the bands are either completely full or empty) excitations take at least as much energy as the gap between the bands, and the system is an insulator. In summary, it is the energy difference between the highest occupied and the lowest unoccupied state that determines whether a system is conducting or insulating.

Band theory is based upon the assumption that there is no interaction in form of the Coulomb repulsion between the electrons, and the only way electrons influence each other is through the Pauli principle which prevents them from occupying the same single particle states. If electron-electron interaction is added perturbatively within band theory, the bands are renormalized but the electrons are still described as non-interacting particles. This does not always work though, and there are materials for which treating the electrons as non-interacting particles does not even *qualitatively* predict the right properties. These materials are said to be strongly correlated or to have strongly correlated electrons because it is the interaction between the electrons that determines their physical properties. Prominent examples of such materials are many transition metal oxides which should be metallic by band theory but turn out to be insulating in experiment [IFT98]. These materials which are made insulating by electron correlation, even though they should be conducting by band theory, are called Mott insulators. [Mot49]

It is surprisingly simple to make a gedankenexperiment that shows how a system that should be conducting by band theory can become Mott insulating. Consider a two-dimensional square lattice of hydrogen atoms. An electron's ability to move from one lattice site to the other depends on the overlap of the atoms' $1s$-orbitals. Within the tight binding framework this is the nearest neighbor hopping matrix element t, which can be calculated using the Wannier orbitals $\phi_i(\vec{r})$ centered at site i through the following equation.

$$t = \int \mathrm{d}\vec{r}\, \phi_i^*(\vec{r}) \left(-\frac{\hbar^2}{2m_e}\vec{\nabla}^2 + V_{\text{ions}}(\vec{r}) \right) \phi_j(\vec{r}) \quad \text{(site } i \text{ and } j \text{ are nearest neighbors)} \tag{1.1}$$

We also want to consider the correlation between the electrons in form of a repulsive Coulomb force. Two electrons on the same site will energetically certainly be unfavorable,

so let us approximate the interaction among the electrons by an energy U that has to be paid for every doubly occupied site. A reasonable way to calculate this energy U is the electrostatic repulsion of the two orbital's electron densities on the same lattice site.

$$U = \int \mathrm{d}\vec{r}_1 \, \mathrm{d}\vec{r}_2 \;\; |\phi_i(\vec{r}_1)| \, \frac{e^2}{|\vec{r}_1 - \vec{r}_2|} \, |\phi_i(\vec{r}_2)| \tag{1.2}$$

The system's behavior is now governed by two opposing tendencies: On the one hand the electrons gain kinetic energy by delocalizing, but on the other hand the repulsion among them tries to localize the electrons on different sites in order to prevent double occupancies. Whether the system ends up being a metal or an insulator depends upon which of the two tendencies wins, or in other words the ratio of U and t. The hopping matrix element t depends on the lattice spacing a between the ions, and decreases when the distance between the ions is increased. The behavior in the limit $a \to \infty$ is obvious: At some point we are dealing with a set of isolated hydrogen atoms which surely are insulating. Since we know the two limiting cases to be conducting at $U/t = 0$ and insulating at $U/t \to \infty$, we can deduce that there must be a metal insulator transition somewhere in between.

Treating the square lattice with band theory we would find one band with the following dispersion relation.

$$\epsilon_{\vec{k}} = -2t\big(\cos(k_x) + \cos(k_y)\big) \tag{1.3}$$

The interesting thing here is that by band theory the lattice of hydrogen atoms should always be conducting. This is most easily seen from the fact that we have one electron but (due to spin) two states per unit cell. This will always give us one band in which exactly half of the available states are occupied, making the system a metal irrespective of the actual value of t, which only controls the width of the band. The fact that band theory predicts a metal but correlation makes it insulating for large enough U/t makes this system a very intuitive example of a Mott insulator.

The system that we have described above is actually just a special case of the so called one-band Hubbard model, which was introduced independently by Hubbard [Hub63], Gutzwiller [Gut63] and Kanamori [Kan63] as a simple microscopic model that takes electron-electron interaction and its competition with the kinetic energy into account. The following is the Hubbard Hamiltonian written in second quantization.

$$\hat{H} = -\sum_{i<j} t_{ij} \sum_{\sigma} \left(\hat{c}^{\dagger}_{i\sigma} \hat{c}_{j\sigma} + \text{h.c.} \right) + U \sum_i \hat{n}_{i\uparrow} \hat{n}_{i\downarrow} \tag{1.4}$$

This Hamiltonian is slightly more general than the two-dimensional hydrogen square lattice considered before, as it can in principle exist on an arbitrary lattice and we also have general hopping matrix elements t_{ij}, which will usually quickly get smaller as the distance between the sites i and j increases. Note that the Hubbard model will only show Mott insulating behavior at half filling, that is if the number of electrons in the system is equal to the number of sites, as is the case for a lattice of hydrogen atoms. Away from half filling the system would always have unoccupied sites (holons) or doubly occupied sites (doublons) which are essentially free positive/negative charges that would start moving when an electric field is applied, making the system metallic. Doping the Hubbard model with holes or additional electrons is certainly interesting, especially

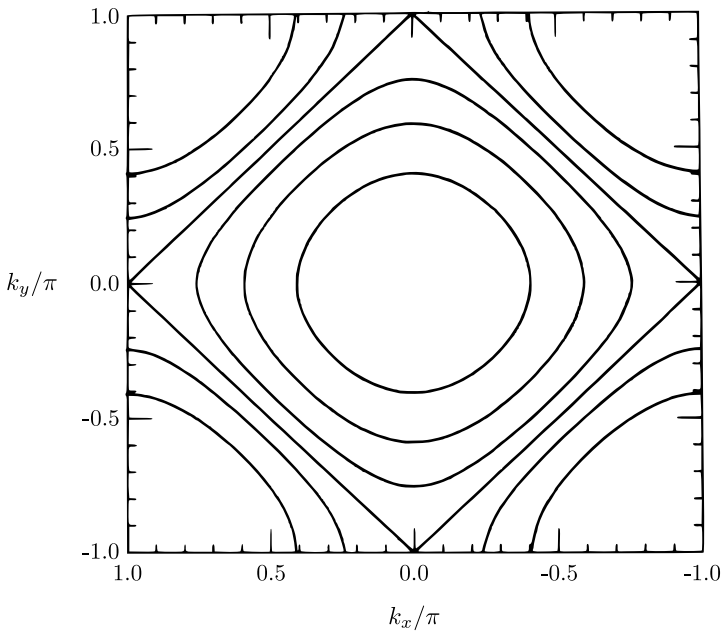

Figure 1.1: Fermi surfaces for electrons on a two-dimensional square lattice with nearest neighbor hopping. Band fillings are 0.25, 0.5, ..., 1.5 from the inside out. Note that the diamond shaped surface corresponding to half filling is perfectly nested with $\vec{q} = (\pi, \pi)$. Image modified from [Hir85].

since it is known that high temperature superconductors are obtained by doping a Mott insulator [LNW06], but within the scope of this thesis we will only look at the half filled case.

One thing that we have not yet mentioned is that systems which are prone to become Mott insulating very often also show an antiferromagnetic ordering of the spins. The problem with this is that antiferromagnetism itself can already in band theory make a system insulating. If we consider the square lattice of hydrogen atoms again, then the introduction of a Néel ordered antiferromagnetism increases the unit cell to two atoms. In band theory we would get two bands and enough electrons to fill one of those, making an insulating behavior possible. In this way the correlation induced Mott insulating behavior can be masked by a magnetic ordering.

It was indeed shown that the Hubbard model on a two-dimensional square lattice with nearest neighbor hopping exhibits a Néel ordered antiferromagnetism at zero temperature for any $U > 0$. For small U this can be seen by looking at the susceptibility $\chi(\vec{q})$ in Random Phase Approximation, which is given by the following equation. [Hir85]

$$\chi(\vec{q}) = \frac{2\chi_0(\vec{q})}{1 - U\chi_0(\vec{q})} \quad \text{with} \quad \chi_0(\vec{q}) = \frac{1}{L}\sum_{\vec{k}} \frac{f(\epsilon_{\vec{k}+\vec{q}}) - f(\epsilon_{\vec{k}})}{\epsilon_{\vec{k}+\vec{q}} - \epsilon_{\vec{k}}} \tag{1.5}$$

Here $\epsilon_{\vec{k}}$ is the dispersion relation of the $U = 0$ case, and f the Fermi distribution. The important property of the square lattice with nearest neighbor hopping is that at half filling the Fermi surface is perfectly nested with the wavevector $\vec{q} = (\pi, \pi)$, as can be seen in figure 1.1. This causes a divergence in the bare susceptibility $\chi_0(\vec{q})$ for $\vec{q} \to (\pi, \pi)$. The susceptibility $\chi(\vec{q})$ itself diverges if $U\chi_0(\vec{q}) = 1$, which happens no matter how

small U is for some vector $\vec{q}$ close to (π, π), due to the divergence of $\chi_0(\vec{q})$ at that point. Therefore the square lattice Hubbard model shows Néel ordered antiferromagnetism at half filling for any small but finite U. In the limit of large U double occupancies are completely forbidden and the half filled Hubbard model goes into the so called spin-$\frac{1}{2}$ Heisenberg model, given by the following Hamiltonian. [CA76]

$$\hat{H} = J \sum_{\langle ij \rangle} \hat{\vec{S}}_i \cdot \hat{\vec{S}}_j \qquad \text{with} \qquad J = \frac{4t^2}{U} \tag{1.6}$$

The Heisenberg model itself has been subject of intense research and it is well established that it shows Néel ordered antiferromagnetism on a square lattice, see [Man91, chapter III.F.2]. As both the large and the small U limit have antiferromagnetism, it is natural to assume that this order is found for any value of U and we will indeed confirm that numerically in a later chapter.

In order to avoid the issue that magnetism is masking the Mott insulation and to be able to study a real metal to Mott insulator transition, one could use a frustrated lattice or alternatively hoppings to more than nearest neighbors, both of which removes the Fermi surface nesting responsible for the magnetism. In two dimensions it could also make sense to artificially suppress magnetism, as the Mermin-Wagner theorem rules out the possibility of having a magnetic order in one and two dimensions for any nonzero temperature. [MW66]

Having presented the Hubbard model, it is now time to look at methods to solve it! Despite its apparent simplicity it is far from easy to solve the Hubbard model, and an exact analytic solution is only available in one dimension [LW68]. The difficulty of finding analytic solutions for strongly correlated electrons has however stimulated the development of various numerical methods which have been applied to the Hubbard model.

The most straightforward of these methods is the exact diagonalization of the Hubbard Hamiltonian. Here one basically writes down the Hamiltonian's matrix in a basis of the Fock space, e.g. the basis of electron configurations where the basis states have fixed particle positions and spins along the z-axis. In principle one can always do this and then diagonalize the resulting matrix numerically, but in practice this is only possible for very, very small systems due to the fact that for the Hubbard model the number of basis states (and therefore the size of the matrix) scales as 4^L, where L is the number of lattice sites. This restriction to tiny systems of course entails large finite size effects, so that overall it can be better to *approximate* the solution of a large, possibly infinite system's Hamiltonian than to solve a small system's Hamiltonian exactly.

It would be out of the scope of this thesis to even list the different numerical methods that have been applied to the Hubbard model, but such an incomplete list can be found in reference [Sca06]. Dynamical Mean Field Theory (see [GKKR96]) and its cluster extensions have recently been especially popular, but then there is also a whole zoo of different Quantum Monte Carlo methods and many, many others. From the abundance of methods around one can already infer that none of them is truly perfect, but they all come with different advantages and limitations.

Within the class of Quantum Monte Carlo methods there is a method called Variational Monte Carlo (VMC). This method was introduced by McMillan in 1965 to calculate the

ground state of liquid ^{4}He [McM65], and in 1977 applied to a fermionic system for the first time [CCK77]. VMC is the method of our choice and its basic idea is to use the Rayleigh-Ritz principle [Rit09] to approximate the ground state through a variational many-body wavefunction. It is a Monte Carlo method because internally a Monte Carlo integration is used to evaluate the sum over a high dimensional configuration space. We will have an in-depth look at the Variational Monte Carlo method, as the primary purpose of this thesis is to demonstrate how VMC can be applied to the Hubbard model.

Software development is something that naturally comes up if one wants to work with numerical methods. Variational Monte Carlo for the Hubbard model is no exception, and as a part of this thesis a VMC implementation for the Hubbard model was written from scratch in C++ and published as free software under the name `hVMC`. `hVMC` is is not the only VMC code for the Hubbard model, and while old and powerful codes already exist, developing from scratch offers a unique insight into the details of the method and especially the aspect of computational efficiency that is mostly hidden to users of ready-made codes. This thesis is also intended to serve as a documentation of the `hVMC` code.

The rest of this thesis is organized into five chapters and three appendices. In chapter 2 we are going to introduce the Variational Monte Carlo method in general, and derive the Metropolis algorithm that is used to sample the configuration space. We also present the Stochastic Reconfiguration method which we are going to use for the optimization of the variational wavefunction, as well as a convergence detection based on the Mann-Kendall test. Chapter 3 is dedicated to the application of VMC to the Hubbard model. A powerful variational wavefunction will be presented, as well as some tricks that reduce the computational complexity by one power. Chapter 4 gives an overview over the `hVMC` code that implements the two preceding chapter's concepts in C++. In chapter 5 we will present simulation results for the square lattice bilayer Hubbard model at half filling, and compare those to results that were obtained using Dynamical Mean Field Theory. Chapter 6 summarizes the thesis. Appendix A.1 approaches the `hVMC` code from a user's perspective and gives some practical information on how to build and run `hVMC`. The second appendix A.2 reviews the different levels of parallelism in today's supercomputers and to what extend they are used by `hVMC`. Finally, the last appendix A.3 proofs some mathematical theorems that were used in the body of the thesis.

There is a bibliography at the end of the thesis which will be referred to at the respective points in the text. There are some sources though that have undoubtedly influenced this thesis, but which can hardly be cited at any specific point. These are all the PhD theses on VMC that were written at SISSA under the supervision of Sandro Sorella and Federico Becca, most importantly the thesis of Manuela Capello [Cap06], and to some degree the thesis of Luca Tocchio [Toc08]. The SISSA lecture notes on numerical methods for strongly correlated electrons by Sandro Sorella and Federico Becca [SB13], have also been very illuminating. Finally I would like to mention that the section on importance sampling and Markov chains is heavily based on my own bachelor's thesis.

2 The Variational Monte Carlo method

2.1 The underlying ideas of VMC

Let us start by stating the basic ideas which power the Variational Monte Carlo method. In order to have a uniform language and notation throughout the thesis, we will assume already in this section that the quantum mechanical system we want to describe are fermions on a lattice. This is not necessary from a mathematical point of view, and one can, for example, easily translate all equations to the continuous case by replacing the sums with integrals.

Let us first define an electron configuration $|x\rangle$ as a state in which all electrons have a defined position and spin along the z-axis. This state can be obtained by simply applying the respective creation operators $\hat{c}^\dagger_{i\sigma}$ to the vacuum state $|\emptyset\rangle$. Consider for example a system with $L = 4$ sites for which the following would be one of the electron configurations $|x\rangle$.[1]

$$|x\rangle = |\downarrow, \updownarrow, 0, \uparrow\rangle = \hat{c}^\dagger_{2\uparrow}\hat{c}^\dagger_{4\uparrow}\hat{c}^\dagger_{1\downarrow}\hat{c}^\dagger_{2\downarrow}\,|\emptyset\rangle \tag{2.1}$$

The set of all possible configurations $|x\rangle$ forms a basis of the many-particle Hilbert space in which any wavefunction can be expanded.

$$|\Psi\rangle = \sum_x |x\rangle\,\langle x|\Psi\rangle \quad \Leftrightarrow \quad \mathbb{1} = \sum_x |x\rangle\,\langle x| \tag{2.2}$$

We will start by inserting the identity operator in this form into the general equation for the expectation value of any quantum mechanical observable $\hat{O} = \hat{O}^\dagger$.

$$\langle\hat{O}\rangle = \frac{\langle\Psi|\hat{O}|\Psi\rangle}{\langle\Psi|\Psi\rangle} = \frac{\sum_x \langle\Psi|x\rangle\,\langle x|\hat{O}|\Psi\rangle}{\sum_x \langle\Psi|x\rangle\,\langle x|\Psi\rangle} = \frac{\sum_x \frac{\langle x|\hat{O}|\Psi\rangle}{\langle x|\Psi\rangle}\,\langle\Psi|x\rangle\,\langle x|\Psi\rangle}{\sum_x \langle\Psi|x\rangle\,\langle x|\Psi\rangle} = \frac{\sum_x \frac{\langle x|\hat{O}|\Psi\rangle}{\langle x|\Psi\rangle}\,|\langle x|\Psi\rangle|^2}{\sum_x |\langle x|\Psi\rangle|^2} \tag{2.3}$$

The quantity $O_{\mathrm{loc}}(x) = \frac{\langle x|\hat{O}|\Psi\rangle}{\langle x|\Psi\rangle}$ is called the local value of the observable $\hat{O}$, and $|\langle x|\Psi\rangle|^2$ is just the probability to find the system with wavefunction $|\Psi\rangle$ in configuration x.

$$\langle\hat{O}\rangle = \frac{\sum_x O_{\mathrm{loc}}(x)\,|\langle x|\Psi\rangle|^2}{\sum_x |\langle x|\Psi\rangle|^2} \tag{2.4}$$

This does not really make things simpler yet. The problem here is that the number of distinguishable configurations x increases exponentially with the number of particles involved, which makes calculating the sum impractical for anything but the smallest systems.

As summing over all possible configurations is unrealistic, an alternative approach might be to only sum over a randomly selected multisubset $\mathcal{S}_1 = \{\, x_1, \ldots, x_N \,\}$ of them.

[1] At this point it does not matter, but to be consistent with the rest of the thesis we have chosen the order of fermionic creation and annihilation operators such that operators corresponding to spin up particles are always left of operators for spin down particles.

This is of course an approximation, but one that disappears in the limit of a large multisubset.

$$\langle \hat{O} \rangle \approx \frac{\sum_{x \in \mathcal{S}_1} O_{\mathrm{loc}}(x) \left| \langle x|\Psi \rangle \right|^2}{\sum_{x \in \mathcal{S}_1} \left| \langle x|\Psi \rangle \right|^2} \tag{2.5}$$

Note that we want $\mathcal{S}_1$ to be a multisubset rather than a simple subset, meaning that we specifically allow the same configuration to occur multiple times within $\mathcal{S}_1$. This does not make much of a difference for the quality of the approximation, but removes the necessity to check if a randomly generated configuration is different from all the previous configurations, which makes the construction of $\mathcal{S}_1$ much easier.

Equation (2.5) in practice means to generate configurations x_i randomly, calculate the local values of the observable $O_{\mathrm{loc}}(x_i)$, and weigh the results with the probabilities $|\langle x_i|\Psi\rangle|^2$. This has a big disadvantage though: If we select the configurations completely randomly, it is not guaranteed that we will actually select any of the configurations which have a large overlap with the wavefunction and therefore a large weight in the sum. Consider for example the case where we correlate our electrons through a strong repulsive potential, so that the correlation would make double occupancies unfavorable. Configurations with a large weight will not have many double occupancies, but a completely random distribution does not take this into account and would make one quarter of the sites doubly occupied at half filling! The solution to this problem is called importance sampling: Instead of generating configurations randomly and weighing the results with their probabilities, we might as well generate configurations according to their probabilities and weigh them evenly. Mathematically speaking, we define a multisubset $\mathcal{S}_\rho$ that contains N configurations distributed according to $\rho \propto |\langle x|\Psi\rangle|^2$, and sum over all of its members.

$$\langle \hat{O} \rangle \approx \frac{1}{N} \sum_{x \in \mathcal{S}_\rho} O_{\mathrm{loc}}(x) \tag{2.6}$$

Since the remaining term in the denominator's sum is the number "1", equation (2.5) is transformed into this very appealing form in which we simply average the local value of the observable over a number of configurations distributed according to $\rho \propto |\langle x|\Psi\rangle|^2$.

At the moment it is still completely unclear how we would actually generate configurations following some probability distribution. This will be the topic of section 2.2, so let us at this point simply assume that we know how to do it.

Equation (2.6) is valid for *any* physical observable, including of course the Hamiltonian $\hat{H}$ of the system itself. As we can evaluate the expectation value of the Hamiltonian, we can now use the Rayleigh-Ritz principle [Rit09] together with a variational wavefunction to find an approximation to the system's ground state. The Rayleigh-Ritz principle states that the energy expectation value of a wavefunction is bounded from below by the exact ground state energy. This can easily be proven for a Hamiltonian with the eigenenergies $E_0 \leq E_1 \leq \ldots \leq E_N$ and the corresponding eigenstates $|n\rangle$.

$$E_\Psi = \frac{\langle \Psi|\hat{H}|\Psi\rangle}{\langle \Psi|\Psi\rangle} = \frac{\sum_{n=0}^{N} E_n \langle \Psi|n\rangle \langle n|\Psi\rangle}{\langle \Psi|\Psi\rangle} \geq \frac{\sum_{n=0}^{N} E_0 \langle \Psi|n\rangle \langle n|\Psi\rangle}{\langle \Psi|\Psi\rangle} = E_0 \tag{2.7}$$

The Rayleigh-Ritz principle tells us that if we have a variational wavefunction $|\Psi\rangle$ that depends on a set parameters $\{\alpha_i\}$, the best choice of the parameters – the one which

makes the variational wavefunction as close as possible to the real ground state – is the one that minimizes the expectation value of the energy.

Having assembled the two main ingredients, we can now state the general Variational Monte Carlo algorithm:

1. Find the variational parameters that minimize the energy expectation value of the trial wavefunction. Any expectation values needed to do this can be evaluated with equation (2.6).

2. Use equation (2.6) again with the optimized wavefunction to calculate the desired observables.

The Variational Monte Carlo method can only be as good as the ansatz that one makes for the variational wavefunction. When we later specialize things for the Hubbard model we will see that we need quite an involved ansatz with dozens of variational parameters to capture the relevant physics. The fact that one can only get from the method what one puts into the ansatz is probably the biggest disadvantage of VMC, as it makes missing the phenomena that one was *not* expecting particularly easy. On the other hand one can always try different variational wavefunctions and identify the best ansatz as the one which gave the lowest energy. The fact that within VMC one can rigorously benchmark different approximations against each other and obtain a clear winner is a huge advantage of the method.

The other approximation that went into the VMC method is the Monte Carlo evaluation of expectation values which gives us a statistical uncertainty in the results. This is not as big of a problem as it may seem: On the one hand one can always decrease the uncertainty by using more configurations to evaluate the local value of the observable, and on the other hand one usually does not need to calculate something to an arbitrarily high precision.

This section has introduced the two basic ideas behind any Variational Monte Carlo calculation. Two things have remained quite vague though, namely how to generate electronic configurations according to some probability distribution and how to find the variational parameters that minimize the energy. These points will be clarified in the next two sections.

2.2 Importance sampling using Markov chains

In order to use equation (2.6) in practice we need a method to generate electron configurations that are distributed according to $\rho(x) \propto |\langle x|\Psi\rangle|^2$. A class of widely used methods to sample a distribution are the so called Markov chain methods, named after the Russian mathematician Andrey Markov, which we briefly want to review in this section. In general a Markov process is a stochastic operation that takes a system from one state to the other. It is stochastic in the sense that only transition probabilities are specified, and the same process acting in the same system in the exact same state does not need to produce the same outcome every time. The series of states a system passes through if the same Markov process is applied over and over again is called the a Markov chain. Note that Markov processes have no memory in the sense that the

transition probabilities only depend on the current state of a system and not on the history of how the system got into that state.

The reason why Markov chains are a suitable tool for our problem is quickly explained: Our goal is to generate electronic configurations that have a large overlap with the wavefunction. If we have found such a configuration x, it is a reasonable assumption that another configuration x' which differs not too much from x (e.g. only by the displacement of a single electron) also has a large overlap with the wavefunction. If we could find a Markov process that generates a subsequent configuration x' through small modifications to the existing configuration x, and whose Markov chain contains states distributed according to $\rho(x) \propto |\langle x|\Psi\rangle|^2$, we could use those states in equation (2.6) to calculate quantum mechanical expectation values. Another rather obvious property of the Markov process we are looking for is its ergodicity, meaning that it must be possible to reach any state from any other state with a finite number of applications of the Markov process. In the remainder of this section, we will present the most popular of such processes, namely the Metropolis algorithm with local updates. [MRR$^+$53]

2.2.1 Derivation of the Metropolis algorithm

Let $T(x \to x')$ be the probability that the Markov process acting on the electrons in configuration x generates configuration x'. This probability is called the transition probability T. Note that the configuration x' can in general be x itself, so the Markov process can have a nonzero probability to leave the system's state unchanged. That said, the sum of all transition probabilities must obviously be unity.

$$\sum_{x'} T(x \to x') = 1 \tag{2.8}$$

Let $P_k(x)$ be the electrons' probability to be in configuration x after we have applied the Markov process k times. The probability $P_{k+1}(x)$ can then be calculated by considering all transitions into and out of configuration x.

$$P_{k+1}(x) = P_k(x) + \sum_{x'} \Big(P_k(x')\, T(x' \to x) - P_k(x)\, T(x \to x') \Big) \tag{2.9}$$

This equation is called the Markov chain's master equation. Since we want the generated states to follow the distribution $\rho(x)$, it is at this point straightforward to demand that $P_k(x) = \rho(x)$ for all values of k. Looking at the master equation, one can see that a sufficient condition for this is that the two terms in the sum cancel one another for each configuration x'. This condition is known as detailed balance.

$$\rho(x')\, T(x' \to x) = \rho(x)\, T(x \to x') \qquad \Leftrightarrow \qquad \frac{\rho(x')}{\rho(x)} = \frac{T(x \to x')}{T(x' \to x)} \tag{2.10}$$

We now have an equation for the transition probabilities that are required to sample the distribution ρ. There is still one problem though: In general it will be extremely difficult to find a simply applicable Markov process that has exactly the right transition probabilities $T(x \to x')$. In order to overcome this problem, we make the transition a two step process where a new configuration is first suggested with a probability $S(x \to x')$ and then accepted or reject with a probability $A(x \to x')$.

$$T(x \to x') = S(x \to x')\, A(x \to x') \tag{2.11}$$

If a suggested configuration x' is rejected, the electrons' configuration remains unchanged, so rejected configurations ultimately only increase the probability $T(x \to x)$ that the system remains in the same state.

The famous Metropolis algorithm together with local updates is a particularly popular choice for the suggestion and acceptance probabilities, and was also used as the Markov process throughout this thesis. To obtain the Metropolis algorithm we choose a symmetric suggestion probability, meaning that $S(x \to x') = S(x' \to x)$. This symmetry is usually very easy to ensure, and a possible symmetric method for electrons on a lattice would be to pick a random electron and propose to hop it to a random neighboring site.[2] Once we have chosen symmetric suggestion probabilities, they cancel out completely from the equation.

$$\frac{\rho(x')}{\rho(x)} = \frac{S(x \to x')}{S(x' \to x)} \frac{A(x \to x')}{A(x' \to x)} = \frac{A(x \to x')}{A(x' \to x)} \tag{2.12}$$

The desired probability distribution ρ now directly determines the ratio of the acceptance probabilities. As $A(x \to x')$ and $A(x' \to x)$ are probabilities, they obviously have to be between zero and one, but since only their ratio is determined by $\rho(x) \propto |\langle x|\Psi\rangle|^2$ we can multiply both with a common factor so that the larger one is exactly unity.

$$A(x \to x') = \begin{cases} \frac{\rho(x')}{\rho(x)} & \text{if } \rho(x') < \rho(x) \\ 1 & \text{otherwise} \end{cases} = \min\left(1, \frac{\rho(x')}{\rho(x)}\right) \tag{2.13}$$

This acceptance probability together with the symmetric suggestion probability defines the Metropolis algorithm, whose popularity stems from the fact that symmetric suggestions probabilities and the above acceptance criterion are very easy to implement for a wide range of applications.

There one issues that we have swept under the rug in this section: Looking again at the master equation (2.9) and the condition of detailed balance, we see that $P_{k+1}(x)$ will only be the same as $P_k(x)$ if the $P_k(x)$ itself was already distributed as $\rho(x)$. The question that comes up at this point is how one gets to the right distribution in the first place. In practice we would initialize the Markov chain in a specific configuration x'' making the initial distribution $P_0(x) = \delta_{x,x''}$ which is of course very different from the desired distribution $\rho(x)$. At this point we would only like to mention that the detailed balance together with the ergodicity is a sufficient condition for the Markov process' property to have a unique limiting distribution to which it will converge independently from the initialization. A proof of this theorem can be found in [Häg02, chapter 5]. A practical consequence of this is that one should always discard a certain number of configurations at the beginning of the Markov chain because in general the probability $P_k(x)$ first had to converge to its limiting distribution $\rho(x)$. This process is called the equilibration of the system.

Summing things up, this is the procedure that we will use to generate electronic configurations on a lattice that follow the distribution $\rho(x) \propto |\langle x|\Psi\rangle|^2$.

1. Pick a random spin-σ electron and propose to hop it to a random site in its vicinity.

[2] Note that this includes propositions that hop electrons to already occupied sites. This is necessary to ensure the symmetry of the suggestion probabilities, but we will simply choose their acceptance probability to be zero in order to prevent those unphysical moves.

2. If the target site not occupied by another spin-σ electron, accept the hop with the probability min $\left(1, \left|\frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle}\right|^2\right)$, otherwise reject.

3. Iterate step **1.** and **2.** and discard the configurations to ensure equilibration.

4. Iterate step **1.** and **2.** until enough configurations have been generated.

2.2.2 Markov chain decorrelation and calculation of uncertainties

The production of states through a Markov process with local updates as explained in the last subsection is a double edged sword: On the one hand the locality of the updates ensures a large acceptance ratio, on the other hand adjacent configurations in the Markov chain are almost the same, which introduces a correlation between local observable measurements.

Correlation between measurements of the local observables should be avoided since it prevents us from calculating the uncertainties in our expectation values, and can in extreme cases (that is if the Markov chain is not much longer than the number of steps needed for decorrelation) even produce biased expectation values. Mathematically, the reason for this is that the variance of a sum is only equal to the sum of the variances if the random variables X_k are uncorrelated.

$$\mathrm{Var}\left(\sum_k X_k\right) = \sum_k \mathrm{Var}(X_k) \quad \text{for uncorrelated } X_k \tag{2.14}$$

This is exactly the property we would need in order to be able to calculate the uncertainty of our expectation values in a simple way.

$$\mathrm{Var}\left(\langle\hat{O}\rangle\right) = \mathrm{Var}\left(\frac{1}{N}\sum_{x\in\mathcal{S}_\rho} O_{\mathrm{loc}}(x)\right) = \frac{1}{N^2}\,\mathrm{Var}\left(\sum_{x\in\mathcal{S}_\rho} O_{\mathrm{loc}}(x)\right) \tag{2.15}$$

$$\underset{\substack{\uparrow\\ \text{no}\\ \text{corr.}}}{=} \frac{1}{N^2}\sum_{x\in\mathcal{S}_\rho} \mathrm{Var}\left(O_{\mathrm{loc}}(x)\right) = \frac{\mathrm{Var}\left(O_{\mathrm{loc}}(x)\right)}{N} \tag{2.16}$$

In the following we will describe a method to decorrelate the measurements of the local observables, so that the above equation holds and we can estimate our uncertainties in this simple fashion.

The first thing we are going to do is to define a Monte Carlo step that subdivides the Markov chain in order to make the resulting sections independent from the size of the system being simulated. A good way to do that is to define a Monte Carlo step as a number of applications of the Markov process equal to the number of particles in the system.[3] Since at least every particle must have had the chance to move before one can expect to have anything decorrelated, it does not make any sense to measure local observables more often than once per Monte Carlo step.

It will usually take multiple Monte Carlo steps until the local values of the observables have decorrelated. One way to get truly uncorrelated measurements, and to be able

[3] We are actually going to define a Monte Carlo step as a number of Markov process applications equal to the number of lattice sites. This does not make much of a difference (and no difference at all at half filling), and was only done to be in line with another VMC code for the Hubbard model that defines it in this way.

to calculate uncertainties with equation (2.15), is to measure only every few Monte Carlo steps. Another option is a method called binning or blocking. Here we divide the Markov chain of N total configurations into N_b so called bins of n_b configurations, where every bin is long enough so that correlation between the bins can be neglected.

$$\langle \hat{O} \rangle \approx \frac{1}{N} \sum_{x \in \mathcal{S}_\rho} O_{\mathrm{loc}}(x) \quad \Rightarrow \quad \langle \hat{O} \rangle \approx \frac{1}{N_b} \sum_{b=1}^{N_b} \langle \hat{O} \rangle_b \quad \text{with} \quad \langle \hat{O} \rangle_b = \frac{1}{n_b} \sum_{x \in \mathcal{S}_\rho}^{n_b} O_{\mathrm{loc}}(x) \tag{2.17}$$

As we are only reordering a sum, this does not make any difference for the expectation value, but since the bins' means $\langle \hat{O} \rangle_b$ are no longer correlated one can use their variance to calculate the uncertainty of the expectation value.

$$\mathrm{Var}\left(\langle \hat{O} \rangle \right) = \frac{\mathrm{Var}\left(\langle \hat{O} \rangle_b \right)}{N_b} = \frac{\frac{1}{N_b - 1} \sum_{b=1}^{N_b} \left(\langle \hat{O} \rangle_b - \langle \hat{O} \rangle \right)^2}{N_b} \tag{2.18}$$

Note that for a given number N of total Monte Carlo steps in the Markov chain, the exact number of Monte Carlo steps per bin does not really matter, as long as the bins remain uncorrelated among each other: Assuming that the bins are already uncorrelated, a further increase of the binlength n_b would decrease the number of bins N_b in the above equation, but at the same time decrease the variance of the $\langle \hat{O} \rangle_b$ themselves, as they now average over more measurements.

In this section we have seen how the Metropolis algorithm can be used to generate a Markov chain of electronic configurations which can be used as $\mathcal{S}_\rho$ in equation (2.6). This was the first of the two main ingredients to the general VMC algorithm that we have identified in section 2.1. The last missing ingredient is now the energy minimization procedure that we will present in the next section.

2.3 Energy minimization with the Stochastic Reconfiguration algorithm

We have seen in section 2.1 that the basic idea of the Variational Monte Carlo method is to find the best approximation to the true ground state of a system through a variational wavefunction and the Rayleigh-Ritz principle. As the variational wavefunctions can be quite complicated many-body objects, this energy minimization can only be done iteratively. In principle it could be done using general purpose optimization methods, but specialized methods have been developed that are particularly efficient in the VMC context. Other than being able to optimize a large number of variational parameters at once, these methods can also handle the statistical uncertainties in the gradients that one can not avoid when one has to calculate them using a Monte Carlo method.

The optimization algorithm of our choice is called Stochastic Reconfiguration and was developed in the group of Sandro Sorella in the early 2000s. [Sor01] [CAS04]

Let $|\Psi\rangle$ be a trial wavefunction that depends on a set of p variational parameters $\{\alpha_1, \ldots, \alpha_p\}$. In order to determine how we need to change the variational parameters in order to get a lower energy, we first make the ansatz of setting the new state $|\Psi'\rangle$ equal to the Taylor expansion of $|\Psi\rangle$ in the variational parameters.

$$|\Psi'\rangle = \delta\alpha_0 |\Psi\rangle + \sum_{k'=1}^{p} \delta\alpha_{k'} \frac{\partial}{\partial \alpha_{k'}} |\Psi\rangle \tag{2.19}$$

We now need to determine the $\delta\alpha_i$ which yield a $|\Psi'\rangle$ with an energy lower than $|\Psi\rangle$.

Equation (2.19) can be rewritten in the following way, since the configurations $|x\rangle$ form a complete basis of the many particle Hilbert space and do not depend on the variational parameters.

$$|\Psi'\rangle = \delta\alpha_0 |\Psi\rangle + \sum_{k'=1}^{p} \delta\alpha_{k'} \frac{\partial}{\partial\alpha_{k'}} \sum_x |x\rangle \langle x|\Psi\rangle \tag{2.20}$$

$$= \delta\alpha_0 |\Psi\rangle + \sum_{k'=1}^{p} \delta\alpha_{k'} \sum_x \frac{\partial \langle x|\Psi\rangle}{\partial\alpha_{k'}} |x\rangle \tag{2.21}$$

$$= \delta\alpha_0 |\Psi\rangle + \sum_{k'=1}^{p} \delta\alpha_{k'} \sum_x \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_{k'}} |x\rangle \langle x|\Psi\rangle \tag{2.22}$$

Let us define a logarithmic derivative operator $\hat{\Delta}_{\Psi k}$ through its spectral representation.

$$\hat{\Delta}_{\Psi k} = \begin{cases} \mathbb{1} & \text{for } k = 0 \\ \sum_x \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_k} |x\rangle \langle x| & \text{for } k \neq 0 \end{cases} \tag{2.23}$$

This operator can easily be identified in (2.22) and the equation rewritten.

$$|\Psi'\rangle = \sum_{k'=0}^{p} \delta\alpha_{k'} \hat{\Delta}_{\Psi k'} |\Psi\rangle \tag{2.24}$$

In order to reduce the energy of the trial wavefunction we will apply the operator $(\Lambda - \hat{H})$ with a large enough Λ to it. This is known as the power method, and we can easily convince ourselves that repeated application of this operator projects out the Hamiltonian's ground state by expanding the trial wavefunction in the eigenstates $\hat{H}|n\rangle = E_n|n\rangle$ of the Hamiltonian.

$$|\Psi\rangle = \sum_{n=0}^{\infty} \langle n|\Psi\rangle |n\rangle \quad \Rightarrow \quad \left(\Lambda - \hat{H}\right)^N |\Psi\rangle = \sum_{n=0}^{\infty} (\Lambda - E_n)^N \langle n|\Psi\rangle |n\rangle \tag{2.25}$$

As $(\Lambda - E_0) \geq (\Lambda - E_i)\ \forall\ i > 0$ the ground state will be projected out in the limit $N \to \infty$. Setting our expression for $|\Psi'\rangle$ equal to $(\Lambda - \hat{H})|\Psi\rangle$ and projecting onto $\hat{\Delta}^\dagger_{\Psi k}|\Psi\rangle$ gives a linear system of equations for the $\delta\alpha$.

$$\sum_{k'=0}^{p} \delta\alpha_{k'} \langle\Psi|\hat{\Delta}_{\Psi k}\hat{\Delta}_{\Psi k'}|\Psi\rangle = \langle\Psi|\hat{\Delta}_{\Psi k}\left(\Lambda - \hat{H}\right)|\Psi\rangle \qquad \text{with } k \in \{0, \ldots, p\} \tag{2.26}$$

Dividing both sides by $\langle\Psi|\Psi\rangle$ this equation can be rewritten in terms of expectation values.

$$\sum_{k'=0}^{p} \delta\alpha_{k'} \langle\hat{\Delta}_{\Psi k}\hat{\Delta}_{\Psi k'}\rangle = \Lambda\langle\hat{\Delta}_{\Psi k}\rangle - \langle\hat{\Delta}_{\Psi k}\hat{H}\rangle \tag{2.27}$$

Let us solve the equation with $k = 0$ for $\delta\alpha_0$ by using the property that $\hat{\Delta}_{\Psi 0} = \mathbb{1}$.

$$\delta\alpha_0 = \Lambda - \langle\hat{H}\rangle - \sum_{k'=1}^{p} \delta\alpha_{k'} \langle\hat{\Delta}_{\Psi k'}\rangle \tag{2.28}$$

We can now substitute the value for $\delta\alpha_0$ into the equations for $k \neq 0$.

$$\left(\Lambda - \langle\hat{H}\rangle - \sum_{k'=1}^{p} \delta\alpha_{k'} \langle\hat{\Delta}_{\Psi k'}\rangle\right) \langle\hat{\Delta}_{\Psi k}\rangle + \sum_{k'=1}^{p} \delta\alpha_{k'} \langle\hat{\Delta}_{\Psi k}\hat{\Delta}_{\Psi k'}\rangle = \Lambda\langle\hat{\Delta}_{\Psi k}\rangle - \langle\hat{\Delta}_{\Psi k}\hat{H}\rangle \tag{2.29}$$

Rearranging the terms yields the final system of equations which determine the $\delta\alpha_k$.

$$\sum_{k'=1}^{p} \delta\alpha_{k'} \left(\langle \hat{\Delta}_{\Psi k} \hat{\Delta}_{\Psi k'} \rangle - \langle \hat{\Delta}_{\Psi k} \rangle \langle \hat{\Delta}_{\Psi k'} \rangle \right) = \langle \hat{\Delta}_{\Psi k} \rangle \langle \hat{H} \rangle - \langle \hat{\Delta}_{\Psi k} \hat{H} \rangle \qquad \text{with } k \in \{ 1, \ldots, p \} \tag{2.30}$$

This is more conveniently written in matrix form.

$$\boldsymbol{S}\, \delta\vec{\alpha} = \vec{f} \qquad \text{with} \qquad \begin{aligned} S_{kk'} &= \langle \hat{\Delta}_{\Psi k} \hat{\Delta}_{\Psi k'} \rangle - \langle \hat{\Delta}_{\Psi k} \rangle \langle \hat{\Delta}_{\Psi k'} \rangle \\ f_k &= \langle \hat{\Delta}_{\Psi k} \rangle \langle \hat{H} \rangle - \langle \hat{\Delta}_{\Psi k} \hat{H} \rangle \end{aligned} \tag{2.31}$$

The expectation values $\langle \hat{H} \rangle$, $\langle \hat{\Delta}_{\Psi k} \rangle$, $\langle \hat{\Delta}_{\Psi k} \hat{H} \rangle$ and $\langle \hat{\Delta}_{\Psi k} \hat{\Delta}_{\Psi k'} \rangle$ can be calculated approximately using equation (2.6). How the local energy $E_{\mathrm{loc}}(x)$ is calculated, depends of course on the Hamiltonian of the system, but substituting the definition of $\hat{\Delta}_{\Psi k}$ and using the orthonormality $\langle x'|x \rangle = \delta_{x'x}$ and completeness $\mathbb{1} = \sum_x |x\rangle\langle x|$ of the configuration basis, we can simplify the expression for $\Delta_{\Psi k,\mathrm{loc}}(x)$ in a general way.

$$\Delta_{\Psi k,\mathrm{loc}}(x) = \frac{\langle x|\hat{\Delta}_{\Psi k}|\Psi\rangle}{\langle x|\Psi\rangle} = \frac{\left\langle x \left| \sum_{x'} \frac{\partial \ln\langle x'|\Psi\rangle}{\partial\alpha_k} \right| x' \right\rangle \langle x'|\Psi\rangle}{\langle x|\Psi\rangle} = \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_k} \tag{2.32}$$

So the local value of the logarithmic derivative operator $\hat{\Delta}_{\Psi k}$ is just the derivative of the overlap $\langle x|\Psi\rangle$ with respect to the variational parameter α_k. Calculating the double operator expectation values $\langle \hat{\Delta}_{\Psi k} \hat{H} \rangle$ and $\langle \hat{\Delta}_{\Psi k} \hat{\Delta}_{\Psi k'} \rangle$ is no more difficult, because we can show that the local value of the double operator is just the product of the two operators' local values.

$$\frac{\langle x|\hat{\Delta}_{\Psi k}\hat{\Delta}_{\Psi k'}|\Psi\rangle}{\langle x|\Psi\rangle} = \frac{\left\langle x \left| \sum_{x'} \frac{\partial \ln\langle x'|\Psi\rangle}{\partial\alpha_k} \right| x' \right\rangle \left\langle x' \left| \sum_{x''} \frac{\partial \ln\langle x''|\Psi\rangle}{\partial\alpha_{k'}} \right| x'' \right\rangle \langle x''|\Psi\rangle}{\langle x|\Psi\rangle} \tag{2.33}$$

$$= \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_k} \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_{k'}} = \Delta_{\Psi k,\mathrm{loc}}(x)\, \Delta_{\Psi k',\mathrm{loc}}(x) \tag{2.34}$$

$$\frac{\langle x|\hat{\Delta}_{\Psi k}\hat{H}|\Psi\rangle}{\langle x|\Psi\rangle} = \sum_{x'} \langle x|\hat{\Delta}_{\Psi k}\hat{H}|x'\rangle \frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle} \tag{2.35}$$

$$= \sum_{x'x''} \left\langle x \left| \frac{\partial \ln\langle x'' \mid \Psi\rangle}{\partial\alpha_{k'}} \right| x'' \right\rangle \langle x''|\hat{H}|x'\rangle \frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle} \tag{2.36}$$

$$= \frac{\partial \ln\langle x|\Psi\rangle}{\partial\alpha_k} \frac{\langle x|\hat{H}|\Psi\rangle}{\langle x|\Psi\rangle} = \Delta_{\Psi k,\mathrm{loc}}(x)\, E_{\mathrm{loc}}(x) \tag{2.37}$$

In principle the solution $\delta\vec{\alpha}$ of equation (2.31) determines the direction in parameter space that decreases the energy. There is one problem though: As the components of $\boldsymbol{S}$ are evaluated using a Monte Carlo method, it can easily happen that its smallest eigenvalues come out as almost zero, making the solution of the system extremely ill-conditioned and the algorithm unstable. Different solutions to this problem have been suggested [Sor01][Att05][Toc08], the simplest of which is to add a scaled down identity matrix $\epsilon\mathbb{1}$ to $\boldsymbol{S}$ before solving the system. This imposes a lower cutoff on the eigenvalues of $\boldsymbol{S}$ and reduces its condition number.

$$(\boldsymbol{S} + \epsilon\mathbb{1})\, \delta\vec{\alpha} = \vec{f} \tag{2.38}$$

Good results where obtained with $\epsilon = 10^{-4}$. The solution $\delta\alpha$ of the above system is then used to update the variational parameters $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$.

$$\vec{\alpha}' = \vec{\alpha} + \mathrm{d}t\ \delta\vec{\alpha} \tag{2.39}$$

Here $\mathrm{d}t$ must be chosen as large as possible to ensure rapid convergence, but small enough for the algorithm to remain stable. It is in general a good idea to start with a large $\mathrm{d}t$ and gradually reduce it the closer one gets to the energy minimum.

How do we know when to stop iterating the optimization? Due to the noisiness of the convergence we can not simply check if the change in the variational parameters from one iteration to the other is below some threshold. A robust procedure is needed to decide when to stop the optimization. The method our our choice is called Mann-Kendall trend test [Ken38][Man45], which is a procedure to measure how much trend there is in a time series $f(t_i)$.

$$\tau = \frac{2\,|\zeta|}{N(N-1)} \qquad \text{with} \qquad \zeta = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \operatorname{sgn}\left(f(t_i) - f(t_j)\right) \tag{2.40}$$

Here τ is unity if $f(t)$ is a strictly monotonic function and zero if there is no trend in $f(t)$. We will perform a Mann-Kendall trend test for each variational parameter on the last $I/2$ Stochastic Reconfiguration iterations, where I is the the total number of iterations since the beginning of the optimization. This has proven to work very well, since it on the one hand removes the very distant past from the test, but on the other hand connects the number of iterations for which the parameter must have been trend-free with the total number of optimization iterations.

The calculation of ζ in equation (2.40) has $\mathcal{O}(N^2)$ complexity in the number of points in the time series. Since we are gradually extending the time series by adding points to its end and shrinking it by removing points from the beginning, there is no need to recalculate ζ every time with $\mathcal{O}(N^2)$ complexity. We can instead update ζ after each iteration with $\mathcal{O}(N)$ complexity.

$$\zeta_{N+1} = \zeta_N + \sum_{i=1}^{N} \operatorname{sgn}\left(f(t_{N+1}) - f(t_i)\right) \qquad \text{(adding at the end)} \tag{2.41}$$

$$\zeta_{N-1} = \zeta_N - \sum_{i=2}^{N} \operatorname{sgn}\left(f(t_i) - f(t_1)\right) \qquad \text{(removing at the beginning)} \tag{2.42}$$

The variational parameters are considered to be converged and the optimization complete if $\tau < \tau_{\text{threshold}}$ for all variational parameters. Values around $\tau_{\text{threshold}} \approx 0.45$ have proven to be reasonable.

Due to the uncertainties in $\boldsymbol{S}$ and $\vec{f}$ one will never exactly reach the energy minimum. The variational parameters will rather fluctuate around their optimal value, and should be averaged over a number of iterations to get more accurate estimates.

In this section we have introduced the Stochastic Reconfiguration method and how it can be used to reliably minimize the energy. At this point we have done everything that can be done generally, and are now ready to specialize things for the Hubbard model.

3 The application of VMC to the Hubbard model

Conceptually, the Variational Monte Carlo method is probably one of the most straightforward numerical methods for correlated systems, and while its fundamental idea can be explained to anyone with an undergraduate's knowledge of quantum mechanics in a matter of minutes, its actual application to any physical system is far from trivial. This chapter attempts to give a detailed description of how the VMC method can be efficiently applied to the Hubbard model. At the end of the chapter the reader should have enough information to start writing his/her own VMC code for the Hubbard model.

3.1 The variational wavefunction

The most important choice that one has to make when one wants to apply VMC to a specific system is the choice of the variational wavefunction. Since VMC can only produce wavefunctions that are possible within the variational ansatz, this choice will ultimately determine whether or not one gets the correct physics. In this section we will present a quite versatile variational wavefunction for the Hubbard model that will allow us to describe a wide range of physics.

Variational wavefunctions $|\Psi\rangle$ used within VMC calculations of fermions generally have two parts: A Slater determinant state $|\Phi\rangle$ on which one or more correlation factors $\hat{\mathcal{P}}_i$ act.

$$|\Psi\rangle = \prod_i \hat{\mathcal{P}}_i \, |\Phi\rangle \tag{3.1}$$

Here the Slater determinant $|\Phi\rangle$ ensures the antisymmetry of the wavefunction while the correlators $\hat{\mathcal{P}}_i$ only modify the amplitude of the wavefunction but leave its antisymmetry intact. In general both the Slater determinant and the correlators will depend on variational parameters.

Let us first have a look at the determinantal part and come back to the correlation factors later.

3.1.1 Determinantal part and the particle-hole transformation

For the determinantal part $|\Phi\rangle$ of the wavefunction, one defines a variational single particle Hamiltonian $\hat{H}_{\text{var}}$ which can easily be diagonalized and its lowest eigenstates used to build the Slater determinant. In order to find a suitable variational Hamiltonian $\hat{H}_{\text{var}}$, let us first look at the Hubbard Hamiltonian $\hat{H}$ itself, since this the Hamiltonian which we want to approximate.

$$\hat{H} = \hat{H}_t + \hat{H}_U = -\sum_{i<j} t_{ij} \sum_\sigma \left(\hat{c}^\dagger_{i\sigma} \hat{c}_{j\sigma} + \text{h.c.} \right) + U \sum_i \hat{n}_{i\uparrow} \hat{n}_{i\downarrow} \tag{3.2}$$

The $\hat{H}_t$ term is no problem as it is quadratic in the creation and annihilation operators. The interaction term $\hat{H}_U$ is not though, and we will therefore to drop it and add two other quadratic terms that are modeled after common correlation induced phenomena: Electron pairing and magnetism.

$$\hat{H}_{\text{var}} = \hat{H}_t^{(\text{var})} + \hat{H}_\Delta + \hat{H}_{\text{mag}} + \hat{H}_\mu \tag{3.3}$$

with
$$\hat{H}_t^{(\text{var})} = -\sum_{i<j} t_{ij}^{(\text{var})} \sum_\sigma \left(\hat{c}_{i\sigma}^\dagger \hat{c}_{j\sigma} + \text{h.c.}\right) \tag{3.4}$$

$$\hat{H}_\Delta = \Delta_0 \sum_i \left(\hat{c}_{i\uparrow}^\dagger \hat{c}_{i\downarrow}^\dagger + \text{h.c.}\right) + \sum_{i<j} \Delta_{ij} \left(\hat{c}_{i\uparrow}^\dagger \hat{c}_{j\downarrow}^\dagger + \hat{c}_{j\uparrow}^\dagger \hat{c}_{i\downarrow}^\dagger + \text{h.c.}\right) \tag{3.5}$$

$$\hat{H}_{\text{mag}} = \mu_m \sum_i (-1)^{\tau(i)} \hat{S}_i^z = \mu_m \sum_i (-1)^{\tau(i)} \frac{\hat{m}_i^z}{2} \tag{3.6}$$

$$\hat{H}_\mu = -\mu \sum_i \hat{n}_i \tag{3.7}$$

Here the $i < j$ sums run over all pairs of lattice sites i and j where $i \neq j$ without double counting.

The individual terms of equation (3.3) certainly require a little more explanation: The hopping term $\hat{H}_t^{(\text{var})}$ is the same as in the original Hubbard Hamiltonian, except that the hopping parameters of the variational Hamiltonian can in general be renormalized by the correlation. Only the hopping parameter between nearest neighbors is the same in the variational and the original Hamiltonian, as we need one parameter to fix the variational Hamiltonian's energy scale.

$$t_{ij}^{(\text{var})} = t_{ij} \quad \text{if } i \text{ and } j \text{ are nearest neigbours} \tag{3.8}$$

The pairing term $\hat{H}_\Delta$ is what one gets if one writes down the mean field decoupling of the BCS Hamiltonian in real space, see [Czy07, chapter 11.4]. In principle this would allow us to describe a superconductor, but more important for the rest of the thesis is that it can also improve our description of a nonmagnetic insulator according to Anderson's Resonating Valence Bond Theory [And87], where pairs of electrons are formed between neighboring sites.

Note that although we write all equations with general variational hopping parameters $t_{ij}^{(\text{var})}$ and pairings Δ_{ij}, it is usually only enough to have hoppings and pairing between sites that are close to each other. We will later restrict ourselves to hopping and pairing between at most third nearest neighbors.

Our magnetic term $\hat{H}_{\text{mag}}$ given by equation (3.6) is only one way to incorporate magnetism into the wavefunction, and also a rather crude one that is only applicable to bipartite lattices. A bipartite lattice is a lattice that can be divided into two sublattices, where every site on sublattice 1 has only nearest neighbours that are on sublattice 2 and vice versa. Examples of bipartite lattices are all hypercubic lattices. The magnetic term $\hat{H}_{\text{mag}}$ contains the spin operator $\hat{S}_i^z$ along the z-axis with a different sign depending on which sublattice $\tau(i) \in \{1, 2\}$ the site i belongs to, which introduces a Néel ordered antiferromagnetism. Néel order is the preferred order of the Hubbard model on hypercubic lattices, and since we will later perform simulations for such lattices, this particular form of $\hat{H}_{\text{mag}}$ is the obvious choice.

We also add a chemical potential term $\hat{H}_\mu$, the purpose of which will become more clear at the end of this subsection.

In order to obtain the Slater determinant we are going to use as the mean field part of our wavefunction, we would have to diagonalize the variational Hamiltonian $\hat{H}_{\text{var}}$ and obtain its eigenvectors. Before we can do so, however, we first need to solve a problem associated with the pairing term $\hat{H}_\Delta$, namely that it contains terms of the form $\hat{c}_i^\dagger \hat{c}_j^\dagger$. These terms prevent us from writing $\hat{H}_{\text{var}}$ as a matrix in the $2L$ dimensional basis $\mathcal{B}_c = \left\{ \hat{c}_{i\sigma}^\dagger \left|\emptyset\right\rangle \mid i \in \{1,\dots,L\} \text{ and } \sigma \in \{\uparrow,\downarrow\} \right\}$ of the single particle Hilbert space. We are therefore going to apply a canonical transformation, called the particle-hole transformation, that brings the $\hat{c}_i^\dagger \hat{c}_j^\dagger$ terms into a $\hat{c}_i^\dagger \hat{c}_j$ form. The particle-hole transformation leaves the operators for spin-$\uparrow$ particles unchanged but defines new creators and annihilators for the spin-$\downarrow$ particles, which are the hermitian conjugate of the old operators.

$$\hat{c}_{i\uparrow} \to \hat{d}_{i\uparrow} \qquad \text{and} \qquad \hat{c}_{i\uparrow}^\dagger \to \hat{d}_{i\uparrow}^\dagger \tag{3.9}$$

$$\hat{c}_{i\downarrow} \to \hat{d}_{i\downarrow}^\dagger \qquad \text{and} \qquad \hat{c}_{i\downarrow}^\dagger \to \hat{d}_{i\downarrow} \tag{3.10}$$

Physically, we are now describing the spin up electrons and spin down holes, instead of spin up and down electrons. Since the operator $\hat{d}_{i\downarrow}^\dagger$ now creates a spin down hole at site i, and therefore annihilates a spin down electron, we get a new vacuum state $\left|\tilde{\emptyset}\right\rangle = \prod_i \hat{c}_{i\downarrow}^\dagger \left|\emptyset\right\rangle$ in which all states for spin down electrons are already occupied. Using the anticommutativity of fermionic operators we can also define new occupation number operators.

$$\hat{u}_{i\uparrow} = \hat{d}_{i\uparrow}^\dagger \hat{d}_{i\uparrow} = \hat{c}_{i\uparrow}^\dagger \hat{c}_{i\uparrow} = \hat{n}_{i\uparrow} \tag{3.11}$$

$$\hat{u}_{i\downarrow} = \hat{d}_{i\downarrow}^\dagger \hat{d}_{i\downarrow} = \hat{c}_{i\downarrow} \hat{c}_{i\downarrow}^\dagger = 1 - \hat{c}_{i\uparrow}^\dagger \hat{c}_{i\uparrow} = 1 - \hat{n}_{i\uparrow} \tag{3.12}$$

Let us calculate how a site's occupation $\hat{n}_i$ and magnetization $\hat{m}_i^z$ can be expressed using the new operators.

$$\hat{n}_i = \hat{n}_{i\uparrow} + \hat{n}_{i\downarrow} = \hat{u}_{i\uparrow} - \hat{u}_{i\downarrow} + 1 = \hat{w}_i^z + 1 \tag{3.13}$$

$$\hat{m}_i^z = \hat{n}_{i\uparrow} - \hat{n}_{i\downarrow} = \hat{u}_{i\uparrow} + \hat{u}_{i\downarrow} - 1 = \hat{u}_i + 1 \tag{3.14}$$

Here we have introduced $\hat{w}_i^z = \hat{u}_{i\uparrow} - \hat{u}_{i\downarrow}$ as the magnetization, and $\hat{u}_i = \hat{u}_{i\uparrow} + \hat{u}_{i\downarrow}$ as the total occupancy for the transformed operators.[4] We see that, except for a shift of one which can often be neglected, the occupation becomes the magnetization under the particle-hole transformation and vice versa.

We will now use the particle-hole transformation to rewrite the individual terms in the variational Hamiltonian. For the hopping term, the particle-hole transformation only changes the sign of the hopping for spin down particles.

$$\hat{H}_t^{(\text{var})} = -\sum_{i<j} t_{ij}^{(\text{var})} \sum_\sigma \left(\hat{c}_{i\sigma}^\dagger \hat{c}_{j\sigma} + \text{h.c.} \right) = -\sum_{i<j} t_{ij}^{(\text{var})} \left(\hat{c}_{i\uparrow}^\dagger \hat{c}_{j\uparrow} + \hat{c}_{i\downarrow}^\dagger \hat{c}_{j\downarrow} + \text{h.c.} \right) \tag{3.15}$$

$$= -\sum_{i<j} t_{ij}^{(\text{var})} \left(\hat{d}_{i\uparrow}^\dagger \hat{d}_{j\uparrow} + \hat{d}_{i\downarrow} \hat{d}_{j\downarrow}^\dagger + \text{h.c.} \right) = -\sum_{i<j} t_{ij}^{(\text{var})} \left(\hat{d}_{i\uparrow}^\dagger \hat{d}_{j\uparrow} - \hat{d}_{j\downarrow}^\dagger \hat{d}_{i\downarrow} + \text{h.c.} \right) \tag{3.16}$$

[4] The letters u and w were chosen because they look like an upside down n and m.

The pairing terms $\hat{c}^\dagger_{i\uparrow}\hat{c}^\dagger_{j\downarrow}$ in $\hat{H}_\Delta$ become $\hat{d}^\dagger_{i\uparrow}\hat{d}_{j\downarrow}$, so they become something like a hopping that also changes the particle's spin.

$$\hat{H}_\Delta = \Delta_0 \sum_i \left(\hat{c}^\dagger_{i\uparrow}\hat{c}^\dagger_{i\downarrow} + \text{h.c.}\right) + \sum_{i<j} \Delta_{ij} \left(\hat{c}^\dagger_{i\uparrow}\hat{c}^\dagger_{j\downarrow} + \hat{c}^\dagger_{j\uparrow}\hat{c}^\dagger_{i\downarrow} + \text{h.c.}\right) \tag{3.17}$$

$$= \Delta_0 \sum_i \left(\hat{d}^\dagger_{i\uparrow}\hat{d}_{i\downarrow} + \text{h.c.}\right) + \sum_{i<j} \Delta_{ij} \left(\hat{d}^\dagger_{i\uparrow}\hat{d}_{j\downarrow} + \hat{d}^\dagger_{j\uparrow}\hat{d}_{i\downarrow} + \text{h.c.}\right) \tag{3.18}$$

The chemical potential and the magnetic term transform in the following way, and kind of reverse their role due to equation (3.13) and (3.14).

$$\hat{H}_{\text{mag}} = \mu_m \sum_i (-1)^{\tau(i)} \frac{\hat{m}^z_i}{2} = \mu_m \sum_i (-1)^{\tau(i)} \frac{\hat{u}_i + 1}{2} = \mu_m \sum_i (-1)^{\tau(i)} \frac{\hat{u}_i}{2} \tag{3.19}$$

$$\hat{H}_\mu = -\mu \sum_i \hat{n}_i = -\mu \sum_i (\hat{w}^z_i + 1) = -\mu \sum_i \hat{w}^z_i - L \equiv -\mu \sum_i \hat{w}^z_i \tag{3.20}$$

The constant one in the magnetic term cancels with itself, since the two sublattices have the same number of sites. In the chemical potential term the constant one produces a shift of $-L$ which we can safely neglect, since we are only interested in the variational Hamiltonian's eigenstates, which are not influenced by a shift of its spectrum.

We see that the variational Hamiltonian can easily be written as a matrix $\boldsymbol{H}_{\text{var}}$ in the basis $\mathcal{B}_d = \left\{\hat{d}^\dagger_{i\sigma}|\tilde{\emptyset}\rangle \mid i \in \{1,\ldots,L\} \text{ and } \sigma \in \{\uparrow,\downarrow\}\right\}$ after one has applied the particle-hole transformation. Let us contract the spin index σ and the lattice site index i into a single index running from 0 to $2L$, with spin major ordering.

$$\hat{d}_{i\uparrow} = \hat{d}_i \qquad \text{and} \qquad \hat{d}_{i\downarrow} = \hat{d}_{i+L} \tag{3.21}$$

The relationship between the variational Hamiltonian $\hat{H}_{\text{var}}$ and its matrix $\boldsymbol{H}_{\text{var}}$ can then be written as follows.

$$\hat{H}_{\text{var}} = \sum_{i,j=1}^{2L} \hat{d}^\dagger_i \left(\boldsymbol{H}_{\text{var}}\right)_{ij} \hat{d}_j \tag{3.22}$$

We can now diagonalize $\hat{H}_{\text{var}}$ using a suitable unitary transform $\hat{U}^\dagger \hat{H}_{\text{var}} \hat{U}$ and define new creation operators that add particles in the variational Hamiltonian's eigenstates.

$$\hat{\gamma}^\dagger_n = \sum_{i=1}^{2L} \boldsymbol{U}_{in}\, \hat{d}^\dagger_i \tag{3.23}$$

Using these operators we can finally write down our determinantal wavefunction explicitly.

$$|\Phi\rangle = \hat{\gamma}^\dagger_1 \hat{\gamma}^\dagger_2 \ldots \hat{\gamma}^\dagger_{N_p} |\tilde{\emptyset}\rangle \tag{3.24}$$

We are going to perform all simulations at a fixed particle number N_p, which is the number of particles *after* the particle-hole transformation. This will in general be different from the number of electrons in the Hubbard model we are simulating, since our new particles are spin up electrons and spin down holes. Suppose we want to run a simulation of the Hubbard model with N_e electrons. Let $N_{e\uparrow}$ of them have spin up and the other $N_{e\downarrow}$ have spin down. The number of spin up particles does not change under the particle-hole transformation since they are still spin up electrons, so $N_{p\uparrow} = N_{e\uparrow}$. The spin down particles now correspond to the sites where there is *no* spin down electron and

hence $N_{p\downarrow} = L - N_{e\downarrow}$, where L is the number of sites. From that we can easily write the relation between the particle numbers before and after the particle hole transformation.

$$N_p = N_{p\uparrow} + N_{p\downarrow} = N_{e\uparrow} + L - N_{e\downarrow} \tag{3.25}$$

Note that the particle numbers do not change at half filling, so in this case not only is $N_{p\uparrow} = N_{e\uparrow}$ but also $N_{p\downarrow} = N_{e\downarrow}$ and $N_p = N_e$.

The particle-hole transformation's effect on our calculations is more profound that it may look at first, and we can not regard the particle-hole transformation as a mathematical trick we employ at one step in the derivation, but should rather think about it in the following way: The particle-hole transformation gives us a set of rules with which we can map our original Hubbard Hamiltonian to a new Hamiltonian which will produce exactly the same physics but for which we can use a more powerful variational wavefunction. Let us have a look at how the Hubbard Hamiltonian itself is transformed by the particle-hole transformation.

$$\hat{H} = -\sum_{i<j} t_{ij} \left(\hat{d}^\dagger_{i\uparrow} \hat{d}_{j\uparrow} - \hat{d}^\dagger_{i\downarrow} \hat{d}_{j\downarrow} + \text{h.c.} \right) + U \sum_i \hat{u}_{i\uparrow}(1 - \hat{u}_{i\downarrow}) \tag{3.26}$$

This is the Hamiltonian that we are *actually* simulating. We also have a different number of particles now and only some of our new particles correspond to the electrons of the original Hamiltonian, but that will not change our results as long as we also particle-hole transform all the observables that we want to measure.

Before we can close this section, there is one more thing we need to address, and that is why we put a chemical potential term into the variational Hamiltonian (3.3). We will perform the simulations with a fixed particle number, which in practice means building the Slater determinant from a fixed number of orbitals and sampling only configurations with this number of particles. For a fixed number of particles the chemical potential only shifts the energies, and seems unnecessary. The reason why we still need to use a chemical potential is the particle-hole transformation: Looking again at equation (3.20) we see that the chemical potential becomes the magnetization after the transformation, which decreases the energy for spin up orbitals but increases it for spin down orbitals. Suppose we had only the hopping and the chemical potential in the variational Hamiltonian, so that it would not be necessary to do the particle-hole transformation. Its eigenstates are certainly degenerate with respect to the spin, so that we would get $N_{e\uparrow} = N_e/2$ spin up orbitals and $N_{e\downarrow} = N_e/2$ spin down orbitals in the Slater determinant if we just take the lowest N_e eigenstates. After the particle-hole transformation we should have $N_{p\uparrow} = N_{e\uparrow}$ spin up orbitals and $N_{p\downarrow} = L - N_{e\downarrow}$ spin down orbitals, but since up and down orbitals are shifted in different directions this will only be the case for the lowest $N_p = N_{p\uparrow} + N_{p\downarrow}$ eigenstates of the transformed variational Hamiltonian if we have the correct chemical potential μ!

There are two mutually complementing ways to determine the right chemical potential. If the pairing term in the variational Hamiltonian is disabled by setting all Δ_{ij} to zero, there is no need to do the particle hole transformation in order to diagonalize the variational Hamiltonian. In this case we can just diagonalize $\hat{H}_{\text{var}}$ without the particle-hole transformation and read off the required chemical potential as the Fermi energy. In the case that we have a pairing term, the eigenvectors of the variational Hamiltonian

will mix the two spins, meaning that we can not have the problem that the Slater determinant has a different number of spin-σ particles than the configurations x that we sample. We will get small overlaps if our chemical potential is too far off, but can in general just treat is as a variational parameter and let the optimization take care of determining its value.

3.1.2 Jastrow factor as a long-ranged correlator

Having dealt with the mean field Slater determinant in the last subsection, we can now introduce correlation into our wavefunction through the use of correlation factors.

A particularly simple example of a correlator is the so called Gutzwiller factor.

$$\hat{\mathcal{P}}_g = \exp\left(-g \sum_i \hat{n}_{i\uparrow}\hat{n}_{i\downarrow}\right) \tag{3.27}$$

It has only a single variational parameter g, and for $g > 0$ it will reduce the wavefunctions overlap with configurations according to their number of double occupancies, essentially preventing the electrons from getting too close to each other. While this is already a step in the right direction, it is not enough to induce the Mott insulator transition in the half-filled Hubbard model: At $U = \infty$ the Gutzwiller parameter g will also be infinite in order to project out any states that have double occupancies, making the system insulating at half filling. At finite U though, the Gutzwiller parameter g will be finite, and the system will have a certain number of double occupancies (doublons) and holes (holons). As doublons have a negative charge and holons a positive charge they will move in opposite directions if an electric field is applied, making the system conducting. Ergo, the system would always be metallic as long as g is finite.

In order to correctly describe the Mott transition at a finite value of U one has to allow for long-range correlations in the form of a so called Jastrow factor. [CBF+05]

$$\hat{\mathcal{P}}_J = \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\hat{n}_i\hat{n}_j\right) = \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\left(\hat{n}_{i\uparrow}+\hat{n}_{i\downarrow}\right)\left(\hat{n}_{j\uparrow}+\hat{n}_{j\downarrow}\right)\right) \tag{3.28}$$

Here the v_{ij} do not depend directly on the indices i and j, but on the absolute distance between the sites, so $v_{ij} = v(|\vec{r}_{ij}|)$. In this way the Jastrow factor has one variational parameter for each possible distance on the finite lattice that is simulated. Note that the parameter corresponding to the largest distance is fixed to zero. This is possible because a shift of all the Jastrow parameters would only change the normalization of the wavefunction, and we therefore need one parameter that sets a scale for the others.

Why the Jastrow factor is superior to the Gutzwiller factor is most easily seen if one rewrites the Jastrow factor in terms of doublon $\hat{D}_i = \hat{n}_{i\uparrow}\hat{n}_{i\downarrow}$ and holon $\hat{H}_i = (1-\hat{n}_{i\uparrow})(1-\hat{n}_{i\downarrow})$ operators.

$$\hat{\mathcal{P}}_J = \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\left[\hat{D}_i\hat{D}_j + \hat{H}_i\hat{H}_j - \hat{H}_i\hat{D}_j - \hat{D}_i\hat{H}_j + \hat{n}_i + \hat{n}_j - 1\right]\right) \tag{3.29}$$

The Jastrow parameters v_{ij} typically increase with the distance between the sites i and j, so that the Jastrow factor describes an attraction between holons and doublons and a repulsion between the doublons and the holons themselves. In this way the doublons

and holons are no longer free charge carriers, and their existence is no longer equivalent with the system being a metal.

We will use the Jastrow factor as defined in equation (3.28) as the only correlator in our wavefunction.

Since we are simulating the particle-hole transformed Hamiltonian, the configurations that we are going to sample are configurations of the transformed particles, and we also need to transform the Jastrow factor, which we had written in equation (3.28) with the operators for the original particles.

$$\hat{\mathcal{P}}_J = \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\hat{n}_i\hat{n}_j\right) \tag{3.30}$$

$$= \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\left(\hat{w}_i^z + 1\right)\left(\hat{w}_j^z + 1\right)\right) \tag{3.31}$$

$$= \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\left(\hat{w}_i^z\hat{w}_j^z + \hat{w}_i^z + \hat{w}_j^z + 1\right)\right) \tag{3.32}$$

$$= \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\hat{w}_i^z\hat{w}_j^z\right)\exp\left(\frac{1}{2}\sum_{ij} v_{ij}\left(\hat{w}_i^z + \hat{w}_j^z + 1\right)\right) \tag{3.33}$$

Let us have a closer look at the second term and proof that it only changes the normalization of the wavefunction and can therefore be neglected since we are working with non-normalized wavefunctions anyway.

$$\sum_{ij} v_{ij}\left(\hat{w}_i^z + \hat{w}_j^z + 1\right) = \sum_{ij} v_{ij}\hat{w}_i^z + \sum_{ij} v_{ij}\hat{w}_j^z + \sum_{ij} v_{ij} \tag{3.34}$$

$$= \sum_{ij} v_{ij}\hat{w}_i^z + \sum_{ij} v_{ji}\hat{w}_i^z + \sum_{ij} v_{ij} \tag{3.35}$$

$$= \sum_{ij} v_{ij}\hat{w}_i^z + \sum_{ij} v_{ij}\hat{w}_i^z + \sum_{ij} v_{ij} \tag{3.36}$$

$$= 2\sum_{ij} v_{ij}\hat{w}_i^z + \sum_{ij} v_{ij} \tag{3.37}$$

$$= 2\sum_{i}\sum_{j} v_{ij}\hat{w}_i^z + \sum_{i}\sum_{j} v_{ij} \tag{3.38}$$

Here the $\sum_j v_{ij}$ is a constant that does not depend on i since v_{ij} only depends on the distance between site i and site j. The remaining sum $\sum_i \hat{w}_i^z$ is also constant since the number of spin up and spin down particles is the same for all sampled configurations. The complete Jastrow factor after the particle-hole transformation can therefore be written as follows.

$$\hat{\mathcal{P}}_J = \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\hat{w}_i^z\hat{w}_j^z\right) \tag{3.39}$$

As we have seen before, the particle-hole transformation switches the occupation number with the magnetization and vice versa.

In this section we have presented the variational wavefunction consisting of a Slater determinant and a correlator in form of the Jastrow factor. In order to have a single

particle variational Hamiltonian from whose eigenstates we can construct the Slater determinant, we have started from the original Hubbard Hamiltonian, dropped the interactive term and reintroduced correlation effects through a pairing and a magnetic term. We had to apply a canonical transformation in order to calculate the eigenstates, and have seen that in practice this means simulating a different Hamiltonian that will have the same physical properties if one also transforms the measured observables.

3.2 The local energy, overlap ratios and complexity reduction

Let us now have a look at how the Hubbard Hamiltonian's local energy $E_{\mathrm{loc}}(x)$ is calculated for a given configuration x. Remember that we did that particle-hole transformation and therefore have to transform all observables, including the Hamiltonian itself, before evaluating them. The particle-hole transformed Hamiltonian is given by equation (3.26) and its local value can be calculated as follows.

$$\begin{aligned} E_{\mathrm{loc}}(x) &= \frac{\langle x|\hat{H}|\Psi\rangle}{\langle x|\Psi\rangle} = \sum_{x'} \langle x|\hat{H}|x'\rangle \, \frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle} \\ &= -\sum_{x \curvearrowright x'} t_{ij}(-1)^{\delta_{\sigma\downarrow}} \frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle} + U \sum_i u_{i\uparrow}(x)\big(1 - u_{i\downarrow}(x)\big) \end{aligned} \tag{3.40}$$

Here $u_{i\sigma}(x)$ is just the number of spin-σ particles at site i in configuration x. The on-site energy term is simple because it is diagonal in the configuration basis. The $x \curvearrowright x'$ sum runs over all configurations x' that are reachable from x by hopping a spin-σ particle from its original site i to an empty site j. The hopping term produces this particular sum because $\langle x|\hat{H}|x'\rangle$ is zero for almost all x' due to the orthonormality $\langle x|x'\rangle = \delta_{xx'}$. The only way to make it nonzero is to have a configuration x' which is turned into x by one of the Hamiltonian's $\hat{d}_j^\dagger \hat{d}_i$ terms, implying that x and x' can not differ by more than a single particle position. The $(-1)^{\delta_{\sigma\downarrow}}$ simply switches the sign of the hopping parameters for the spin-$\downarrow$ particles, which is necessary because of the particle-hole transformation, see equation (3.15).

The only part of equation (3.40) which is not trivially calculable is the ratio of the overlaps $\langle x'|\Psi\rangle$ and $\langle x|\Psi\rangle$. Looking back at subsection 2.2.1, we see that this is also precisely the Metropolis probability of going from configuration x to x'. The calculation of these overlap ratios is computationally the single most expensive part in the entire simulation, and since these ratios are needed en masse for the Metropolis algorithm as well as the calculation of the local energy, their evaluation must be extremely efficient.

The problem of calculating $\langle x'|\Psi\rangle \,/\, \langle x|\Psi\rangle$ can be broken down into two subproblems if we substitute the form of our variational wavefunction $|\Psi\rangle = \hat{P}_J\,|\Phi\rangle$ into the equation.

$$\frac{\langle x'|\Psi\rangle}{\langle x|\Psi\rangle} = \frac{\langle x'|\hat{P}_J|\Phi\rangle}{\langle x|\hat{P}_J|\Phi\rangle} = \frac{P_J(x')}{P_J(x)} \frac{\langle x'|\Phi\rangle}{\langle x|\Phi\rangle} \tag{3.41}$$

Here we have used the hermiticity of $\hat{P}_J$ (because $\hat{u}_i = \hat{u}_i^\dagger$ and $[\hat{u}_i, \hat{u}_j] = 0$) and the fact that the configurations $|x\rangle$ are its eigenstates $\hat{P}_J\,|x\rangle = P_J(x)\,|x\rangle$. Rewritten in this way, we can calculate the ratios among the Jastrows and the Slater determinants separately. Let us start with the determinantal part.

3.2.1 Ratios among Slater determinants

The determinantal part of the wavefunction is constructed from the N_p lowest eigenstates of the variational single particle Hamiltonian $\hat{H}_{\text{var}}$, which was introduced in subsection 3.1.1. Let $|\phi_n\rangle = \hat{\gamma}_n^\dagger |\tilde{\emptyset}\rangle$ be the n-th lowest eigenstate of $\hat{H}_{\text{var}}$. Written in the basis[5] $\mathcal{B}_d = \left\{ |i\rangle \equiv \hat{d}_i^\dagger |\tilde{\emptyset}\rangle \mid i \in \{1, \dots, 2L\} \right\}$, the unitary $(2L \times 2L)$ matrix $\boldsymbol{U}$ which diagonalizes $\boldsymbol{H}_{\text{var}}$ has the eigenstates $|\phi_n\rangle$ written in $\mathcal{B}_d$ as its column vectors, which we will assume to be ordered ascending by energy.

$$\boldsymbol{U} = \begin{pmatrix} \langle 1|\phi_1\rangle & \langle 1|\phi_2\rangle & \cdots & \langle 1|\phi_{2L}\rangle \\ \langle 2|\phi_1\rangle & \langle 2|\phi_2\rangle & \cdots & \langle 2|\phi_{2L}\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle 2L|\phi_1\rangle & \langle 2L|\phi_2\rangle & \cdots & \langle 2L|\phi_{2L}\rangle \end{pmatrix} \tag{3.42}$$

We are only going to need the N_p energetically lowest eigenstates $|\phi_n\rangle$, so let us delete everything else from the matrix $\boldsymbol{U}$ and call the resulting $(2L \times N_p)$ matrix $\boldsymbol{M}$.

$$\boldsymbol{M} = \begin{pmatrix} \langle 1|\phi_1\rangle & \langle 1|\phi_2\rangle & \cdots & \langle 1|\phi_{N_p}\rangle \\ \langle 2|\phi_1\rangle & \langle 2|\phi_2\rangle & \cdots & \langle 2|\phi_{N_p}\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle 2L|\phi_1\rangle & \langle 2L|\phi_2\rangle & \cdots & \langle 2L|\phi_{N_p}\rangle \end{pmatrix} \tag{3.43}$$

Let us assign greek labels $\alpha \in \{1, \dots, N_p\}$ to our N_p particles and let $x_\alpha \in \{1, \dots, 2L\}$ be the position of the α-th particle. Note that no two x_α and x_β can be the same, since we contracted site and spin into a single index, and that the set of the x_α completely determines the many-particle configuration $|x\rangle$. Our mean field part $|\Phi\rangle$ of the wavefunction is now just the Slater determinant of $|\phi_1\rangle$ to $|\phi_{N_p}\rangle$, which can be written in the many-particle configuration basis $|x\rangle$ as follows.

$$\langle x|\Phi\rangle = \det \boldsymbol{D} \qquad \text{with} \qquad \boldsymbol{D} = \begin{pmatrix} \langle x_1|\phi_1\rangle & \langle x_1|\phi_2\rangle & \cdots & \langle x_1|\phi_{N_p}\rangle \\ \langle x_2|\phi_1\rangle & \langle x_2|\phi_2\rangle & \cdots & \langle x_2|\phi_{N_p}\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_{N_p}|\phi_1\rangle & \langle x_{N_p}|\phi_2\rangle & \cdots & \langle x_{N_p}|\phi_{N_p}\rangle \end{pmatrix} \tag{3.44}$$

Note that this $(N_p \times N_p)$ matrix $\boldsymbol{D}$ can be obtained from $\boldsymbol{M}$ by dropping the rows corresponding to sites which are unoccupied in configuration $|x\rangle$, and that the row index of $\boldsymbol{D}$ corresponds to the particle label, while the column index corresponds to the occupied orbital.

In principle we could be finished now and just calculate the determinant of $\boldsymbol{D}$ explicitly every time we need it. This would be terribly inefficient though, since $\mathcal{O}(N_p^3)$ operations are necessary to calculate it, and also prone to floating point overflow issues, especially for large particle numbers. Luckily we never need the absolute overlap $\langle x|\Phi\rangle$, but only ratios of the type $\langle x'|\Psi\rangle \,/\, \langle x|\Psi\rangle$, where x and x' are the same except for the position of a single particle. We are now going to present a way to exploit this similarity of x and x' by following the derivation in the PhD thesis of Manuela Capello. [Cap06]

[5] We have contracted the spin and site indices into one index from 1 to $2L$ again, where the first L correspond to spin up and the rest to spin down.

Consider two configurations x and x' where x' is obtained from x by hopping the particle number β from the k-th site to the l-th site. Then $\boldsymbol{D}$ and $\boldsymbol{D}'$ are related to each other by the following equation.

$$D'_{\alpha j} = D_{\alpha j} + \delta_{\alpha\beta}\left(M_{lj} - D_{\alpha j}\right) = D_{\alpha j} + \delta_{\alpha\beta} v_j^{(l\beta)} \tag{3.45}$$

In words this just means to replace the β-th row of $\boldsymbol{D}$ with the l-th row of $\boldsymbol{M}$. The vector $v_j^{(l\beta)} = M_{lj} - D_{\alpha j}$ is what needs to be added to the β-th row of $\boldsymbol{D}$ if particle number β hops to site l. Equation (3.45) can be rewritten in the following way.

$$D'_{\alpha j} = D_{\alpha j} + \sum_m D_{\alpha m} D^{-1}_{m\beta}\, v_j^{(l\beta)} = \sum_m D_{\alpha m}\left(\delta_{mj} + D^{-1}_{m\beta}\, v_j^{(l\beta)}\right) \tag{3.46}$$

We now define $K_{mj} = \delta_{mj} + D^{-1}_{m\beta}\, v_j^{(l\beta)}$ to write it compactly in matrix notation.

$$\boldsymbol{D}' = \boldsymbol{D}\boldsymbol{K} \tag{3.47}$$

The calculation of the ratio of the determinants of $\boldsymbol{D}'$ and $\boldsymbol{D}$ is therefore equivalent to the calculation of the determinant of $\boldsymbol{K}$.

$$\det \boldsymbol{D}' = \det(\boldsymbol{D}\boldsymbol{K}) = \det \boldsymbol{D}\ \det \boldsymbol{K} \quad \Rightarrow \quad \frac{\det \boldsymbol{D}'}{\det \boldsymbol{D}} = \det \boldsymbol{K} \tag{3.48}$$

Looking at the definition of the matrix $\boldsymbol{K}$ we see that it is essentially the identity matrix plus the outer product of the vector $\vec{v}^{(l\beta)}$ and the β-th column vector of $\boldsymbol{D}^{-1}$. We can therefore use the matrix determinant lemma $\left(\det\left(\mathbb{1} + \vec{u}\vec{v}^T\right) = 1 + \vec{v}^T\vec{u}\right)$, for which a proof can be found in appendix A.3.1, to write down the determinant of $\boldsymbol{K}$ explicitly.

$$\det \boldsymbol{K} = 1 + \sum_m D^{-1}_{m\beta}\, v_m^{(l\beta)} = 1 + \sum_m D^{-1}_{m\beta}\left(M_{lm} - D_{\alpha m}\right) = \sum_m M_{lm} D^{-1}_{m\beta} \tag{3.49}$$

In matrix notation this can be written as follows.

$$\frac{\langle x'|\Phi\rangle}{\langle x|\Phi\rangle} = \frac{\det \boldsymbol{D}'}{\det \boldsymbol{D}} = W_{l\beta} \quad \text{with} \quad \boldsymbol{W} = \boldsymbol{M}\,\boldsymbol{D}^{-1} \tag{3.50}$$

Once the matrix $\boldsymbol{W}$ is calculated we can get the overlap with any adjacent configuration by looking at the corresponding matrix element, which is $\mathcal{O}(1)$ compared to the $\mathcal{O}(N_p^3)$ complexity of calculating a determinant. Numerically there is hardly ever a reason to explicitly calculate the inverse of a matrix: In the above equation the inversion can be replaced by solving a linear system for the transposed matrices, which is not only faster but possibly also more accurate in terms of floating point errors. [DT12]

$$\boldsymbol{W} = \boldsymbol{M}\,\boldsymbol{D}^{-1} \quad \Leftrightarrow \quad \boldsymbol{W}\boldsymbol{D} = \boldsymbol{M} \quad \Leftrightarrow \quad \boldsymbol{D}^T\,\boldsymbol{W}^T = \boldsymbol{M}^T \tag{3.51}$$

Once the Metropolis algorithm accepts a move and changes the particle configuration, we would have to recalculate the matrix $\boldsymbol{W}$ using equation (3.51), which is of complexity $\mathcal{O}(N_p^3)$. Instead of expensively recalculating it, there is a way to update $\boldsymbol{W}$ that scales as $\mathcal{O}(N_p^2)$ and relies on the fact that the configurations x and x' are related to each other.

Let us start from equation (3.46) again.

$$D'_{\alpha j} = \sum_m D_{\alpha m} K_{mj} \quad \text{with} \quad K_{mj} = \delta_{mj} + D^{-1}_{m\beta}\, v_j^{(l\beta)} \tag{3.52}$$

$$\Rightarrow \quad D'^{-1}_{j\alpha} = \sum_m K^{-1}_{jm} D^{-1}_{m\alpha} \quad \text{with} \quad K^{-1}_{jm} = \delta_{jm} - \frac{D^{-1}_{j\beta} v_m^{(l\beta)}}{1 + \sum_i D^{-1}_{i\beta}\, v_i^{(l\beta)}} = \delta_{jm} - \frac{D^{-1}_{j\beta}}{W_{l\beta}} v_m^{(l\beta)} \tag{3.53}$$

Here we have used a special case of the Sherman-Morrison formula to calculate $\boldsymbol{K}^{-1}$, a proof of which can be found in appendix A.3.2.

$$D'^{-1}_{j\alpha} = \sum_m \left(\delta_{jm} - \frac{D^{-1}_{j\beta}}{W_{l\beta}} v_m^{(l\beta)} \right) D^{-1}_{m\alpha} = D^{-1}_{j\alpha} - \frac{D^{-1}_{j\beta}}{W_{l\beta}} \sum_m v_m^{(l\beta)} D^{-1}_{m\alpha} \tag{3.54}$$

By multiplying from the left with M_{ij} and summing over j, we can obtain an update formula for the elements of the matrix $\boldsymbol{W}$.

$$W'_{i\alpha} = \sum_j M_{ij} D'^{-1}_{j\alpha} \tag{3.55}$$

$$= \sum_j M_{ij} D^{-1}_{j\alpha} - \frac{1}{W_{l\beta}} \sum_j M_{ij} D^{-1}_{j\beta} \sum_m v_m^{(l\beta)} D^{-1}_{m\alpha} \tag{3.56}$$

$$= W_{i\alpha} - \frac{W_{i\beta}}{W_{l\beta}} \sum_m \Big(M_{lm} - D_{\beta m} \Big) D^{-1}_{m\alpha} \tag{3.57}$$

$$= W_{i\alpha} - \frac{W_{i\beta}}{W_{l\beta}} \Big(W_{l\alpha} - \delta_{\alpha\beta} \Big) \tag{3.58}$$

Since the matrix W has only N_p^2 elements, this update is of $\mathcal{O}(N_p^2)$ complexity compared to the $\mathcal{O}(N_p^3)$ complexity of the initial calculation of $\boldsymbol{W}$.

Note that in practice floating point errors will slowly build up in the matrix $\boldsymbol{W}$, so that one has to recalculate it after a certain number of updates. We will discuss this point in more detail later, specifically in subsection 4.2.5.

3.2.2 Ratios among Jastrow factors

Having treated the determinantal ratios in the last subsection, we are now going to focus on the ratios among the Jastrow factors.

Let us start by inserting the definition of the particle-hole transformed Jastrow factor and simplifying the equation.

$$\frac{\mathcal{P}_J(x')}{\mathcal{P}_J(x)} = \frac{\exp \frac{1}{2} \sum_{ij} v_{ij} w_i^z(x') w_j^z(x')}{\exp \frac{1}{2} \sum_{ij} v_{ij} w_i^z(x) w_j^z(x)} = \exp \frac{1}{2} \sum_{ij} v_{ij} \Big(w_i^z(x') w_j^z(x') - w_i^z(x) w_j^z(x) \Big) \tag{3.59}$$

The double sum makes the calculation of the Jastrow ratio an $\mathcal{O}(L^2)$ operation. As for the case of the determinantal ratio, the complexity can be reduced to $\mathcal{O}(L)$ if we exploit the relationship between the configurations x and x'.

Let us consider the case where configuration x' is created from x by hopping a spin σ particle from site k to site l. We will have to keep track of a sign that depends on whether it is a spin up or spin down particle that is hopping, so let us just define $s = (-1)^{\delta_{\downarrow\sigma}}$.

The new magnetization $w_i^z(x')$ can then be updated from its old value $w_i^z(x)$ through the following equation.

$$w_i^z(x') = w_i^z(x) + s\,(\delta_{il} - \delta_{ik}) \tag{3.60}$$

Substituting this into the first equation, everything can be simplified.

$$\frac{\mathcal{P}_J(x')}{\mathcal{P}_J(x)} = \exp\frac{1}{2}\sum_{ij} v_{ij}\left[\Big(w_i^z(x) + s\,(\delta_{il} - \delta_{ik})\Big)\Big(w_j^z(x) + s\,(\delta_{jl} - \delta_{jk})\Big) - w_i^z(x)w_j^z(x)\right] \tag{3.61}$$

$$= \exp\frac{1}{2}\sum_{ij} v_{ij}\Big(s\delta_{jl}w_i^z(x) - s\delta_{jk}w_i^z(x) + s\delta_{il}w_j^z(x) - s\delta_{ik}w_j^z(x) + \delta_{il}\delta_{jl} - \delta_{il}\delta_{jk} + \delta_{ik}\delta_{jk} - \delta_{ik}\delta_{jl}\Big) \tag{3.62}$$

$$= \exp\frac{1}{2}\Big(s\sum_i v_{il}w_i^z(x) - s\sum_i v_{ik}w_i^z(x) + s\sum_j v_{lj}w_j^z(x) - s\sum_j v_{kj}w_j^z(x) + v_{ll} - v_{lk} + v_{kk} - v_{kl}\Big) \tag{3.63}$$

Now we use the fact that the Jastrow parameters v_{ij} only depend on the distance r_{ij}, hence $v_{ij} = v_{ji}$ and $v_{ii} = v_{jj}$.

$$\frac{\mathcal{P}_J(x')}{\mathcal{P}_J(x)} = \exp\frac{1}{2}\Big(s\sum_i v_{il}w_i^z(x) - s\sum_i v_{ik}w_i^z(x) + s\sum_i v_{il}w_i^z(x) - s\sum_i v_{ik}w_i^z(x) + v_{ll} - v_{lk} + v_{ll} - v_{lk}\Big) \tag{3.64}$$

$$= \exp\Big(s\sum_i v_{il}w_i^z(x) - s\sum_i v_{ik}w_i^z(x) + v_{ll} - v_{lk}\Big) \tag{3.65}$$

Defining a vector $\vec{T}$ in analogy to the determinantal part's matrix $\boldsymbol{W}$, we can write the equation in the following way.

$$\frac{\mathcal{P}_J(x')}{\mathcal{P}_J(x)} = \exp\left(s\,(T_l(x) - T_k(x)) + v_{ll} - v_{lk}\right) \qquad \text{with} \qquad T_i(x) = \sum_j v_{ij}w_i^z(x) \tag{3.66}$$

Calculating the vector $\vec{T}$ is still an $\mathcal{O}(L^2)$ operation, but we can update it after every accepted Metropolis move in $\mathcal{O}(L)$ time through the following equation.

$$T_i(x') = \sum_j v_{ij}w_i^z(x') = \sum_j v_{ij}\big(w_i^z(x) + s\,(\delta_{il} - \delta_{ik})\big) = T_i(x) + s\,(v_{il} - v_{ik}) \tag{3.67}$$

Exactly as with the matrix $\boldsymbol{W}$ from the last section, it is necessary to recalculate $\vec{T}$ after a certain number of quick updates to avoid the accumulation of floating point errors.

In this section we have identified the ratios of the variational wavefunctions as a key ingredient in an efficient VMC implementation, and have then presented methods that reduce the computational complexity of their evaluation by one power.

3.3 Calculation of the logarithmic derivatives

In order to minimize the energy of the variational wavefunction through the Stochastic Reconfiguration algorithm, we will have to calculate the expectation values of the energy

and the logarithmic derivatives with respect to the variational parameters. We have already treated the local energy in the last section, and now there should only be one open question that separates us from a more or less complete description of VMC for the Hubbard model: The calculation of the logarithmic derivatives.

3.3.1 Logarithmic derivatives of the Jastrow parameters

The complete variational wavefunction is given by the Jastrow factor $\hat{\mathcal{P}}_J$ acting on a determinantal wavefunction $|\Phi\rangle$.

$$|\Psi\rangle = \hat{\mathcal{P}}_J |\Phi\rangle \qquad \text{with} \qquad \hat{\mathcal{P}}_J = \exp\left(\frac{1}{2}\sum_{ij} v(r_{ij})\hat{w}_i^z \hat{w}_j^z\right) \tag{3.68}$$

If the variational parameter α_k for which we want to calculate the logarithmic derivative is the Jastrow parameter $v_{lm} = v(r_{lm})$, the local value $\Delta_{\Psi k,\mathrm{loc}}(x)$ of the logarithmic derivative operator can be calculated by straightforwardly substituting the above into equation (2.32) and simplifying the result.

$$\frac{\partial \ln\langle x|\Psi\rangle}{\partial v_{lm}} = \frac{\partial}{\partial v_{lm}} \ln\langle x|\hat{\mathcal{P}}_J|\Phi\rangle = \frac{\partial}{\partial v_{lm}} \ln\langle\Phi|\hat{\mathcal{P}}_J|x\rangle^* \tag{3.69}$$

$$= \frac{\partial}{\partial v_{lm}} \ln\left\langle \Phi \middle| \exp\left(\frac{1}{2}\sum_{ij} v_{ij}\hat{w}_i^z\hat{w}_j^z\right) \middle| x\right\rangle^* \tag{3.70}$$

$$= \frac{\partial}{\partial v_{lm}} \ln\left\langle \Phi \middle| \exp\left(\frac{1}{2}\sum_{ij} v_{ij} w_i^z(x) w_j^z(x)\right) \middle| x\right\rangle^* \tag{3.71}$$

$$= \frac{\partial}{\partial v_{lm}} \left(\frac{1}{2}\sum_{ij} v_{ij} w_i^z(x) w_j^z(x) + \ln\langle\Phi\,|\,x\rangle^*\right) \tag{3.72}$$

$$= \frac{1}{2}\sum_{ij} \delta_{r_{ij},r_{lm}} w_i^z(x) w_j^z(x) \tag{3.73}$$

In words this just means to calculate the sum of $w_i^z(x)w_j^z(x)$ for all sites i and j that are linked by the respective Jastrow parameter. Note that since there are $L \times L$ pairs of sites, the calculation of *all* derivatives of Jastrow parameters is an $\mathcal{O}(L^2)$ operation.

3.3.2 Logarithmic derivatives of the determinantal parameters

In order to calculate the logarithmic derivatives with respect to the variational parameters of the determinantal part of the wavefunction, we use perturbation theory and treat a small change in the variational parameter α_k as a perturbation $\hat{V}_k$ of the variational single particle Hamiltonian $\hat{H}_{\mathrm{var}}$. [YS06]

$$\hat{H}'_{\mathrm{var}} = \hat{H}_{\mathrm{var}} + \sum_{k=1}^{p} \delta\alpha_k \hat{V}_k \tag{3.74}$$

Here $\hat{V}_k$ is an operator chosen proportional to the variational parameter α_k. Consider for example the chemical potential μ which enters into $\hat{H}_{\mathrm{var}}$ through the term $\hat{H}_\mu = -\mu\sum_i \hat{w}_i^z$. In this case $\hat{V}_\mu = -\sum_i \hat{w}_i^z$ would be a suitable perturbative operator. Up to

first order the correction to the Slater determinant $|\Phi\rangle$ can be written in the variational Hamiltonian's eigenbasis $\mathcal{B}_\gamma = \left\{\hat{\gamma}_n^\dagger |\tilde{\emptyset}\rangle \mid n \in \{1,\dots,2L\}\right\}$ as follows.

$$|\Phi'\rangle = |\Phi\rangle + \left(\sum_{k=1}^{p} \delta\alpha_k \sum_{\eta,\nu=1}^{2L} (\boldsymbol{Q}_k)_{\eta\nu} \hat{\gamma}_\eta^\dagger \hat{\gamma}_\nu\right) |\Phi\rangle + \mathcal{O}(\delta\alpha_k^2) \tag{3.75}$$

$$\text{with} \quad \boldsymbol{Q}_k = \begin{cases} \frac{\left(\boldsymbol{U}^\dagger \boldsymbol{V}_k \boldsymbol{U}\right)_{\eta\nu}}{\epsilon_\nu - \epsilon_\eta} & \text{if } \eta > N_p \text{ and } \nu \leq N_p \\ 0 & \text{otherwise} \end{cases} \tag{3.76}$$

Here $\hat{U}$ is the unitary transformation that diagonalizes $\hat{H}_{\text{var}}$. In equation (3.75) we can identify $\sum_{\eta,\nu=1}^{2L} (\boldsymbol{Q}_k)_{\eta\nu} \hat{\gamma}_\eta^\dagger \hat{\gamma}_\nu$ as the logarithmic derivative operator $\hat{\Delta}_{\Phi k}$ from equation (2.24). In order to calculate its expectation value conveniently within the VMC framework, we switch back to the $\mathcal{B}_d = \left\{\hat{d}_i^\dagger |\tilde{\emptyset}\rangle \mid i \in \{1,\dots,2L\}\right\}$ basis.

$$\hat{\Delta}_{\Phi k} = \sum_{i,j=1}^{2L} (\boldsymbol{A}_k)_{ij} \hat{d}_i^\dagger \hat{d}_j \quad \text{with} \quad \boldsymbol{A}_k = \boldsymbol{U}\boldsymbol{Q}_k\boldsymbol{U}^\dagger \tag{3.77}$$

Let us calculate its local value that we will evaluate during the Monte Carlo cycle.

$$\Delta_{\Psi k,\text{loc}}(x) = \frac{\langle x|\hat{\Delta}_{\Phi k}|\Phi\rangle}{\langle x|\Phi\rangle} = \sum_{i,j=1}^{2L} (\boldsymbol{A}_k)_{ij} \frac{\langle x|\hat{d}_i^\dagger \hat{d}_j|\Phi\rangle}{\langle x|\Phi\rangle} \tag{3.78}$$

Luckily we have already calculated the ratio $\langle x|\hat{d}_i^\dagger \hat{d}_j|\Phi\rangle \,/\, \langle x|\Phi\rangle$ from the above equation in the last section, because it is exactly $\langle x'|\Phi\rangle \,/\, \langle x|\Phi\rangle$ where x' is obtained from x by hopping a particle from site[6] i to site j. This ratio is either zero if there is no particle at site i, or contained somewhere in the matrix $\boldsymbol{W}$. Looking at equation (3.50) we see that the matrix element $W_{j\beta}$ is the ratio between the Slater determinants if the particle with the label β hops to site j. If we define i_β to be the position of the particle labeled β, we can rewrite the above equation to use the elements of $\boldsymbol{W}$.

$$\Delta_{\Psi k,\text{loc}}(x) = \sum_{\beta=1}^{N_p} \sum_{j=1}^{2L} (\boldsymbol{A}_k)_{i_\beta j} W_{j\beta} \tag{3.79}$$

The matrices $\boldsymbol{A}_k$ only have to be evaluated once in the beginning, and $\boldsymbol{W}$ is available anyway since one has to calculate it for the Metropolis algorithm, making the evaluation of $\Delta_{\Psi k,\text{loc}}(x)$ an $\mathcal{O}(N_p \times L)$ operation for determinantal parameters.

In this section the last piece of the VMC puzzle for the Hubbard model has fallen into place, namely the evaluation of the logarithmic derivatives that are needed for the Stochastic Reconfiguration optimization. In principle we could start writing a code now, and the next chapter will present such a code, but for now there is one last thing that is not really related to performing the simulation itself but rather to the interpretation of its results: We need some way to distinguish an insulator from a metal.

[6] We are using the contracted indices, so in this case "site" means a combination of site and spin.

3.4 Charge gap calculation

As we have seen in the introduction, the Hubbard model can for increasing U exhibit a transition from a metal to a Mott insulator at some critical value of U. In band theory, we could just check whether the Fermi energy crosses a band, but having only the optimized wavefunction, how can we distinguish between a metal and an insulator within VMC? What is the qualitative difference between a conducting and an insulating wavefunction? It was Walter Kohn who pointed out in 1964 that it is the organization of the electrons in the ground state that distinguishes metals from insulators. [Koh64]

Relying on this important insight, we now want to present a method that calculates an upper bound for the charge gap by looking at the long-range behavior of the charge-charge correlation $\langle \hat{n}_i \hat{n}_j \rangle$. Let us make the following single mode ansatz for the wavefunction of the excited state, which goes back to a paper by Richard Feynman on excitations in liquid Helium [Fey54], and was later also successfully applied to fermionic systems. [Ove71][GMP86]

$$|\Psi_{\vec{q}}\rangle = \hat{n}_{\vec{q}}\,|\Psi_0\rangle \qquad \text{with} \qquad \hat{n}_{\vec{q}} = \frac{1}{\sqrt{L}} \sum_i \mathrm{e}^{\mathrm{i}\vec{q}\cdot\vec{r}_i} \hat{n}_i \tag{3.80}$$

The energy of such an excitation can then be calculated as follows.

$$E_{\vec{q}} = \frac{\langle \Psi_{\vec{q}} | (\hat{H} - E_0) | \Psi_{\vec{q}} \rangle}{\langle \Psi_{\vec{q}} | \Psi_{\vec{q}} \rangle} = \frac{\langle \Psi_0 | \hat{n}_{-\vec{q}} (\hat{H} - E_0) \hat{n}_{\vec{q}} | \Psi_0 \rangle}{\langle \Psi_0 | \hat{n}_{-\vec{q}} \hat{n}_{\vec{q}} | \Psi_0 \rangle} \tag{3.81}$$

$$= \frac{\langle \Psi_0 | (\hat{n}_{-\vec{q}} \hat{H} \hat{n}_{\vec{q}} - \hat{n}_{-\vec{q}} \hat{n}_{\vec{q}} \hat{H}) | \Psi_0 \rangle}{\langle \Psi_0 | \hat{n}_{-\vec{q}} \hat{n}_{\vec{q}} | \Psi_0 \rangle} = \frac{\langle \Psi_0 | \hat{n}_{-\vec{q}} [\hat{H}, \hat{n}_{\vec{q}}] | \Psi_0 \rangle}{\langle \Psi_0 | \hat{n}_{-\vec{q}} \hat{n}_{\vec{q}} | \Psi_0 \rangle} \tag{3.82}$$

The object in the denominator is called the static structure factor $N(\vec{q}) = \langle \hat{n}_{-\vec{q}}\, \hat{n}_{\vec{q}} \rangle$. The expectation value of $\hat{n}_{-\vec{q}} [\hat{H}, \hat{n}_{\vec{q}}]$ can be calculated analytically [TBG11] for small $|\vec{q}|$ and turns out to be proportional to $|\vec{q}|^2$. Therefore, the charge gap Δ in the limit $\vec{q} \to 0$ is given by the following equation.

$$\Delta \propto \lim_{\vec{q} \to 0} \frac{|\vec{q}|^2}{N(\vec{q})} \tag{3.83}$$

Note that because of the specific form of our ansatz, this is only an upper bound for the real charge gap. Therefore, we can only state with certainty that the system is metallic for $\Delta = 0$, but reversely can not say that the system *must* be an insulator if $\Delta > 0$.

The static structure factor $N(\vec{q})$ can be calculated easily within VMC as the Fourier transform of the charge-charge correlation $\langle \hat{n}_i \hat{n}_j \rangle$.

$$N(\vec{q}) = \frac{1}{L} \sum_{ij} \mathrm{e}^{\mathrm{i}\vec{q}\cdot(\vec{r}_i - \vec{r}_j)} \langle \hat{n}_i \hat{n}_j \rangle \tag{3.84}$$

$$= \frac{1}{L} \sum_{i \neq j} \mathrm{e}^{\mathrm{i}\vec{q}\cdot(\vec{r}_i - \vec{r}_j)} \langle \hat{n}_i \hat{n}_j \rangle + \frac{1}{L} \sum_i \langle \hat{n}_i^2 \rangle \tag{3.85}$$

$$= \frac{1}{L} \sum_{i > j} \left(\mathrm{e}^{\mathrm{i}\vec{q}\cdot(\vec{r}_i - \vec{r}_j)} + \mathrm{e}^{-\mathrm{i}\vec{q}\cdot(\vec{r}_i - \vec{r}_j)} \right) \langle \hat{n}_i \hat{n}_j \rangle + \frac{1}{L} \sum_i \langle \hat{n}_i^2 \rangle \tag{3.86}$$

$$= \frac{2}{L} \sum_{i > j} \cos\left(\vec{q} \cdot (\vec{r}_i - \vec{r}_j)\right) \langle \hat{n}_i \hat{n}_j \rangle + \frac{1}{L} \sum_i \langle \hat{n}_i^2 \rangle \tag{3.87}$$

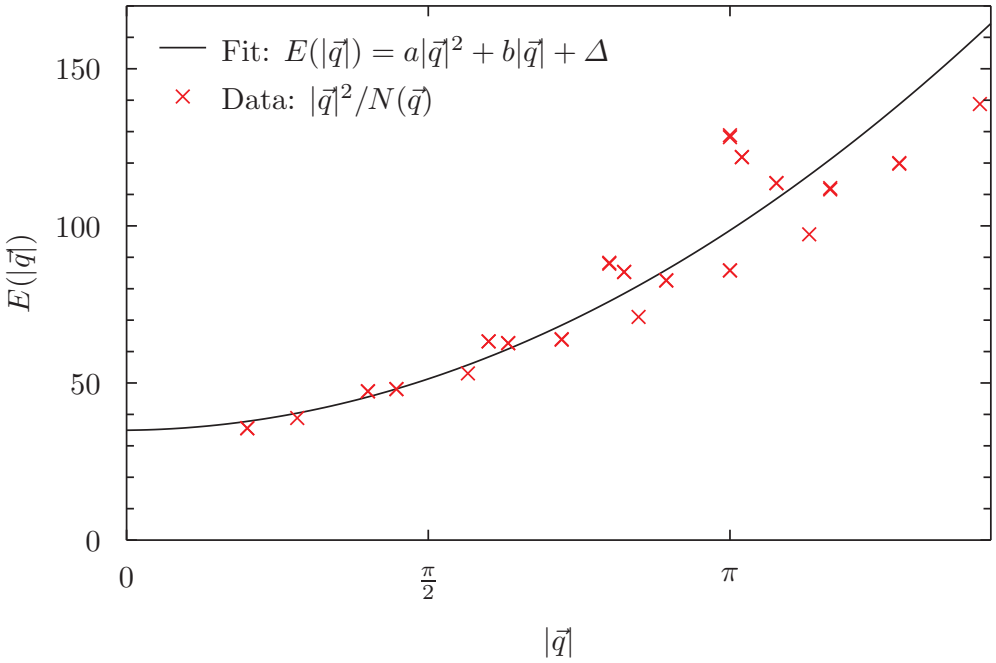

Figure 3.1: Extrapolation of the charge gap Δ by a fit of $E(|\vec{q}|) = a\,|\vec{q}|^2 + b\,|\vec{q}| + \Delta$ to the $\frac{|\vec{q}|^2}{N(\vec{q})}$ data of a 10×10 square lattice Hubbard model at $U = 8$. Note that the data points split up with increasing $|\vec{q}|$ since all directions in k-space are considered, and $N(\vec{q})$ is a function of the vector $\vec{q}$ and not only its norm $|\vec{q}|$.

In order to extract the charge gap Δ one also needs to perform the limit $|\vec{q}| \to 0$. This can in practice be rather tricky as we will only be simulating finite systems (for which there is only a limited number of $\vec{q}$-points) and performing the limit means to extrapolate $\vec{q}$ to zero. One way to do this is to fit the function $E(|\vec{q}|) = a\,|\vec{q}|^2 + b\,|\vec{q}| + \Delta$ with the constraints $a > 0$, $b > 0$ and $\Delta > 0$ to the $|\vec{q}|^2/N(\vec{q})$ data. If one weights the data points with $|\vec{q}|^{-\kappa}$, the κ can be chosen so that data points with small $|\vec{q}|$ dominate the outcome of the fit. Values of $\kappa \approx 2$ generally give good results. An example of such an extrapolation can be seen in figure 3.1.

In this chapter we have investigated how the Variational Monte Carlo method can be applied to the Hubbard model. In the next chapter we will present `hVMC`, an actual VMC code for the Hubbard model which was written as a part of this thesis.

4 hVMC – a free VMC code for the Hubbard model

In chapter 2 and 3 we have introduced the Variational Monte Carlo method in general and then investigated how it can efficiently be applied to the Hubbard model. This chapter is to give an example of how those ideas could be translated into actual source code. To this purpose a VMC code for the Hubbard model was developed from scratch in C++, and can be downloaded from:

`https://github.com/robertrueger/hVMC`

Considering that this is a thesis in physics, this chapter will only document those parts of `hVMC` which are directly related to the simulation, and which more or less map to the mathematical concepts of chapter 2 and 3. Complementing this rather theoretical view of the code, there is an `hVMC` quick start guide in appendix A.1, that approaches the codebase from a user's perspective and gives some practical information on how to build and run `hVMC`.

Note that in order to enable the reader to back and forth between the description of the code in this chapter and the actual source code, there will be margin notes in the form `filename:line` that point to the respective lines in the code.[7]

4.1 Design goals

Every software project should have a clear idea of what it is aiming for. Let us therefore as a very first step identify a set of fundamental principles, that have governed the design and implementation of the `hVMC` code.

Correctness & Flexibility: The correctness of the results goes without saying for a scientific code and was verified repeatedly by testing against an existing implementation and known analytic results. In order to be useful in a wide range of applications, the code must be flexible, at least in the sense that it can easily be modified to fit a particular problem. In general an object-oriented modular design approach was used to ensure flexibility with respect to the underlying lattice, the variational wavefunction and the measured observables.

Clarity & Brevity: The code should be easy to understand in order to encourage its modification by users other than the original author. The amount of "overhead code" (that is code not immediately related to

[7] In the PDF version of this thesis these margin notes are actually clickable links, which point to the respective file on GitHub, where the lines are highlighted.

the simulation itself, e.g. input/output, logging, MPI communication) should be minimal. To this purpose hVMC makes heavy use of external C++ template libraries.

Efficiency & Scalability: The code should make efficient use of the available hardware and be suitable for execution on large distributed memory machines. The hVMC code was profiled repeatedly and hotspots optimized or offloaded to optimized libraries. The potential of the different levels of parallelism available in modern supercomputers was evaluated and appropriate parallelizations implemented, see appendix A.2.

4.2 Core components

The innermost part of the hVMC code consists of a number of classes that can be mapped directly to the mathematical objects introduced in chapter 2 and 3. These classes are then wrapped into a class called the ModelManager, that exposes their functionality through an intuitive interface to the rest of the code. Let us refer to the ModelManager and the aforementioned classes as the hVMC core components, and the rest of the code as peripheral components.

Figure 4.1 shows all the core components and the relations among them. After construction only the ModelManager is used directly by the peripheral components. All other core components are hidden behind the ModelManager's interface, and are automatically destructed when the ModelManager goes out of scope.

Let us first have a look at the individual core components, treating the ModelManager last, since its job is mostly to glue the other components together and hide them behind a convenient interface.

4.2.1 The Lattice interface and derived classes

The underlying lattice not only has a tremendous influence on the physics of the system, but also has an influence on many parts of the simulation: The Hubbard Hamiltonian as well as the variational single particle Hamiltonian have hoppings and pairings between neighboring sites, so we will need to be able to find the n-th nearest neighbors of a specific site. Hopping to neighboring sites is also required by the Metropolis updates. The Jastrow parameters v_{ij} actually only depend on the distance between site i and j, so we will need to determine which of the parameters connects two sites. There are even more places where the lattice plays a role, but we can already see that we need a lot of information about the lattice in order to perform the simulation.

On the other hand we want to be extremely flexible with regard to the lattice and be able to add new lattices without any changes to the existing code. We are going to achieve this through runtime polymorphism, specifically
lattice.hpp:34 a Lattice class that has almost only pure virtual functions. In this way the Lattice class serves as an interface, which the real implementations of the desired lattices (1d chain, 2d square, etc.) can then implement.

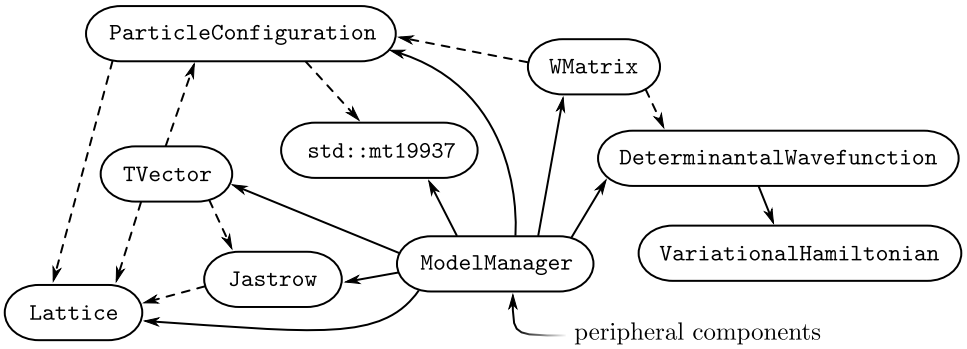

Figure 4.1: The core components of `hVMC` and the relations among them. Solid arrows can be read as "owns", while the dashed arrows mean "makes use of". The class `std::mt19936` is just the C++ standard library version of the Mersenne twister pseudo random number generator.

4.2.1.1 Index assignment and neigborhood relations

Our approach to the lattices will be quite different from the way periodic structures are usually introduced, where one defines a Bravais lattice through translation vectors and possibly a basis of atoms attached to each point of the Bravais lattice. This is a rather geometric approach to the lattices. We are going to take another approach, completely ignore these geometric aspects of the lattices and only think in terms of indices. In principle there are two different types of indices that we will have to distinguish: On the one hand we have the combined spin and site indices in the range 0 to $2L-1$, where the first L indices label the spin-$\uparrow$ states and the last L the spin-$\downarrow$ states. On the other hand there are the indices in the range 0 to $L-1$ that only label the different sites and ignore the spin. Both types of indices are used in the code, often at the same time, and in order not to be terribly confusing it is necessary to distinguish strictly between the two. This is an ideal use of the `typedef` keyword in C++, which we used to define `lattice:spindex` for the combined indices in the range 0 to $2L-1$ and `lattice:index` for the site indices in the range 0 to $L-1$. Both are essentially just new names for `unsigned int`, implying that this is no real type safety. It makes the code much more readable though, and we even want to use the term "spindex" for the remainder of this chapter to refer to the combined site and spin indices.

lattice.hpp:53

lattice.hpp:52

Let us look at the example of a two-dimensional square lattice of size 6×6 that is implemented in the `Lattice2DSquare` class to see how this index oriented approach works in practice. Figure 4.2 shows such a lattice and how `hVMC`'s internal representation looks like. Let us go through the elements of the figure step by step.

lattice_2dsqr.hpp:33

The first thing one has to do is to assign the spindices. We have already said that we want the first L spindices to be for the spin-$\uparrow$ states, but how those are assigned to the lattice sites is up to us. In principle it does not matter, but a unfavorable choice can make everything unnecessarily complicated. For the square lattice we have chosen to assign contiuous spindices to the rows of sites, as shown in figure 4.2.

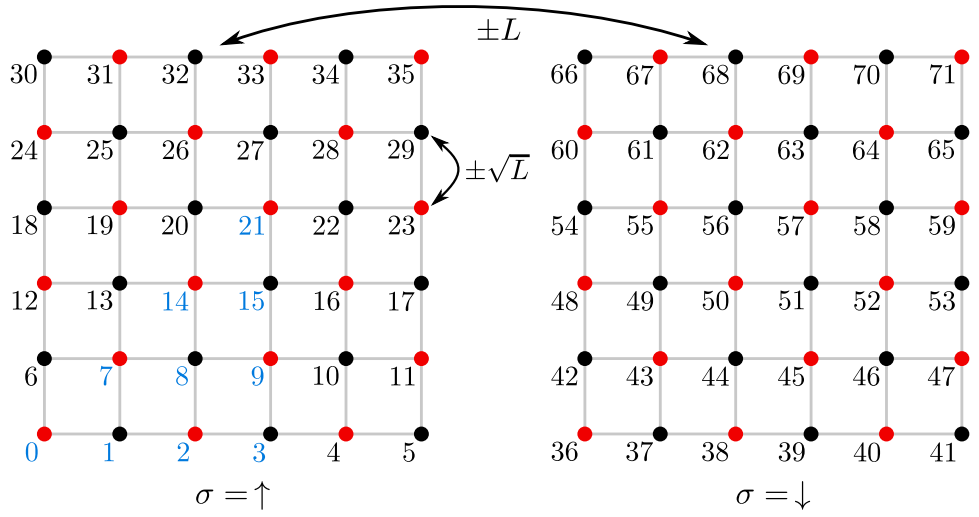

Figure 4.2: Visualization of the internal representation of an $L = 36$ square lattice. The numbers in the range 0 to 71 are the spindices (combined site and spin indices). The two sublattices are shown on red/black. Irreducible spindex relations are marked in blue. Note that their choice is not unique.

Having assigned the spindices, we can now think about how to find out the neighborhood relations between them. Looking at figure 4.2 we see that for spindices which are not on the boundary of the lattice we can find the nearest neighbors of spindex i at $i \pm 1$ and $i \pm \sqrt{L}$. We can modify these rules to also work on the boundaries of the lattice. For the spindices at the upper edge of the lattice, we can for example find its top neighbor at $i + \sqrt{L} - L$. Similar rules can be found for all boundaries. Note that in order to check whether a spindex i is on the boundary of the lattice, one can easily calculate its corresponding index j as the remainder of the integer division i/L, and then calculate the position through an integer division $j/\sqrt{L}$, where the quotient is the site's y position and the remainder its x position. All of the neighborhood relations in hVMC are determined through rules like the ones above, which of course depend on the actual lattice that is being simulated. The downside of this is that one has to implement these rules for every new lattice that one adds, but on the other hand they are very quick to execute as they usually only involve integer arithmetic and possibly some branches.

lattice_2dsqr.cpp:91-126

lattice.cpp:28

lattice_2dsqr.cpp:42-54

4.2.1.2 Mapping of index pairs to Jastrow parameters

The use of the Jastrow factor generates the need for another feature of the lattice implementation: The Jastrow parameters v_{ij} only depend on the distance between the sites with the indices i and j, so the number of Jastrow parameters is much smaller than the number of index combinations, which is L^2. We therefore need to be able to map an index pair ij to the correct Jastrow parameter. In principle the right parameter is selected according to the distance r_{ij}, but it is not a good idea to implement it in that straightforward way for two reasons: First we would have to do the calculation with floating

point numbers since the positions $\vec{r}_i$ and $\vec{r}_j$ are in general completely arbitrary and the distance r_{ij} will probably involve some square root that is not an integer anyway. The use of floating point numbers is especially painful at this point, because we would like to make checks for equality, but there is no guarantee that two distances r_{ij} and r_{lm} that should be equal in an unlimited precision arithmetic are still equal in floating point arithmetic. We could probably work around this issue by using some kind of fuzzy test for equality, but this is where the second problem strikes: Even if we could calculate the distance r_{ij} with unlimited precision and express the Jastrow parameters as $v(r_{ij})$, we would still be doing a mapping from a real number r_{ij} to another real number $v(r_{ij})$. Something like this could certainly be implemented, e.g. using a `std::map<double,double>` from the C++ standard library, but it is certainly much more complicated than mapping an `int` to a `double`, which is basically what any plain `double[]` array does!

We are going to implement the mapping from the index pairs ij to the Jastrow parameters v_{ij} without directly relying on calculations of the distance r_{ij}. Instead we are going to first map the pair (i, j) to another pair $(0, k)$, in a way that $r_{ij} = r_{0k}$, and then use the integer k to access the elements of an array holding the Jastrow parameters. Let us look at figure 4.2 again, and consider the index pair $(13, 25)$. These sites are third nearest neighbors, so we might say that the relation between 13 and 25 is the same as the relation between 0 and 2, which are also third nearest neighbors, and hence $v_{13,25} = v_{0,2}$. This also works across the periodic boundaries, where for example $(7, 35)$ is equivalent to $(0, 14)$. In this way we have reduced the pair of indices (i, j) to a pair $(0, k)$, which is fully determined by a single integer k. Let us call the number k an irreducible index relation. Note that not all pairs $(0, i)$ are irreducible: The pair $(0, 13)$ is for example equivalent to $(0, 8)$, so that one can chose *either* 8 *or* 13 as an irreducible index relation. A possible choice of irreducible index relations is shown in blue in figure 4.2.

This method of identifying irreducible index relations has a lot of advantages, most importantly that the irreducible index relation is just an integer and can therefore be used as an array index, but it also comes with two downsides: First a specialized method that calculates the irreducible index relation from a pair of indices must be implemented for every lattice. Second it is only applicable to lattices, where every relation between to sites (i, j) can be reduced to a relation $(0, k)$, that is a relation between a site k and the site with index 0. This is certainly possible for any Bravais lattice and a number of other important lattices (e.g. the honeycomb lattice), but it is not possible in general. lattice.hpp:69

Compared to the neighborhood relations and the Jastrow parameter reduction, the other methods of the `Lattice` interface are relatively straightforward, and can be picked up by reading the actual code.

4.2.2 The `Jastrow` class

The Jastrow class is very simple and its main purpose is to save the Jastrow parameters v_{ij} and then overload the `operator()` so that one can conveniently jastrow.hpp:53

access the Jastrow parameters via `Jastrow( i, j )`. This is in strict analogy to how we have written down the equations in the last chapter and makes the code more readable. Internally the `Jastrow` class stores the Jastrow parameters
jastrow.hpp:41 in a simple array according to the irreducible index relation that they apply to. For the square lattice in figure 4.2 this is an array of 22 elements (index 0 to 21), not all of which are really used. For example 13 is not an irreducible index relation, meaning that the 13-th element of the array is never actually accessed. In principle we are wasting a tiny bit of memory here, but on the other hand benefit massively from being able to access the Jastrow parameters very quickly by simply indexing into an array.

Note that the `Jastrow` class uses methods from the `Lattice` implementation to reduce the index relations, explaining the "makes use of" arrow between the two in figure 4.1.

4.2.3 The `ParticleConfiguration` class

The `ParticleConfiguration` class takes care of remembering the current positions of the N_p particles on the lattice. `hVMC`'s core components work completely within the particle-hole transformation, so `ParticleConfiguration` tracks the spin-$\uparrow$ electrons and the sites not occupied by a spin-$\downarrow$ electron. There are two common questions that the `ParticleConfiguration` answers: Is the spindex i occupied by a particle or unoccupied? Where is particle α? In order to answer both questions efficiently – that is in $\mathcal{O}(1)$ time – one needs to
pconf.hpp:79-80 store the particle positions as well as the site occupations. Of course we are storing unnecessarily much information here, as one could easily find out if spindex i is occupied by checking the positions of all N_p particles. This would be a $\mathcal{O}(N_p)$ operation though, which we would like to avoid, and the simplest way to do this is by storing particle positions as well as site occupations, and
pconf.cpp:192-195 keeping both in sync once the configuration is updated.

It is also the `ParticleConfiguration`'s job to suggest new particle config-
pconf.cpp:108-181 urations for the Metropolis algorithm. This is done by selecting a particle randomly and then suggesting to hop it to a neighboring site. In order to do this, the `ParticleConfiguration` needs access to the `Lattice` implementation for finding out the neighborhood relations, and the random number generator, see figure 4.1.

4.2.4 The `VariationalHamiltonian` and `DeterminantalWavefunction` classes

The `VariationalHamiltonian` and `DeterminantalWavefunction` classes together describe the mean field part of the wavefunction.

Let us start with the `VariationalHamiltonian` class, which is basically a
detwf.hpp:45 wrapper around the matrix representation $\boldsymbol{H}_{\text{var}}$ of the variational Hamiltonian $\hat{H}_{\text{var}}$. The stored matrix is initialized to zero on construction and not directly accessible from the outside, but it can be modified by adding other matrices to it. The variational Hamiltonian introduced in subsection 3.1.1 consists of a number of different terms anyway (hopping and pairing terms for

neighboring sites, $\hat{H}_{\text{mag}}$, and $\hat{H}_{\mu}$), so it is quite natural to build their matrices individually and then add them together.

There are two different ways of adding terms to the variational Hamiltonian's matrix $\boldsymbol{H}_{\text{var}}$. The first one simply adds a term to the matrix without any additional functionality. The second one adds a term of the form $\alpha_k \boldsymbol{V}_k$ to the Hamiltonian, where α_k is a variational parameter. In this case the term is not only added, but the matrix $\boldsymbol{V}_k$ is also saved for future reference, since all of the perturbation theory from section 3.3.2 can only be done after the diagonalization of the *complete* variational Hamiltonian. detwf.hpp:54 detwf.hpp:55 detwf.hpp:48

Once the variational Hamiltonian has been built, it is passed into the constructor of the `DeterminantalWavefunction`, which then diagonalizes $\boldsymbol{H}_{\text{var}}$ to obtain its eigenenergies ϵ_n and the matrices $\boldsymbol{U}$ and $\boldsymbol{M}$ as defined in equation (3.42) and (3.43). It also calculates the matrices $\boldsymbol{A}_k$, that will be needed for the evaluation of the logarithmic derivatives of the mean field parameters, from the stored $\boldsymbol{V}_k$ through the equations (3.76) and (3.77). detwf.hpp:87-88 detwf.cpp:70-71 detwf.cpp:74-92

Note that while the neighborhood relations from the `Lattice` class are needed to build the terms that are added to the variational Hamiltonian, once constructed neither `VariationalHamiltonian` nor `DeterminantalWavefunction` need access to the `Lattice` implementation, and hence there is no connection between them in figure 4.1.

4.2.5 The `WMatrix` and `TVector` classes

The two classes `WMatrix` and `TVector` are wrappers around the matrix $\boldsymbol{W}$ as defined in equation (3.50) and the vector $\vec{T}$ from equation (3.66), both of which are needed to efficiently calculate overlap ratios between different particle configurations. Both classes have a method to calculate $\boldsymbol{M}$ (or $\vec{T}$) from scratch for a given particle configuration, as well as a method to update them on accepted Metropolis steps. wmatrix.hpp:58-59 tvector.hpp:46-47

Let us first have a look at the implementation of the `WMatrix` class. At the beginning of the simulation, the `WMatrix` is constructed for the initial particle configuration by solving the linear system $\boldsymbol{D}^T \boldsymbol{W}^T = \boldsymbol{M}^T$ from equation (3.51) using a full pivoting LU decomposition of $\boldsymbol{D}^T$ from the Eigen3 linear algebra library [GJ+10]. This decomposition is rank revealing, which is necessary at this point to check if $\boldsymbol{D}^T$ is invertible. If it is not, the randomly generated initial particle configuration $|x\rangle$ does not have any overlap with the determinantal wavefunction $|\Phi\rangle$, in which case new configurations are generated until one with $\langle x|\Phi\rangle \neq 0$ is found. wmatrix.cpp:66-77 modman.cpp:55 modman.cpp:70-80

The update of the matrix $\boldsymbol{W}$ on accepted Metropolis steps is the computational hotspot within `hVMC` and its efficient execution is therefore of the utmost importance. Luckily, the entire update as written in equation (3.58) can be done in place and with a single BLAS call: Let $\boldsymbol{r}_l$ be the l-th row vector of $\boldsymbol{W}$ and $\boldsymbol{c}_\beta$ the β-th column vector, both of which can be extracted from $\boldsymbol{W}$ in linear time. Next we will subtract 1 from the β-th component of $\boldsymbol{r}_l$, the result of which we will call $\tilde{\boldsymbol{r}}_l$. As we only modify a single element, this can of course be done in constant time. The updated matrix $\boldsymbol{W}'$ can now be calculated by wmatrix.cpp:94-172

subtracting from $\boldsymbol{W}$ the outer product of $\boldsymbol{c}_\beta$ and $\tilde{\boldsymbol{r}}_l$ divided by $W_{l\beta}$.

$$\boldsymbol{W}' = \boldsymbol{W} - \frac{1}{W_{l\beta}} \boldsymbol{c}_\beta \tilde{\boldsymbol{r}}_l \tag{4.1}$$

This last step has $\mathcal{O}(N_p^2)$ complexity, but can be performed by a single call to
wmatrix.cpp:198-221 the `(s/d)ger` method of an optimized CBLAS implementation. `hVMC` can link against any implementation of CBLAS, but benchmarks of different libraries on different computers suggest a slight overall lead of the ATLAS library. [WD97]

Updating $\boldsymbol{W}$ along with the particle configuration is much faster than recalculating it after every change, but also comes with the problem of slowly accumulating floating point errors. This can be controlled be calculating $\boldsymbol{W}$ from scratch every once in a while in order to "reset" the floating point errors. The question here is of course how often one can update $\boldsymbol{W}$ before one should recalculate it. This question is rather difficult to answer and also depends on which particle configurations the system goes through, but the least we could do is to compare at every recalculation the recalculated $\boldsymbol{W}_r$ with the value $\boldsymbol{W}_u$ of the matrix that was updated a number of times. They should be the same in unlimited precision arithmetic, but if the two are not too different, we can at least state in hindsight that our limited precision did not cause any problems
fpctrl.hpp:87-99 since the last recalculation. `hVMC` uses the following measure to judge what "not too different" means.

$$\Delta W = \sqrt{\frac{\sum_{i\alpha} (\boldsymbol{W}_u - \boldsymbol{W}_r)_{i\alpha}^2}{\sum_{i\alpha} (\boldsymbol{W}_r)_{i\alpha}^2}} \tag{4.2}$$

options.cpp:208-210 Values of $\Delta W \approx 0.001$ are still completely safe, and one can then tune the number of updates between recalculations to stay below this threshold. This of course also depends on whether the calculations in `WMatrix` are done in single or double precision. For performance reasons we will by default use single precision (see appendix A.2.2 for a discussion), for which 500 updates
options.cpp:212-219 between recalculations is a generally safe value. In double precision one can perform up to 5000 updates. In principle one could also adjust the number of updates dynamically during the simulation based on the previous deviations,
fpctrl.cpp:23-36 but such a system was not implemented into `hVMC`. Instead it simply checks
mccrun.cpp:197-233 the deviation and outputs warnings if it is above a user defined threshold.

Note that at the point where the matrix $\boldsymbol{W}$ is recalculated, we already know that the current particle configuration has a nonzero overlap with the determinantal wavefunction, and therefore there is no need to check the rank of the matrix $\boldsymbol{D}^T$. We can turn this knowledge into performance by now using a
wmatrix.cpp:178-179 partial pivoting LU decomposition that executes quicker than its full pivoting counterpart, but is not rank revealing and would fail silently on a non-invertible matrix $\boldsymbol{D}^T$.

The `TVector` class has almost the exact same interface as `WMatrix` but its implementation is much simpler and more or less a straightforward translation of the equations in subsection 3.2.2 into C++. It uses the same floating point precision control as `WMatrix`, but in practice errors on the vector $\vec{T}$ accumulate much slower.

4.2.6 The `ModelManager` class

Now that we have described all the other classes, we can finally have a look at the `ModelManager` class which glues all of the core components together.

The construction of the `ModelManager` is a rather complicated process as all of the relations shown in figure 4.1 have to be setup up. The `Lattice` implementation, the `Jastow`, the `DeterminantalWavefunction`, and the random number generator are first set up on their own and then passed into the constructor of the `ModelManager` class which takes care of creating the `WMatrix` and `TVector`. Even though some of the core components are constructed outside of the `ModelManager`, they can still be thought of as being owned by the `ModelManager`, which will take care of cleaning them up when it is destructed. mccrun_prepare.cpp:35-84

The public interface of the `ModelManager` class is based on how one thinks about Monte Carlo simulations: There is a method that performs a Monte Carlo step and methods to measure the current local value of the observables. Note that the methods of the `ModelManager` measure the observables for the original Hubbard Hamiltonian, and not its particle-hole transformed version[8]. The `dblocc_dens()` method measures for example the local value of $1/L \sum_i \hat{n}_{i\uparrow}\hat{n}_{i\downarrow}$, that is the double occupancy of the original electrons and not the transformed particles. All of the core components work completely in the particle-hole transformation, but the `ModelManager`'s interface exposes the original observables to the rest of the code. In this way it kind of splits the code into two parts where the inner one works in particle-hole transformation, whereas the outer part works with the original system. modman.hpp:109 modman.hpp:112-117

4.3 Peripheral components

In the last section we have presented `hVMC`'s core components, effectively explaining the codebase from the inside out. In this section we are going to start from the outside, working our way in until we again arrive at the interface of the `ModelManager` class.

4.3.1 Scheduler programs

In principle one can think about a VMC simulation as a three step process: **Optimizing** the variational wavefunction. **Simulating** the system using the optimized wavefunction. **Analyzing** the simulation's results. This is a typical `hVMC` workflow, where it is run three times, each step writing its output to disk so that it can be read as the input of the next stage. Let us refer to the layer of the code where this distinction between the different modes is made as the scheduler. We will see why this is an appropriate name in a second.

[8] There is one exception to this rule: The particle configurations are always recorded for the transformed particles. There is little use in recording them at all, except for using them as the initial configurations of other simulations, in which case it is more convenient to already have the transformed configurations.

hVMC is parallelized via MPI and uses a master/slave model. This difference between the running processes is reflected in the scheduler. The master process has three different scheduler programs that are selected depending on which of the three modes described above is running. The slave processes only have a single program in which they are stuck in a loop, waiting for a messages from master. There are currently only two messages that master would send to the slaves: One to instruct the slaves to help master run a Monte Carlo cycle and the other one to tell them that there is no more work and that they are allowed break from the loop and exit. In summary one could say that the slaves wait for jobs which are scheduled by the master according to what needs to be calculated.

sched.cpp:60-85 sched.cpp:474-504 msgtags.hpp:32-35

Before we go on to explain how the master runs a Monte Carlo cycle together with the slaves, let us have a closer look at the three scheduler programs of the master.

The optimization program is by far the most complex. It starts with some initial variational parameters and then continually schedules Monte Carlo cycles which measure the logarithmic derivatives and the energy, obtains new variational parameters through the equations (2.38) and (2.39), and finally checks for convergence by performing a Mann-Kendall trend test on the variational parameter evolution. Once convergence is reached, the Stochastic Reconfiguration cycle is continued for a number of iterations during which the variational parameters are averaged. The averaged parameters are then written to disk.

sched.cpp:89-384 sched.cpp:92 sched.cpp:175-214 sched.cpp:232-246 sched.cpp:265-293 sched.cpp:360-365 sched.cpp:378-383

The simulation program will typically read some variational parameters from disk and schedule a single Monte Carlo cycle which measures a set of user specified observables. The results are then written to disk.

sched.cpp:388-453

The analysis program is even simpler. It just checks which analysis modules the user has selected, and runs those. Currently there is only one analysis module: The calculation of the static structure factor as a Fourier transform of the charge-charge correlation. This is a very quick operation, so no work is scheduled to the slaves. If a more expensive analysis would be required, one could easily offload parts of it to the slaves.

sched.cpp:457-470 analysis.cpp:42-85

4.3.2 The distributed Monte Carlo cycle

Both the optimization and the simulation program perform a Monte Carlo cycle that does exactly what one would expect it to do: It sets up a `ModelManager` instance and then uses its interface to equilibrate the system and measure the desired observables. The only non-trivial thing about hVMC's implementation of the Monte Carlo cycle is that it runs in parallel on the master and its slaves. The idea behind the parallelization is very straightforward: Instead of having one long Markov chain with N_b bins it sets up a number of Markov chains equal to the number of processes and propagates those chains in parallel until a total number of N_b bins have been generated, at which point the results from all the different Markov chains are averaged. This is a very common way to parallelize Monte Carlo simulations and very easy to implement.

mccrun.cpp:48 mccrun.cpp:109 mccrun.cpp:149-153

The easiest way to split the Markov chain would be to assume that all n_{proc} processes are equally fast and to split the Markov chain into n_{proc} subchains of length N_b/n_{proc}. In practice this does not work too well: If the number of involved processes is large enough, one of them is almost always slower than the rest for an unknown and not reproducible reason, in which case all processes have to wait until the last one has finished. `hVMC` therefore implements a dynamic load balance, where the master hands out new bins to the slaves which have reported the completion of their previous work, until all N_b bins have been distributed. In this way faster processes simply calculate more bins, while the slowest process does not hinder the others very much. Note that the master process itself also calculates its own Markov chain, but in order to reduce communication latency communicates with the finished slaves after every Monte Carlo step instead of after every bin. The communication latency is further reduced on the slave's side: A new bin is not requested when the former bin has been completed, but while it is still calculating, so that the slave ideally never has to wait for an answer from master. Benchmarks on different machines have shown that the interconnect (within machine, Infiniband, Ethernet) has very litte influence on `hVMC`'s overall performance.

`mccrun.cpp:66-91`

`mccrun.cpp:138-154`

`mccrun.cpp:277-286`

Let us use Amdahl's law [Amd67] to calculate the maximum speedup possible with our parallelization. Let t_{prep} be the time needed by a process to prepare the Markov chain (e.g. diagonalizing the variational Hamiltonian, etc.), and t_b the time needed to calculate one bin. Let N_{equil} be the number of bins needed for equilibration and N_b the number of bins used for measuring the observables. The completely serial runtime can then easily be calculated.

$$t(1) = t_{\text{prep}} + t_b \left(N_{\text{equil}} + N_b\right) \tag{4.3}$$

When splitting the Markov chain one still has to prepare and equilibrate every one of the subchains, so that the parallel runtime with n_{proc} processes is given by the following expression.

$$t(n_{\text{proc}}) = t_{\text{prep}} + t_b \left(N_{\text{equil}} + \frac{N_b}{n_{\text{proc}}}\right) \tag{4.4}$$

The speedup $\mathcal{S}(n_{\text{proc}})$ is the ratio between the two, and its maximum value $\mathcal{S}_{max}$ is obtained if one process is used for every bin.

$$\mathcal{S}(n_{\text{proc}}) = \frac{t(1)}{t(n_{\text{proc}})} \qquad \Rightarrow \qquad \mathcal{S}_{max} = \mathcal{S}(N_b) \tag{4.5}$$

Let us plug in some realistic numbers to get an impression of the possible speedups. The preparation of the Markov chain is usually quite cheap, so let us set $t_{\text{prep}} = t_b$. For the wavefunction optimization it is usually enough to calculate around $N_b = 100$ statistically independent bins to get a reasonably accurate estimate of the energy and the logarithmic derivatives. For a binlength which is not much larger than the number of Monte Carlo steps needed for decorrelation, one should do at least $N_{\text{equil}} = 10$ bins for equilibration. The

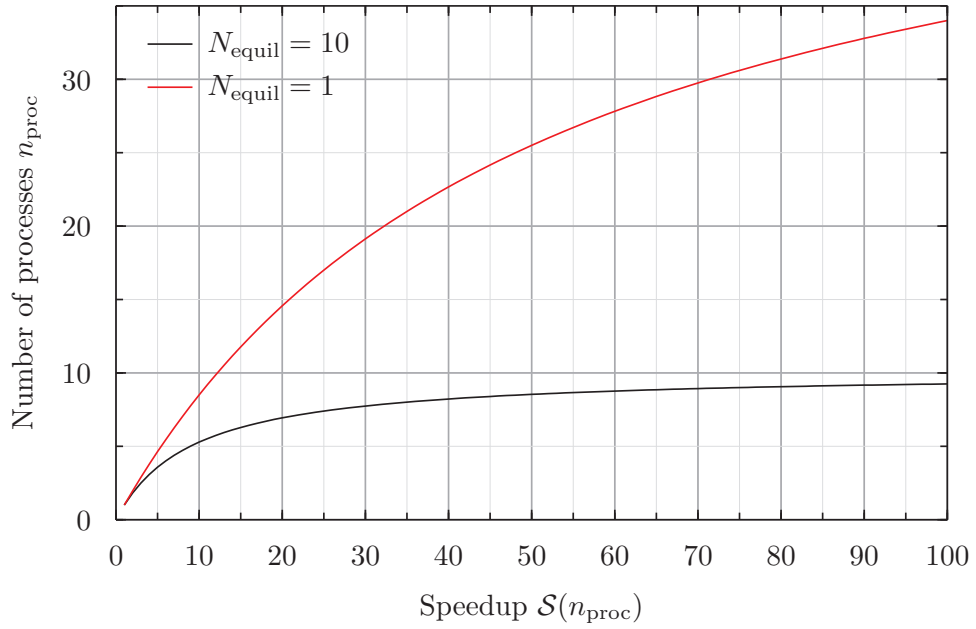

Figure 4.3: Estimation of the speedup for a Monte Carlo cycle of $N_b = 100$ bins. Reducing the inherently serial part of the calculation, that is number of bins used for equilibration, massively improves the scalability of the distributed Monte Carlo cycle

black line in figure 4.3 shows the resulting speedups for these numbers. We can see that the necessity to equilibrate every Markov chain severely limits the possible speedup, and for $N_b = 10$ it is probably not worth it to use more than a few processes. In order to obtain better speedups and be able to use more processes, one should try to reduce the number of bins needed for equilibration N_{equil}, since those are the bulk of the inherently serial part of the calculation. Note that Amdahl's law only limits our speedup during the optimization stage: Once the optimized wavefunction has been found, one simply wants to calculate a Markov chain that is as long as possible in order to reduce the errors. In this case the serial part of the calculation is only a tiny fraction, and one can get practically linear speedups for several hundred processes.

sched.cpp:156-158
sched.cpp:191-200
mccrun.cpp:96-101

In order to reduce the effect of Amdahl's law on the scalability of the optimization stage, `hVMC` uses particle configurations from the former iteration in the Stochastic Reconfiguration algorithm as the initial configurations for the current Monte Carlo cycle. The idea here is that the wavefunction does not change a lot between iterations, so configurations generated with the old wavefunction are almost correctly distributed for the new wavefunction. Since the distribution of the initial configurations is already almost correct, the number of bins can be shortened drastically. `hVMC` reduces the number of equilibration bins by a factor of 10 if an old configuration was used as the initial state. In the above example this would mean that we only use a single bin for the equilibration, which gives a much better scalability as shown in figure 4.3.

4.3.3 The `Observable` interface and derived classes

The last aspect of `hVMC` we have to look at is the implementation of the observables. We have already seen that the `ModelManager` class exposes the current local value of some observables through its public interface. There is a lot more to measuring observables though: The local values must be averaged, and in case one wants to calculate the uncertainties one also has to do the binning. In general we also have to average the different Markov chains in the distributed Monte Carlo cycle, each one weighted with the length of its Markov chain. Then there are observables like correlation functions $\langle \hat{o}_i \hat{o}_j \rangle$, where the local value $o_i(x)o_j(x)$ can be calculated from the local value of another observable, in this case $o_k(x)$. We see that while the `ModelManager` class returns local values of some observables, there is still a lot to do until one has the final result. Last but not least we would also like to make the observable system highly modular, so that it is easy to add new observables to `hVMC`.

`hVMC` implements a pure virtual `Observable` class that serves as an interface which the different observables then implement through inheritance. The interface is relatively simple: There is a function that accepts a reference to the `ModelManager` class and then uses its public interface to make a single measurement of the local value of the observable. Another function is used to signal to the observable that a bin has been completed. Finally there are two functions only one of which is called depending on whether the process is the master or a slave in the distributed Monte Carlo cycle. The slaves' function simply sends the recorded results to master, while the master's function gathers all the slaves' data, does the averaging and if necessary the binning, and saves everything in a small utility class called `MCCResults`. Having the observables as different implementations of a common interface allows us to treat all of them with the exact same code, limiting the amount of changes needed to add an observable and greatly increasing the readability of the `hVMC` code.

obs.hpp:63-83
obs.hpp:69-72
obs.hpp:74
obs.hpp:80-82
obs.hpp:76-79
mccresults.hpp:85-97
mccrun.cpp:149-159

There is one last issue that is related to the performance of related observables: Suppose that we want to measure the logarithmic derivatives $\langle \hat{\Delta}_{\Psi k} \rangle$ as well as their correlation matrix $\langle \hat{\Delta}_{\Psi k} \hat{\Delta}_{\Psi k'} \rangle$. Both are implemented into `hVMC` as separate observables, but on a local level they both just need to measure the local value of $\Delta_{\Psi k}(x)$, for which there is a method in the `ModelManager` interface. In order to avoid calling this method twice and thereby actually calculating the exact same thing twice, `hVMC` implements a small cache between the `ModelManager` class and the observables.

obs.hpp:45-60

In this chapter we have presented `hVMC`, the VMC code for the Hubbard model that was written as a part of this thesis. Now that we finally have all the theory as well as its implementation we can finally test it on an actual system: The square lattice bilayer Hubbard model.

5 The bilayer Hubbard model

5.1 Motivation and previous results

Transitions between Mott and band insulators have been a topic of intensive debate recently, as the transition between the two phases is a useful tool to study the differences in the nature of the phases themselves. Different models that undergo a Mott to band insulator transition have been investigated, for example the extended Hubbard model [EN07], the ionic Hubbard model in one and two dimensions [BA04][KD07], and the bilayer Hubbard model [FHM06][KO07][HKL09]. In this chapter we want to apply the Variational Monte Carlo method to the square lattice bilayer Hubbard model at half filling. Previous investigations of this system have mostly used DMFT and its cluster extensions, so that using a completely unrelated method like VMC can serve to validate or complement the existing results.

The bilayer Hubbard model with only nearest neighbor hopping in the plane and a hopping between the planes is given by the following Hamiltonian.

$$\hat{H} = \underbrace{-t\sum_{l,\sigma}\sum_{\langle ij\rangle}\left(\hat{c}^\dagger_{j,l,\sigma}\hat{c}_{i,l,\sigma} + \text{h.c.}\right)}_{\hat{H}_t} \underbrace{- t_\perp \sum_{i,\sigma}\left(\hat{c}^\dagger_{i,2,\sigma}\hat{c}_{i,1,\sigma} + \text{h.c.}\right)}_{\hat{H}_{t_\perp}} \underbrace{+ U\sum_{i,l}\hat{n}_{i,l,\uparrow}\hat{n}_{i,l,\downarrow}}_{\hat{H}_U} \tag{5.1}$$

Here the index $l \in \{1, 2\}$ labels the plane of the site. Note that in the following we measure all energies in units of t, so that for example $U > 8$ would actually mean $U/t > 8$.

Let us as a very first step look at what band theory predicts for the $U = 0$ case, where $\hat{H} = \hat{H}_t + \hat{H}_{t_\perp}$. To this purpose it is more convenient to sum over the positions $\vec{r} \equiv \vec{r}_i$ instead of the unit cell labels i. Note that we can also drop the spin index σ as the entire Hamiltonian is just a sum over spin.

$$\hat{H}_t = -t\sum_{\vec{r},l}\left(\hat{c}^\dagger_{\vec{r}+\vec{e}_x,l} + \hat{c}^\dagger_{\vec{r}-\vec{e}_x,l} + \hat{c}^\dagger_{\vec{r}+\vec{e}_y,l} + \hat{c}^\dagger_{\vec{r}-\vec{e}_y,l}\right)\hat{c}_{\vec{r},l} \tag{5.2}$$

$$\hat{H}_{t_\perp} = -t_\perp\sum_{\vec{r}}\left(\hat{c}^\dagger_{\vec{r},2}\hat{c}_{\vec{r},1} + \text{h.c.}\right) \tag{5.3}$$

Let L be the total number of lattice sites. That gives $L/2$ sites per layer and that is the number of vectors in the $\vec{r}$ sum. The hermitian conjugate of the in-plane hopping term is now absorbed into the $\vec{r}$ sum that takes care of the reverse hopping direction. Going to k-space by substituting the following operators almost diagonalizes the Hamiltonian.

$$\hat{c}_{\vec{r},l} = \frac{1}{\sqrt{L/2}}\sum_{\vec{k}} \mathrm{e}^{\mathrm{i}\vec{k}\cdot\vec{r}}\hat{c}_{\vec{k},l} \qquad \text{and} \qquad \hat{c}^\dagger_{\vec{r},l} = \frac{1}{\sqrt{L/2}}\sum_{\vec{k}} \mathrm{e}^{-\mathrm{i}\vec{k}\cdot\vec{r}}\hat{c}^\dagger_{\vec{k},l} \tag{5.4}$$

Using $2\cos(x) = \mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x}$ we get the following terms for $\hat{H}_t$ and $\hat{H}_{t_\perp}$.

$$\implies \qquad \hat{H}_t = -2t \sum_{\vec{k},l} \big(\cos(k_x) + \cos(k_y)\big)\, \hat{c}^\dagger_{\vec{k},l} \hat{c}_{\vec{k},l} \tag{5.5}$$

$$\hat{H}_{t_\perp} = -t_\perp \sum_{\vec{k}} \left(\hat{c}^\dagger_{\vec{k},2} \hat{c}_{\vec{k},1} + \text{h.c.} \right) \tag{5.6}$$

For every point in k-space this can now be written as a 2×2 matrix.

$$\hat{H} = \sum_{\vec{k}} \begin{pmatrix} \hat{c}^\dagger_{\vec{k},1} \\ \hat{c}^\dagger_{\vec{k},2} \end{pmatrix}^T \begin{pmatrix} -2t\big(\cos(k_x) + \cos(k_y)\big) & -t_\perp \\ -t_\perp & -2t\big(\cos(k_x) + \cos(k_y)\big) \end{pmatrix} \begin{pmatrix} \hat{c}_{\vec{k},1} \\ \hat{c}_{\vec{k},2} \end{pmatrix} \tag{5.7}$$

Diagonalizing this matrix gives the bandstructure $\epsilon_{\vec{k}}$ of the square lattice bilayer.

$$\epsilon^{(1,2)}_{\vec{k}} = -2t\big(\cos(k_x) + \cos(k_y)\big) \pm t_\perp \tag{5.8}$$

We see that we get two bands of the same shape called the bonding ($-t_\perp$) and antibonding ($+t_\perp$) band that are shifted by $t_\perp$ in opposite directions. At half filling the system is metallic as long as the two bands overlap. The energetically highest state in the bonding band has the energy $4t - t_\perp$ at $\vec{k} = M \equiv (\pi, \pi)$, and the lowest state in the antibonding band is at $\vec{k} = \Gamma \equiv (0, 0)$ and has the energy $-4t + t_\perp$. Ergo the system undergoes a metal to band insulator transition at $t_\perp = 4$ (in units of t) for $U = 0$.

We have seen in the introduction that the perfect nesting of the Fermi surface makes the monolayer square lattice at half filling instable towards a Néel ordered antiferromagnetism. We now want to show that the bilayer's Fermi surfaces are also perfectly nested for any $t_\perp < 4$. Let us begin by noticing that without any $t_\perp$ the bandstructure is invariant under a shift of (π, π) and a reversal of the sign.

$$-2t\big(\cos(k_x) + \cos(k_y)\big) = 2t\big(\cos(k_x + \pi) + \cos(k_y + \pi)\big) \tag{5.9}$$

At half filling and $t_\perp = 0$ the Fermi surface is a diamond around $\Gamma \equiv (0, 0)$. If we now increase $t_\perp$ then the Fermi surface of the antibonding band shrinks around Γ, while the Fermi surface of the bonding band expands. Looking at it in a different way, we could also say that the bonding band's Fermi surface shrinks around $M \equiv (\pi, \pi)$, and because of the symmetry it does so in exactly the same way as the antibonding band's Fermi surface shrinks around Γ. This is shown in figure 5.1. This symmetry implies that the Fermi energy at half filling is zero for any value of $t_\perp$. The Fermi surface of the bonding band is then defined implicitly through the following equation.

$$0 = -2t\big(\cos(k_x) + \cos(k_y)\big) - t_\perp \tag{5.10}$$

Adding a vector $\vec{q} = (\pi, \pi)$ and multiplying with -1 produces the implicit definition of the antibonding band's Fermi surface.

$$0 = -2t\big(\cos(k_x + \pi) + \cos(k_y + \pi)\big) - t_\perp = -2t\big(\cos(k_x) + \cos(k_y)\big) + t_\perp \tag{5.11}$$

Ergo, we have a perfect Fermi surface nesting with $\vec{q} = (\pi, \pi)$ between the two bands, which signals an instability towards Néel ordered antiferromagnetism.

Having treated the $U = 0$ case which is exactly described by band theory, we can have a look at the other extreme case, which is $t_\perp = 0$. This case is equivalent to

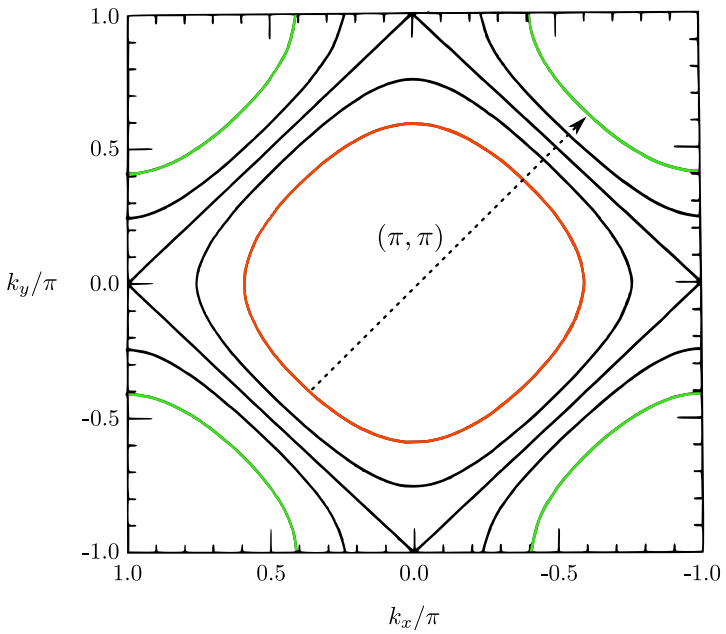

Figure 5.1: Fermi surfaces of the bonding (green) and antibonding (red) band for the square lattice bilayer with nearest neighbor hopping. The symmetry of the dispersion relation ensures a perfect Fermi surface nesting between the bands as long as $t_\perp < 4$. The black curves are Fermi surfaces for smaller values of $t_\perp$. Image modified from [Hir85].

the square lattice monolayer, as there is no coupling at all between the two planes. The system's behavior in the $t_\perp = 0$ case depends on whether we allow magnetic order. If we forbid magnetic order the system is just like the hydrogen lattice from the gedankenexperiment in the introduction, which we expect it to be metallic for small U and Mott insulating for large U, so that a Mott insulator transition should be found somewhere in between. Suppressing magnetic ordering is not as unphysical as it might seem, since the Mermin-Wagner theorem [MW66] forbids magnetic order in two dimensions for any $T > 0$. If we allow magnetic order, we have already seen in the introduction that the system will be a Néel ordered antiferromagnetic insulator for any value of $U > 0$.

Another interesting edge case we might want to look at is the $U \to \infty$ limit with magnetism, where the Hubbard Hamiltonian from equation (5.1) reduces to the following Heisenberg Hamiltonian.

$$\hat{H} = J \sum_l \sum_{\langle ij \rangle} \hat{\vec{S}}_{i,l} \cdot \hat{\vec{S}}_{j,l} + J_\perp \sum_i \hat{\vec{S}}_{i,1} \cdot \hat{\vec{S}}_{i,2} \qquad \text{with} \quad J = \frac{4t^2}{U}, \quad J_\perp = \frac{4t_\perp^2}{U} \tag{5.12}$$

The bilayer Heisenberg Hamiltonian has itself been subject to extensive research, and it has been found to undergo an order-disorder transition at $J_\perp/J = 2.552$, which is $t_\perp/t = 1.588$ in terms of the Hubbard Hamiltonian's hopping parameters. [SS94][WBS06]

The bilayer Hubbard model is certainly metallic for $t_\perp = U = 0$, and by looking at the edge cases we have seen that it can be made band insulating by increasing $t_\perp$ and Mott insulating by increasing U while suppressing magnetic order. The question that naturally comes up at this point is the behavior in the intermediate region where $t_\perp > 0$

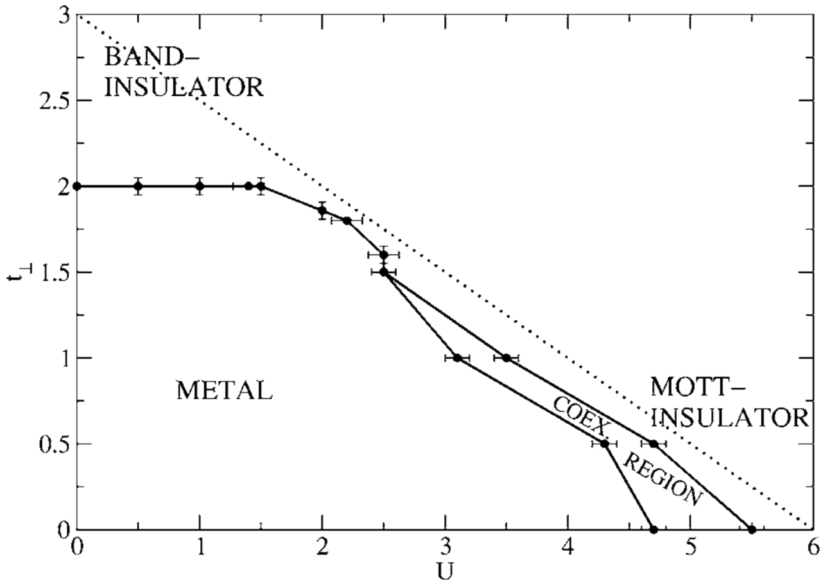

Figure 5.2: DMFT calculated phase diagram for the bilayer Hubbard model at a finite temperature of $T = 0.025$ as published by Fuhrmann, Heilmann and Monien [FHM06]. Magnetic order is forbidden by the Mermin-Wagner theorem [MW66]. No clear separation between Mott and band insulator was found. Note that the definition of the $t_\perp$ in the plot differs from ours by a factor of two.

and $U > 0$. Especially interesting is the case where U is large enough to induce Mott insulation, so that we could see a transition from a Mott insulator to a band insulator by increasing $t_\perp$.

So far the phase diagram has been calculated using DMFT by Fuhrmann, Heilmann and Monien [FHM06] and through Cluster DMFT by Kancharla and Okamoto [KO07]. Let us have a quick look at the phase diagrams they obtained.

The DMFT phase diagram published by Fuhrmann, Heilmann and Monien is shown in figure 5.2. They used a Quantum Monte Carlo impurity solver in the DMFT cycle and were therefore unable to perform simulations at zero temperature. Long range magnetic order is forbidden by the Mermin-Wagner theorem in two dimensions at finite temperatures [MW66], and therefore they present a nonmagnetic phase diagram. A metallic and an insulating phase were found, were the insulating phase surely is band insulating at $U = 0$, and Mott insulating at $t_\perp = 0$ and large U. No clear separation between the Mott and the band insulating phase was found.

Figure 5.3 shows the Cluster DMFT calculated phase diagram published by Kancharla and Okamoto. They used a finite bath and exact diagonalization to solve the impurity problem, which allowed them to go to $T = 0$ and to obtain both a magnetic phase diagram as well as a $T = 0$ phase diagram where magnetism is artificially suppressed. There are several important differences with respect to the earlier phase diagram by Fuhrmann, Heilmann and Monien: First of all they are able to distinguish between a Mott and a band insulator by looking at the charge gap as a function of $t_\perp$. Another difference is that they find a Mott insulator at $t_\perp = 0$ for any value of $U > 0$, even if magnetism is artificially suppressed. They mention the CDMFT's intralayer spatial

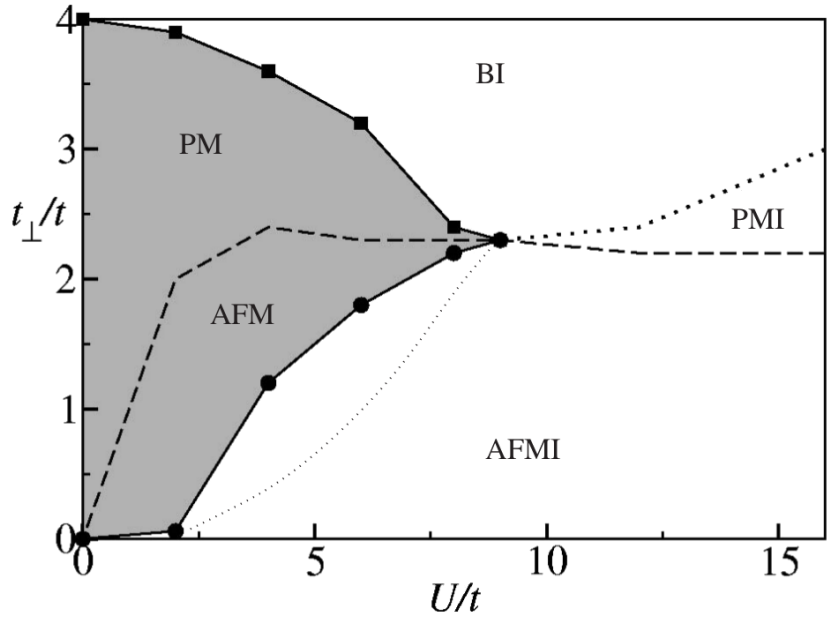

Figure 5.3: Cluster DMFT calculated phase diagram for the bilayer Hubbard model at $T = 0$ as published by Kancharla and Okamoto [KO07]. The region below the dashed line is magnetically ordered. The shaded region is metallic and extends down to lower $t_\perp$ if magnetic order is artificially suppressed, as shown by the closely spaced dotted curve.

correlation as the reason why this is found with CDMFT but not in the DMFT results by Fuhrmann, Heilmann and Monien. However, at $t_\perp = 0$ the planes are completely decoupled and should have the same properties as a single plane. For the single plane it is well established that it becomes Mott insulating at some finite U, see for example a recent publication by Gull, Parcollet and Millis [GPM13], where the critical U was calculated using the DCA (Dynamical Cluster Approximation) extension of DMFT and a variety of cluster sizes. A possible reason for having a Mott insulator for any finite U might be the small cluster size of 2×2 (that is two sites per plane) used by Kancharla and Okamoto. Having two sites in each plane breaks the fourfold rotational symmetry of the square lattice and results in an artificially enhanced local pair within each plane for any $U > 0$. This can noticeably affect the phase diagram of a system, as reported by Lee et al. for the two-orbital Hubbard model [LZJ+10]. A particularly interesting feature of the phase diagram published by Kancharla and Okamoto is that for a certain range of U the system goes through two phase transitions as $t_\perp$ is increased, first from a Mott insulator to a metal and then from a metal to a band insulator. At large U the system exhibits a direct transition from a Mott to a band insulator. It is also noteworthy that no magnetic ordering was found by Kancharla and Okamoto for small U and $2 \lesssim t_\perp < 4$, even though we have seen that there is an instability towards Néel ordered antiferromagnetism due to the perfect nesting between the Fermi surfaces of the bonding and antibonding band.

Overall, the DMFT results do not give an entirely conclusive picture yet, making it worthwhile to also employ other methods in order to compare the result and thereby gain a deeper understanding of the physics in the bilayer Hubbard model.

5.2 Variational Monte Carlo results

We now want to present some results for the square lattice bilayer Hubbard model that were obtained with the Variational Monte Carlo method and the `hVMC` code.

The variational wavefunction that was used is in principle the one that was introduced in section 3.1, but the determinantal part only contains hopping and pairing between nearest neighbors within the planes (t and Δ), as well as between planes ($t_\perp^{(\mathrm{var})}$ and $\Delta_\perp$). Note that in order to set the energy scale for the variational wavefunction the hopping parameter t within the planes is the same in the variational single particle Hamiltonian and in the original Hubbard Hamiltonian, so that only the interplane hopping $t_\perp^{(\mathrm{var})}$ is a variational parameter. d-wave symmetry was used for the interplane pairing term as it is known to be the preferred symmetry for the two-dimensional square lattice [BK92]. The magnetic potential μ_m was either fixed at $\mu_m = 0$ for the nonmagnetic phase diagram or optimized along with the other parameters for the magnetic case. Note that the number of Jastrow parameters increases by a factor of two for the bilayer model, as an in-plane and interplane parameter are now needed for every possible distance in the lattice.

Before we look at the simulation data itself, let us first present the final result, namely the full nonmagnetic and magnetic phase diagrams that can be seen in figure 5.4. The nonmagnetic phase diagram shows a metallic phase at small U and $t_\perp$, which goes into a band insulating phase at $t_\perp = 4$. This is compatible with the band theory calculation from the last section. The critical value of $t_\perp$ that is needed to make the system band insulating decreases as U is increased. At small $t_\perp$ the system undergoes a metal to Mott insulator transition as U is increased, where the critical U increases from about 5.5 at $t_\perp = 0$ to 7.5 at $t_\perp = 0.7$. For large U the system undergoes a Mott to band insulator transition at $t_\perp \approx 0.7$, where the critical value of $t_\perp$ is independent from U and only weakly dependent on the system's size. Hysteresis in the variational parameters when going through the Mott to band insulator transition suggest the transition to be of first order. An interesting feature is that for $5.5 \lesssim U \lesssim 7.5$ the system first undergoes a Mott insulator to metal transition and then a metal to band insulator transition as the $t_\perp$ is increased.

Only two phases are found in the magnetic phase diagram. The system is a Néel ordered antiferromagnetic Mott insulator as long as $t_\perp$ is smaller than some critical value, and a paramagnetic band insulator for larger $t_\perp$. The critical interlayer hopping is $t_\perp = 4$ at $U = 0$ and decreases as U is increased. At $U = 10$ it reaches a value of about 1.9, which suggests that it approaches the Heisenberg limit of 1.588 for $U \to \infty$.

Let us now present the data and the considerations that have led us to these results.

5.2.1 Nonmagnetic phase diagram

In order to obtain the nonmagnetic phase diagram we have first analytically diagonalized the variational single particle Hamiltonian whose eigenstates are used to build the determinantal part of the wavefunction. The argument here is that if the Slater determinant is already gapped, no correlator can make the wavefunction conducting again. Ergo, we can identify a band insulator purely by looking for a gap in the band structure of the determinantal part. Of course there is the superconductor that has a gap in the mean field state and is not an insulator. The only parameter that could introduce

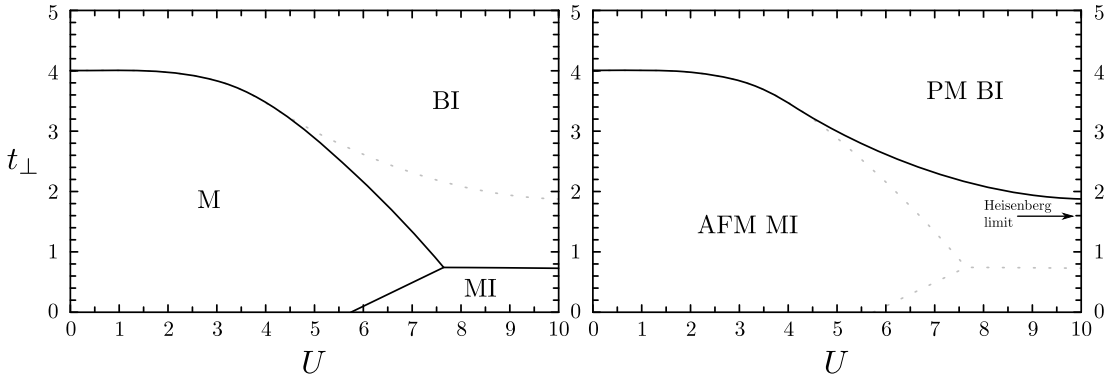

Figure 5.4: VMC calculated phase diagrams for the square lattice bilayer Hubbard model. The nonmagnetic phase diagram (left) shows a metallic phase (M), a band insulating phase (BI) and a Mott insulating phase (MI). The magnetic phase diagram (right) shows an antiferromagnetic Mott insulator (AFM MI) and a paramagnetic band insulator (PM BI).

superconductivity in our variational wavefunction is the in-plane pairing Δ, but since it has d-wave symmetry it will never open a gap in the nodal direction. Therefore, we do not run the risk of accidentally classifying a superconductor as an insulator, and the existence of a gap in the mean field state is indeed equivalent to the system being insulating.

In the nonmagnetic case the variational Hamiltonian consists of four terms for the intraplane/interplane hopping and pairing, where the hopping terms are given by equation (5.2) and (5.3).

$$\hat{H}_{\text{var}} = \hat{H}_t + \hat{H}_{t_\perp}^{(\text{var})} + \hat{H}_\Delta + \hat{H}_{\Delta_\perp} \tag{5.13}$$

with
$$\hat{H}_\Delta = \Delta \sum_{\vec{r},l} \hat{c}^\dagger_{\vec{r},l,\uparrow}\Big(\hat{c}^\dagger_{\vec{r}+\vec{e}_x,l,\downarrow} + \hat{c}^\dagger_{\vec{r}-\vec{e}_x,l,\downarrow} - \hat{c}^\dagger_{\vec{r}+\vec{e}_y,l,\downarrow} - \hat{c}^\dagger_{\vec{r}-\vec{e}_y,l,\downarrow}\Big) + \text{h.c.} \tag{5.14}$$

$$\hat{H}_{\Delta_\perp} = \Delta_\perp \sum_{\vec{r}} \Big(\hat{c}^\dagger_{\vec{r},1,\uparrow}\hat{c}^\dagger_{\vec{r},2,\downarrow} + \hat{c}^\dagger_{\vec{r},2,\uparrow}\hat{c}^\dagger_{\vec{r},1,\downarrow} + \text{h.c.}\Big) \tag{5.15}$$

We have already treated the two hopping terms in the last section where we diagonalized the original Hubbard Hamiltonian for $U = 0$. The only thing that is different now is that we can no longer drop the spin index due to the fact that the pairing terms mix the two spins, and we get a minus sign in front of the spin down hopping because of the particle-hole transformation, see subsection 3.1.1. Applying the particle-hole transformation from equation (3.9) and (3.10) to the individual terms and then Fourier transforming them gives the following results.

$$\hat{H}_t = -2t \sum_{\vec{k},l} \big(\cos(k_x) + \cos(k_y)\big)\Big(\hat{d}^\dagger_{\vec{k},l,\uparrow}\hat{d}_{\vec{k},l,\uparrow} - \hat{d}^\dagger_{\vec{k},l,\downarrow}\hat{d}_{\vec{k},l,\downarrow}\Big) \tag{5.16}$$

$$\hat{H}_{t_\perp} = -t_\perp^{(\text{var})} \sum_{\vec{k}} \Big(\hat{d}^\dagger_{\vec{k},2,\uparrow}\hat{d}_{\vec{k},1,\uparrow} - \hat{d}^\dagger_{\vec{k},1,\downarrow}\hat{d}_{\vec{k},2,\downarrow} + \text{h.c.}\Big) \tag{5.17}$$

$$\hat{H}_\Delta = 2\Delta \sum_{\vec{k},l} \big(\cos(k_x) - \cos(k_y)\big)\Big(\hat{d}^\dagger_{\vec{k},l,\uparrow}\hat{d}_{\vec{k},l,\downarrow} + \text{h.c.}\Big) \tag{5.18}$$

$$\hat{H}_{\Delta_\perp} = \Delta_\perp \sum_{\vec{k}} \Big(\hat{d}^\dagger_{\vec{k},2,\uparrow}\hat{d}_{\vec{k},1,\downarrow} + \hat{d}^\dagger_{\vec{k},1,\uparrow}\hat{d}_{\vec{k},2,\downarrow} + \text{h.c.}\Big) \tag{5.19}$$

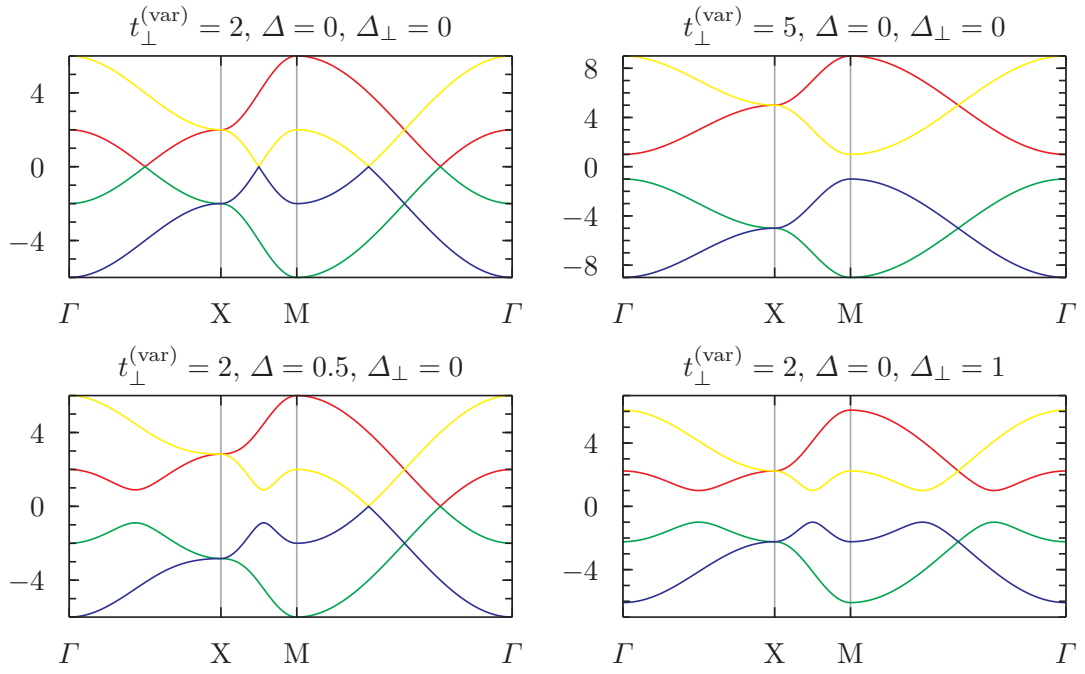

Figure 5.5: Band structure of the Slater determinant for different values of the variational parameters. The symmetry points are $\Gamma \equiv (0,0)$, $X \equiv (\pi,0)$ and $M \equiv (\pi,\pi)$. The blue and green band are occupied, and a gap opens for $t_\perp^{(\mathrm{var})} > 4$ or $\Delta_\perp \neq 0$. As expected, the in-plane pairing Δ does not open a gap in the d-wave's nodal direction $\Gamma \to M$.

This can again be written in matrix form for every value of $\vec{k}$, where we use the abbreviation $\mathrm{c}_i \equiv \cos(k_i)$.

$$\hat{H} = \sum_{\vec{k}} \begin{pmatrix} \hat{d}^\dagger_{\vec{k},1,\uparrow} \\ \hat{d}^\dagger_{\vec{k},2,\uparrow} \\ \hat{d}^\dagger_{\vec{k},1,\downarrow} \\ \hat{d}^\dagger_{\vec{k},2,\downarrow} \end{pmatrix}^T \begin{pmatrix} -2t(\mathrm{c}_x + \mathrm{c}_y) & -t_\perp^{(\mathrm{var})} & 2\Delta(\mathrm{c}_x - \mathrm{c}_y) & \Delta_\perp \\ -t_\perp^{(\mathrm{var})} & -2t(\mathrm{c}_x + \mathrm{c}_y) & \Delta_\perp & 2\Delta(\mathrm{c}_x - \mathrm{c}_y) \\ 2\Delta(\mathrm{c}_x - \mathrm{c}_y) & \Delta_\perp & 2t(\mathrm{c}_x + \mathrm{c}_y) & t_\perp^{(\mathrm{var})} \\ \Delta_\perp & 2\Delta(\mathrm{c}_x - \mathrm{c}_y) & t_\perp^{(\mathrm{var})} & 2t(\mathrm{c}_x + \mathrm{c}_y) \end{pmatrix} \begin{pmatrix} \hat{d}_{\vec{k},1,\uparrow} \\ \hat{d}_{\vec{k},2,\uparrow} \\ \hat{d}_{\vec{k},1,\downarrow} \\ \hat{d}_{\vec{k},2,\downarrow} \end{pmatrix} \tag{5.20}$$

This matrix has the following eigenvalues.

$$\epsilon_{\vec{k}}^{(1,3)} = \pm\sqrt{\left(-2t(\cos k_x + \cos k_y) + t_\perp^{(\mathrm{var})}\right)^2 + \left(2\Delta(\cos k_x - \cos k_y) + \Delta_\perp\right)^2} \tag{5.21}$$

$$\epsilon_{\vec{k}}^{(2,4)} = \pm\sqrt{\left(-2t(\cos k_x + \cos k_y) - t_\perp^{(\mathrm{var})}\right)^2 + \left(2\Delta(\cos k_x - \cos k_y) - \Delta_\perp\right)^2} \tag{5.22}$$

Two of the bands have a plus sign in front of the (always positive) square root, while the other two have a minus sign. Ergo, we have two bands with only positive energies and two bands with only negative energies. At half filling we have enough electrons to populate exactly two bands, so that the two negative bands are always completely full and the two positive bands entirely empty. Consequently, the only way not to have a bandgap is by having the bands touch each other at zero energy. The easiest way to understand the influence of the different parameters is to look at figure 5.5 where the bands are plotted for different values of the variational parameters. We can see that there are two ways of opening a gap in the mean field part of our variational

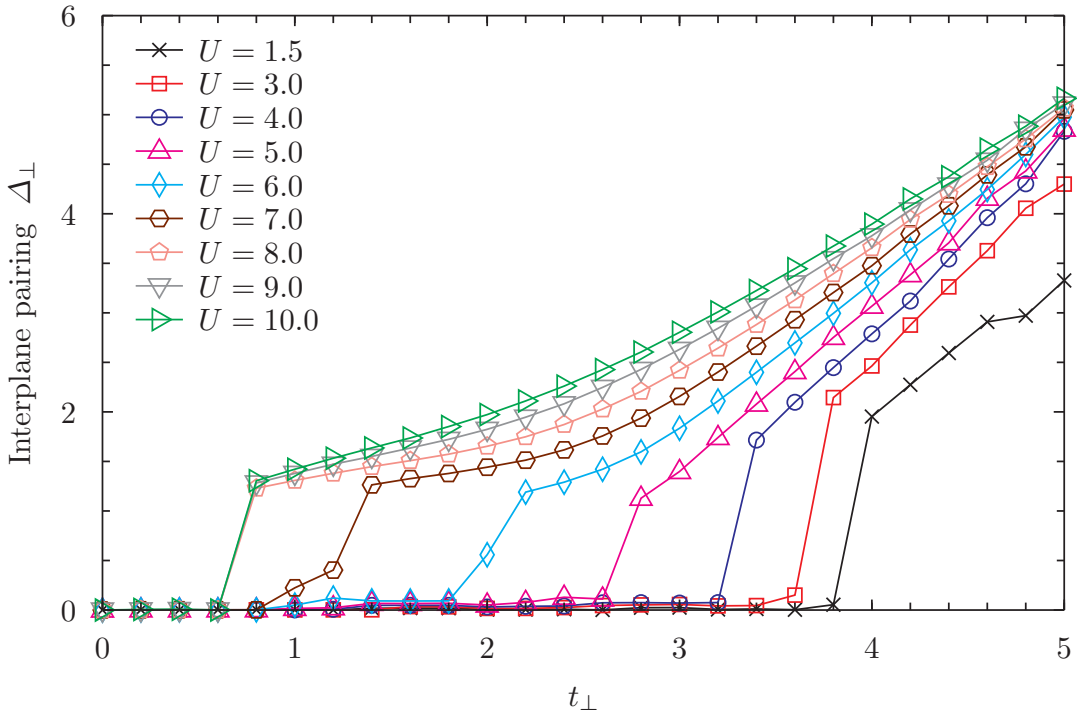

Figure 5.6: The interplane pairing $\Delta_\perp$ as a function of the interplane hopping $t_\perp$.

wavefunction, namely having a $t_\perp^{(\mathrm{var})} > 4$, or a nonzero $\Delta_\perp$ which corresponds to the formation of singlets between the planes. In summary, we have found that we can check if our optimized wavefunction is band insulating by looking for a $t_\perp^{(\mathrm{var})} > 4$ or a $\Delta_\perp \neq 0$ (or both) in the optimized variational parameters.

Let us now have a look at our Variational Monte Carlo simulation results. Figure 5.6 shows the interplane pairing $\Delta_\perp$ as a function of the interplane hopping $t_\perp$. We have seen that any nonzero $\Delta_\perp$ opens a gap in the mean field part of the wavefunction and therefore indicates a band insulator. Starting with $t_\perp = 4$ at $U = 0$ the region with a nonzero $\Delta_\perp$ extends to lower $t_\perp$ as U is increased. For $U \geq 8$ the jump is at a constant $t_\perp \approx 0.7$.

Figure 5.7 shows the intraplane Mott gap calculated according to section 3.4 as a function of the interplane hopping $t_\perp$. A measurable Mott gap is found for $U \geq 6$ and increases as the Coulomb repulsion U is increased. Note that for $U \geq 8$ the maximum $t_\perp$ for which a Mott gap is found at a constant $t_\perp \approx 0.7$. An in-plane Δ is only found in the region with a finite Mott gap, indicating that the pairing is to be understood in terms of the Resonating Valence Bond theory.

Figure 5.8 shows the renormalization of the variational interplane $t_\perp^{(\mathrm{var})}$ hopping. In general the variational $t_\perp^{(\mathrm{var})}$ is not too different from the $t_\perp$ in the original Hubbard Hamiltonian, but there are two exceptions to this rule. The first one is that in the Mott insulating phase at large U and small $t_\perp$ the variational $t_\perp^{(\mathrm{var})}$ is renormalized to quite small values. Together with the lack of a $\Delta_\perp$ in this region, this indicates that the mean field part of the wavefunction is almost the one of two decoupled Hubbard planes. The other exception is that at small U and large $t_\perp$ the variational $t_\perp^{(\mathrm{var})}$ is renormalized to values larger than the original $t_\perp$. This should probably not be overinterpreted as the

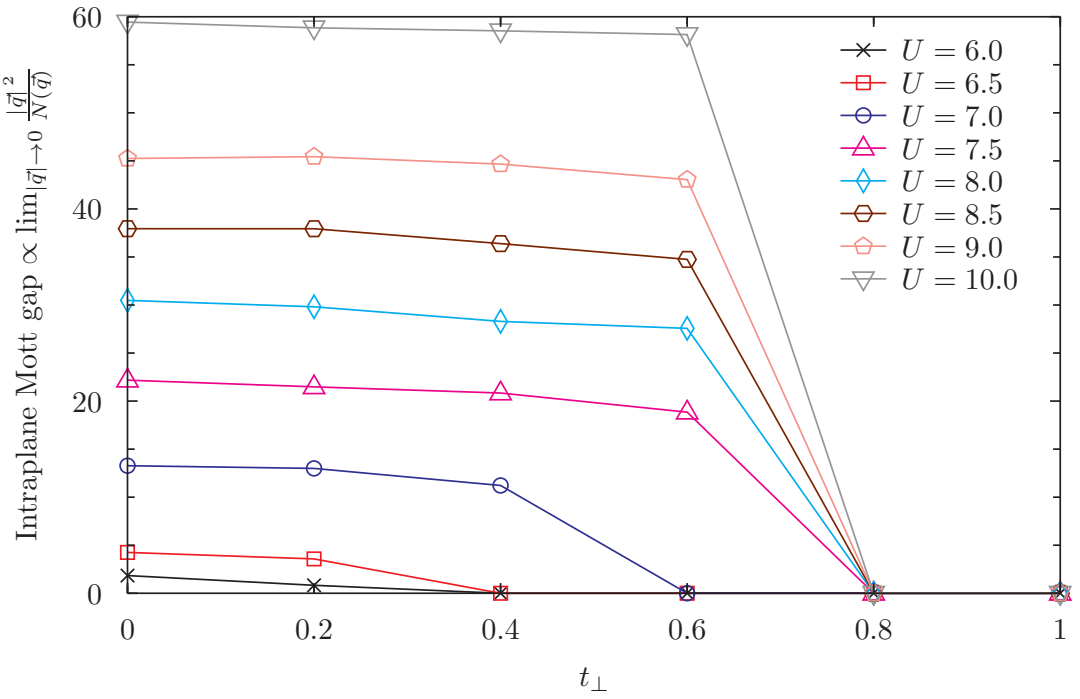

Figure 5.7: Intraplane Mott gap as a function of the interplane hopping $t_\perp$.

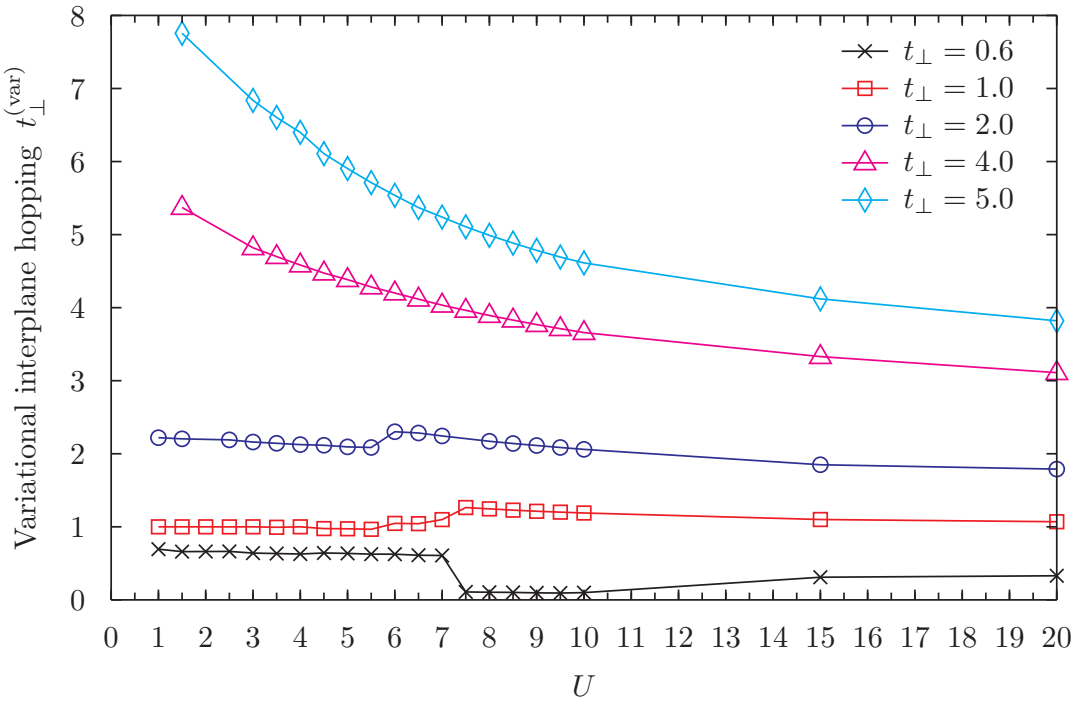

Figure 5.8: The variational interplane hopping $t_\perp^{(\mathrm{var})}$ as a function of the the Coulomb repulsion U for different values of the interplane hopping $t_\perp$ in the original Hubbard Hamiltonian.

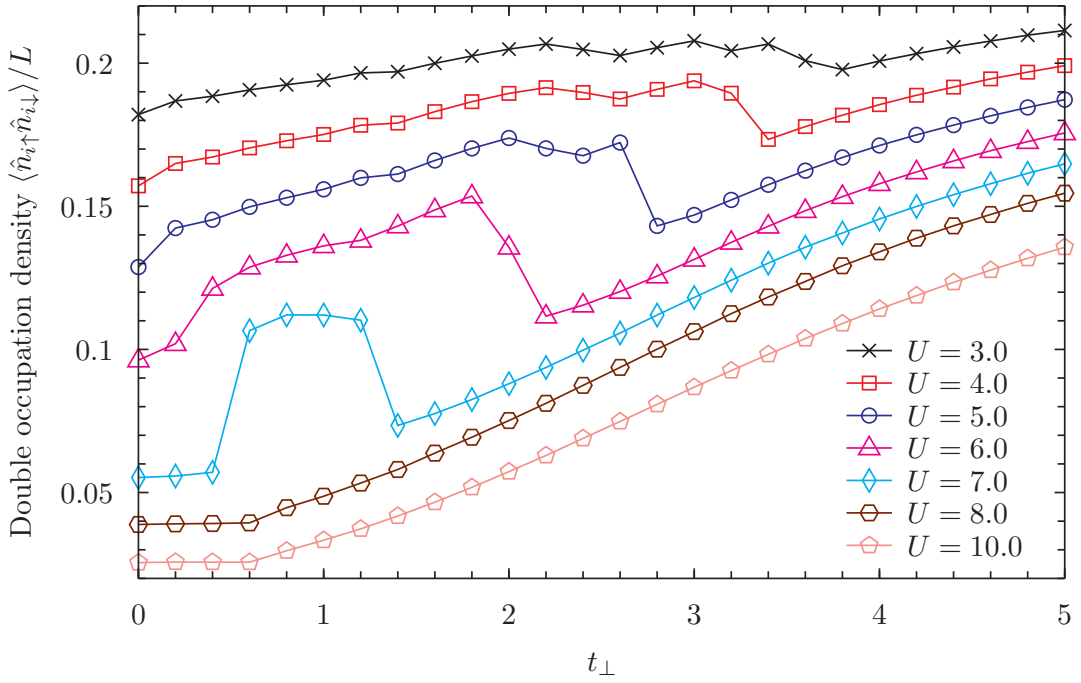

Figure 5.9: The density of double occupancies as a function of the interplane hopping $t_\perp$.

wavefunction's energy is only extremely weakly dependent on the actual value of $t_\perp^{(\mathrm{var})}$ in this region, and the same energy can be obtained within the statistical uncertainty if the $t_\perp^{(\mathrm{var})}$ is fixed to any value $t_\perp^{(\mathrm{var})} \geq 4$. The reason for this is most easily seen in the $U = 0$ limit where energy is independent from $t_\perp^{(\mathrm{var})}$ if $t_\perp^{(\mathrm{var})} \geq 4$. Increasing $t_\perp^{(\mathrm{var})}$ beyond four only shifts the already disconnected bands further apart, which does not change the occupied $\vec{k}$-points used to build the determinantal part of the wavefunction, and hence does not change the wavefunction at all.

The in-plane Mott gap and the band gap opening $\Delta_\perp$ already map out the regions of Mott and band insulation in the phase diagram, which is confirmed by looking at the system's double occupancy. Figure 5.9 shows the density of double occupancies as a function of the interplane hopping $t_\perp$. For intermediate values of U the double occupancy first rises sharply and then drops off abruptly again as the $t_\perp$ is increased, signaling that there is a metallic phase in between the Mott and band insulating phases for $5.5 \lesssim U \lesssim 7.5$. The double occupancy was also calculated by Kancharla and Okamoto, who have, however, found that the double occupancy *decreases* in the metallic phase. This is a rather puzzling result as the essence of a Mott insulator is that the suppression of double occupancies produces an insulating behavior, but no explanation was given by the authors.

At large $U \gtrsim 8$ we have seen that the Mott insulator goes directly into the band insulator at $t_\perp \approx 0.7$. The Mott insulating wavefunction is characterized by an in-plane $\Delta > 0$ and a $\Delta_\perp = 0$, while the band insulating wavefunction has a sizeable $\Delta_\perp$ but no in-plane Δ. A strong hysteresis in these variational parameters was observed when going through the transition, so that both a Mott and a band insulator could be obtained for $t_\perp$ close to the transition. The correct wavefunction is the one with the lower energy, which is plotted in figure 5.10. Finding hysteresis means that both the

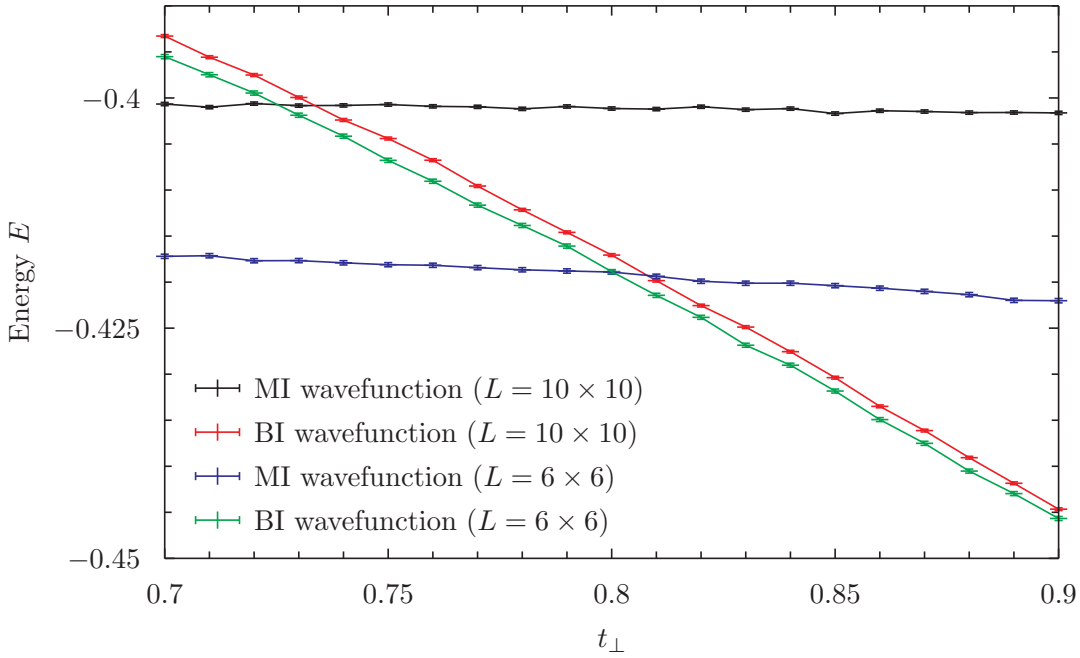

Figure 5.10: The energy of both Mott and band insulating wavefunctions for different value of $t_\perp$ close to the Mott to band insulator transition. Note that increasing the system size from 36 to 100 sites per plane slightly decreases the critical $t_\perp$ from 0.8 to 0.73.

Mott and the band insulating wavefunctions are local energy minima in our variational parameter space. This strongly suggests the Mott to band insulator transition to be of first order, though a continuous transition can not be ruled out based on our variational approach as it might be the variational ansatz that does not allow for a smooth transition between the two phases.

5.2.2 Magnetic phase diagram

The simulations through which we obtained the magnetic phase diagram were performed in exactly the same way as for the nonmagnetic case, only that the site and spin dependent chemical potential μ_m was no longer fixed to zero during the optimization.

Figure 5.11 shows the magnetic potential μ_m as a function of the interplane hopping $t_\perp$. For small U an antiferromagnetic order is only found for $t_\perp < 4$, which makes sense as the ordering is driven by the perfect nesting of the bonding and antibonding bands' Fermi surfaces, which no longer exist if the bonding band is completely full and the antibonding band completely empty at $t_\perp > 4$. Increasing the U pushes the antiferromagnetic region to smaller $t_\perp$, indicating that the critical interplane hopping indeed goes to the Heisenberg limit of $t_\perp = 1.588$ for $U \to \infty$.

In the following we would like to show that having a nonzero μ_m opens a gap in the mean field part of the wavefunction and therefore makes the system an insulator. The calculation here is geometrically more involved as we now have four sites in our unit cell, that we would like to label from 1 to 4. The unit cell and the lattice vectors $\vec{a}$ are shown in figure 5.13.

Writing down a Hamiltonian for the hopping and the magnetism, we get the following

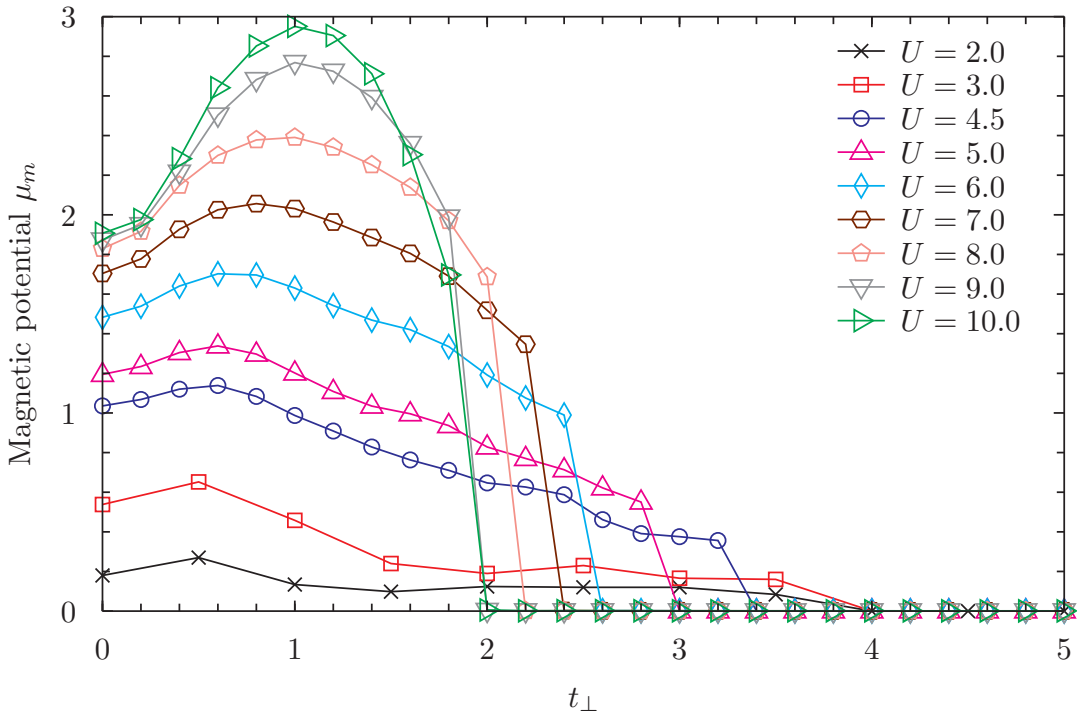

Figure 5.11: The magnetic potential μ_m as a function of the interplane hopping $t_\perp$.

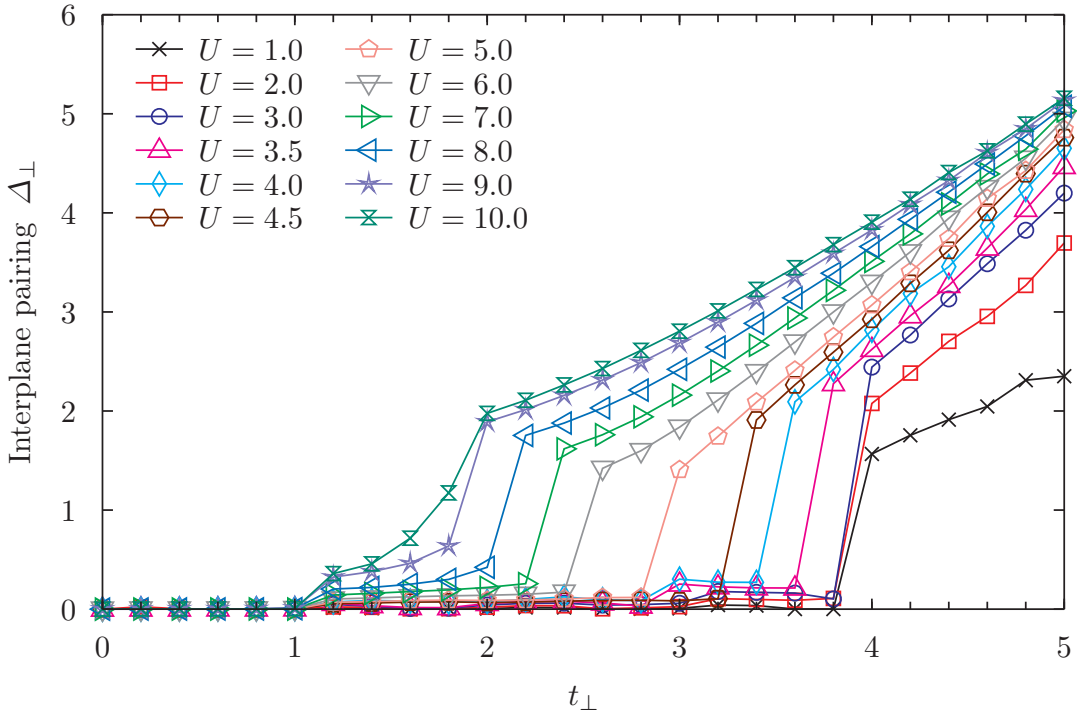

Figure 5.12: The interplane pairing $\Delta_\perp$ as a function of the interplane hopping $t_\perp$ in the magnetic case.

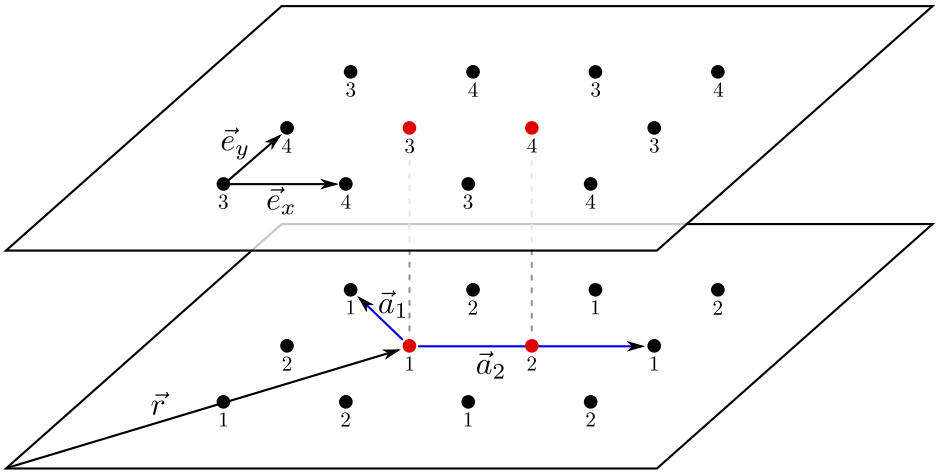

Figure 5.13: Four atom unit cell of the square lattice bilayer. The lattice vectors are shown as $\vec{a}_1 = -\vec{e}_x + \vec{e}_y$ and $\vec{a}_2 = 2\vec{e}_x$.

terms, where $\hat{H}_{t,l}$ is the term for hopping in the l-th plane. Note that the operator $\hat{c}^\dagger_{\vec{r},2,\sigma}$ creates a spin-σ electron on site 2 in the unit cell at $\vec{r}$, where the vector $\vec{r}$ is the position of the unit cell and *not* the position of the site 2, which is at $\vec{r} + \vec{e}_x$.

$$\hat{H}_{\text{var}} = \sum_l \hat{H}_{t,l} + \hat{H}^{(\text{var})}_{t_\perp} + \hat{H}_{\text{mag}} \tag{5.23}$$

$$\begin{aligned}\hat{H}_{t,1} = -t \sum_{\vec{r},\sigma} \Big[&\left(\hat{c}^\dagger_{\vec{r}-2\vec{e}_x,2,\sigma} + \hat{c}^\dagger_{\vec{r}-\vec{e}_x+\vec{e}_y,2,\sigma} + \hat{c}^\dagger_{\vec{r}-\vec{e}_x-\vec{e}_y,2,\sigma} + \hat{c}^\dagger_{\vec{r},2,\sigma}\right) \hat{c}_{\vec{r},1,\sigma} \\ &+ \left(\hat{c}^\dagger_{\vec{r},1,\sigma} + \hat{c}^\dagger_{\vec{r}+\vec{e}_x+\vec{e}_y,1,\sigma} + \hat{c}^\dagger_{\vec{r}+\vec{e}_x-\vec{e}_y,1,\sigma} + \hat{c}^\dagger_{\vec{r}+2\vec{e}_x,1,\sigma}\right) \hat{c}_{\vec{r},2,\sigma} \Big]\end{aligned} \tag{5.24}$$

$$\hat{H}^{(\text{var})}_{t_\perp} = -t^{(\text{var})}_\perp \sum_{\vec{r},\sigma} \left(\hat{c}^\dagger_{\vec{r},3,\sigma}\hat{c}_{\vec{r},1,\sigma} + \hat{c}^\dagger_{\vec{r},4,\sigma}\hat{c}_{\vec{r},2,\sigma} + \text{h.c.}\right) \tag{5.25}$$

$$\begin{aligned}\hat{H}_{\text{mag}} = -\mu_m \sum_{\vec{r}} \Big[&\left(\hat{c}^\dagger_{\vec{r},1,\uparrow}\hat{c}_{\vec{r},1,\uparrow} - \hat{c}^\dagger_{\vec{r},1,\downarrow}\hat{c}_{\vec{r},1,\downarrow}\right) - \left(\hat{c}^\dagger_{\vec{r},2,\uparrow}\hat{c}_{\vec{r},2,\uparrow} - \hat{c}^\dagger_{\vec{r},2,\downarrow}\hat{c}_{\vec{r},2,\downarrow}\right) \\ &- \left(\hat{c}^\dagger_{\vec{r},3,\uparrow}\hat{c}_{\vec{r},3,\uparrow} - \hat{c}^\dagger_{\vec{r},3,\downarrow}\hat{c}_{\vec{r},3,\downarrow}\right) + \left(\hat{c}^\dagger_{\vec{r},4,\uparrow}\hat{c}_{\vec{r},4,\uparrow} + \hat{c}^\dagger_{\vec{r},4,\downarrow}\hat{c}_{\vec{r},4,\downarrow}\right) \Big]\end{aligned} \tag{5.26}$$

Note that we have only written out $\hat{H}_{t,1}$ explicitly, as $\hat{H}_{t,2}$ can be obtained by simply changing the indices. The Hamiltonian is a sum over spins, so let us drop the spin index and substitute the following Fourier transformed operators.

$$\hat{c}_{\vec{r},l} = \begin{cases} \frac{1}{\sqrt{L/4}} \sum_{\vec{k}} \mathrm{e}^{\mathrm{i}\vec{k}\cdot\vec{r}} \hat{c}_{\vec{k},l} & \text{if } l \in \{1,3\} \\ \frac{1}{\sqrt{L/4}} \sum_{\vec{k}} \mathrm{e}^{\mathrm{i}\vec{k}\cdot(\vec{r}+\vec{e}_x)} \hat{c}_{\vec{k},l} & \text{if } l \in \{2,4\} \end{cases} \tag{5.27}$$

The distinction is necessary because site 2 and 4 are not at the origin of the unit cell.

$$\implies \qquad \hat{H}^1_t = -2t \sum_{\vec{k}} \left(\cos(k_x) + \cos(k_y)\right) \left(\hat{c}^\dagger_{\vec{k},2}\hat{c}_{\vec{k},1} + \hat{c}^\dagger_{\vec{k},1}\hat{c}_{\vec{k},2}\right) \tag{5.28}$$

$$\hat{H}^{(\text{var})}_{t_\perp} = -t^{(\text{var})}_\perp \sum_{\vec{k}} \left(\hat{c}^\dagger_{\vec{k},3}\hat{c}_{\vec{k},1} + \hat{c}^\dagger_{\vec{k},4}\hat{c}_{\vec{k},2} + \text{h.c.}\right) \tag{5.29}$$

$$\hat{H}_{\text{mag}} = -\mu_m \sum_{\vec{k}} \left(\hat{c}^\dagger_{\vec{k},1}\hat{c}_{\vec{k},1} - \hat{c}^\dagger_{\vec{k},2}\hat{c}_{\vec{k},2} - \hat{c}^\dagger_{\vec{k},3}\hat{c}_{\vec{k},3} + \hat{c}^\dagger_{\vec{k},4}\hat{c}_{\vec{k},4}\right) \tag{5.30}$$

Adding the second plane again and writing everything in matrix notation gives the following equation.

$$\hat{H} = \sum_{\vec{k}} \begin{pmatrix} \hat{c}^{\dagger}_{\vec{k},1} \\ \hat{c}^{\dagger}_{\vec{k},2} \\ \hat{c}^{\dagger}_{\vec{k},3} \\ \hat{c}^{\dagger}_{\vec{k},4} \end{pmatrix}^{T} \begin{pmatrix} -\mu_m & -2t(\mathrm{c}_x + \mathrm{c}_y) & -t_{\perp}^{(\mathrm{var})} & 0 \\ -2t(\mathrm{c}_x + \mathrm{c}_y) & \mu_m & 0 & -t_{\perp}^{(\mathrm{var})} \\ -t_{\perp}^{(\mathrm{var})} & 0 & \mu_m & -2t(\mathrm{c}_x + \mathrm{c}_y) \\ 0 & -t_{\perp}^{(\mathrm{var})} & -2t(\mathrm{c}_x + \mathrm{c}_y) & -\mu_m \end{pmatrix} \begin{pmatrix} \hat{c}_{\vec{k},1} \\ \hat{c}_{\vec{k},2} \\ \hat{c}_{\vec{k},3} \\ \hat{c}_{\vec{k},4} \end{pmatrix} \tag{5.31}$$

Diagonalizing the above matrix one obtains the following four bands.

$$\epsilon_{\vec{k}}^{(1,3)} = \pm\sqrt{\big(-2t(\cos k_x + \cos k_y) + t_{\perp}\big)^2 + \mu_m^2} \tag{5.32}$$

$$\epsilon_{\vec{k}}^{(2,4)} = \pm\sqrt{\big(-2t(\cos k_x + \cos k_y) - t_{\perp}\big)^2 + \mu_m^2} \tag{5.33}$$

We can now make the exact same argument as in the nonmagnetic case, namely that the two negative bands are filled and the system can only be conducting if they touch the empty bands at zero energy. Looking at the equations for the bands, it is easy to see that this can not happen for $\mu_m \neq 0$, and hence we can use a nonzero μ_m as a criterion for an insulating state.

Looking at the interplane $\Delta_{\perp}$ in figure 5.12 and comparing it with the plot of the magnetic potential μ_m in figure 5.11, we see that a nonzero $\Delta_{\perp}$ is found in the entire paramagnetic region. Both the μ_m and the $\Delta_{\perp}$ open a gap in the mean field part of the wavefunction, so that the square lattice bilayer Hubbard model is *always* an insulator at $U > 0$ if magnetic order is allowed. Note that in the phase diagram we classify the ordered phase as a Mott insulator, even though we have gap in the mean field state. This is due to the fact that the antiferromagnetic ordering is correlation induced, which also makes the insulating behavior an effect of electron correlation.

It is interesting to note that at large $U \gtrsim 8$ and $1 < t_{\perp} \lesssim 2$ there is a region where we find both a nonzero magnetic potential μ_m and an interplane pairing $\Delta_{\perp}$. This together with the fact that we did not see any hysteresis in the variational parameters at the order-disorder transition suggests that the transition is continuous.

Comparing our phase diagrams to the ones obtained with DMFT [FHM06] and Cluster DMFT [KO07], we see that our nonmagnetic phase diagram is somewhere in between the DMFT and CDMFT results: While we agree with Kancharla and Okamoto that there is a region in which the system first goes from a Mott insulator to a metal and then to a band insulator as the $t_{\perp}$ is increased, we do not find that this region extends down to $U = 0$. Instead for the decoupled planes we find a metal to Mott insulator transition at a critical $U \approx 5.5$, which agrees with the DMFT results by Fuhrmann, Heilmann and Monien, and the DCA results for the single layer by Gull, Parcollet and Millis [GPM13]. For large U our results agree with those of Kancharla and Okamoto in that there is a direct transition from a Mott to a band insulator, but our critical $t_{\perp}$ is smaller by about a factor of three. The reason for this might be the cluster with two sites in each plane used by Kancharla and Okamoto that breaks the fourfold rotational symmetry of the square lattice and creates an artificially enhanced local pair within each plane, ultimately stabilizing the in-plane Mott phase against the interplane dimers of the band insulating phase.

The magnetic phase diagram we obtained is quite different from the one published by Kancharla and Okamoto. The most obvious difference is that we no longer find a metallic phase if magnetic ordering is allowed. Instead we find a Néel ordered Mott insulator, which we attribute to the perfect nesting between the bonding and antibonding band's Fermi surfaces. The reason why Variational Monte Carlo stabilizes magnetic ordering compared to Cluster DMFT with two sites per plane might be the much more explicit treatment of long range correlations.

Overall, it can be said that the Variational Monte Carlo results certainly improve our understanding of the bilayer Hubbard model, but that further investigation is necessary to clarify the sizeable differences between the VMC and DMFT results.

6 Summary and conclusion

The purpose of this chapter is to summarize the most important points of the thesis.

In the introduction we have presented the Hubbard model at half filling as a model for systems where conventional band theory fails, namely the Mott insulators. From the variety of numerical methods that are available to treat the Hubbard model, we have chosen Variational Monte Carlo (VMC), a variational method that relies on the Rayleigh-Ritz principle to approximate the ground state through a variational many-body wavefunction and which uses Monte Carlo integration to evaluate sums over a high dimensional configuration space.

Chapter 2 has described the VMC method in general. Having stated the fundamental idea, we have then taken an in-depth look at how a Markov chain can be used to implement an importance sampling in configuration space, and have derived the famous Metropolis algorithm. We have also presented binning as a method of calculating the uncertainties in the results in spite of the fact that there is correlation between adjacent configurations of the Markov chain. VMC relies on finding the variational parameters that minimize the expectation value of the wavefunction's energy, to which purpose we have presented the Stochastic Reconfiguration optimization, an iterative scheme based on the power method. The fact that the convergence to the optimal variational parameters is noisy within a Monte Carlo method makes detecting actual convergence nontrivial, and we have chosen to use a Mann-Kendall test on the evolution of the variational parameters as a convergence criterion.

Chapter 3 has explored how the VMC method can be applied to the Hubbard model. We have presented a wavefunction consisting of a Slater determinant and a correlator. The Slater determinant was constructed from the eigenstates of a variational single particle Hamiltonian which itself was obtained from the Hubbard Hamiltonian by dropping the interaction term and adding a number of terms modeled after correlation induced phenomena. In order to deal with the pairing term in the variational Hamiltonian we had to apply a canonical transformation called the particle-hole transformation. As a correlator we have used the so called Jastrow factor, a long range correlator that basically acts as an attractive force between holons and doublons, binding them together and making the wavefunction Mott insulating. The logarithmic derivatives of this wavefunction with respect to the variational parameters, as needed by the Stochastic Reconfiguration optimizer, have been calculated either directly in case of the Jastrow factor, or using first order perturbation theory in case of the Slater determinant. We have then identified the computationally most demanding part of the algorithm as the evaluation of overlap ratios $\langle x'|\Psi\rangle \,/\, \langle x|\Psi\rangle$, where the particle configurations x and x' only differ by the displacement of a single electron. By using the relation between x and x' we have been able to reduce the computational complexity of this step by one power, and in case of the Slater determinant also reformulated everything in terms of linear algebra operations that can easily be offloaded to highly optimized libraries.

Finally we have shown how one can use a single mode approximation based ansatz for the excited state to detect a Mott insulating wavefunction through measurements of the density-density correlation.

In chapter 4 we have looked at how the concepts from the previous two chapters are implemented in `hVMC`, a free Variational Monte Carlo code that was written from scratch in C++ as a part of this thesis.

Chapter 5 has investigated the square lattice bilayer Hubbard model that has recently been discussed as a model for a Mott to band insulator transition. We have first looked at the known edge cases and have then performed VMC simulations to calculate both a nonmagnetic and a magnetic phase diagram of the system. In the nonmagnetic case and at intermediate values of U we have found that the system undergoes a Mott insulator to metal and then a metal to band insulator transition as the $t_\perp$ is increased. At large values of U there is a direct Mott to band insulator transition, which appears to be of first order. For the magnetic phase diagram we find a Néel ordered antiferromagnetic Mott insulator at small $t_\perp$ due to the perfect nesting of the Fermi surfaces of the bonding and antibonding bands, while a paramagnetic band insulator was found for large $t_\perp$. The bilayer Hubbard model's phase diagrams had previously been calculated by Kancharla and Okamoto [KO07] using Cluster Dynamical Mean Field Theory, and while our VMC results appear to be self-consistent and correctly reproduce the known edge cases, we have found some substantial differences with respect to the CDMFT results, most prominently the absence of a metallic phase in the magnetic phase diagram.

In summary we have demonstrated how the Variational Monte Carlo method can be used to simulate the Hubbard model, taking the entire road from theory through implementation to application. For the two layer system we have found some discrepancies between our VMC results and the results others have obtained earlier using DMFT, and while our results certainly improve our understanding of the bilayer Hubbard model, this only highlights the fact that strongly correlated electrons are still hard, and none of the present methods can be considered the last word on the subject.

On a personal note I have to say that I enjoyed working on a thesis that had such a big software development component. Implementing `hVMC` took a lot of time and effort, but as a result I not only gained more knowledge about C++ and software development in general, but also learned to solve a multitude of little problems I ran into along the way: I surely would not have thought about how to detect the convergence of a noisy time series and would not have discovered the Mann-Kendall test if I had not written my own code. Spending so much time programming of course meant that not much time was left to investigate physical problems, but I think this trade-off was well worth it in my case. I myself will not have the opportunity to work with Variational Monte Carlo in the foreseeable future, but would be delighted to see a future student from the group of Professor Valentí pick up where I left off. I hope the `hVMC` code proves useful and that this thesis gives him/her a fighting chance in using, understanding and expanding it.

Acknowledgments

I would like thank the following people for their contributions to this thesis.

- Prof. Dr. Roser Valentí for giving me the opportunity to work on this exciting project, and for bearing with me through the long time I spent developing and tweaking `hVMC`. Meeting her was always encouraging and motivating, and her insightful comments and suggestions contributed a lot to this thesis.
- Dr. Luca Tocchio for many interesting discussions and his continued support during the time I was working on this thesis. He constantly tested `hVMC` against his own implementation, which was a huge help during development.
- Prof. Dr. Claudius Gros and Dr. Federico Becca for discussions that offered some additional insight into the VMC method.
- Matthias Bach for contributing his GPGPU expertise to the development of an OpenCL based version of `hVMC`, and Nvidia for donating a GPU on which it was tested.
- Prof. Dr. Marc Wagner, Prof. Dr. Eberhard Engel and Prof. Dr. Volker Lindenstruth for their amazing lectures on computational physics and high performance computing.
- The developers and supporters of free and open source software all around the globe without whose relentless effort this thesis simply would not have been possible.
- The entire condensed matter theory group for many interesting (and at times hilarious) lunch conversations, and especially my office mates Steffen Backes, Daniel Barragan Yani and Jean Diehl for quick sanity checks of my ideas, and for bringing delicious food from all over the world to our office.
- My girlfriend for her sympathetic handling of any thesis-induced mood swings, and my family for their loving lifelong support.

Bibliography

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[And87] P. W. Anderson. The Resonating Valence Bond State in La_2CuO_4 and Superconductivity. *Science*, 235(4793):1196–1198, 1987.

[Att05] Claudio Attaccalite. *RVB phase of hydrogen at high pressure: towards the first ab-initio Molecular Dynamics by Quantum Monte Carlo*. PhD thesis, SISSA/ISAS Trieste Italy, 2005.

[BA04] C. D. Batista and A. A. Aligia. Exact Bond Ordered Ground State for the Transition between the Band and the Mott Insulator. *Phys. Rev. Lett.*, 92:246405, Jun 2004.

[BCL] Boost C++ libraries. `http://www.boost.org/`.

[BK92] M.A. Baranov and M.Yu. Kagan. D-wave pairing in the two-dimensional Hubbard model with low filling. *Zeitschrift für Physik B Condensed Matter*, 86(2):237–239, 1992.

[CA76] Charles L. Cleveland and Rodrigo Medina A. Obtaining a Heisenberg Hamiltonian from the Hubbard model. *American Journal of Physics*, 44(1):44–46, 1976.

[Cap06] Manuela Capello. *Variational description of Mott insulators*. PhD thesis, SISSA/ISAS Trieste Italy, 2006.

[CAS04] Michele Casula, Claudio Attaccalite, and Sandro Sorella. Correlated geminal wave function for molecules: An efficient resonating valence bond approach. *The Journal of Chemical Physics*, 121(15):7110–7126, 2004.

[CBF+05] Manuela Capello, Federico Becca, Michele Fabrizio, Sandro Sorella, and Erio Tosatti. Variational Description of Mott Insulators. *Phys. Rev. Lett.*, 94:026406, January 2005.

[CCK77] D. Ceperley, G. V. Chester, and M. H. Kalos. Monte Carlo simulation of a many-fermion study. *Phys. Rev. B*, 16:3081–3099, Oct 1977.

[Czy07] G. Czycholl. *Theoretische Festkörperphysik: Von den klassischen Modellen zu modernen Forschungsthemen*. Springer-Lehrbuch. Springer, 2007.

[DT12] A. Druinsky and S. Toledo. How Accurate is inv(a)*b? arXiv:1201.6035v1 [cs.NA], January 2012.

[EN07] Satoshi Ejima and Satoshi Nishimoto. Phase Diagram of the One-Dimensional Half-Filled Extended Hubbard Model. *Phys. Rev. Lett.*, 99:216403, Nov 2007.

[Fey54] R. P. Feynman. Atomic Theory of the Two-Fluid Model of Liquid Helium. *Phys. Rev.*, 94:262–277, April 1954.

[FHM06] Andreas Fuhrmann, David Heilmann, and Hartmut Monien. From Mott insulator to band insulator: A dynamical mean-field theory study. *Phys. Rev. B*, 73:245118, Jun 2006.

[GFB+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010. `http://eigen.tuxfamily.org/`.

[GKKR96] Antoine Georges, Gabriel Kotliar, Werner Krauth, and Marcelo J. Rozenberg. Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions. *Rev. Mod. Phys.*, 68:13–125, Jan 1996.

[GMP86] S. M. Girvin, A. H. MacDonald, and P. M. Platzman. Magneto-roton theory of collective excitations in the fractional quantum Hall effect. *Phys. Rev. B*, 33:2481–2494, Feb 1986.

[GPM13] Emanuel Gull, Olivier Parcollet, and Andrew J. Millis. Superconductivity and the Pseudogap in the Two-Dimensional Hubbard Model. *Phys. Rev. Lett.*, 110:216405, May 2013.

[Gut63] Martin C. Gutzwiller. Effect of Correlation on the Ferromagnetism of Transition Metals. *Phys. Rev. Lett.*, 10:159–162, Mar 1963.

[Häg02] Olle Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 1. edition, 2002.

[Hir85] J. E. Hirsch. Two-dimensional Hubbard model: Numerical simulation study. *Phys. Rev. B*, 31:4403–4419, Apr 1985.

[HKL09] Hafermann, H., Katsnelson, M. I., and Lichtenstein, A. I. Metal-insulator transition by suppression of spin fluctuations. *EPL*, 85(3):37006, 2009.

[Hub63] J. Hubbard. Electron Correlations in Narrow Energy Bands. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 276(1365):238–257, 1963.

[IFT98] Masatoshi Imada, Atsushi Fujimori, and Yoshinori Tokura. Metal-insulator transitions. *Rev. Mod. Phys.*, 70:1039–1263, Oct 1998.

[Kan63] Junjiro Kanamori. Electron Correlation and Ferromagnetism of Transition Metals. *Progress of Theoretical Physics*, 30(3):275–289, 1963.

[KD07] S. S. Kancharla and E. Dagotto. Correlated Insulated Phase Suggests Bond Order between Band and Mott Insulators in Two Dimensions. *Phys. Rev. Lett.*, 98:016402, Jan 2007.

[Ken38] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[KL12] Matthias Kretz and Volker Lindenstruth. Vc: A C++ library for explicit vectorization. *Software: Practice and Experience*, 42(11):1409–1430, 2012.

[KO07] S. S. Kancharla and S. Okamoto. Band insulator to Mott insulator transition in a bilayer Hubbard model. *Phys. Rev. B*, 75:193103, May 2007.

[Koh64] Walter Kohn. Theory of the Insulating State. *Phys. Rev.*, 133:A171–A181, January 1964.

[LNW06] Patrick A. Lee, Naoto Nagaosa, and Xiao-Gang Wen. Doping a Mott insulator: Physics of high-temperature superconductivity. *Rev. Mod. Phys.*, 78:17–85, Jan 2006.

[LW68] Elliott H. Lieb and F. Y. Wu. Absence of Mott Transition in an Exact Solution of the Short-Range, One-Band Model in One Dimension. *Phys. Rev. Lett.*, 20:1445–1448, Jun 1968.

[LZJ+10] Hunpyo Lee, Yu-Zhong Zhang, Harald O. Jeschke, Roser Valentí, and Hartmut Monien. Dynamical Cluster Approximation Study of the Anisotropic Two-Orbital Hubbard Model. *Phys. Rev. Lett.*, 104:026402, Jan 2010.

[Man45] Henry B. Mann. Nonparametric Tests Against Trend. *Econometrica*, 13(3):245–259, 1945.

[Man91] Efstratios Manousakis. The spin-$\frac{1}{2}$ Heisenberg antiferromagnet on a square lattice and its application to the cuprous oxides. *Rev. Mod. Phys.*, 63:1–62, Jan 1991.

[McM65] W. L. McMillan. Ground State of Liquid He^4. *Phys. Rev.*, 138:A442–A451, Apr 1965.

[Mot49] N. F. Mott. The Basis of the Electron Theory of Metals, with Special Reference to the Transition Metals. *Proceedings of the Physical Society. Section A*, 62(7):416, 1949.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

[MW66] N. D. Mermin and H. Wagner. Absence of Ferromagnetism or Antiferromagnetism in One- or Two-Dimensional Isotropic Heisenberg Models. *Phys. Rev. Lett.*, 17:1133–1136, Nov 1966.

[Ove71] A. W. Overhauser. Simplified Theory of Electron Correlations in Metals. *Phys. Rev. B*, 3:1888–1898, Mar 1971.

[Rit09] Walter Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik. *Journal für die reine und angewandte Mathematik*, 135:1–61, 1909.

[SB13] Sandro Sorella and Federico Becca. Lecture notes on numerical methods for strongly correlated electrons. Scuola Internazionale Superiore di Studi Avanzati, 2013.

[Sca06] D. J. Scalapino. Numerical Studies of the 2D Hubbard Model. arXiv:cond-mat/0610710, October 2006.

[Sor01] Sandro Sorella. Generalized Lanczos algorithm for variational quantum Monte Carlo. *Phys. Rev. B*, 64:024512, June 2001.

[SS94] A. W. Sandvik and D. J. Scalapino. Order-disorder transition in a two-layer quantum antiferromagnet. *Phys. Rev. Lett.*, 72:2777–2780, Apr 1994.

[StGDC13] Richard M. Stallman and the GCC Developer Community. *Using the GNU Compiler Collection*. Free Software Foundation, Boston, 2013. `http://gcc.gnu.org/onlinedocs/`.

[Sut05] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. `http://www.gotw.ca/publications/concurrency-ddj.htm`, March 2005.

[TBG11] Luca F. Tocchio, Federico Becca, and Claudius Gros. Backflow correlations in the Hubbard model: An efficient tool for the study of the metal-insulator transition and the large-U limit. *Phys. Rev. B*, 83:195138, May 2011.

[Toc08] Luca F. Tocchio. *A new variational wave function with backflow correlations for frustrated Hubbard models*. PhD thesis, SISSA/ISAS Trieste Italy, 2008.

[WBS06] Ling Wang, K. S. D. Beach, and Anders W. Sandvik. High-precision finite-size scaling analysis of the quantum-critical point of $S = 1/2$ Heisenberg antiferromagnetic bilayers. *Phys. Rev. B*, 73:014431, Jan 2006.

[WD97] R. Clint Whaley and Jack Dongarra. Automatically Tuned Linear Algebra Software. Technical Report UT-CS-97-366, University of Tennessee, December 1997. `http://www.netlib.org/lapack/lawns/lawn131.ps`.

[YS06] Seiji Yunoki and Sandro Sorella. Two spin liquid phases in the spatially anisotropic triangular Heisenberg model. *Phys. Rev. B*, 74:014408, July 2006.

Erklärung

Nach § 28 (12) der Prüfungsordnung für den Bachelor- und Masterstudiengang Physik der Johann-Wolfgang-Goethe-Universität Frankfurt am Main erkläre ich, dass diese Arbeit von mir selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst wurde. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, sind als solche kenntlich gemacht. Weiter erkläre ich, dass die Arbeit nicht – auch nicht auszugsweise – für eine andere Prüfung verwendet worden ist.

Frankfurt am Main, den 19. August 2013

(Robert Rüger)

Appendix

A.1 hVMC quick start guide

Chapter 4 gave an overview of the different modules and classes in the `hVMC` codebase. In this appendix we want to approach the code from a user's perspective and go through the steps of building and running `hVMC`.

A.1.1 Building

`hVMC` is kept in a repository on GitHub and the easiest way to get the latest source code is to clone this repository using `git`.

```
$ git clone https://github.com/robertrueger/hVMC.git && cd hVMC
```

Building `hVMC` should be as easy as invoking `make` in its directory, provided that the following external dependencies are satisfied.

- The GNU Compiler Collection [StGDC13] C++ compiler in version 4.7 or newer. `hVMC` uses features from the new C++11 standard which have only recently been implemented into GCC, but there is a special `git` branch dedicated to making building possible with GCC 4.6. It can be checked out by running the following command.

  ```
  $ git checkout -b gcc46compat remotes/origin/gcc46compat
  ```

  `hVMC` has been successfully built with the Intel C++ compiler and a modified `makefile`, but no speedup in comparison to GCC 4.7 was observed

- An MPI implementation, preferably OpenMPI [GFB+04]. The `makefile` that comes with `hVMC` would need to be modified to use another implementation.

- The Eigen3 linear algebra C++ template library [GJ+10].

- Any CBLAS implementation. Very good performance has been achieved with ATLAS [WD97].

- The Boost C++ libraries [BCL] in version 1.48.0.3 or newer, specifically Boost Filesystem, Boost MPI, Boost ProgramOptions and Boost Optional.

There are three different versions of the `hVMC` executable.

- A debugging build (`make BUILD=DEBUG`) that is optimized for quick compilation and a good debugging experience. Note that due the execution of assertions and the lack of compiler optimization the debugging build is *extremely* slow.

- A profiling build (`make BUILD=PROFILE`) that has most compiler optimizations enabled (except those that interfere with profiling) and also includes debugging symbols. Profiling this build should produce good data on where the computational hotspots are.
- A release build (default, or `BUILD=RELEASE`) with full optimization and no debugging symbols.

Finally there are two options, that can be passed to `make`.

- `make DBLPREC=ENABLED` builds a double precision implementation of the `WMatrix` class. Will most likely do nothing but reduce the performance, and is therefore disabled by default.
- `make CBLAS=DISABLED` disables the linking to an external CBLAS library. Note that disabling also removes the dependency and enables `hVMC` to be built on systems without an installed CBLAS implementation. Only some of `hVMC`'s linear algebra calculations are done by the external CBLAS, while the bulk is performed by the Eigen template library. If the external CBLAS is disabled, Eigen will be used for everything. This is not recommended since it will most likely result in a severe performance penalty. The default is to link against CBLAS.

Simply invoking `make` in the `hVMC` directory will build in release mode with a single precision `WMatrix` implementation and link to an external CBLAS. The options that an `hVMC` executable was built with can be checked by invoking it with the `--version` switch.

A.1.2 Running

Now that we have built `hVMC`, we can have a look at how to use it. `hVMC`'s command line interface is based on the Boost ProgramOptions library, and all available options will be displayed when `hVMC` is launched with the `--help` switch. Options can either be passed directly from the command line or read from a jobfile or both, in which case the command like takes precedence over options from the jobfile.

Let us run a simulation of a 10×10 square lattice Hubbard model with $U/t = 8$ at half-filling with a nonmagnetic wavefunction in order to get to know `hVMC`. The nonmagnetic system is metallic at small U/t and Mott insulating for $U/t \gtrsim 6$, we should therefore be able to see a Mottgap by extrapolating $|\vec{q}|^2 / N(\vec{q})$ to $|\vec{q}| \to 0$. Everything needed to run the above simulation already comes with the `hVMC` code and can be found in the `example` directory.

```
$ cd example
```

We will have to run `hVMC` three times: First we are going to optimize the variational wavefunction, then we are going to run a simulation with this wavefunction, measuring the density-density correlation, and finally we are going to run it a third time to Fourier transform the density-density correlation in order to obtain the static structure factor. Most of the options passed to `hVMC` are the same for all three runs, so in order not to have a ridiculously long command line we are going to write those common options to a jobfile that is read by all three stages.

Let us now have a look at the file `example.job`. Most of the options are fairly self-explanatory and there are some comments in the file, so we only want to make a few remarks here. Note that the example jobfile contains all `hVMC` options, some of which are commented out though and are only present to express our intention to use their default values.

- `hVMC`'s options are grouped into three categories: Physical parameters, calculation settings, and floating point precision control. Depending on what group an option belongs to, it has a different prefix (`phys`, `calc` or `fpctrl`) when specified on the command line, or goes to a different section of the jobfile.

- No matter which lattice is selected, the size of the lattice is always specified by setting the total number of lattice sites, which then might be subject to some constraints depending on the actual selected lattice; e.g. a perfect square for the two-dimensional square lattice.

- There are three ways to set the variational parameters that will be used:

  1. A variational parameter can be set manually by specifying the respective `phys.vpar-ovwrt` option. Note that only the determinantal parameters and the onsite Jastrow can be set in this way.

  2. Variational parameters can be read from the output file of a previous `hVMC` run. Note that `hVMC` will by default look in the simulation's working directory for a file called `opt_vpar_final.dat` if no other file is specified via the `phys.vpar-file` option. Values read from a file can still be overwritten manually using the `phys.vpar-ovwrt` options.

  3. If no variational parameter file is found and a parameter is not specified directly, it will be set to some default value. By default, the variational hopping parameters will be set to the value in the original Hubbard Hamiltonian, all the pairings, the magnetic parameter and all the Jastrows will be set to zero, and the chemical potential will be set to the correct value (see the discussion on page 25).

  For our example, we are setting the chemical potential $\mu = 0$ manually, due to the particle-hole symmetry. We are not specifying a variational parameter file and instead rely on the fact that `hVMC` will automatically look for it in the simulation directory.

- Not all of the options make sense for all of the three modes (`optimization`, `simulation`, `analysis`). For example it does not make sense to select analysis modules via `--calc.analysis` if `hVMC` is running in the `optimization` or `simulation` mode. `hVMC` will ignore any options that are not meaningful in the current mode.

- The `--calc.optimizers` option can be thought of as a bitfield. A power of two is assigned to each determinantal variational parameter and the Jastrow, and the choice which parameters are to be optimized is made by specifying the *sum* of the respective numbers. In our example we want to optimize the Jastrow factor (256) and the nearest neighbor pairing (8) and hence set `--calc.optimizers` to 264.

- The `--calc.vpar-minabs-select` option can be used to prevent the absolute value of determinantal parameters from becoming smaller than a value set with `--calc.vpar-minabs-value`. In the example case this is necessary for the nearest neighbor pairing parameter Δ, because for $\Delta = 0$ the system would be open shell and therefore the choice of the eigenvectors included in the Slater determinant arbitrary. An open shell mean field part can lead to several problems and should generally be avoided. This can either be done by choosing a system size for which it is closed shell, or by imposing a tiny minimum absolute value on some gap opening variational parameter.

- The default values of the options in the `fpctrl` group should generally be a good choice.

Let us now run the optimization of the wavefunction:

```
$ mpirun -n 2 ../hvmc example.job \
      --calc.mode opt --calc.num-bins 100 --phys.vpar-ovwrt-D1 0.01
```

This will run `hVMC` in optimization mode and read the options from the `example.job` file. The number of bins per Monte Carlo cycle is different for the optimization and the simulation stage, so it is not included in the common jobfile. Instead we are setting it here from the command line. The last parameter sets the initial value of the nearest neighbor pairing variational parameter Δ to a small but nonzero value in order to avoid the closed shell problem. The above command runs two `hVMC` processes (master and 1 slave) but can easily be changed to use more processors.

It should take around 300 iterations for the optimization to converge, and its progress can be monitored by running the `plot_opthist.gnu` file in the `example` directory through `gnuplot`.

```
$ gnuplot plot_opthist.gnu
```

This will read the `opt_vpar_hist.txt` and `opt_E_hist.txt` files and plot the values of the variational parameters and the energy against the number of Stochastic Reconfiguration iterations. These files are continuously written to disk during the optimization, so that the script can be run even before the optimization is complete. Note that hitting the enter key in the `gnuplot` terminal will show the next plot.

Once the optimization is complete, `hVMC` will display the final averaged variational parameters and write those to the file `opt_vpar_final.dat`. The final variational parameters should be similar to these.

```
0 0 \
0 0.456486 0 0 \
0 0 \
-3.40792 -1.94301 -1.088 -0.637025 -0.366933 -0.295135 -1.51286 \
-0.92416 -0.560911 -0.328811 -0.263739 -0.627053 -0.393868 -0.229249 \
-0.182451 -0.247297 -0.12915 -0.0930439 -0.0502015 -0.0239823
```

The first two are the variational hopping parameters $t^{(\mathrm{var})}$ between 2nd and 3rd nearest neighbors. There is no such hopping in this example's Hubbard Hamiltonian, so they were initialized to zero and stayed there since we did not optimize them. The next

four parameters are the pairing parameters Δ. We optimized only the pairing between nearest neighbor sites. All the others pairings (on-site, second and third n.n.) were initialized to zero and were not optimized. The chemical potential μ is fixed at zero due to particle-hole symmetry, and the magnetic parameter is also zero because we do not want magnetism in this example. All the others are Jastrow parameters. Note that these are *not* ordered ascending by distance, but ascending by irreducible index relation.

There are two types of output files: `*.txt` and `*.dat`. The former are intended to be human readable and easy to plot, while the latter a generated by the Boost Serialization library and are intended as machine readable input files for consecutive `hVMC` runs. Now that we have the optimized wavefunction we can run the simulation to measure the density-density correlation and the energy.

```
$ mpirun -n 2 ../hvmc example.job --calc.mode sim --calc.num-bins 10000
```

We do not have to specify the variational parameter file that the optimization stage wrote to disk explicitly: `hVMC` will automatically look for a file called `opt_vpar_final.dat` in the simulation's working directory.

Once the simulation has finished we should have a bunch of output files called `sim_res_*.*` in the working directory, depending on which observables have been measured. Let us check the obtained energy which is written to the `sim_res_E.txt` file.

```
$ cat sim_res_E.txt
-0.39972 0.000232416
```

The first number is the mean value $\langle E \rangle$, and the second one the standard deviation σ of the mean. Different results within a few standard deviations will be obtained for every simulation run depending on the system time which is used as the random number generator seed. Note that even for a predefined seed using the `--calc.rng-seed` option, one will still get different results for parallel runs due to the dynamic scheduling of bins in the distributed Monte Carlo cycle. In order to get the exact same result twice, one has to specify a seed and run `hVMC` without MPI parallelization.

The last step is now to run `hVMC` a third time in analysis mode.

```
$ ../hvmc example.job --calc.mode ana
```

This will read the density-density correlation data from the `sim_res_nncorr.dat` file, Fourier transform it, and write the results to `ana_res_ssfac.txt`. The analysis is not parallelized, so there is no point in running it with `mpirun`. We can then use the `plot_mottgap.gnu` script through `gnuplot` to plot $|\vec{q}|^2 / N(\vec{q})$ and extrapolate to $|\vec{q}| \to 0$.

```
$ gnuplot plot_mottgap.gnu
```

This should produce a plot similar to figure 3.1 on page 36. The dimension on the y-axis is completely arbitrary, but the fact that the extrapolation intersects it at a nonzero value strongly hints that the system is Mott insulating. We can easily check that this is no longer the case for small values of $U \lesssim 4$, by modifying the jobfile and running everything again.

A.2 Levels of parallelism in modern high performance computers and their utilization in the `hVMC` code

Increased parallelism has been one aspect of the increasing performance of supercomputers since the mid 1980s. This situation has however changed in the early 2000s, and today ever increasing parallelism is the one driving force behind faster computers. This phenomenon can not only be observed in the realm of high performance computing, but also in mainstream consumer hardware: Personal computers with multiple processors were quite uncommon up until 2004, but as of 2013 multi-core processors are absolutely prevalent and have even started spreading into mobile devices. How did we get here?

For the last decades of the 20th century processors simply got faster in the sense that each new CPU generation reduced the execution time for a given stream of instructions. For software this meant that if something was not performing well, one could simply wait for the next CPU generation to solve the problem. Several factors were contributing to this increase in CPU performance, but it was the ever increasing clock speed of the processors that was the most important force. Unfortunately those days are over: It was Herb Sutter who pointed out in 2005 in a now famous article that the clock speed driven serial processing speed of CPUs was running into "not just one but several physical issues, notably heat (too much of it and too hard to dissipate), power consumption (too high), and current leakage problems" [Sut05]. Looking at figure A.1 we see that Moore's law holds for the total number number of transistors in a CPU, but the exponential increase of the clock speed and the power consumption has definitely ended in the early 2000s, with the clock speed being stuck around 3GHz for the last 10 years. As of 2013 the serial processing speed of CPUs only increases by a few percent with every new generation, while their power consumption is more or less constant in the 100W region, which is about the value that can be air-cooled with a reasonable effort.

Even though serial processing performance does not increase that much anymore, computers still get faster in the sense that they can execute more and more things in parallel. This parallelism happens on several different levels: Within one CPU core, within one computer and then in case of supercomputer also between the different computers in the cluster. Finally, there are also microprocessors dedicated to parallel computing in the form of GPUs. All of these parallelization opportunities have in common that software can only benefit from them, if it is especially designed to do so. Quite often the most difficult part of parallelizing a calculation is not the implementation itself, but identifying the concurrency in the problem, and in some cases with too little concurrency even rethinking the entire algorithm to make it more parallel.

Making efficient use of the available hardware resource is especially important in scientific computing, as performance directly translates to which is the biggest problem one can solve in a reasonable time. A scientific code should therefore try to exploit *all* available levels of parallelism. In this appendix we quickly want to review these different levels and to what extend they are utilized in the `hVMC` code.

A.2.1 Shared and distributed memory multiprocessing

The obvious parallelism inside modern computers is the fact that the different processor cores within the same computer (or in case of a cluster even across the network) can do

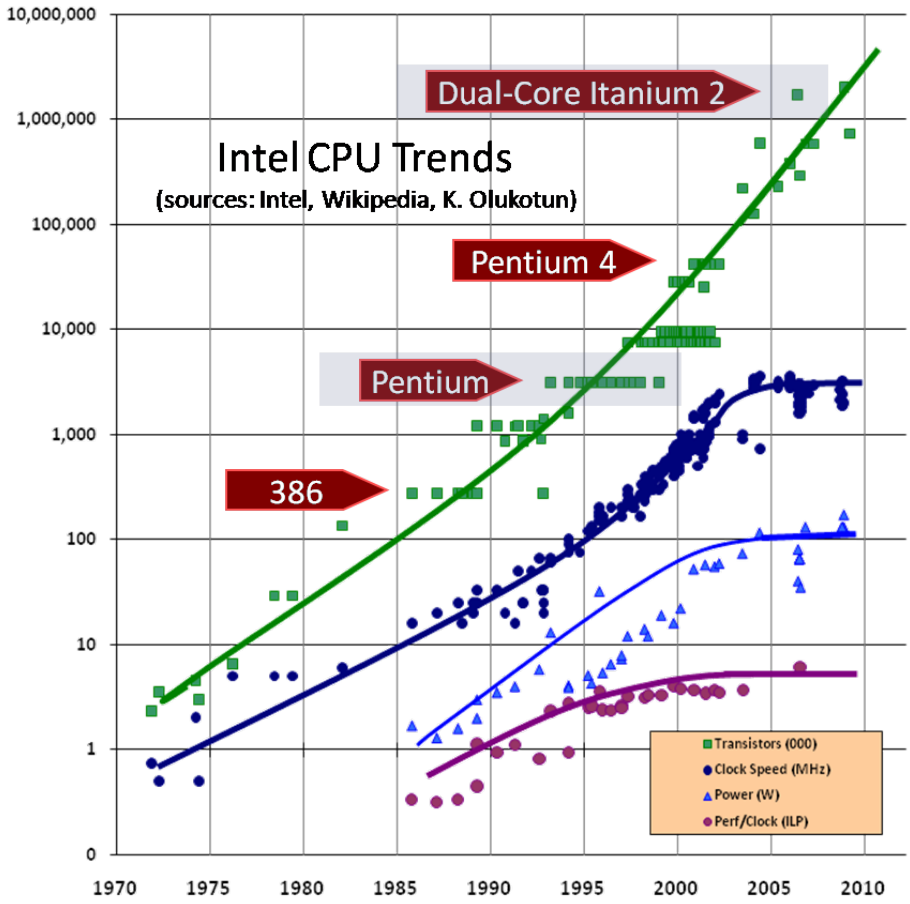

Figure A.1: Key properties of Intel CPUs from 1970 until today. Image: [Sut05]

different things at the same time. This kind of parallelism is called MIMD, which is an acronym for Multiple Instruction Multiple Data, and comes in two different flavors called shared and distributed memory multiprocessing.

Different cores working on the same problem will certainly require some communications between them, and the main difference between shared and distributed memory multiprocessing is how this communication works: In the shared memory paradigm all threads read from and write to the same memory, and because the threads can in principle see what everyone else is doing, synchronization between them is commonly also implemented across this shared memory, and can be thought of as happening implicitly. We are sweeping a lot of things under the rug here: Programming in this shared memory paradigm can be very hard, and it is all too easy to introduce subtle and difficult to reproduce bugs, e.g. data structures that are changed by one thread while another thread is reading them. It is also quite complicated from the hardware point of view, due to the necessity to keep the caches of the different cores in sync.

The other type of MIMD parallelism is distributed memory multiprocessing. Here the different parts of the calculation do not have a shared memory and communicate with each other through explicitly sent messages, usually by calling methods from an MPI (Message Passing Interface) library. Since the different processes have no way of sharing data with each other, all of the data needed by one process first needs to be

transferred there and possibly transferred back upon completion. These transfers of course require additional effort from the programmer, but the fact that there is no shared data prevents most of the subtle bug associated with shared memory programming. One very important thing is that only distributed memory multiprocessing is suitable for both parallelization within the same computer and between nodes in a cluster, while shared memory multiprocessing can only be used inside a single computer, due to the fact that within a cluster the node's memory is already physically distributed.

`hVMC` is parallelized using MPI that works in the distributed memory paradigm and its parallelization was already explained in detail in subsection 4.3.2. The choice of MPI over a shared memory framework like OpenMP was in case of `hVMC` a very easy one for a number of reasons: Since every process is running its own Monte Carlo cycle, there is almost no communication between them and little possibility to share data. Another point is that only a parallelization using MPI works both inside a node and across the network. If we had gone for a shared memory parallelization within the node, we would still need MPI for internode parallelization, essentially doubling the development effort.

There are some things in `hVMC` which could probably benefit from a shared memory parallelization, but none of them have been implemented: Processes running within the same node could share the `Jastrow` and `DeterminantalWavefunction` classes, which – once constructed – are read only anyway. The `Jastrow` is tiny, so it is probably not worth it, but the `DeterminantalWavefunction` class containes the matrix M that can be quite large compared to the caches we would like to keep it in, and which is needed for every recalculation of the $\boldsymbol{W}$ matrix. Another thing that could probably benefit from a shared memory parallelization is the initial diagonalization of the variational Hamiltonian. `hVMC` currently transfers only the variational parameters to each process, each of which is then in charge of building and diagonalizing its own variational Hamiltonian. A much better way to do this would be to transfer the variational parameters only to each node, and then have the cores within each node diagonalize the variational Hamiltonian together in a shared memory fashion. This would probably be a lot faster, but due to the mixing of shared and distributed memory programming also much more complicated to implement. Considering that the initial diagonalization of the variational Hamiltonian is only a tiny fraction of the total runtime, this optimization is probably not worth it.

A.2.2 Vectorization

The innermost level of parallelism hides inside each CPU core itself and is known as vectorization. Vectorization is a so called SIMD parallelization, where SIMD is an acronym for Single Instruction Multiple Data: Instead of instructions that act on single operands, the same operation is applied to a whole vector of operands at the same time, potentially speeding things up by a factor equal to the number of elements in the vector.

This kind of parallelism can be found in recent CPUs as a set of vector instructions called SSE or AVX. We do not want to go into too much detail here, but internally the CPU has dedicated extra wide registers to store the vectors on which the vector instructions then operate. These registers are typically 128bit (SSE) / 256bit (AVX) wide, so that they would be able to hold either 4/8 single precision or 2/4 double precision floating point numbers, implying that there is a factor of two performance difference between single and double precision for vectorized code. Increasing the size

of the vector registers is relatively cheap and simple for the hardware manufacturers, and one can therefore expect that vectorization will become more important in the future. A widening of the registers to 512bit is already scheduled by Intel for 2015, but even today we can not neglect vectorization: A serial single precision code running on a processor with AVX would only use $1/8 = 12.5\%$ of its capacity.

Vectorization can noticeably improve the performance of a code, but at times it can also be quite hard to implement: Historically it was quite common to write the vectorized parts of the code directly in assembly or using compiler intrinsics that directly map to vector instructions. This was of course rather uncomfortable and not portable between different vector instruction sets. At least in C++ this situation has improved a lot with the appearance of template libraries like Vc [KL12] that provide an additional layer of abstraction, greatly improving readability and portability of the vectorized code. Vectorization can still be quite tricky due to more fundamental reasons: One problem is that branch heavy code does not lend itself well to vectorization. Something like branches can be accomplished through the use of masks that restrict the effect of an instruction to parts of a vector, but this of course means that the execution is serial if different parts of a vector take different branches, possibly reducing the performance to a point where simple serial execution would be faster. Another possible issue is the data locality: An AVX vector is essentially a 256bit chunk of data and should ideally be loaded from and stored to memory as a 256bit block. Consider for example a matrix where one needs to access rows as well as columns, but depending on the storage order only one of the two is continuous in memory and could be loaded directly into a vector register. Hardware manufacturers are trying to minimize this problem through the introduction of gather and scatter instructions that allow loading vectors from and writing vectors to non-contiguous memory locations, but these instructions are not widely available yet.

Let us now have a look at where the `hVMC` code could benefit from vectorization. The computationally most expensive things that have to be done are the update and recalculation of the matrix $\boldsymbol{W}$ and the vector $\vec{T}$ from section 3.2.

Both updating and recalculating the matrix $\boldsymbol{W}$ comes down to performing linear algebra calculations. Numerical linear algebra is usually very suitable for vectorization, and luckily there is already a myriad of vectorized and highly optimized libraries available to perform these calculations, making it possible to exploit the available vector instructions without ever writing explicitly vectorized code. `hVMC` takes exactly this lazy approach and offloads practically all calculations related to $\boldsymbol{W}$ to either the Eigen3 library or an external CBLAS, both of which are vectorized and highly optimized.

We have already seen that the speedup attainable by vectorization depends on the number of elements that fit into the vector registers of a given size, and that therefore single precision floating point calculations have the potential to be twice as fast as calculations in double precision, making it preferable to do everything in single precision. For the matrix $\boldsymbol{W}$ in `hVMC` we have already seen that we will need to recalculate it from time to time in order to reset the floating point errors that accumulate during the quick updates. The fact *that* we have to recalculate it does not depend on the precision of the arithmetic, but the precision determines *how many* quick updates can be done before $\boldsymbol{W}$ has to be recalculated. In summary, this means that we can calculate in single

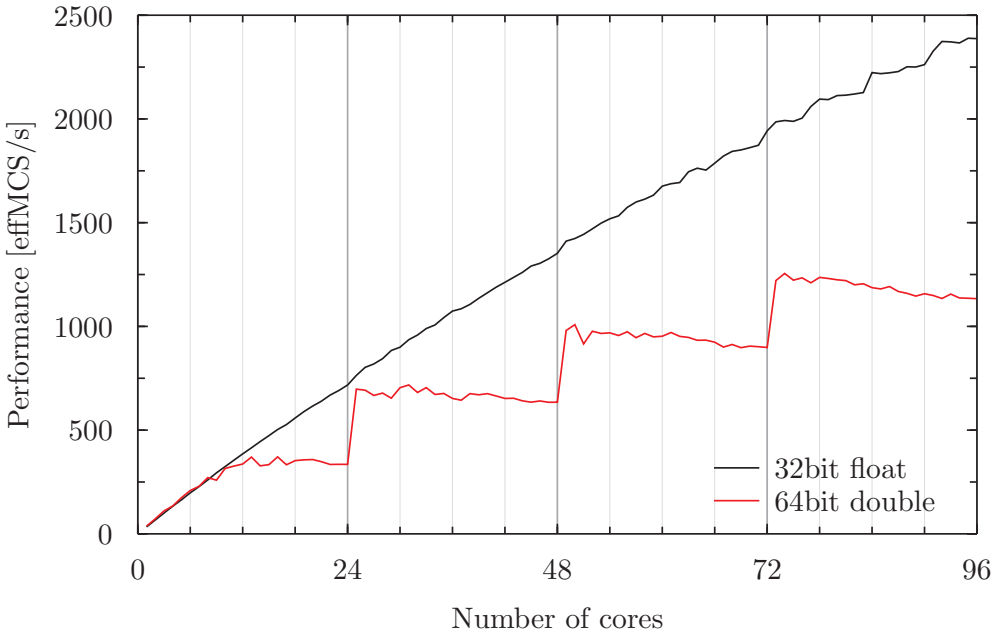

Figure A.2: Scaling of the single and double precision `WMatrix` implementations on LOEWE-CSC. Note the jumps in the double precision performance for every additional node, indicating that in double precision the performance is limited by the different core's competition for memory bandwidth and cache.

precision (potentially speeding updates and recalculations up by a factor of two) if we compensate for the reduced precision by recalculating $\boldsymbol{W}$ more often. Recalculating $\boldsymbol{W}$ is a very expensive operation, so naturally the question comes up whether single or double precision is faster overall. Clarification of exactly this point was the primary motivation for the implementation of `hVMC`'s floating point error control presented on page 44. Tests show that for realistic system sizes the speedup through the increased vector capacity in single precision is roughly balanced by the necessity to recalculate more often, making the single and double precision implementations of the `WMatrix` class equally fast. What is very interesting is that this is only true as long as not many instances of `hVMC` run on different cores in the same CPU at a time, which is exactly what `hVMC`'s distributed Monte Carlo cycle does! Figure A.2 shows the performance as a function of the number of processor cores used in the LOEWE-CSC cluster, which consists of nodes with 24 cores each. We can see that for a small number of cores, single and double precision practically give the same performance. This trend continues until around 12 cores, at which point the speed of the single precision implementation is still increasing linearly, but the performance of the double precision version is almost constant. As soon as the 25th core is used, the double precision performance makes quite a jump and almost catches up with single precision. What happened here? What we see here is the competition of the different processes for the shared cache and memory bandwidth. The more working cores there are within a node, the more memory bandwidth and cache they consume. This is not a problem as long as there is enough bandwidth and cache for everyone, but it starts to limit the performance when these shared resources are depleted. The sudden jump at 25 cores can be explained by the

fact that the LOEWE-CSC nodes have 24 cores each, and requesting 25 of cores will allocate two entire nodes and distribute the 25 processes evenly among them, essentially doubling the available cache and memory bandwidth. In view of this scaling behavior it is easy to see why the single precision implementation of the `WMatrix` class is `hVMC`'s default.

Using vectorization to speed up the `TVector` class is more difficult, since it does not map directly to numerical linear algebra and we can therefore not use ready-made vectorized libraries. Explicit vectorization of the `TVector` class was considered, but ultimately not implemented due to a number of different reasons: First of all the computational complexity of updating and recalculating the vector $\vec{T}$ is one power smaller than for the matrix $\boldsymbol{W}$, implying that its optimization will become less important with larger systems. Second, profiling `hVMC` shows that even for small systems most of the time is still spend dealing with the matrix $\boldsymbol{W}$. Suppose `hVMC` spends 20% of its runtime with the vector $\vec{T}$. Even if we could make that 4 times faster, it would still only reduce the total runtime of `hVMC` by 15%. Finally, the equations for $\vec{T}$ also include the Jastrow parameters v_{ij}, for which the `Jastrow` class internally has to perform the reduction of the index relations, meaning that this would also need to be vectorized. This might reduce `hVMC`'s extensibility, since the reduction of the index relations needs to be implemented for every new lattice. In summary, the vectorization of the calculations for the Jastrow part probably has too little potential to warrant the additional development effort.

A.2.3 General-purpose computing on graphics processing units

So far we have only looked at MIMD parallelism between different processors and the SIMD parallelism inside each processor core in the form of vectorization. Recently another type of parallelism has come into fashion that does not even involve the CPU and goes under the name GPGPU, an acronym for General-Purpose Computing on Graphics Processing Units. What is it that makes GPUs suitable for parallel processing?

CPUs have traditionally been optimized for executing a serial stream of instructions as fast as possible. In general the instructions are dependent on each other, and therefore have to be executed in the correct order. At every branch in the code it also happens, that the instructions which need to be executed themselves depend on the results of former instructions, making everything even more interdependent. There is a huge problem associated with all those dependencies: Suppose that the CPU has to fetch some data from main memory. Main memory is extremely slow compared to the CPU itself, so it will have to wait a long time for the data, in which it could in principle execute a lot of instructions, but practically can not do anything because everything depends on the data that it is waiting for! This is of course horrible for performance, and hardware manufacturers have made a lot of effort to optimize these things and have come up with caches, pipelining, branch prediction, instruction-level parallelism, etc. Today's CPUs are so complex that it is beyond the scope of this thesis to describe all those features, so let us just state the important point: CPUs are extremely efficient at executing a stream of interdependent instructions!

This efficiency at serial execution comes at a very high cost, as can be easily seen in figure A.3. The left picture is a magnified view of the integrated circuit of a CPU,

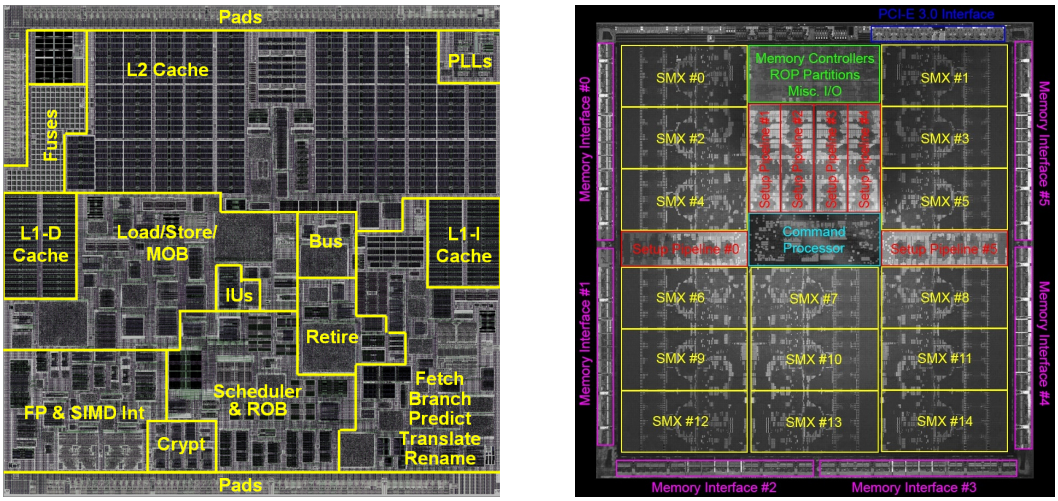

Figure A.3: Die shots of a VIA Nano CPU (left) [viagallery.com] and a NVIDIA Kepler GPU (right) [fellix @ forum.beyond3d.com]. Note the difference in the relative area occupied by arithmetic units (labeled "FP & SIMD Int" on Nano and "SMX" on Kepler).

where the different parts are labeled according to their function. The actual arithmetic happens in the part that is labeled "FP & SIMD Int", and the entire rest of the chip is more or less dedicated to keeping these execution units busy. The problem with CPUs in scientific computing is that here the work that needs to be done is very often not even of this sequentially dependent type for which CPUs were designed. Consider for example `hVMC`'s update of the $\boldsymbol{W}$ matrix through equation (3.58). A straightforward CPU implementation of this update would consist of two nested for-loops, but this kind of serialization is *completely* artificial as there is no dependence between the updates of the individual matrix entries whatsoever! Instead of serializing this inherently parallel update, it would be much better to update the entries in parallel. In the end, all that we care about is the total time until all of them are updated, and not the order in which they are updated or the time it takes to update one of them.

Luckily, processors which are exactly designed for this kind of work already exist in form of the GPU. These devices were originally designed for rendering computer games, but have evolved into general purpose parallel processors in the last years. GPUs are in contrast to CPUs not designed to execute a serial stream of instructions as fast as possible, but are instead built to achieve a high throughput of mostly independent instructions. Internally they consist of a number of independent processors with very wide vectors, essentially combining MIMD and SIMD parallelism, and have dedicated memory of a few gigabytes that can be accessed with a very high bandwidth. The internal processors are relatively simple compared to a CPU and lack its complicated, serial processing speed increasing features like a cache or branch prediction. What they are very good at is quickly switching between different streams of instructions in order to hide memory latency: If one stream of instructions has to wait for data from main memory, the scheduler will put it to rest and continue working on another stream that is ready for execution. In this way a lot more surface area of the chip can be dedicated to actual execution units, as visible in the right picture in figure A.3.

Where could hVMC benefit from the massively parallel execution on a GPU? The most promising point is the update and the recalculation of the matrix $\boldsymbol{W}$, as it is the part of the calculation with the highest power computational complexity. The update of its $2L \times N_p$ entries can be done in parallel, and its recalculation through the solution of a system of linear equations should also work quite well on a GPU.

In the git repository of hVMC there is an experimental branch called cl that implements an update of the matrix $\boldsymbol{W}$ in OpenCL, which is one of the two big frameworks for GPU programming, the other one being Nvidia's proprietary Cuda. The implementation is very basic and only features a few lines of actual GPU code, namely the $\boldsymbol{W}$ update equation (3.58). The idea is to keep the matrix $\boldsymbol{W}$ mostly in GPU memory, and at every Metropolis step only to transfer the single element needed to calculate the Metropolis probabilities back to the CPU, in this way minimizing the transfer between CPU and GPU. The recalculation of $\boldsymbol{W}$ still happens on the CPU, but since this does not need to be done too often, the transfer of the new matrix to the GPU should not affect performance too much.

Benchmarking this, one quickly realizes that this simple idea of having part of the algorithm on the GPU and part on the CPU does not work very well: At every Metropolis step we have to transfer the matrix element $W_{l\beta}$ from the GPU to the CPU, which of course does have a certain latency. We have measured the time needed to fetch a single floating point number from the GPU memory to be about 13μs, which unfortunately is already much to long: Suppose we want to do 10000 Monte Carlo steps for a system of 576 sites, amounting to $5.76 \cdot 10^6$ Metropolis steps in total. The total time spent transferring the $W_{l\beta}$ matrix elements from the GPU is therefore $5.76 \cdot 10^6 \cdot 13\mu\text{s} = 75\text{s}$, which is also about the measured runtime of the GPU version. If we on the other hand just run such a simulation on a single CPU core, we find that it finishes in about 60s! We can therefore conclude that our GPU implementation's performance is heavily limited by the latency of the many transfers, or even more general that any GPU implementation where transfers are required at every Metropolis step would be limited by the transfer times.

A possible way around this issue could be to port not just the $\boldsymbol{W}$ update but essentially everything to the GPU, and only fetch the finished results back to the CPU once every Monte Carlo step, or even once every bin. That is probably more promising than just the $\boldsymbol{W}$ update, but also a lot more development effort, especially since all the inherently serial parts also have to be ported to the GPU. Such an implementation was started in the cl2 branch of hVMC, but was never really finished due to its complexity (it is essentially a complete rewrite of the core components) and the limited time.

On a personal note, I still think that Variational Monte Carlo for the Hubbard model offers more than enough parallelism so that it should be possible to write a very efficient GPU implementation. What I by now have some doubts about, is whether it is a good idea to start writing such an implementation by basically modifying an existing CPU version and porting parts of it to the GPU. I think its probably more worthwhile to start the GPU version from scratch and really take the time to rethink everything with GPUs in mind. This is a lot of work and certainly out of the scope of this thesis, but it is also something I would definitely like to get back to in the future!

A.3 Proofs of mathematical theorems

A.3.1 Matrix determinant lemma

Let us prove the lemma we used on page 30 to calculate the determinant of a matrix that is the sum of the identity matrix and an outer vector product. Consider the following identity.

$$\begin{pmatrix} \mathbb{1} & 0 \\ \vec{v}^T & 1 \end{pmatrix} \begin{pmatrix} \mathbb{1} + \vec{u}\vec{v}^T & \vec{u} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbb{1} & 0 \\ -\vec{v}^T & 1 \end{pmatrix} = \begin{pmatrix} \mathbb{1} & 0 \\ \vec{v}^T & 1 \end{pmatrix} \begin{pmatrix} \mathbb{1} & \vec{u} \\ -\vec{v} & 1 \end{pmatrix} = \begin{pmatrix} \mathbb{1} & \vec{u} \\ 0 & 1 + \vec{v}^T\vec{u} \end{pmatrix} \tag{A.1}$$

The two outer matrices on the left hand side are triangular with only ones on the diagonal and hence their determinant is one. Using the multiplicativity of the determinant we can state the desired relation.

$$\det\left(\mathbb{1} + \vec{u}\vec{v}^T\right) = 1 + \vec{v}^T\vec{u} \tag{A.2}$$

A.3.2 Sherman-Morrison formula

On page 31 we have used a special case of the Sherman-Morrison formula to calculate the inverse of a matrix that is the sum of the identity matrix and an outer vector product, which we now want to prove. Let us make an ansatz for the inverse and then verify that multiplication with it indeed produces the identity matrix.

$$\left(\mathbb{1} + \vec{u}\vec{v}^T\right)\left(\mathbb{1} - \frac{\vec{u}\vec{v}^T}{1 + \vec{v}^T\vec{u}}\right) = \mathbb{1} - \frac{\vec{u}\vec{v}^T}{1 + \vec{v}^T\vec{u}} + \vec{u}\vec{v}^T - \frac{\vec{u}\vec{v}^T\vec{u}\vec{v}^T}{1 + \vec{v}^T\vec{u}} \tag{A.3}$$

$$= \mathbb{1} + \vec{u}\left[-\frac{1}{1 + \vec{v}^T\vec{u}} + 1 - \frac{\vec{v}^T\vec{u}}{1 + \vec{v}^T\vec{u}}\right]\vec{v}^T \tag{A.4}$$

$$= \mathbb{1} + \vec{u}\left[\frac{-1 + 1 + \vec{v}^T\vec{u} - \vec{v}^T\vec{u}}{1 + \vec{v}^T\vec{u}}\right]\vec{v}^T \tag{A.5}$$

$$= \mathbb{1} \tag{A.6}$$

The ansatz is correct and we can therefore state the following relation, which is a special case of the more general Sherman-Morrison formula.

$$\left(\mathbb{1} + \vec{u}\vec{v}^T\right)^{-1} = \mathbb{1} - \frac{\vec{u}\vec{v}^T}{1 + \vec{v}^T\vec{u}^T} \tag{A.7}$$