Full Configuration Interaction Monte Carlo Studies of Quantum Dots

by

Karl Roald Leikanger

THESIS

for the degree of

MASTER OF SCIENCE

(Master in Computational Physics)

Faculty of Mathematics and Natural Sciences
Department of Physics
University of Oslo

June 2013

•

This thesis is dedicated to all near and dear for putting up with me during the last year. And especially to Lisen, Oskar and Kaspar.

I would also like to thank Morten Hjorth-Jensen for the supervision and the motivation, Simen Kvaal for helping me out with his article and to Sarah, Sigve, Jørgen, Veronika, Gustav and everybody in the corridor for a good time.

•

Contents

Chapter 1

Introduction

1.1 Motivation

Full Configuration Interaction Monte Carlo (FCIQMC) is an *ab initio* method to calculate ground state properties of quantum many body systems. The algorithm was first made famous by Booth *et. al.* (2009) [3], and has since then seen great success within the quantum chemistry community. Our motivation was to implement FCIQMC and apply the algorithm on a new physical system, namely two dimensional quantum dots in parabolic potentials. This is a well explored physical system, which have been simulated using a range of numerical methods like Variational Monte Carlo [11], Diffusion Monte Carlo [22], Coupled Cluster [24] and Full Configuration Interaction [20, 15, 16], and therefore provides a good benchmark for the FCIQMC algorithm. Furthermore, the degree of correlation can be tuned by increasing or decreasing the strength of the confining potential, and thus allows us to study how the algorithm performs with systems of varying correlation strength.

1.2 Achievements

Our main achievement is to successfully have implemented FCIQMC, which is a rather novel algorithm with few implementations. The implementation was a challenging task, and quite a lot of thinking and experimenting was necessary to find viable solutions. The main obstacles was to find an efficient numerical representation of the state vectors with a low memory footprint, and to develop a fast parallel algorithm with a small overhead using a hybrid approach with multithreading and MPI. Some work has also been invested in finding a practical and reliable way of analyzing the stochastic error of the so called projected estimator of the energy and developing a fast algorithm for storing and retrieving the Hamiltonian matrix elements.

We have studied how the algorithm performs when applied on quantum dots with a different degree of correlation and a different number of particles. We believe that these results can be generalized to other systems as well, and this may be seen as our main results. We have also made a few predictions of the energy of quantum dots with $N_P \leq 6$ particles using extrapolation of the energies, and have demonstrated that open shell Diffusion Monte Carlo calculations of the same systems produce ground state energies that are slightly too high.

1.3 The structure of the thesis

This thesis is organized in six parts

I: In the first part we discuss the theoretical background for our project. An introduction to quantum mechanics is followed by a short review of the physics of quantum dots.

II: In the second part we discuss different numerical many body methods such as Full Configuration Interaction, Diffusion Monte Carlo and Hartree-Fock. It is important to have a certain knowledge of these methods, both as a theoretical basis to understand the FCIQMC algorithm and as a theoretical background when we interpret our results.

III: The third part is devoted to the theory and the numerical methods that is directly relevant for the FCIQMC method. Three chapters are devoted to the FCIQMC algorithm, the initiator adaption i-FCIQMC and sampling methods. In the last chapter we have introduce an indexation and storage scheme for the Hamiltonian Matrix elements.

IV: In the fourth part we are dealing with methods to analyse output data from the simulations. We have discussed how the statistical error can be calculated and how we can fit parametrized curves to extrapolate the results.

V: In the fifth part we discuss different aspects of our implementation of the algorithm. The numerical representation of the state vector and the parallelization of the algorithm are discussed in the first chapters. Next we look at the class structure and implementation details. One chapter is devoted to benchmarking and testing of the code, and in the last chapter we discuss how it can be modified to simulate other physical systems.

VI: In the last part we study how the algorithm performs when applied to two dimensional quantum dots. This is an excellent test case since the degree of correlation in the systems can be varied by changing the potential strength.We have demonstrated that the code is reproducing Full Configuration Interaction, Coupled Cluster and Diffusion Monte Carlo results, and we have calculated the ground state energies for a few open shell systems with $N_P \leq 6$ particles by extrapolating the energies. We have also explored the scaling and efficiency of the algorithm for systems with a different number of particles and different interaction strengths.

Part I

General theory

In this part we give a short introduction to non- relativistic quantum mechanics and quantum many body theory. We will also introduce the so called second quantized notation and discuss how quantum dots are modelled mathematically.

The following chapters are not in any way meant to be a thorough introduction to the subjects that we discuss. Our goal here is twofold. First we want to introduce the necessary concepts, theorems and equations that we need later in the thesis, and second, we want to establish a certain mathematical notation.

Chapter 2

A brief introduction to Quantum Mechanics

This chapter is not meant to be a self-contained introduction to quantum mechanics, but rather a brief introduction to the concepts that have a direct relevance to this thesis. Those who are looking for a complete introduction to quantum mechanics are referred to an introductory text on the subject. See for example Ref. [31] for a thorough introduction to the field. We will first state the fundamentals of quantum theory followed by the introduction of a few important properties of quantum mechanical systems.

2.1 The postulates of quantum mechanics

Quantum mechanics is an axiomatic theory, meaning that it is based on a set of postulates from which the theory is logically derived. In this chapter we are presenting the fundamental axioms or postulates of quantum mechanics and explain their meaning. This will serve as an extremely brief introduction to the main concepts of the theory.

The first postulate concerns the mathematical representation of a quantum mechanical system.

Postulate *1: To every quantum system there is an associated separable infinite dimensional complex Hilbert space $\mathscr{H}$. A quantum state is represented as a normalized state vector $|\Psi\rangle \in \mathscr{H}$.*

The Hilbert space, also called the state space, is a linear vector space with an inner product, and can be viewed as a generalization of Euclidian spaces. Hilbert spaces are characterized by a set of properties that we will now present. Assume that the vectors $\{|a\rangle, |b\rangle\}$ are in $\mathscr{H}$. Then the inner product of two elements $|a\rangle, |b\rangle$ is denoted $\langle a|b\rangle$, and obeys the following relations

(i): The Hermitian symmetric property: $\langle a|b\rangle = \langle b|a\rangle^*$,

(ii): Linearity in the first element : $\langle ca|b\rangle = c\langle a|b\rangle$,

(iii): Conjugate linearity in the second element : $\langle a|cb\rangle = c^*\langle a|b\rangle$,

(iv): Additivity in the first element: $\langle a_1 + a_2|b\rangle = \langle a_1|b\rangle + \langle a_2|b\rangle$,

(v): Additivity in the second element: $\langle a|b_1 + b_2\rangle = \langle a|b_1\rangle + \langle a|b_2\rangle$.

(vi): Positivity: $\langle a|a\rangle \geq 0$.

This is a general definition of Hilbert spaces, but as we will see later, the exact properties of the state spaces are defined by the physical system itself.

Note that any linear combination of vectors in $\mathscr{H}$ (except the null vector) represents a physical state. If we for example have a basis $\{|i\rangle\}_{i=0}^{n}$ for the state space, any state can be represented as

$$|\Psi\rangle = \sum_{i=1}^{n} c_i|i\rangle, \tag{2.1}$$

where $\{c_i\}$ are complex weights, or amplitudes, with the property $\sum_{i=1}^{n} |c_i|^2 = 1$ to normalize the state.

The next postulate concerns the joint Hilbert spaces of composite systems. An example of such systems is the many electron quantum dots that we are dealing with in this thesis.

***Postulate** 2: If $\mathscr{H}_1$ and $\mathscr{H}_2$ are the Hilbert spaces of two quantum systems, then the Hilbert space of the composite system is the tensor product $\mathscr{H}_1 \otimes \mathscr{H}_2$.*

For example, for a system consisting of a number of distinguishable particles in the states $|a\rangle$, $|b\rangle$ and $|c\rangle$, the composite state is the tensor product $|\Psi\rangle = |a\rangle \otimes |b\rangle \otimes |c\rangle$. We will come back to this subject later when we discuss many particle systems, and as we will see, this theorem provides us with an easy way of constructing the Hilbert spaces of many particle quantum dots.

The state vector encapsulates all there is to know about the physical system, but quantum mechanical systems are not well defined in a classical sense where all measurable quantities can be known simultaneously. They are abstract quantities that contain information about the probability of the different outcomes of a measurement. The link between the abstract state vectors and measurable quantities are defined by Hermitian[1] operators as stated in the third postulate.

***Postulate** 3: To every observable of a quantum mechanical system, there is associated a linear Hermitian operator. The spectrum of eigenvalues of the operator represents the measurable values of the observable.*

An observable is defined as a measurable quantity such as the angular momentum or the energy of the state. Hermitian operators have the properties that all eigenvalues are real and that the eigenvectors span the space in which the operators act. An example is the Hamiltonian operator, $\hat{H}$ which represents the energy of the state. Assume that the operator $\hat{H}$ has the eigenvalues $\{\varepsilon_i\}_{i=1}^{n}$ and the eigenvectors $\{|\varphi_i\rangle\}_{i=1}^{n}$ which are a basis for $\mathscr{H}$. The eigenvalues represents the possible outcome of a measurement of the energy and the corresponding eigenvectors are the states in which the observable is "sharp" or well defined. The probability of measuring the energy ε_i is given by the function

$$P(\varepsilon_i) = \langle\Psi|\varphi_i\rangle\langle\varphi_i|\Psi\rangle, \tag{2.2}$$

[1] An operator $\hat{O}$ is Hermitian if $\hat{O}^\dagger = \hat{O}$, where $\hat{O}^\dagger$ is the adjoint operator of $\hat{O}$.

which means that the energy is well defined only in the case that the state vector is parallel to an eigenvector of $\hat{H}$, or in the subspace spanned by the eigenvectors $\{|\varphi_{i_1}\rangle, |\varphi_{i_2}\rangle, \dots, \}$ with degenerated energies $\varepsilon_{i_1} = \varepsilon_{i_2} = \dots$. As a consequence of this, it is impossible to simultaneously measure the observables of two operators $\hat{H}$ and $\hat{O}$ with non-parallel eigenvectors, or more precisely, which are not commuting $[\hat{O}, \hat{H}] \neq 0$. This property is commonly known as the uncertainty principle.

A measurement in quantum mechanics is a subtle concept which often gives rise to philosophical difficulties. The fourth postulate bypasses these problems by providing an operational definition of an ideal measurement.

***Postulate** 4: A ideal measurement of an observable o with the corresponding operator $\hat{O}$ leaves the quantum system in the state $|o\rangle$ where $\hat{O}|o\rangle = o|o\rangle$.*

As an example, if the quantum state $|\Psi\rangle$ is measured to have the energy ε_i, we say that the state has collapsed into the corresponding eigen state $|\varphi_i\rangle$. This is modeled as an instantaneous process where the state changes from $|\Psi\rangle$ to $|\varphi_i\rangle$. Note that both the concept of an ideal measurement and the idea of an instantaneous change of the physical state is controversial, and that this is an active field of research (see for example Ref. [32]).

The fifth and last postulate states that the dynamics of a quantum system is described by a famous partial differential equation.

***Postulate** 5: The time evolution of the state vector is described by the time dependent Schrödinger equation.*

The Schrödinger equation can be written

$$i\hbar \frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle, \tag{2.3}$$

where $\hbar$ is the reduced Planck's constant and $\hat{H}$ is the Hamiltonian operator. The quantum mechanical wave function $|\Psi(t)\rangle$ is a *probability amplitude* that describes the state of a quantum mechanical system at time t. If $|\Psi(t)\rangle$ is an eigenstate of the Hamiltonian $\hat{H}$, then $|\Psi(t)\rangle$ is said to be stationary and is represented by the wave function

$$|\Psi(t)\rangle \to |\varphi_i\rangle e^{-i\varepsilon_i t/\hbar}, \tag{2.4}$$

where $\{|\varphi_i\rangle\}$ are the eigenstates of the Hamiltonian and $\{\varepsilon_i\}$ are the eigenvalues

$$\hat{H}|\varphi_i\rangle = \varepsilon_i|\varphi_i\rangle. \tag{2.5}$$

Obviously, this can only be true if the Hamiltonian has no explicit time dependence. Since the Hamiltonian operator is a hermitian operator, the set of eigenstates $|\varphi_i\rangle$ is a basis. This means that as long as the Hamiltonian operator is time independent, all quantum states can be written as a linear combination of the stationary states

$$|\Psi(t)\rangle = \sum_i |\phi_i\rangle e^{-i\varepsilon_i t/\hbar}, \tag{2.6}$$

which always solves the Schrödinger equation.

In this thesis we are only dealing with time independent quantum mechanical systems. Therefore we are only solving the eigen value problem Eq. (2.5) which we will refer to as the stationary Schrödinger equation from now on. Since the stationary states are eigenstates of the Hamiltonian, the eigenvalues ε_i must be interpreted as the possible energies of the system. The lowest eigenvalue ε_i will be referred to as the ground state energy of the system and the corresponding eigenstate will be referred to as the ground state.

2.2 Antisymmetric state vectors and fermions

A state vector which describes a system of identical particles can be shown to be either symmetric or antisymmetric with regard to the interchange of two particles. And as we will show in this section, this seemingly innocent property has important consequences for the behaviour of quantum particles.

We will start with a short derivation. Consider an N particle quantum system of identical particles

$$|\Psi\rangle = |1, 2, 3, \ldots, N\rangle. \tag{2.7}$$

This state must have the same physical properties when two particles are permuted, which means that for any observable $\hat{O}$,

$$\langle\Psi|\hat{P}_{ij}^{\dagger}\hat{O}\hat{P}_{ij}|\Psi\rangle = \langle\Psi|\hat{O}|\Psi\rangle, \tag{2.8}$$

where $\hat{P}_{ij}$ is the interchange operator which has the property that it interchanges particle i and j. This equation has the solution

$$\hat{P}_{ij}|\Psi\rangle = \pm|\Psi\rangle. \tag{2.9}$$

All elementary particles are separated in two categories of particles called fermions, as for example electrons, and bosons, with the property that

$$\hat{P}_{ij}|\Psi\rangle = -|\Psi\rangle \text{ for fermions and } \hat{P}_{ij}|\Psi\rangle = +|\Psi\rangle \text{ for bosons.} \tag{2.10}$$

A state vector describing a system of fermions is therefore said to be antisymmetric with regard to the exchange of two particles. A property of such states is that two identical fermions never can be in the same quantum state. This is easy to see if we consider the two particle state $|1, 1\rangle$ consisting of two identical fermions. If we apply the interchange operator on this state we see that

$$|1, 1\rangle = \hat{P}_{12}|1, 1\rangle = -|1, 1\rangle \Rightarrow |1, 1\rangle = 0, \tag{2.11}$$

which shows that this state can not exist.

2.3 Spin and the spin statistics theorem

Spin is an intrinsic property of elementary particles which have no classic counterpart but is a pure quantum phenomenon. The spin quantum number s can only have values $s \in \hbar n/2$ where n is a positive integer and $\hbar$ is the reduced Planck constant. Any given class of elementary particles has a fixed spin value that can not be changed, and according to the spin statistics theorem, all particles with half integer spins are fermions [31].

The spin of a single electron is described by a spinor $|\chi\rangle \in \mathbb{C}^2$ which is usually represented as a linear combination of the eigen functions of the spin projection operator $\hat{S}_z$

$$\hat{S}_z|\uparrow\rangle = s|\uparrow\rangle, \qquad \hat{S}_z|\downarrow\rangle = -s|\downarrow\rangle, \tag{2.12}$$

and a many particle wave function has a spinor

$$|\chi\rangle \in \{|\uparrow\rangle, |\downarrow\rangle\} \otimes \{|\uparrow\rangle, |\downarrow\rangle\} \otimes \{|\uparrow\rangle, |\downarrow\rangle\} \otimes \ldots \{|\uparrow\rangle, |\downarrow\rangle\}. \tag{2.13}$$

There is a magnetic moment associated to the spin of an electron. The magnetic moment contribute to the total energy of an electronic system, but this effect is usually very small and is often neglected. In this thesis, we use a Hamiltonian without a spin coupling, meaning that we ignore the magnetic moment of the electrons. However, the spin of the particles still have a very important impact on the behaviour of a quantum system. Because of the Pauli exclusion principle, the many particle basis is limited to states where any one particle state is occupied by at most one electron.

2.4 The many particle basis

The bosonic and fermionic many particle wave functions live in different Hilbert spaces and follow different mathematical rules. We will only focus on fermionic systems since these are the systems that we have dealt with in this thesis.

We assume that an orthonormal single particle basis is available

$$\{|\varphi_1\rangle, |\varphi_2\rangle, |\varphi_3\rangle, \dots\} \in \mathscr{H}_1, \tag{2.14}$$

where $\{\varphi_i\}$ are eigenfunctions of the single particle hamiltonian $\hat{H}_{HO}$, and the spinors are assumed to be contained in the single particle wave functions. According to the second postulate, the many particle Hilbert space is

$$\mathscr{H} = \mathscr{H}_1 \otimes \mathscr{H}_1 \otimes \mathscr{H}_1 \otimes \dots \mathscr{H}_1. \tag{2.15}$$

In the case that the single particle states are representing fermions, the many particle states must be antisymmetric with regard to a permutation of two particles. The anti symmetric Hilbert space of an N particle system is a subspace of $\mathscr{H}$ and can be written

$$\mathscr{H}_{AS} = \left\{|\Psi\rangle \in \mathscr{H} \,;\, \hat{P}_{ij}|\Psi\rangle = -|\Psi\rangle \;\forall i \neq j \in 1, 2, 3, \dots, N\right\}. \tag{2.16}$$

To construct the antisymmetric states we will define the antisymmetrization operator $\hat{A}$. And to do so, we must first define the permutation operator $\hat{P}_{i_1,i_2,\dots,i_N}$ which changes the sequence of particles $(1, 2, 3, \dots, N)$ to $(i_1, i_2, \dots, i_N)$. Any permutation of the particles can now be written as a product of p interchange operators

$$\hat{P}_{i_1,i_2,\dots,i_N} = \hat{P}_{m_1,n_1}\hat{P}_{m_2,n_2}\dots\hat{P}_{n_p,m_p}, \tag{2.17}$$

where the sign of the permutation is

$$S_{i_1,i_2,\dots,i_N} = (-1)^p. \tag{2.18}$$

The permutation operator is defined as

$$\hat{A} = \sum_{\text{All possible permutations } \hat{P}_{\dots}} \frac{1}{\sqrt{N!}} \hat{P}_{i_1,i_2,\dots,i_N} S_{i_1,i_2,\dots,i_N}, \tag{2.19}$$

where $1/\sqrt{(N!)}$ is a normalization factor (note that the total number of possible permutations is $N!$). Since $\hat{P}_{ij}\hat{A} = -\hat{A}$, we see that the states

$$|\varphi_{i_1}\varphi_{i_2}\varphi_{i_3}\dots\varphi_{i_N}\rangle \equiv \hat{A}|\varphi_{i_1}\rangle \otimes |\varphi_{i_2}\rangle \otimes |\varphi_{i_3}\rangle \otimes \dots |\varphi_{i_N}\rangle, \tag{2.20}$$

are antisymmetric. The adjoint of P_{ij} is the permutation operator defined by the inverse permutation $P_{ij}^{-1} = P_{ji}$ meaning that $P^T = P^{-1}$. Consequently, each permutation operator is a unitary operator and therefore $\hat{A}$ must be as well. Since $\hat{A}$ is a unitary operator the normalization and orthogonality of the product states are conserved.

The antisymmetrized product states, with $i_1 < i_2 < i_3 < \dots i_N$, constitute a basis for $\mathscr{H}_{AS}$,

$$\mathscr{H}_{AS} = \text{span}\{|\varphi_{i_1}\varphi_{i_2}\varphi_{i_3}\dots\varphi_{i_N}\rangle;\, i_1 < i_2 < i_3 < \dots i_N\}, \tag{2.21}$$

where we have set $i_1 < i_2 < i_3 < \dots i_N$ to assure that all particles are in a unique state and that every state is represented only once.

The states $|\varphi_{i_1}\varphi_{i_2}\varphi_{i_3}\dots\varphi_{i_N}\rangle$ are often written as determinants

$$|\varphi_{i_1}\varphi_{i_2}\varphi_{i_3}\dots\varphi_{i_N}\rangle = \frac{1}{\sqrt{N!}}\det(\varphi_{i_1},\varphi_{i_2},\varphi_{i_3},\dots,\varphi_{i_N}) \equiv \frac{1}{\sqrt{N!}}\begin{vmatrix} \varphi_{i_1}^{(1)} & \varphi_{i_2}^{(1)} & \dots & \varphi_{i_N}^{(1)} \\ \varphi_{i_1}^{(2)} & \varphi_{i_2}^{(2)} & \dots & \varphi_{i_N}^{(2)} \\ \vdots & & & \vdots \\ \varphi_{i_1}^{(N)} & \varphi_{i_2}^{(N)} & \dots & \varphi_{i_N}^{(N)} \end{vmatrix}, \tag{2.22}$$

where the superscript (i) denotes the i'th state in a direct product. Determinants are very useful since they have the desired antisymmetric properties, and the many particle state vectors are often sloppily referred to as determinants.

2.5 The many particle operators

As we have already discussed, observables are represented by Hermitian operators which map $\mathscr{H}_{AS}$ to itself. Many particle operators are often categorized after the number of particles that are involved in an interaction. The one body operators involve single particle interactions, the two body operators involve two particle interactions and so on. Our Hamiltonian, as we will see later, only contains one and two body operators, and we will therefore take a closer look at these cases.

The one body operators takes the form

$$\hat{C} = \hat{c}\otimes 1\otimes 1\otimes\dots 1 + 1\otimes\hat{c}\otimes 1\otimes\dots 1 + \dots + 1\otimes 1\otimes 1\otimes\dots\hat{c}, \tag{2.23}$$

where $\hat{c}$ is an operator in the single particle Hilbert space $\mathscr{H}_1$. On the single particle basis $\{|\varphi_i\rangle\}$ the operator is written

$$\hat{c} = \sum_{ij}|\varphi_i\rangle\langle\varphi_j|\,\langle\varphi_i|\hat{c}|\varphi_j\rangle, \tag{2.24}$$

and the amplitude for an interaction between the two many body states $|\varphi_{n_1}\dots\varphi_{n_N}\rangle$ takes the simple form

$$\langle\varphi_{n_1}\dots\varphi_{n_N}|\hat{C}|\varphi_{m_1}\dots\varphi_{m_N}\rangle = \sum_{i,j=1}^{N}\langle\varphi_{n_i}|\hat{c}|\varphi_{m_j}\rangle. \tag{2.25}$$

A general two particle operator $\hat{D}$ can be written as a sum of operators

$$\hat{D} = \frac{1}{2}\sum_{i\neq j}\hat{D}_{ij} \tag{2.26}$$

where D_{ij} is defined on the two particle space $\mathscr{H}_1 \otimes \mathscr{H}_1$ of particle i and j and can be written on the form.

$$\hat{D}_{ij} = \sum_{klmn} d_{klmn} 1 \otimes \ldots 1 \otimes \underbrace{|\varphi_k\rangle\langle\varphi_m|}_{i\text{'th place}} \otimes 1 \otimes \ldots 1 \otimes \underbrace{|\varphi_l\rangle\langle\varphi_n|}_{j\text{'th place}} \otimes 1 \otimes \ldots 1, \tag{2.27}$$

and the interaction amplitude is

$$\langle\varphi_{n_1}\ldots\varphi_{n_N}|\hat{D}|\varphi_{m_1}\ldots\varphi_{m_N}\rangle = \sum_{i,j,k,l=1}^{N} d_{n_i n_j m_k m_l}. \tag{2.28}$$

The operators representing an observable must obviously map $\mathscr{H}_{AS}$ to itself. This implies, as is straight forward to show, that the operators must commute with the antisymmetrization operator $\hat{A}$. This is always the case for the one body operator $\hat{C}$, but for the two body operator $\hat{D}$ it sets some restrictions on the form of d_{klmn}. In our case, the two body operator is the Coulomb interaction for which $d_{klmn} = d_{mnkl}$, which means that $\hat{D}$ and $\hat{A}$ commute. In fact, our Hamiltonian is completely symmetric with regard to the exchange of two particles since all electrons have the same mass and charge. This means that the interchange operators have no effect on the operator, and consequently it commutes with the antisymmetrization operator.

2.6 The variational principle

The variational principle is a general property of quantum systems, and is also the foundation of several numerical many body methods. It states that any state $|\Psi\rangle \in \mathscr{H}$ has an expectation value for the energy $\langle\Psi|\hat{H}|\Psi\rangle$ which is larger or equal to the ground state energy. Furthermore, if the expectation value is equal to the ground state energy, $|\Psi\rangle$ is a ground state of the Hilbert space. We will show why this is true for nondegenerate ground states. Assume that $|\Psi\rangle$ is expanded in the basis of eigenfunctions $\{|\varphi_i\rangle\}$ of the Hamiltonian operator

$$|\Psi\rangle = \sum_i c_i|\varphi_i\rangle. \tag{2.29}$$

The expectation value of the Hamiltonian is

$$\langle\Psi|\hat{H}|\Psi\rangle = \sum_i |c_i|^2\varepsilon_i, \tag{2.30}$$

where $\{\varepsilon_i\}$ are the eigenvalues of $\hat{H}$ and $\{c_i\}$ are complex weights. Since the ground state energy ε_0 is smaller or equal to all other ε_i, the normalization requirement $\sum_i |c_i|^2 = 1$ implies that

$$\sum_i |c_i|^2\varepsilon_i \leq \varepsilon_0, \tag{2.31}$$

and only equal in the case that $|c_0| = 1$ and $c_i = 0$ for $i > 0$, in which case $|\Psi\rangle$ is the ground state $|\varphi_0\rangle$. The proof is easily generalized to systems with degenerate ground states, but in this case the ground state is no longer unique.

Chapter 3

A brief introduction to the second quantization formalism

We will now introduce a formalism which is very useful when dealing with systems of many interacting particles. The formalism will be presented very briefly and with no proofs. There are numerous texts which give a good introduction to the second quantization formalism together with relevant theorems and proofs. See for example Shavitt *et. al.* (2009) [33] or Gross *et. al.* (1991) [9].

3.1 Introduction

The second quantization formalism makes use of creation and annihilation operators to add or remove particles to a quantum state. The annihilation operator a_i removes a state φ_i from a state, and the creation operator $a_i^\dagger$ adds a state φ_i,

$$a_2|\varphi_1\varphi_2\varphi_3\dots\varphi_N\rangle = |\varphi_1\varphi_3\dots\varphi_N\rangle, \tag{3.1}$$

$$a_2^\dagger|\varphi_1\varphi_3\dots\varphi_N\rangle = |\varphi_1\varphi_2\varphi_3\dots\varphi_N\rangle. \tag{3.2}$$

By defining a set of anticommutation relations for these operators, we can ensure that the state vectors have the correct antisymmetric properties. The following equations can be shown to apply to fermionic systems

$$\{a_i^\dagger, a_j^\dagger\} = \{a_i, a_j\} = 0, \tag{3.3}$$

$$\{a_i^\dagger, a_j\} = \delta_{ij}. \tag{3.4}$$

It follows that

$$a_i^\dagger|\varphi_{n_1}\varphi_{n_2}\varphi_{n_3}\dots\varphi_{n_N}\rangle = 0 \text{ if } i \in \{n_1, n_2, \dots, n_N\}, \tag{3.5}$$

$$a_i|\varphi_{n_1}\varphi_{n_2}\varphi_{n_3}\dots\varphi_{n_N}\rangle = 0 \text{ if } i \notin \{n_1, n_2, \dots, n_N\}. \tag{3.6}$$

As follows from Eqs. (3.3), (3.4), these relations assure that the state vectors have the correct symmetry properties since any permutation of two different operators (particles) will lead to a sign change. The antisymmetric properties are now contained in the algebraic expressions for the state vectors. Also note that Eq. (3.3) is an expression of the Pauli principle.

Although this is not strictly correct, these state vectors are often simply referred to as determinants. The reason is that state vectors that are represented as determinants have many of the same mathematical properties as the second quantization state vectors. We will, in the rest of this thesis, comply to the sloppy language and refer to a general state vector as a determinant. Any such determinant can be written

$$a_{n_1}^\dagger a_{n_2}^\dagger a_{n_3}^\dagger \dots a_{n_N}^\dagger |\,\rangle = |\varphi_{n_1}\varphi_{n_2}\varphi_{n_3}\dots\varphi_{n_N}\rangle, \tag{3.7}$$

where the indices n_i must be ordered in some way to make sure that each determinant is only represented once. We use the ordering $n_1 < n_2 < n_3 < \cdots < n_N$. The state $|\,\rangle$ represents the vacuum state which symbolizes that there are zero particles present in the system. Note that the vacuum state is normalized $\langle\,|\,\rangle = 1$ and that an annihilation operator working on the vacuum state, $a_i|\,\rangle$, is defined to be zero.

An important property that follows from the anticommutation relations is that the orthogonality of the determinants are preserved. This can be shown by evaluating the inner product of $a_{m1}^\dagger \dots a_{mN}^\dagger|\,\rangle$ and $a_{n1}^\dagger \dots a_{nN}^\dagger|\,\rangle$ which yields

$$\langle\,|a_{mN}\dots a_{m1}a_{n1}^\dagger \dots a_{nM}^\dagger|\,\rangle = \delta_{m1,n1}\dots\delta_{mN,nN}, \tag{3.8}$$

as follows from Eq. (3.4).

3.2 Operators in the second quantization notation

Operators can also be represented in terms of creation and annihilation operators. One body operators can be written on the form

$$\hat{C} = \sum_{ij}\langle\varphi_i|\hat{C}|\varphi_j\rangle a_i^\dagger a_j, \tag{3.9}$$

while two body operators can be written on the form

$$\hat{D} = \frac{1}{2}\sum_{ijkl}\langle\varphi_i\varphi_j|\hat{D}|\varphi_k\varphi_l\rangle a_i^\dagger a_j^\dagger a_l a_k. \tag{3.10}$$

The last two equations can be verified by considering the earlier section about many particle operators. It is often convenient to rewrite the two body operators as

$$\hat{D} = \frac{1}{4}\sum_{ijkl}\langle\varphi_i\varphi_j|\hat{D}|\varphi_k\varphi_l\rangle_{AS} a_i^\dagger a_j^\dagger a_l a_k, \tag{3.11}$$

$$\langle\varphi_i\varphi_j|\hat{D}|\varphi_k\varphi_l\rangle_{AS} \equiv \langle\varphi_i\varphi_j|\hat{D}(|\varphi_k\varphi_l\rangle - |\varphi_k\varphi_l\rangle). \tag{3.12}$$

A derivation of the above identities can be found in most texts covering many body quantum mechanics, and a good reference is Shavitt (2009) [33].

The evaluation of expectation values of products of operators are often made much simpler by finding the so called normal ordered form. We will make use of normal ordered operators later, and we will therefore spend the rest of this section to define normal ordering. A product of operators are said to be normal ordered when all creation operators are to the

left of all annihilation operators. As an example, consider the operators $\hat{A}, \hat{B}, \hat{C}$ which are products of annihilation and creation operators

$$\hat{A} = a^\dagger_{a1} a_{a2}, \tag{3.13}$$

$$\hat{B} = a^\dagger_{b1} a_{b2}, \tag{3.14}$$

$$\hat{C} = a^\dagger_{c1} a_{c2}. \tag{3.15}$$

The normal ordering of these operators are

$$\{\hat{A}\hat{B}\hat{C}\} = (-1)^p a^\dagger_{a1} a^\dagger_{b1} a^\dagger_{c1} a_{a2} a_{b2} a_{c2}. \tag{3.16}$$

This is a reordering where the annihilation operators are set to the right of the creation operators, and p is the number of permutations necessary to reorder the operators. An interchange of two creation or two annihilation operators would not destroy the normal order, thus the normal ordering is not unique.

3.3 The time independent Wick's theorem

Wick's theorem provides an efficient way of calculating expectation values. We will state this theorem without a proof, but the full proof can be found in many texts as for example Gross *et. al.* (1991) [9]. Before we state the theorem we must introduce contractions. A contraction of two creation or annihilation operators $\hat{A}, \hat{B}$ is defined as

$$\overbrace{\hat{A}\hat{B}} = \hat{A}\hat{B} - \left\{\hat{A}\hat{B}\right\}. \tag{3.17}$$

Both $\hat{A}$ and $\hat{B}$ are either a creation or an annihilation operator, which means that there are four possible combinations

$$\overbrace{a^\dagger_i a^\dagger_j} = a^\dagger_i a^\dagger_j - a^\dagger_i a^\dagger_j = 0, \tag{3.18}$$

$$\overbrace{a^\dagger_i a_j} = a_i a^\dagger_j - a^\dagger_i a_j = 0, \tag{3.19}$$

$$\overbrace{a_i a_j} = a_i a_j - a_i a_j = 0, \tag{3.20}$$

$$\overbrace{a^\dagger_i a_j} = a_i a^\dagger_j - (-) a^\dagger_i a_j = \left\{a_i, a^\dagger_j\right\} = \delta_{ij}. \tag{3.21}$$

Further more, contraction of operators within a normal product can be written

$$\left\{\hat{A}\ldots\hat{B}\ldots\hat{C}\ldots\hat{D}\ldots\hat{E}\ldots\hat{F}\right\} = (-1)^p \hat{B}\hat{D}\hat{C}\hat{E}\ldots\left\{\hat{A}\ldots\hat{F}\right\}, \tag{3.22}$$

where p is the number of permutations which are needed to move the contracted operators in front of the normal ordered product. Wick's theorem says that a product of creation and annihilation operators is equivalent to the normal ordered product plus the sum of the normal ordered products with all possible contractions.

$$\hat{A}\hat{B}\ldots = \left\{\hat{A}\hat{B}\ldots\right\} + \sum_{\text{all single c.}} \left\{\overbrace{\hat{A}\hat{B}}\ldots\ldots\right\} + \sum_{\text{all double c.}} \left\{\overbrace{\hat{A}\hat{B}}\ldots\ldots\ldots\right\} + \ldots \tag{3.23}$$

We can use Wick's theorem to rewrite operators as a sum of normal ordered products. This is a convenient form since vacuum expectation values $\langle\,|\{\hat{A}\ldots\}|\,\rangle$ always are zero.

3.4 Particle- hole formalism

Until now, we have used the vacuum state as the reference state. It is however often natural to use other states as the reference state in which case it is convenient to use the so called particle-hole formalism. Mathematically, the particle hole formalism is very similar to the "vacuum reference" formalism, but some mathematical concepts, like normal ordering of the operators and contractions, have to be redefined.

Assume that we want to use the state $|c\rangle = a^\dagger_{c1}\dots a^\dagger_{cN}|\,\rangle$, where N is the number of particles, as our new vacuum state, and define all other states relative to this state. The states $\{|\varphi_{c_i}\rangle\}_{i=1}^N$ are now referred to as hole states while all other states are referred to as particle states. In the rest of this section we will let indices i, j, k, l symbolize hole states and indices a, b, c, d symbolize particle states. Now, any state can be written as an excitation of the reference state

$$|c^{ab\dots}_{ij\dots}\rangle \equiv a^\dagger_a a^\dagger_b \dots a_j a_i |c\rangle. \tag{3.24}$$

The creation operator $a^\dagger_i$ is sometimes called a pseudo annihilation operator since it annihilates a hole. Likewise, an annihilation operator of a hole state, a_i, can be called a pseudo creation operator since it creates a hole. The full set of psuedo operators are defined

$$\left.\begin{aligned} b^\dagger_a &= a^\dagger_a \\ b_a &= a_a \end{aligned}\right\} \text{ (acts on particle states)},$$
$$\left.\begin{aligned} b^\dagger_i &= a_i \\ b_i &= a^\dagger_i \end{aligned}\right\} \text{ (acts on hole states)}. \tag{3.25}$$

The exited states are now written

$$|c^{ab\dots}_{ij\dots}\rangle \equiv b^\dagger_a b^\dagger_b \dots b^\dagger_j b^\dagger_i |c\rangle. \tag{3.26}$$

With this new set of operators, the anticommutation relations, contractions and Wick's theorem will have exactly the same definition as in the "vacuum reference" formalism.

Chapter 4

Mathematical Modelling of two dimensional Quantum dots

4.1 Introduction

In the most general sense of the term, a quantum dot is a small quantum system which is confined in space. But the term is most often used to describe electronic systems that are trapped in semiconductor structures. During the last few decades, advanced processing techniques have made it possible to manufacture artificial quantum dots which are trapped in one or two spatial dimensions, and confinement to less than three dimensions was experimentally verified already in the early 1970s in GaAs-A1GaAs semiconductors [25]. Confinement by potentials set up by electrostatic gates also allows experimentalists to control their shape and size and the number of electrons which ranges from one to hundreds [26].

Our idealized quantum dots are modelled as two dimensional systems, which can be justified both theoretically and experimentally. As we will come back to later, our quantum dot wave function is separable in the spatial dimensions. Consequently, if the component perpendicular to the semiconductor layers are always in the ground state, this component can then be left out of the description.

We will not discuss the "real" quantum dots any further, but will from now on concentrate on our idealized mathematical model. For more information on quantum dots we refer to the review article of Ref.[26], which discusses both numerical and experimental results.

4.2 The Hamiltonian

The two dimensional quantum dots can be described by a Hamiltonian operator on the form

$$\hat{H} = \hat{H}_0 + \hat{V}, \tag{4.1}$$

where $\hat{H}_0$ is an one-body operator which accounts for the single particle kinetic energies and the interaction with an external electromagnetic field, and $\hat{V}$ is a two-body operator which accounts for the Coulomb interaction between the electrons. The single particle

Hamiltonian can be written on the general form

$$\hat{H}_0 \to H_0(\mathbf{r}_1, \ldots, \mathbf{r}_N, \mathbf{p}_1, \ldots, \mathbf{p}_N) = \sum_i \left(\frac{1}{m^*} \left[\mathbf{p}_i - \frac{e}{c} \mathbf{A}(\mathbf{r}_i) \right]^2 + V_{ext}(\mathbf{r}_i) \right), \tag{4.2}$$

where $\mathbf{A}(\mathbf{r})$ is the vector potential of an external magnetic field, and $(\mathbf{r}_i, \mathbf{p}_i)$ is the position and momentum coordinates. The magnetic moment of the electrons is ignored, thus the spin-spin and the spin-orbit couplings are assumed to be negligible. We also assume that there is no external magnetic field and set the vector potential $\mathbf{A}(\mathbf{r})$ to zero.

The external electrostatic field $V_{ext}(\mathbf{r})$ is taken to be a parabolic potential. This is a much used approximation in theoretical physics, since the shape of any conservative potential is approximately parabolic close to the minima. With these assumptions, the one-body part of the Hamiltonian becomes

$$\hat{H}_{H0} = \hat{T} + \hat{V}_{ext}, \tag{4.3}$$

$$\hat{T} \to \sum_i \frac{1}{m} \mathbf{p}_i = -\frac{\hbar^2 \nabla^2}{2m}, \tag{4.4}$$

$$\hat{V}_{ext} \to \frac{m\omega^2}{2} \mathbf{R}^2, \, \mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_N), \tag{4.5}$$

where ω is the oscillator frequency, $\hbar$ is the reduced Planck constant and m is the electron mass. To simplify the expressions we choose to measure energies in units of $\hbar\omega$ and lengths in the units of $(\hbar/(m\omega))^{1/2}$. The single particle Hamiltonian can then be written on the dimensionless form

$$\hat{H}_{HO}(\mathbf{r}) = \frac{1}{2}\mathbf{r}^2 - \frac{\nabla_{\mathbf{r}}^2}{2}, \tag{4.6}$$

and the Coulomb operator can be written

$$\hat{V} \to \sum_{i<j} V(\mathbf{r}_i, \mathbf{r}_j) = \sum_{i<j} \frac{\lambda}{r_{ij}}, \quad r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|, \lambda = \frac{1}{\hbar\omega} \left(\frac{e^2}{4\pi\epsilon_0 \epsilon} \right). \tag{4.7}$$

4.3 The single particle wave functions

We will first take a look at the eigenstates of the single particle Hamiltonian $\hat{H}_0$ which we will refer to as spin orbitals or simply as the single particle wave functions. We will later use these to construct the many particle basis using the same strategy as we discussed in the previous chapters. Since the Hamiltonian has no spin coupling, the electron spins does not affect the description of the single electrons. Therefore, we will leave them out for now and explicitly include them in the mathematical description in the next section where we discuss the many particle basis.

We denote the single particle wave functions $\{\varphi_1, \varphi_2, \ldots\}$. These are found by first noting that the single particle Hamiltonian can be written as a sum

$$\hat{H}_{HO} = \frac{1}{2} \sum_i \left(x_i^2 - \frac{\delta^2}{\delta x_i^2} \right) + \frac{1}{2} \left(y_i^2 - \frac{\delta^2}{\delta y_i^2} \right), \, \mathbf{r}_i = (x_i, y_i). \tag{4.8}$$

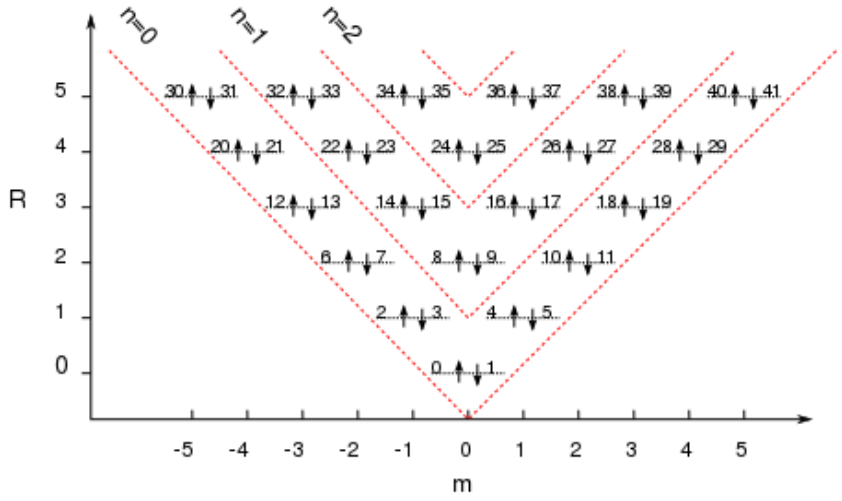


Figure 4.1: Illustration of the single particle basis, and an example of how the spin orbitals can be indexed. Each arrow ↑ corresponds to a spin $+1/2$ state, and each ↓ corresponds to a spin $-1/2$ state. Each vertical line corresponds to an orbital with a given magnetic quantum number m and principal quantum number n. The numbers next to the arrows corresponds to the index of the spin orbital. For example, φ_0 has spin $-1/2$, $m = n = 0$ and φ_{10} has spin $+1/2$ and $n = 0$, $m = 2$.

Thus the solution is separable, and can be written as a product of functions that depends only on one of the variables $\{x_i\}$ or $\{y_i\}$. These are well known equations which describe the behaviour of the so called quantum harmonic oscillator [31], and the solutions can be represented in many ways. Because of the rotational symmetry of the quantum dots, it is often convenient to represent the single particle wave functions as the eigenfunctions of the angular momentum operator $\hat{L}_Z$ which can be shown to commute with $\hat{H}_{H0}$. In two dimensions and in polar coordinates φ_i can be written

$$\varphi_i(\mathbf{r}) \to \varphi_{n,m}(r,\theta) = \left[\frac{2n!}{(n+|m|)!}\right]^{1/2} \frac{1}{2\pi} e^{im\theta} r^{|m|} L_n^{|m|}(r^2) e^{-r^2/2}, \quad r = |\mathbf{r}|\,, \tag{4.9}$$

which are the so called Fock- Darwin orbitals, and where L_m are the Laguerre polynomials. Each i is mapped to a unique pair of quantum numbers (n,m), where m is the magnetic quantum number and n is the principal quantum number

$$n \in 0,1,2,\ldots, \qquad m \in \{-n,-n+2,\ldots,n\}. \tag{4.10}$$

The single particle energy can be shown to be

$$\int dr\, r d\theta\, \varphi^\dagger_{n,m}(r,\theta) H_{HO}(r,\theta) \varphi_{n,m}(r,\theta) = 2n + |m| + 1, \tag{4.11}$$

in units of $\hbar\omega$.

4.4 The many body wave functions

The many body wave functions are constructed from the harmonic oscillator eigenfunctions. But to describe the many particle system, we must also include spin. Each single particle

wave function has a triplet of quantum numbers (m, n, s) which uniquely specifies the quantum state.

$$\varphi_i = \varphi_{m_i,n_i} \otimes \chi_{s_i}, \tag{4.12}$$

where χ_{s_i} is the spinor. The N particle wave function is the determinant

$$|D_i\rangle = |\varphi_{i_1}, \ldots, \varphi_{i_N}\rangle \equiv \frac{1}{N!}\det(\varphi_{i_1}, \ldots, \varphi_{i_N}), \tag{4.13}$$

which we will write in second quantized form as

$$|D_i\rangle = a^\dagger_{i_1} \ldots a^\dagger_{i_N}|\,\rangle. \tag{4.14}$$

4.5 The Normal ordered Hamiltonian

To simplify the calculation of different expectation values, we want to express the Hamiltonian on normal ordered form in the particle-hole formalism. Our Hamiltonian can be written on the general form

$$\hat{H} = \hat{H}_{HO} + \hat{V} = \sum_{pq}\langle\varphi_p|\hat{H}_{HO}|\varphi_q\rangle a^\dagger_p a_q + \frac{1}{4}\sum_{pqrs}\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS} a^\dagger_p a^\dagger_q a_s a_r. \tag{4.15}$$

We will now rewrite the Hamiltonian in normal ordered form in the particle-hole formalism. We define the contractions

$$\overbrace{a^\dagger_p a_q} = \langle D_\alpha|a^\dagger_p a_q - \{a^\dagger_p, a_q\}|D_\alpha\rangle, \tag{4.16}$$

where $|D_\alpha\rangle$ can be any state

$$|D_\alpha\rangle = a^\dagger_{\alpha_1} a^\dagger_{\alpha_2} \ldots a^\dagger_{\alpha_N}|\,\rangle. \tag{4.17}$$

We use the convention that the indices

$$p, q, r, s \in [0, 2M], \tag{4.18}$$

can represent any single particle wave function while

$$\begin{aligned} &i, j, k, l \in \{\alpha_1, \alpha_2, \ldots, \alpha_N\}, \\ &a, b, c, d \notin \{\alpha_1, \alpha_2, \ldots, \alpha_N\}. \end{aligned} \tag{4.19}$$

Here, $2M$ is the total number of spin orbitals, and $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$ are the indices of the occupied orbitals. Note that we do not necessarily take i, j, k, l to be states below the Fermi level.

According to Wick's theorem and using the fact that $|\varphi_p\rangle$ are the eigenfunctions of $\hat{H}_{HO}$ we arrive at the expression

$$\hat{H}_{HO} = \sum_{pq}\langle\varphi_p|\hat{H}_{HO}|\varphi_q\rangle\left(\{a^\dagger_p a_q\} + \{\overbrace{a^\dagger_p a_q}\}\right) = \sum_p\langle\varphi_p|\hat{H}_{HO}|\varphi_p\rangle\{a^\dagger_p a_p\} + \sum_i\langle\varphi_i|\hat{H}_{HO}|\varphi_i\rangle. \tag{4.20}$$

The Coulomb energy operator $\hat{V}$ is also rewritten on normal form using Wick's theorem. Only the nonzero contractions are included in the derivation

$$\hat{V} = \frac{1}{4}\sum_{pqrs}\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS} \times \Big(\{a_p^\dagger a_q^\dagger a_s a_r\} + \{\overbracket{a_p^\dagger a_q^\dagger} a_s a_r\} + \{\overbracket{a_p^\dagger a_q^\dagger a_s a_r}\}$$
$$+ \{a_p^\dagger \overbracket{a_q^\dagger a_s} a_r\} + \{a_p^\dagger \overbracket{a_q^\dagger a_s a_r}\} + \{\overbracket{a_p^\dagger a_q^\dagger} a_s a_r\} + \{\overbracket{a_p^\dagger a_q^\dagger a_s} a_r\} \Big) \tag{4.21}$$
$$= \frac{1}{4}\sum_{pqrs}\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS}\{a_p^\dagger a_q^\dagger a_s a_r\} + \sum_{pqi}\langle\varphi_p\varphi_i|\hat{V}|\varphi_q\varphi_i\rangle_{AS}\{a_p^\dagger a_q\}$$
$$+ \frac{1}{2}\sum_{ij}\langle\varphi_i\varphi_j|\hat{V}|\varphi_i\varphi_j\rangle_{AS}. \tag{4.22}$$

The total Hamiltonian can now be written

$$\hat{H} = \sum_p\langle\varphi_p|\hat{H}_{HO}|\varphi_p\rangle\{a_p^\dagger a_p\} + \frac{1}{4}\sum_{pqrs}\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS}\{a_p^\dagger a_q^\dagger a_s a_r\}$$
$$+ \sum_{pqi}\langle\varphi_p\varphi_i|\hat{V}|\varphi_q\varphi_i\rangle_{AS}\{a_p^\dagger a_q\} + E_\alpha \tag{4.23}$$
$$E_\alpha = \sum_i\langle\varphi_i|\hat{H}_{HO}|\varphi_i\rangle + \frac{1}{2}\sum_{ij}\langle\varphi_i\varphi_j|\hat{V}|\varphi_i\varphi_j\rangle_{AS}. \tag{4.24}$$

4.6 The Hamiltonian matrix elements

We will now use the normal ordered Hamiltonian to find the general expressions for the Hamiltonian matrix elements $\langle D_\alpha|\hat{H}|D_\beta\rangle$. From Eq. (4.24) we see that only states that differs by less than three orbitals are connected which means that all non zero matrix elements can be written either as a diagonal element

$$\langle D_\alpha|\hat{H}|D_\alpha\rangle, \tag{4.25}$$

as the amplitude of a single excitation

$$\langle D_\alpha|\hat{H}\{a_a^\dagger a_i\}|D_\alpha\rangle, \tag{4.26}$$

or as the amplitude of a double excitation

$$\langle D_\alpha|\hat{H}\{a_a^\dagger a_b^\dagger a_i a_j\}|D_\alpha\rangle. \tag{4.27}$$

Here we have used the same conventions for the indexing as in Eq. (4.19), with $|D_\alpha\rangle$ as the reference state. The closed form expressions of the matrix elements are calculated by inserting the normal ordered Hamiltonian (4.24). For the diagonal elements we obtain the expression

$$\langle D_\alpha|\hat{H}|D_\alpha\rangle = \sum_i\langle\varphi_i|\hat{H}_{HO}|\varphi_i\rangle + \frac{1}{2}\sum_{ij}\langle\varphi_i\varphi_j|\hat{V}|\varphi_i\varphi_j\rangle_{AS}. \tag{4.28}$$

The single excitations have the amplitude

$$\langle D_\alpha|\hat{H}\{a_a^\dagger a_i\}|D_\alpha\rangle = \sum_{pqj}\langle\varphi_p\varphi_j|\hat{V}|\varphi_q\varphi_j\rangle_{AS}\{\overbracket{a_p^\dagger a_q}\}\{\overbracket{a_a^\dagger a_i}\} = \sum_j\langle\varphi_a\varphi_j|\hat{V}|\varphi_i\varphi_j\rangle_{AS}, \tag{4.29}$$

and the double excitations have the amplitude

$$\langle D_\alpha|\hat{H}\{a_a^\dagger a_b^\dagger a_i a_j\}|D_\alpha\rangle = \frac{1}{4}\sum_{pqrs}\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS}$$
$$\times\Big(\{a_p^\dagger a_q^\dagger a_s a_r\}\{a_a^\dagger a_b^\dagger a_i a_j\} + \{a_p^\dagger a_q^\dagger a_s a_r\}\{a_a^\dagger a_b^\dagger a_i a_j\}$$
$$+\{a_p^\dagger a_q^\dagger a_s a_r\}\{a_a^\dagger a_b^\dagger a_i a_j\} + \{a_p^\dagger a_q^\dagger a_s a_r\}\{a_a^\dagger a_b^\dagger a_i a_j\}\Big)$$
$$= \langle\varphi_a\varphi_b|\hat{V}|\varphi_i\varphi_j\rangle_{AS}. \tag{4.30}$$

The single particle energies $\langle\varphi_i|\hat{H}_{HO}|\varphi_i\rangle$ are calculated using Eq. (4.11) while the anti-symmetric Coulomb matrix elements $\langle\varphi_i\varphi_j|\hat{V}|\varphi_i\varphi_j\rangle_{AS}$ are defined as the integrals

$$\langle\varphi_p\varphi_q|\hat{V}|\varphi_r\varphi_s\rangle_{AS} = I_{pqrs}\langle\chi_{s_p}|\chi_{s_r}\rangle\langle\chi_{s_q}|\chi_{s_s}\rangle - I_{pqsr}\langle\chi_{s_p}|\chi_{s_s}\rangle\langle\chi_{s_q}|\chi_{s_r}\rangle, \tag{4.31}$$
$$I_{klpq} \equiv \frac{1}{4}\int d\mathbf{r}_1 d\mathbf{r}_2 \varphi_k^\dagger(\mathbf{r}_1)\varphi_l^\dagger(\mathbf{r}_2)V(\mathbf{r}_1,\mathbf{r}_2)\varphi_p(\mathbf{r}_1)\varphi_q(\mathbf{r}_2). \tag{4.32}$$

where the spinors are orthonormal and obey the relation $\langle\chi_{s_1}|\chi_{s_2}\rangle = \delta_{s_1 s_2}$. Kvaal (2008) has shown how the integrals I_{klpq} can be efficiently calculated in Ref. [15]. In our implementation, we have used use the open source C++ library `OpenFCI`, which is described in the same article, to calculate these integrals.

Part II

Numerical many body methods

Quantum many body systems are notoriously difficult to solve, and only the simplest systems, such as the two particle quanum dot, can be solved on a closed form. More complex systems are always solved using numerical methods, and in most cases some approximation or simplification must be done to reduce the computational cost to a realistic level. Typical examples of such approximations are basis truncations, meaning that the only a subspace of the full state space is used. Another example is the fixed node approximation of Diffusion Monte Carlo that we will discuss later in this chapter.

In this part we will take a closer look at three classes of many body methods, namely Full Configuration Interaction Theory (FCI), Projector Monte Carlo (PMC) methods and Hartree-Fock (HF) methods. These use different approaches to calculate ground state values of different many body systems, and have different advantages and weaknesses. The reason why we want to present FCI and PMC is that the Full Configuration Interaction Quantum Monte Carlo Method (FCIQMC), which we will present in the next part, can be seen as a hybrid between these approaches. It shares some of the strengths and weaknesses of both methods, and it is therefore instructive to take a closer look at these first. The HF method is introduced because it provides a method to unitarily transform the spin orbitals to a more favourable basis, which we hope will speed up the convergence of our simulations.

Chapter 5

The Full Configuration Interaction method

5.1 Introduction to the method

In this section we will introduce a numerical method which is called Full Configuration Interaction Theory (FCI). This is the conceptually simplest of the many body methods and yields exact solutions to the stationary Schrödinger equation within a given subspace of the full state space. To be more specific, FCI solves the eigen problem

$$\hat{P}_R \hat{H} \hat{P}_R |\Psi_i\rangle = \varepsilon_i |\Psi_i\rangle, \tag{5.1}$$

where $\hat{P}_R$ is a projection operator [1] which projects the full Hilbert space onto a subspace which we call the FCI space. These subspaces can be defined in several ways, and two commonly used FCI spaces are the energy cut space $\mathscr{P}_R$ and the direct product space $\mathscr{M}_R$, which in our case are defined as

$$\mathscr{P}_R = \text{span}\left\{|D_i\rangle\,;\, \langle D_i|\hat{H}_{HO}|D_i\rangle \leq R+1\right\}, \tag{5.2}$$

$$\mathscr{M}_R = \text{span}\left\{|D_i\rangle = |\varphi_{n_1}\varphi_{n_2}\ldots\varphi_{n_N}\rangle\,;\, \langle\varphi_{n_i}|\hat{H}_{HO}|\varphi_{n_i}\rangle \leq R+1\right\}, \tag{5.3}$$

where $|\varphi_i\rangle$ are the harmonic oscillator eigenfunctions as defined earlier. Note that it is not clear which of these truncations give the most accurate result in terms of the dimensionality of the subspace. Although it can be argued that more physical relevant states are included in the energy cut spaces, it is not obvious which of the truncations that is the most favourable [14].

According to the variational principle, solving the eigen problem Eq. (5.1) is analogous to solving the minimization problem [8]

$$E(R) = \min(\langle\Psi|\hat{P}_R\hat{H}\hat{P}_R|\Psi\rangle). \tag{5.4}$$

This has two important implications. The first is that $E(R_1) \leq E(R_2)$ when $R_1 > R_2$, such that the FCI ground state energy is either improved or unchanged when we increase the size of the FCI space. Thus the FCI ground state energy can be systematically improved by

[1] The projection $\hat{P}$ onto the subspace span$\{|\varphi_i\rangle\}$ is defined as $\hat{P} = \sum_i |\varphi_i\rangle\langle\varphi_i|$. Note that $\hat{P}^\dagger = \hat{P} = \hat{P}\hat{P}$. The projection onto the excluded or truncated space $\hat{Q} = 1 - \hat{P}$ has the same properties.

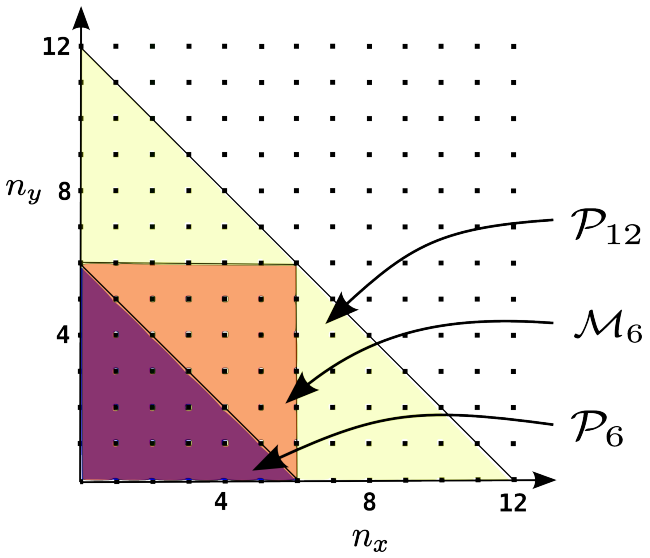


Figure 5.1: Modelspaces $\mathcal{P}_6$, $\mathcal{M}_6$ and $\mathcal{P}_{12}$ for quantumdots with $N_P = 1$ particle. Here $\mathcal{P}_N$ and $\mathcal{M}_N$ is the energy cut space and the direct product space respectively, truncated on $N + 1$ shells. Note that $\mathcal{P}_6 \subset \mathcal{M}_6 \subset M_{12}$ and that in general $\mathcal{P}_N \subset \mathcal{M}_N \subset M_{NN_P}$ for all N_P

increasing the number of shells in the truncated basis which makes it possible to check if the energies have converged. And secondly, the method is variational, such that any FCI energy sets an upper bound to the energies of a system. Variational methods makes it possible to do a systematic search for lower ground state energies, knowing that the lower result always is the better one.

Implementations of the FCI algorithm involves large scale diagonalization of the Hamiltonian matrix

$$H_{ij} = \langle D_i | \hat{P}_R \hat{H} \hat{P}_R | D_j \rangle. \tag{5.5}$$

The most common method is to use the Lanczos algorithm [8], which requires at least two vectors of length equal to the dimension of the FCI space to be stored. The problem is that the dimensionality of the FCI spaces scales as

$$\binom{M}{N}, \tag{5.6}$$

where N is the number of particles and M is the number of spin orbitals in the basis [12]. This limits the FCI algorithm to small systems with only a few particles or crude approximations with a low R. As an example we look at the 6 particle quantum dot in 20 shells. The basis is filtered such that only the states with $m = s = 0$ are included. The filtered basis has a direct product space with dimensionality $\dim(\mathscr{M}_{19}) \sim 4 \times 10^{10}$ (see Tab. 5.1) which is already at the absolute upper limit of what an FCI implementation can tackle on a modern super computer [20].

5.2 Error analysis of the FCI energies

Kvaal (2009)[16] has analyzed the truncation error of the FCI method with the harmonic oscillator basis in energy cut spaces and with Coulomb particle- particle interactions. A

R	$\dim(\mathscr{M}_R)$
1	1
2	64
3	1490
4	16451
5	115148
6	594118
7	2459910
8	8621028
9	26502841
10	73293890
11	185741360
12	437408455
13	967633556
14	2028188748
15	4055864256
16	7782042912
17	14393796941
18	25765329848
19	44783684274
20	75798808967

Table 5.1: This table shows the dimension of "filtered" FCI spaces with different R for a six particle quantum dot. By filtered we mean that only the physically relevant states with $m = s = 0$ are included. At R = 20, the dimensionality is already at the upper limit of what is possible to simulate with FCI on high level supercomputing facilities. It is worth mentioning that the FCI energy at R = 19 shells with the harmonic oscillator basis is $20.17H^*$, approximately $10mH^*$ higher that the true ground state energy at approximately $20.16H^*$. These results are listed in chapter .

parametrization of the approximate error of the energy as function of R is also derived. Without going into details, we will present Kvaals error estimate. Kvaal has shown that the following relation holds for two dimensional systems

$$\Delta E(R) \leq [1 + \nu(R)] \sum_{r=R+1}^{\infty} (N_P + r)\, r^{-c}, \tag{5.7}$$

where $\Delta E(R)$ is the error in the energy $E(\infty) - E(R)$ resulting from the basis truncation, c is a real constant, $\nu(R)$ is an unknown function with the property

$$\nu(R) \underset{R\to\infty}{\longrightarrow} 0, \tag{5.8}$$

and N_P is the number of particles. Kvaal has analyzed the error in the energy cut basis, but his results will also apply to FCI energies calculated in the direct product basis. The reason for this is that the direct product basis energies are bounded from below and above by energy cut energies since

$$\mathscr{P}_R \subset \mathscr{M}_R \subset \mathscr{P}_{N_P R}. \tag{5.9}$$

In other words, the direct product basis error will be somewhere in the range

$$[\Delta E(R), \Delta E(N_P R)].$$

The Hilbert spaces $\mathscr{M}_R$ and $\mathscr{P}_R$ is much closer in dimensionality than $\mathscr{M}_R$ and $\mathscr{P}_{N_P R}$. We will therefore assume that we can use the same error formula for the spaces $\mathscr{M}_R$ and $\mathscr{P}_R$. As we will see later, this formula seems to hold very well for the systems we are studying.

We will also assume that $\nu(R)$ is negligible and rewrite Eq. (5.7) on the form

$$\Delta E \approx b \sum_{r=R+1}^{\infty} (N_P + r)\, r^{-c}, b \in \mathbb{R}. \tag{5.10}$$

This assumption is justified by Kvaal's [16] results. He has performed FCI calculations on two dimensional quantum dots in the energy cut basis with $3, 4$ and 5 particles, and for these systems $\nu(R)$ is shown to be small. Also, since $\mathscr{P}_R \subset \mathscr{M}_R$, we expect $\nu(R)$ to decay even faster in the direct product basis.

5.3 Extrapolation Formulas

We have used Eq. (5.10) to parametrize the energy as a function of the number of shells.

$$E(R) \approx a - b \sum_{r=1}^{R} (N_P + r) r^{-|c|}, \tag{5.11}$$

where a, b, c are real constants. This formula has an error $\sim E(R)\nu(R)$, and consequently it will become more accurate as R increases and $\nu(R)$ becomes small. This means that we should avoid extrapolations with energies $\langle \hat{P}_R \hat{H} \hat{P}_R \rangle$ for small R.

We also want to find the extrapolated energy at $R \to \infty$ given that we know the optimal values of a, b, c. We notice that $E(\infty)$ can be written in terms of the Riemann zeta function $\zeta(c)$ such that

$$\lim_{R\to\infty} E(R) \approx a - b[N_P \zeta(c) + \zeta(c-1)], \tag{5.12}$$

where $\zeta(c) = \sum_{r=1}^{\infty} r^{-c}$, which is known to converge for all $c > 1$.

Chapter 6

Projector Monte Carlo Methods

6.1 Introduction

The projector Monte Carlo (PMC) Methods includes a range of methods which are tailored to calculate properties of low energy states of quantum mechanical systems, and includes algorithms like Diffusion Monte Carlo (DMC), Green's Function Monte Carlo and the recently developed Full Configuration Interaction Quantum Monte Carlo (FCIQMC). These methods are all based on the same theoretical foundation, namely that the so called projection operator $\hat{P}_t$, will project out the lowest energy state in the Hilbert space. The projection operator is defined as

$$\hat{P}_t = e^{-t(\hat{H}-S)},\ S \in \mathbb{R}, \tag{6.1}$$

where S is a real shift of the energy which we will soon come back to, and the projected state is

$$|\Psi(t)\rangle = \hat{P}_t|\Psi_0\rangle. \tag{6.2}$$

Assumed that the initial state $|\Psi_0\rangle$ has a nonzero overlap with the ground state, the large time $|\Psi(t)\rangle$ will be proportional to the ground state. To see why this is so we consider the general wave function

$$|\Psi\rangle = \sum_i c_i|\psi_i\rangle, \tag{6.3}$$

where $\{|\psi_i\rangle\}$ is the eigenfunctions of the Hamiltonian $\hat{H}$ with the corresponding eigenvalues $\{E_i\}$. The projected state can now be written

$$|\Psi(t)\rangle = \hat{P}_t|\Psi\rangle = \sum_i c_i|\psi_i\rangle e^{-t(E_i-s)}. \tag{6.4}$$

The vector $c_i|\psi_i\rangle e^{-t(E_i-S)}$ will decay to zero if $E_i > S$, will be constant if $E_i = S$ and will increase exponentially if $E_i < S$. In the case that $S = E_0$ and the overlap between the initial state and the ground state c_0 is nonzero, it is evident that

$$\lim_{t\to\infty} \hat{P}_t|\Psi\rangle = \lim_{t\to\infty} \sum_i e^{-(E_i-S)t} c_i|\psi_i\rangle \to c_0|\psi_0\rangle. \tag{6.5}$$

Also note that in the case of a $S \neq \varepsilon_0$, the ground state will still dominate since the ground state always has the greatest exponent. The basic idea of PMC algorithms is to solve the above equation by expressing the ground state as Markov chain, and to find the steady state solution iteratively. This can be done by noting that

$$\hat{P}_{n\tau} = e^{-(\hat{H}-S)n\tau} = (\hat{P}_\tau)^n, \tag{6.6}$$

where τ is a small time step and n is a positive integer. By repeatedly multiplying with the operator $\hat{P}_\tau$ we arrive at the large t solution

$$\lim_{n\to\infty} (\hat{P}_\tau)^n |\Psi\rangle \to |\psi_0\rangle. \tag{6.7}$$

From this equation we see that we can express the time evolution of the projected state as a Markov chain

$$|\Psi^{(n+1)}\rangle \leftarrow \hat{P}_\tau |\Psi^{(n)}\rangle, \text{ where } \lim_{n\to\infty} |\Psi^{(n)}\rangle \to k|\psi_0\rangle,\ k \in \mathbb{C}. \tag{6.8}$$

where the steady state solution is the ground state. This equation is the starting point for all PMC algorithms. In general, the different PMC algorithms utilize different methods to approximate the projection operator. For example in FCIQMC, the projection operator is approximated as a first order Taylor expansion, while in DMC it is approximated as a Green's function.

We will look closer at two Projector Monte Carlo algorithms, namely DMC and FCIQMC. We will first look at DMC and discuss some of the advantages and shortcomings of this algorithm. This will also serve as an introduction to the next part where we give a thorough presentation of the FCIQMC algorithm.

6.2 Diffusion Monte Carlo

In this section we will explain the basics of the DMC algorithm and also discuss the main difficulties and sources of error.

6.2.1 The projection operator and the short time approximation

DMC finds the projected state by solving the integral

$$\Psi(\mathbf{R}, t) = \int G(\mathbf{R}, \mathbf{R}', t)\Psi(\mathbf{R}')d\mathbf{R}', \tag{6.9}$$

where G is the Green's function

$$G(\mathbf{R}, \mathbf{R}', t) = \langle R|e^{-t(\hat{H}-S)}|R'\rangle = \sum_i e^{-t(E_i-S)}\psi_i(\mathbf{R}')\psi_i^*(\mathbf{R}), \quad \Psi(R) = \langle R|\Psi\rangle. \tag{6.10}$$

Because the eigenfunctions $|\psi_i\rangle$ are orthogonal

$$\int \psi_i(\mathbf{R})\psi_j^*(\mathbf{R})dR = \langle\psi_i|\psi_j\rangle = \delta_{ij}, \tag{6.11}$$

it is straight forward to show that

$$\int G(\mathbf{R}, \mathbf{R}', t)\Psi(\mathbf{R})d\mathbf{R}' = \sum_i c_i\psi_i(\mathbf{R})e^{-t(\varepsilon_i-S)}, \tag{6.12}$$

which is the definition of the projected state in Eq. (6.4).

This integral can be sampled directly using Monte Carlo techniques, but this has proven to be an inefficient approach, mainly because the Green's function contains singularities that makes the simulations very unstable [10]. It is known to be a more efficient approach to use a modified Green's function and to solve the integral

$$f(\mathbf{R},t) = \int \tilde{G}(\mathbf{R},\mathbf{R}',t)f(\mathbf{R}',0)d\mathbf{R}', \tag{6.13}$$

where

$$f(\mathbf{R},t) = \Psi_T(\mathbf{R})\Psi(\mathbf{R},t), \tag{6.14}$$

$$\tilde{G}(\mathbf{R},\mathbf{R}',t) = \Psi_T(\mathbf{R})G(\mathbf{R},\mathbf{R}',t)\frac{1}{\Psi_T(\mathbf{R}')}. \tag{6.15}$$

The trial wave function Ψ_T must be chosen as a wave function which closely resembles the ground state for this approach to be efficient. As we see, $\Psi(\mathbf{R},t)$ has exactly the same time dependency as before.

By assuming that the Green's function is constant during a time interval τ, it can be shown that it can be written on the form [39]

$$\tilde{G}(\mathbf{R}_1,\mathbf{R}_2,\tau) = \mathcal{M}e^{-\frac{1}{2\tau}\left|\mathbf{R}_2-\mathbf{R}_1-\frac{1}{2}\mathbf{F}(\mathbf{R}_1)\right|^2}e^{-\tau\left[\frac{E_L(\mathbf{R}_2)+E_L(\mathbf{R}_1)}{2}-S\right]}+\mathcal{O}(\tau^2). \tag{6.16}$$

where $\mathcal{M}$ is a normalization factor. This is known as the short time approximation and is, as we will discuss later, one of two main sources of numerical error. The first factor in eq. (6.16) is a Gaussian distribution

$$\mathcal{N}(\mathbf{R}_2,\mathbf{R}_1,\tau) = e^{-\frac{1}{2\tau}\left|\mathbf{R}_2-\mathbf{R}_1-\frac{1}{2}\mathbf{F}(\mathbf{R}_1)\right|^2}, \tag{6.17}$$

where $\mathbf{F}$ is the so called quantum force

$$\mathbf{F}(\mathbf{R}) = 2\frac{\nabla\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}. \tag{6.18}$$

The second factor is the so called the weight function

$$\mathcal{W}(\mathbf{R}_2,\mathbf{R}_1,\tau) = e^{-\tau\left[\frac{E_L(\mathbf{R}_2)+E_L(\mathbf{R}_1)}{2}-E_T\right]}, \tag{6.19}$$

where E_L is the local energy

$$E_L(\mathbf{R}) = \frac{\hat{H}\Psi(\mathbf{R})}{\Psi(\mathbf{R})}. \tag{6.20}$$

The trial wave function is constructed such that E_L has no singularities, typically by introducing the so called Jastrow factor, since this reduces the numerical instabilities of the DMC algorithm.

6.2.2 The Diffusion Monte Carlo Algorithm

In DMC, the function $f(\mathbf{R}, t)$ is represented as a distribution of walkers with the coordinates $\{\mathbf{R}^i\}_{i=1}^N$, such that

$$f(\mathbf{R}, t) \approx \int d\mathbf{R} \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{R} - \mathbf{R}^i), \tag{6.21}$$

which is only exact in the limit $N \to \infty$. The walkers follow a set of dynamical rules which are derived from Eqs. (6.16), (6.13), and will converge to a distribution which is proportional to $\Psi_T(\mathbf{R})\psi_0(\mathbf{R})$. We will now summarize the DMC algorithm in a few steps to illustrate how it works.

(i): First, the initial distribution is chosen to be $f(\mathbf{R}, 0) = |\Psi(\mathbf{R})|^2$, which should be close to the ground state. Such distributions are typically obtained using variational techniques like Variational Monte Carlo [10]. A number of N samples (or walkers) $\{\mathbf{R}_0^i\}_{i=1}^N$ are drawn from this distribution.

(ii): During a single time step, f evolves according to the equation

$$\begin{aligned} f(\mathbf{R}, \tau(n+1)) &= \int d\mathbf{R}' \tilde{G}(\mathbf{R}, \mathbf{R}', \tau) f(\mathbf{R}', \tau n) \\ &= \int d\mathbf{R}' \mathcal{N}(\mathbf{R}, \mathbf{R}', \tau) \mathcal{W}(\mathbf{R}, \mathbf{R}', \tau) f(\mathbf{R}', \tau n). \end{aligned} \tag{6.22}$$

In the limit of a large number of walkers, $f(\mathbf{R}, \tau(n+1))$ is proportional to the distribution

$$\{\mathcal{W}(\mathbf{R}_{n+1}^i, \mathbf{R}_n^i, \tau)\}_{i=1}^N, \tag{6.23}$$

where $\mathbf{R}_{n+1}^i$ is sampled from the probability distribution $\mathcal{N}(\mathbf{R}, \mathbf{R}_n^i)$. Instead of storing the weights $\mathcal{W}$ each walker is either cloned or removed from the simulation according to the branching rule:

- Calculate the number $M_i = \text{floor}(\mathcal{W}(\mathbf{R}_{n+1}^i, \mathbf{R}_n^i, \tau) + (\mathcal{U} - 1/2))$ where $\mathcal{U}$ is a random uniform between 0 and 1.
- If $M_i = 0$ the i'th walker is removed from the simulation. If $M > 1$, $M - 1$ new walkers are added to the simulation in the same coordinate as the i'th walker.

In the limit of a large number of walkers, this yields the correct results since

$$\langle \text{floor}(a + (\mathcal{U} - 1/2) \rangle = a, \tag{6.24}$$

for any number a.

(iii): The shift S should be equal to E_0 to project out the ground state. But since E_0 is unknown S is adjusted to stabilize the population of walkers. From Eq. (6.4) we see that a large S results in a population growth and vice versa, and that the S that keeps the number of walkers constant is the ground state energy. For this purpose it is common to use an equation on the form [10]

$$S \leftarrow S - \frac{k}{\tau} \log(\frac{N_W}{N_P}), \tag{6.25}$$

where N_W is the current number of walkers, N_P is the desired number of walkers and k is some real number which should be chosen such that the fluctuations of S are minimized.

(iv): Steps (ii), (iii) are repeated a number of times to thermalize the distribution. After this step, the distribution should be $\Psi_T(\mathbf{R})\psi_0(\mathbf{R})$.

(v): When the thermalization is finished, one starts collecting observables. As an example, the energy can be found by taking the average of the shift S during a simulation.

(vi): Steps (ii), (iii) and (v) are repeated until a sufficient number of samples are collected.

There are many details and possible optimizations that are left out here, but a thorough discussion of this algorithm can be found in Res. [39].

6.2.3 Systematic errors

DMC is subject to a time step error which stems from the short time approximation of the Green's function, but this problem can often be overcome by extrapolating the results to the limit $\tau \to 0$. More severe is the so called sign problem, which is the main difficulty of DMC and PMC methods in general, and which is connected to the antisymmetry of fermionic wave functions. As we remember, fermionic wave functions with identical particles must be antisymmetric with regard to the exchange of two particles, which means that such wave functions has regions of both negative and positive sign. These regions are separated by the nodal surfaces where $\psi_0 = 0$ and changes its sign (has a nonzero gradient). In fact, the product $f = \Psi_T\psi_0$ should be negative in the regions where Ψ_T and ψ_0 have different signs. From our interpretation of f as a distribution of walkers, it is restricted to be positive everywhere, meaning that our algorithm only can be exact in the case where Ψ_T and ψ_0 have exactly the same nodal surfaces or ψ_0 is positive everywhere (bosonic systems).

The first to propose a method to deal with the sign problem was Anderson (1975) [1] who set $\mathcal{W} = 0$ (removing the walker) for any step crossing a node of the trial wave function, thus imposing the nodal surfaces of the trial function. By using the fixed node approximation, the assumption is made that Ψ_T has the same nodes as ψ_0. This is the only fundamental assumption made in the theory behind DMC, and if the exact nodes ψ_0 where known, DMC would in principle be exact for systems of identical fermions.

The fixed node approximation is shown to produce energies that are higher than the ground state [27], thus the method is variational. However, since there are no systematic methods of improving the nodes, the method is not systematically improvable and this means that it is difficult to control the fixed node error.

Chapter 7

The Hartree-Fock method

7.1 Overview

The Hartree- Fock (HF) method [12], is an approximate method to solve the Schrödinger equation, and can be seen as a mean field approximation where the interactions between the particles are replaced by an averaged interaction. The result is that the many particle problem is effectively reduced to a single particle problem where each particle only sees the mean field. The HF method is numerically cheap, but since the method includes only selected correlations, it does in general provide less accurate results than FCI or the DMC.

Formally, the HF ansatz assumes that the wave function can be written as a single determinant

$$|\Psi_{HF}\rangle = |\tilde{\varphi}_1, \ldots, \tilde{\varphi}_{N_P}\rangle, \tag{7.1}$$

where N_P is the number of particles in the system. The HF determinant $|\Psi_{HF}\rangle$ is the determinant that solves the equation

$$E_{HF} = \min\left[\langle\Psi_{HF}|\hat{H}|\Psi_{HF}\rangle\right]. \tag{7.2}$$

According to the variational principle, this is the single determinant with the energy that is closest to the ground state energy. In practice, the HF determinant can be found by varying the single particle basis $\{\varphi_j\}_{i=1}^{M}$

$$|\tilde{\varphi}_i\rangle = \sum_j C_{ij}|\varphi_j\rangle, \tag{7.3}$$

where the basis $\{\tilde{\varphi}_j\}_{i=1}^{M}$ that solves Eq. (7.2) is the so called HF basis, and C_{ij} is unitary in the sense that

$$\sum_j C_{ij}^* C_{jk} = 1. \tag{7.4}$$

Note that unitarity assures that expectation values and the normalization of the wave function are conserved.

The reason why we are interested in the HF method is that it can be used to optimize the single particle basis. The HF basis yields a better starting point for other numerical methods compared with the harmonic oscillator basis. The HF basis is routinely used together with Diffusion Monte Carlo [22], Coupled Cluster Theory [24], Full Configuration Interaction Quantum Monte Carlo [3] as well as other numerical many body methods.

7.2 The Roothaan-Hartree-Fock equations

In this section we will take a closer look at how the minimization of the single determinant energy can be performed numerically. We will see that the most costly operation is the diagonalization of an $M \times M$ matrix, where M is the number of single particle orbitals in the basis. Note that M never becomes very large, and that even for simulations with 30 shells, M does not exceed 10^3.

For a Hamiltonian on the form $\hat{H} = \hat{H}_{HO} + \hat{V}$, where $\hat{H}_{HO}$ is an one-body operator and $\hat{V}$ is a two-body operator, it can be shown that the minimimization problem of Eq. (7.2) can be written as an eigenvalue problem [30]

$$\sum_{\gamma} H^{HF}_{\alpha\gamma} C_{\kappa\gamma} = e_k C_{\kappa\alpha}, \tag{7.5}$$

$$H^{HF}_{\alpha\gamma} = \langle\varphi_\alpha|\hat{H}_{HO}|\varphi_\gamma\rangle + \sum_{\alpha=1}^{N_P}\sum_{\beta\delta} C^*_{\alpha\beta} C_{\alpha\delta} \langle\varphi_\alpha\varphi_\beta|\hat{V}|\varphi_\gamma\varphi_\delta\rangle_{AS}, \tag{7.6}$$

which can be expressed as a matrix equation

$$\mathbf{H}^{HF}\mathbf{C} = \mathbf{C}\mathbf{e}, \tag{7.7}$$

where $\mathbf{e}$ is a diagonal matrix and $\mathbf{C}$ and $\mathbf{H}^{HF}$ are matrices with elements C_{ij} and H^{HF}_{ij}. This equation is a non linear eigenvalue problem that must be solved iteratively. The columns of $\mathbf{C}$ and the diagonal elements of $\mathbf{e}$ are simply the eigenvectors and the eigenvalues of the matrix $\mathbf{H}^{HF}$, and can thus be found by diagonalizing $\mathbf{H}^{HF}$. However, $\mathbf{H}^{HF}$ is dependent on the elements of $\mathbf{C}$, and must be recalculated every time $\mathbf{C}$ is updated. This iterative procedure will converge to the self consistent solution when certain stability criteria are fulfilled. The convergence properties are discussed in Ref. [12].

Part III

Full Configuration Interaction Quantum Monte Carlo

Full Configuration Interaction Quantum Monte Carlo (FCIQMC) is an "ab initio" method which is rooted in basic quantum mechanical principles without making any assumptions or approximations. The FCIQMC algorithm promises to calculate the exact Full Configuration Interaction (FCI) value within a given Hilbert space, but usually at a much lower computational cost than FCI methods. Similar to the approach used in DMC, the ground state is found by calculating the projected state $\hat{P}_t|\Psi\rangle$, but the integration is not performed in the coordinate space but in the discrete space of Slater determinants. Because of the inherent anti- symmetric properties of this basis, the infamous sign problem is not as severe for FCIQMC as for other projector methods. See for example Ref. [13] for a thorough analysis of this subject.

The algorithm was first made famous by Booth *et. al.* (2009) [3]. They were able to find the FCI values for different molucules in very large Hilbert spaces, like for example the NaH molecule with an FCI basis of more than 10^{12} determinants. Although these results are impressive and far out of the scope of FCI, the basic FCIQMC algorithm is limited due to the unfavourable scaling with increased basis sizes. In 2010, an improvement to the algorithm was proposed by Cleland *et. al.* [4]. The improved algorithm, dubbed i-FCIQMC, made it possible to do simulations with Hilbert spaces vastly larger than the original algorithm could handle. A series of articles have since then been published where the new method is applied to different systems, and produce what are believed to be nearly unbiased results. It is worth mentioning the article by Shepherd *et. al.* (2012) [34], where ground state values of the homogenous electron gas with a Hilbert space with 10^{108} determinants are calculated.

In the following chapters, we aim to give a thorough introduction to the FCIQMC algorithm and the optimized algorithm i-FCIQMC.

Chapter 8

The FCIQMC algorithm

In this chapter will use the following notation. The general wave function is written as

$$|\Psi\rangle = \sum_i v_i |\psi_i\rangle, \tag{8.1}$$

where $\{v_i\}$ are complex coefficients and $\{\psi_i\}$ is the eigenfunctions of the Hamiltonian with the corresponding eigenvalues $\{E_i\}$. The ground state is expanded in the determinants $\{|D_i\rangle\}$, which is constructed from the spin orbitals $\{\varphi_i\}$.

$$\psi_0 = \sum_i C_i |D_i\rangle, \tag{8.2}$$

the CI coefficients of the ground state wave function satisfy the stationary Schrödinger equation in terms of the eigenvalue problem

$$\sum_j \langle D_i|\hat{H}|D_j\rangle C_j = E_0 C_i, \tag{8.3}$$

where E_0 is the lowest eigenvalue of $\hat{H}$ on the FCI basis.

8.1 The mathematical approach

From the last chapter we remember that the projected state can be found by iterating over the equation

$$|\Psi^{(n)}\rangle \leftarrow \hat{P}_\tau |\Psi^{(n)}\rangle, \tag{8.4}$$

where $\lim_{n\to\infty} |\Psi^{(n)}\rangle$ is proportional to the ground state $|\psi_0\rangle$ if the overlap $\langle\Psi^{(0)}|\psi_0\rangle$ is different from zero and the energy shift S of the projection operator $\hat{P}_\tau = e^{-\tau(\hat{H}-S)}$ is equal to the ground state value E_0. Our first aim is to formulate this equation in the FCI basis and in terms of the coefficients C_i. The projection operator is approximated as the first order Taylor expansion

$$\hat{P}_\tau = 1 - (\hat{H} - S)\tau + \mathcal{O}([\hat{H} - S]^2\tau^2). \tag{8.5}$$

We now apply this operator on the initial state expressed on the FCI basis

$$|\Psi^{(n)}\rangle \to \sum_i C_i^{(n)} |D_i\rangle, \tag{8.6}$$

and end up with the expression

$$\sum_i C_i^{(n+1)}|D_i\rangle \approx (1-(\hat{H}-S)\tau)\sum_i C_i^{(n)}|D_i\rangle \tag{8.7}$$

$$\Downarrow$$

$$\begin{aligned} C_i^{(n+1)} &\approx C_i^{(n)} - \sum_j (H_{ij}-\partial_{ij}S)\tau C_j^{(n)} \\ &= C_i^{(n)} - (H_{ii}-S)\tau C_i^{(n)} - \sum_{j\neq i} H_{ij}\tau C_j^{(n)}. \end{aligned} \tag{8.8}$$

Here we have used $H_{ij} = \langle D_i|\hat{H}|D_j\rangle$. We will from now on omit the index n in our equations.

A DMC inspired approach is used to find the steady state solution of Eq. (8.8). The coefficients $\{C_i\}$ are represented by a population of N_W walkers α which are distributed on the determinants D_{i_α} and with a sign $s_\alpha = \pm$. The amplitude C_i is then defined to be

$$C_i \propto n_i = \sum_\alpha s_\alpha \delta_{ii_\alpha}, \tag{8.9}$$

which is a signed sum where two walkers of opposite signs on the same determinant will make zero contribution, and the number of walkers is

$$N_W = \sum_i |n_i|. \tag{8.10}$$

Similar to DMC, we define a set of dynamical rules of such a nature that the walkers converge to a distribution $\langle n_i\rangle \propto C_i$. These dynamical rules are derived from the two last terms of the right hand side of Eq. (8.8) which describes the rate of which C_i changes. The first of these therms is

$$p_d(i|i) = \tau(H_{ii}-S), \tag{8.11}$$

where S is the real shift from the projection operator and τ is the timestep. This term represents the rate of which C_i is projected on it self. Numerically it represents the probability for a walker to either be copied (cloned) or to be removed from the simulation (killed). If $p_d(i|i) < 0$ the walker is cloned with probability $|p_d(i|i)|$ and if $p_d(i|i) > 0$ the walker is killed with the probability $p_d(i|i)$. The second term is

$$p_d(i|j) = \tau H_{ij},\ i \neq j, \tag{8.12}$$

which is the rate of which other $C_{j\neq i}$ is projected onto C_i. We say that $p_d(i|j)$ represents the probability that a walker on $|D_j\rangle$ (the parent) spawns a walker on $|D_i\rangle$ (the child). Note that the newly spawned walkers can have both a positive and a negative sign. From Eq. (8.8) we see that the child must have the opposite sign of its parent if $p_d(i|j)$ is negative, and the sign same if $p_d(i|j)$ is positive. If two walkers with the opposite sign happen to populate the same determinant $|D_i\rangle$, these walkers add zero to the amplitude C_i, and both walkers can therefore be removed from the simulation.

The sums over all elements τH_{ij} involve $\mathcal{O}(\dim(\mathscr{H}) \times N_w)$ operations since the spawning probabilities from all walkers to all determinants must be evaluated. This is a

numerically expensive task since both $\dim(\mathscr{H})$ and N_w can be very big, and for large simulations we often have that $\dim(\mathscr{H}) \geq 10^{10}$ and $N_W \geq 10^7$. According to Booth *et. al.* [3] it is numerically more efficient to sample the sum

$$\sum_j \sum_{i \neq j} \tau H_{ij} C_j \rightarrow \sum_j \tau H_{sj} C_j / p_{gen}(s|j), \text{ where } \sum_s p_{gen}(s|j) = 1, \tag{8.13}$$

reducing the number of operations to $\mathcal{O}(N_w)$ for each iteration. Even though the number of iterations before convergence will increase, the cost of each iteration will become so much cheaper that the overall numerical efficiency is improved. In Eq. (8.13) $p_{gen}(s|j)$ is the suggestion probability of s from j, and this equation is only correct when $p_{gen}(s|j)$ is different from zero for all connected determinants $|D_j\rangle$ and $|D_s\rangle$

$$p_{gen}(s|j) \neq 0 \quad \forall \quad \langle D_j|\hat{H}|D_s\rangle \neq 0. \tag{8.14}$$

In principle, there is no other constraint on $p_{gen}(s|j)$, and as long as the number of iterations is large enough the simulations will eventually converge to the correct results, but in practice certain choices of suggestion probability distributions will lead to a more efficient sampling. We will return to this topic in a later chapter.

8.2 Population control and the statistical estimators

As in DMC, we do not know the shift $S = E_0$ that gives the desired convergence to the ground state distribution. The population of walkers will increase if the shift S is larger than the ground state energy E_0 and decrease if S is smaller than E_0. We can therefore control the population by varying S. We use the formula of Umrigar *et. al.* [39] which has been used for population control in the context of DMC

$$S \rightarrow S^{(i)} = S^{(i-1)} - \frac{\xi}{\tau} \log \left[\frac{N_W^{(i)}}{N_W^{(i-1)}} \right], \tag{8.15}$$

where ξ is a real number of which the optimal value must be found by experiment, and $N_W^{(i)}$ and $S^{(i)}$ are the population and shift at the time $i\tau$. According to Eq. (6.5), the shift S that stabilizes the number of walkers is the ground state energy, and this means that the above equation provides us with a measure of the ground state energy E_0.

We use two different statistical estimators to calculate the energy. The first is the one that we just mentioned, which can be taken as the long time average of the shift

$$E_0 \approx \langle S \rangle = (1/N) \sum_{i=1}^{N} S^{(i)}, \tag{8.16}$$

and which we will refer to as the generational estimator of the energy. The second is the projected estimator that is calculated by sampling the projected energy

$$E_0 = \frac{\langle D_0|\hat{H}|\psi_0\rangle}{\langle D_0|\psi_0\rangle} = \frac{\langle D_0|\hat{H} \sum_i C_i|D_i\rangle}{\langle D_0| \sum_i C_i|D_i\rangle}. \tag{8.17}$$

We rewrite the operator on the form

$$E_0 \approx \langle E_P \rangle = H_{00} + \sum_{i \neq 0} H_{0i} \langle n_i \rangle / \langle n_0 \rangle, \tag{8.18}$$

which means that we need to sample n_0 and $\sum_i n_i H_{0i}$ each iteration. These estimators are statistical measures, and there will always be a certain statistical error associated with them. In a later chapters we will discuss how the statistical errors can be calculated. It is also worth commenting that these two estimators, as we will see, to a large degree are uncorrelated and therefore supplement each other in a valuable way.

8.3 The FCIQMC algorithm and simulation procedures

The first step of a simulation is always to initialize a population of walkers. Before we start collecting statistical samples, the population must be brought to the steady state distribution of Eq. (8.4), which can be done by following a simple procedure which we will now describe. First, we start with one walker on the Fermi determinant $|D_0\rangle$, and a shift that is larger than E_0 to increase the spawning probability and obtain a rapid population growth. When N_W has reached a predetermined value, we start varying the shift S according to Eq. (8.16) to keep the population constant. After the population has reached the desired number of walkers, the simulation still have to run for a while to assure that the population is properly thermalized. The thermalization phase is assumed to be finished when the fluctuations of the instantaneous values of the projected energy and the shift are stabilized around some energy.

For each iteration of the simulation, the following five steps are performed:

1. (Die/clone/spawn): Each walker lives permanently on the slater determinant where it was spawned. Cycle over all walkers i and do:

   a. Attempt to kill or clone the walker. First $p_d(i|i) = \tau(H_{ii} - S)$ is calculated. If $p_d(i|i) < 0$ the walker is killed with probability $|p_d(i|i)|$ and if $p_d(i|i) \geq 0$ the walker is cloned with probability $p_d(i|i)$.

   b. Attempt to spawn a new walker on a connected determinant $|D_j\rangle$ with the probability $\tau|H_{ji}/p_{gen}(j|i)$ where $p_{gen}(j|i)$ is the generation probability of i from j. The child has the same sign as its parent if P_s is positive and the opposite sign otherwise.

2. (Annihilate): Annihilate pairs of walkers of opposite sign that lives on the same determinant.

3. (Update S): Two possible strategies: 1) In constant shift (S) mode, do nothing, 2) in constant population (N_w) mode, adjust S to keep N_w constant.

4. (Sample statistical estimators): Store samples to calculate the energy. We need to store the shifts $S^{(i)}$ to calculate the generational estimator, and $n_0^{(i)}$ and $\sum_i n_i H_{0i}^{(i)}$ to calculate the projected energy.

5. (Additional rule): The efficiency of the algorithm can be improved considerably if additional rules for the walker dynamics are implemented. The most used optimization of the algorithm, i-FCIQMC, will be discuss in Chapter 9.

This algorithm is also illustrated in the flowchart in Fig. 8.1.

8.4 Convergence criteria and the time step error

The projection operator $\hat{P}_\tau$ will always project out the ground state regardless of the size of the time step τ, but this is not the case when the projection operator is approximated as the first order Taylor expansion

$$\hat{P}_\tau = e^{-(\hat{H}-S)\tau} \approx 1 - (\hat{H} - S)\tau. \tag{8.19}$$

To analyze the convergence criteria we can equally well view the Taylor expansion as our new projection operator instead of analyzing the truncation error. The stationary states $|\psi_i\rangle$ are eigenstates of the projection operator as well as its first order Taylor expansion, and the eigenvalue equations are

$$[1 - (\hat{H} - S)\tau]|\psi_i\rangle = [1 - (E_i - S)\tau]|\psi_i\rangle. \tag{8.20}$$

Recall that FCIQMC will project out the state

$$\begin{aligned}[1 - (\hat{H} - S)\tau]^n|D_0\rangle &= \sum_i [1 - (\hat{H} - S)\tau]^n v_i|\psi_i\rangle \\ &= \sum_i [1 - (E_i - S)\tau]^n v_i|\psi_i\rangle,\ S \approx E_0,\ \sum_i |v_i|^2 = 1,\end{aligned} \tag{8.21}$$

which will converge to the ground state only in the case that the absolute values $|1 - (\hat{E}_i - S)\tau|$ are smaller than one for all $i > 0$. This sets the upper bound for the time step

$$\tau < \tau_m = 2/(E_m - S),\ E_m = \max \langle\psi_i|\hat{H}|\psi_i\rangle, \tag{8.22}$$

where E_m is bounded in our truncated FCI spaces. Since we do not know the value of the excited energies we can not compute τ_m. But in general we can conclude that τ_m decreases when E_m increases, as for example when we increase the number of particles, the number of shells or the interaction strength. In our simulations, we have used a timestep $\tau = 10^{-3}$ or smaller, which means that E_m in practice can be larger than $2 \times 10^3 H^*$, which is greater than the most excited energies E_m in any of the FCI spaces that we have used.

Note that there is no time step error, and as long as the time step is smaller than τ_m, the simulations will converge to the exact ground state. This is a consequence of the fact that the projected estimator and its first order Taylor expansion have the same eigenstates.

8.5 The FCIQMC sign problem

Walker annihilation is a necessary ingredient of the FCIQMC algorithm, and as we will show, if we had turned off the annihilation the simulations would not converge to the ground state. Thus, when the number of walkers is "small" and the density of walkers becomes too low for efficient annihilation, the simulations will no longer yield the correct results. Note that this problem does not occur in DMC where most system can be sampled using a small number of walkers. The reason is that the walkers do not carry a sign as in FCIQMC, but instead the sign structure of the wave function is forced by applying the fixed node

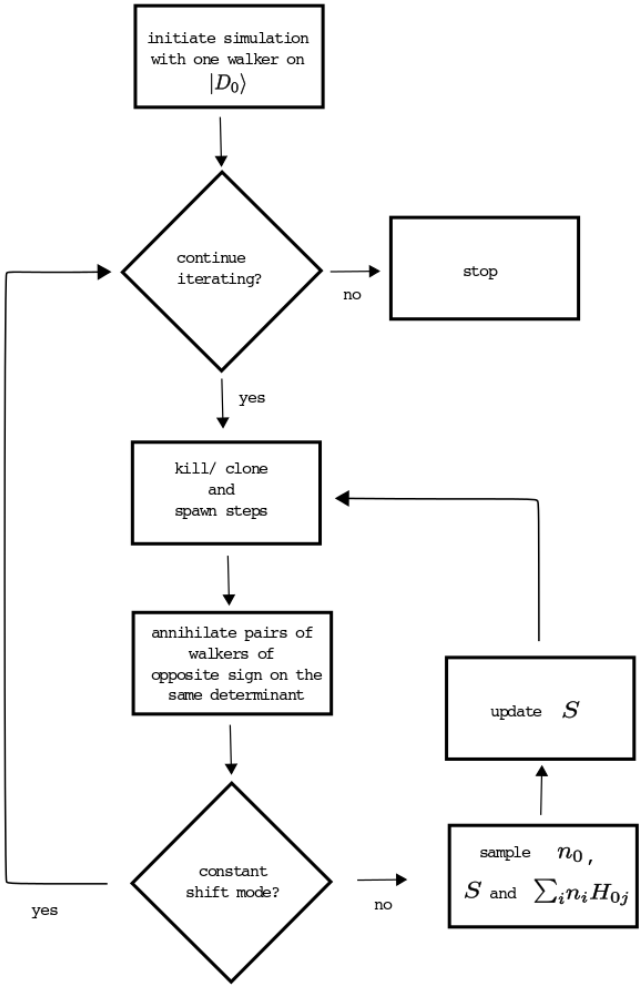


Figure 8.1: Flowchart that illustrates the basic FCIQMC algorithm.

approximation. To explain the mechanisms behind the FCIQMC sign problem, we take a closer look at what would happen if we turned off annihilation. This will also provide an explanation of why the simulations do not converge to the ground state when the number of determinants is too low. To analyze this situation we will follow the derivation of Spencer *et. al.* [36].

Assume that n_i^+ and n_i^- represents the positive and the negative population of walkers on the determinant $|D_i\rangle$. We define the "transition matrix"

$$T_{ij} = -(H_{ij} - \delta_{ij}S) = T_{ij}^+ - T_{ij}^-, \tag{8.23}$$

where $T_{ij}^+ = \max(T_{ij}, 0)$ and $T_{ij}^- = |\min(T_{ij}, 0)|$. From Eq. (8.8) we see that after one iteration, the distributions will change from n^+ to $n_i^+ + \Delta n_i^+$ and from n^- to $n_i^- + \Delta n_i^-$ where

$$\Delta n_i^+ = \tau \sum_j \left[T_{ij}^+ n_j^+ + T_{ij}^- n_j^-\right], \tag{8.24}$$

$$\Delta n_i^- = \tau \sum_j \left[T_{ij}^+ n_j^- + T_{ij}^- n_j^+\right]. \tag{8.25}$$

We rearrange these equations on the form

$$\Delta n_i^+ + \Delta n_i^- = \tau \sum_j \left[T_{ij}^+ + T_{ij}^-\right]\left[n_j^+ + n_j^-\right], \tag{8.26}$$

$$\Delta n_i^+ - \Delta n_i^- = \tau \sum_j \left[T_{ij}^+ - T_{ij}^-\right]\left[n_j^+ - n_j^-\right], \tag{8.27}$$

which tells us how the distributions $n_i^+ + n_i^-$ and $n_i^+ - n_i^-$ will evolve. Here $n_i^+ \pm n_i^-$ represents the distribution without annihilation (+) and with annihilation (−). The steady state solutions are the eigenstates of the operators

$$P_{ij}^+ = 1 + \tau \sum_j \left[T_{ij}^+ + T_{ij}^-\right], \tag{8.28}$$

$$P_{ij}^- = 1 + \tau \sum_j \left[T_{ij}^+ - T_{ij}^-\right], \tag{8.29}$$

with the corresponding eigenvalues e^+ and e^-, and it can be shown that $e^+ \geq e^-$ (See Ref. [36] for the proof). Thus, if we turn off annihilation, $n^+ + n^-$ will become exponentially larger than $n^+ - n^-$. It is in principle still possible to find $n^+ - n^-$ by subtracting the negative population n^- from the positive population n^+, but in practice $n^+ - n^-$ will be lost due to statistical noise. Note that P_{ij}^- is simply the first order Taylor expansion of the projection operator which means that $n^+ - n^-$ is the ground state distribution.

We have now demonstrated that the sign problem in FCIQMC stems from the emergence of the dominant eigenstate of the operator P^+, while the desired state is the dominant eigenstate of the operator P^-. In a simulation, the growth rate of these eigenstates will be proportional to the corresponding eigenvalues e^+ and e^-.

According to Spencer *et. al.* , if e^+ is much larger than e^-, a large concentration of walkers is typically necessary to achieve a high enough rate of annihilation events. Such systems are said to have a severe sign problem, and are not easily simulated with the FCIQMC algorithm. They also comment that it is difficult to predict the severity of the sign problem for a specific system. A difficult system is not necessarily a highly correlated system, but the sign problem rather depends on the structure and occurrence of negative elements in the Hamiltonian matrix.

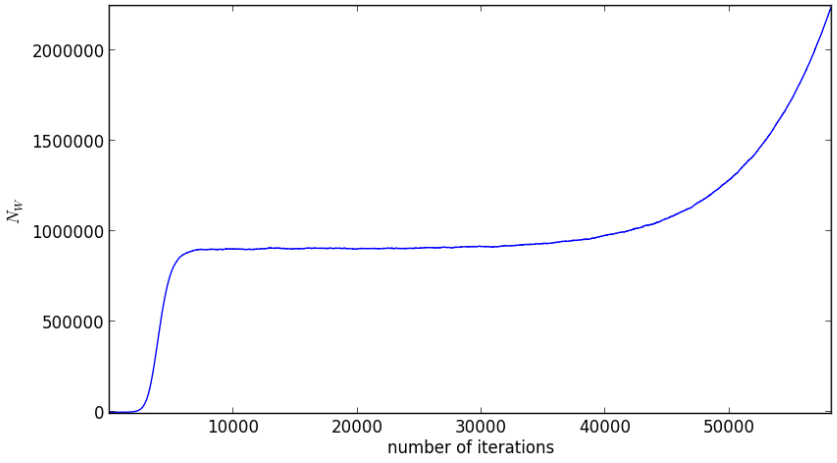


Figure 8.2: A typical evolution of the population size in a simulation where S is kept constant at a value that is slightly greater than the ground state. In the first phase of the simulation the population is too small for efficient annihilation, and the population grows exponentially with a growth rate e^+. When the population reaches a certain limit, annihilation events become more probable, and the state $n^+ - n^-$ emerges. The population growth is still exponential, but with a lower growth rate e^-, and this can be observed as a plateau in the population size. If the plateau occurs at a large population relative to the dimension of the Hilbert space, the sign problem is severe.

Chapter 9

Initiator-FCIQMC

As we discussed in the last chapter, the number of walkers must be above a certain threshold N_C for the population to converge to the ground state distribution. This threshold is, as we will see later, system dependent and often close to the number of determinants in the FCI space. In practice this is the main limitation of FCIQMC since the numerical cost increase as a function of the number of walkers.

Cleland *et. al.* (2010) [4] demonstrated that by introducing a set of new dynamical rules for the walkers, the performance of the algorithm can be improved drastically while still obtaining results with a precision of 10^{-3} Hartree or better. The number of walkers that is necessary in such simulations are typically several orders of magnitude smaller than N_C. The improved algorithm is called initiator-FCIQMC or i-FCIQMC, and is identical to FCIQMC except that we use different rules to calculate the spawning probability. In essence, only walkers that fulfill certain criteria are allowed to spawn on determinants which are previously unpopulated, and such walkers are called initiators.

First, a so called initiator space is defined. This is a set of determinants which are assumed to have a relatively large average population (large ampiludes $|C_i|$). All walkers that live on a determinant in the initiator space become initiators, and initiators follows the same rules as walkers in the basic FCIQMC algorithm. The first rule of i-FCIQMC concerns the non-initiators and states:

- (*Spawning rule I*): Non-initiators are only allowed to spawn on determinants that are previously populated.

With this rule the Hilbert space is in effect reduced to the states in the initiator space and the single and double excitations of these. Next, two additional rules are added to improve the energy and to give the walkers access to the entire FCI space:

- (*Dynamic enlargement of the initiator space*): All determinants $|D_i\rangle$ with a population $|n_i| > N_I$ are automatically included in the initiator space for as long the population is above this limit. Here n_i is the signed population of $|D_i\rangle$ and N_i is a positive integer that we will refer to as the initiator threshold. The initiator threshold is set before a simulation. Note that with $N_I = 1$ i-FCIQMC is identical to FCIQMC.

- (*Spawning rule II*): If the absolute value of sum of all walkers that are spawned on an unpopulated determinant by non- initiators is larger than two, then the spawning event is accepted.

Although the initiator adaption of the algorithm has been shown to increase the efficiency, it is only correct according to Eq. (8.8) in the limit of a large number of of walkers. In this case the initiator space will grow to include the entire FCI space, and i-FCIQMC will tend to FCIQMC. This means that the i-FCIQMC energies are systematically improvable and that the so called initiator error will decay to zero when the number of walkers is increased.

9.1 The initiator spaces

The initiator spaces are defined using a so called Complete Active Space (CAS) criterion, which is often used within quantum chemistry together with methods like many-body pertubation theory or the configuration interaction method to pick out physically relevant states of a Hilbert space [12]. The CAS is defined by specifying two set of orbitals, the frozen and the active orbitals. The CAS is the set of determinants where:

(i): The frozen orbitals are doubly occupied (by both a spin up and a spin down electron).

(ii): The rest of the electrons are occupying one of the active orbitals.

In our simulations, the frozen orbitals are chosen to be the lowest lying orbitals in the N_F first shells and the active space consists of all orbitals in the N_A first shells. Here N_F and N_A are preset parameters which defines the CAS. See Fig 9.1 for an illustration.

Because of the dynamic enlargement of the initiator space, the final result will be the same regardles of which CAS we choose. But, as Cleland et. al. [4] has demonstrated, an appropriate CAS might sometimes improve the convergence.

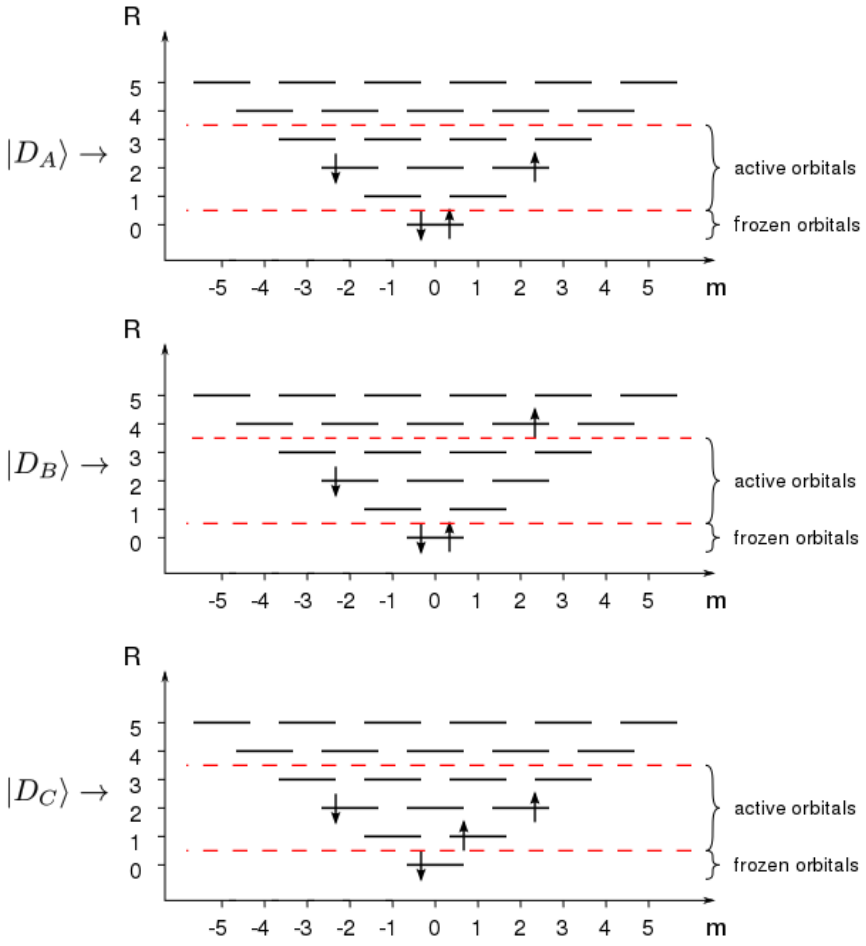


Figure 9.1: These figures are schematic representations of the three determinants $|D_A\rangle$, $|D_B\rangle$ and $|D_C\rangle$ of a four particle system. Here R is the shell number and m is the magnetic quantum number. The vertical lines represents an orbital that can be occupied by a spin up ($\uparrow$) and a spin down ($\downarrow$) electron. The CAS is defined by the frozen orbitals which are the two spinorbitals with the lowest energy, and the active orbitals which are all states in the second, third and fourth shell. The determinant $|D_A\rangle$ is in the CAS since both of the frozen spin orbitals are occupied. The determinant $|D_B\rangle$ is not in the CAS space since there are one electron which is occupying an orbital which is neither an active orbital or a frozen orbital. The determinant $|D_C\rangle$ is not in the CAS since only one of the frozen orbitals are occupied.

Chapter 10

Sampling rules

10.1 The suggestion probability distribution

Recall that earlier in this part we introduced the suggestion probability $p_{gen}(i|j)$ which is the probability of sampling (trying to spawn a walker on) the determinant $|D_i\rangle$ from the determinant $|D_j\rangle$. Before we can implement the algorithm it is necessary to find a "smart" suggestion probability, and there are two main factors that should be considered.

(i): From a sampling perspective, there exist certain suggestion probabilities that are more favourable than others. For example, using a p_{gen} where the most physically relevant states have a larger probability of being sampled is expected to lead to a faster convergence in terms of the number of iterations. This method is commonly referred to as importance sampling [10].

(ii): We must find a $p_{gen}(i|j)$ that can be sampled with a low numerical cost. As an example of the opposite, assume that we want to use a $p_{gen}(i|j)$ where all connected determinants $|D_i\rangle$ ($\langle D_i|D_j\rangle \neq 0$) have an uniform probability. This might be efficient from a sampling perspective but not from a numerical perspective. The reason is that we would have to explicitly find all connected determinants which is a numerically expensive operation.

We have used a strategy that was introduced by Booth *et. al.* (2009) [3], where the suggestion probability distribution has the form

$$p_{gen}(i|j) = \begin{cases} p_s \times A_j(pk), & \text{for } |D_i\rangle = (-)^p a_p^\dagger a_k |D_i\rangle,\ p \neq q, \\ (1-p_s) \times B_j(pqkl), & \text{for } |D_i\rangle = (-)^p a_p^\dagger a_q^\dagger a_l a_k |D_j\rangle,\ \{k,l\} \cap \{p,q\} = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{10.1}$$

Here p_s is a predetermined probability which represents the probability of a single excitation versus a double excitation, and $A_j(pk)$ and $B_j(pqkl)$ are probability distributions which we will look closer at in the next section. Only single and double excitations are included here since all pairs of unconnected determinants corresponds to a spawning probability $p_d(i|j) \propto H_{ij} = 0$. The sampling will be correct regardless of the choice of p_s, but optimally

it should reflect the "true" probability of a single excitation. Since the single excitations are more probable than the double excitations, it is natural to choose a p_s that is close to one, but a bad choice of p_s would, at least in principle, only lead to a less efficient sampling and a slower convergence. The distributions $A_j(pk)$ and $B_j(pqkl)$ are chosen such that they are numerically cheap to sample. But the resulting suggestion probability distribution $p_{gen}(i|j)$ is not based on any sampling considerations and will probably not give the fastest convergence in terms of the number of iterations.

10.2 Sampling of the suggestion probabilities

We will now discuss the distributions $A_j(pk)$ and $B_j(pqkl)$ and show how they can be sampled in an efficient manner. In the case of a single excitation, one occupied orbital φ_k is chosen with the uniform probability $A_i(k) = 1/N_P$, and a new orbital φ_p is then picked randomly from the set

$$\mathcal{S}_0 = \left\{\varphi_p\,;\ a_p^\dagger a_k|D_i\rangle \in \mathscr{H},\, p \neq k\right\}. \tag{10.2}$$

The conditional probability of drawing p is $A_j(p|k) = 1/\text{size}(\mathcal{S}_0)$ and the total probability becomes

$$A_j(pk) = A_j(p|k)A_i(k) = [N_P \times \text{size}(S_0)]^{-1}. \tag{10.3}$$

In the case of a double excitation, two random occupied orbitals φ_k, φ_l are chosen with the uniform probability

$$B_j(kl) = \binom{N_P}{2}^{-1} = \frac{2}{[N_P(N_P-1)]} \tag{10.4}$$

We now need to consider if there are any constants on which q we can pick. We must find the subset $\mathcal{S}_1$ of orbitals that fulfills these constraints and pick a random orbital $\varphi_q \in \mathcal{S}_1$ with the probability

$$B_j(q|kl) = 1/\text{size}(\mathcal{S}_1). \tag{10.5}$$

The next step is to follow the same procedure for p. We find the subset $\mathcal{S}_2$ of orbitals that fulfills our constraints given that φ_q is already picked, and pick a random $\varphi_p \in \mathcal{S}_2$ with the probability

$$B_j(p|qkl) = 1/\text{size}(\mathcal{S}_2). \tag{10.6}$$

Since the probability of drawing the pair (p, q) in general is dependent of which order they are picked, we must explicitly calculate the probability of drawing p, q in the reverse order. We follow the same procedure as above and find

$$B_j(p|kl) = 1/\text{size}(\mathcal{S}_3), \tag{10.7}$$

$$B_j(q|pkl) = 1/\text{size}(\mathcal{S}_4). \tag{10.8}$$

The total probability becomes

$$B_j(pqkl) = B_j(kl)\Big(B_j(q|pkl)B_j(p|kl) + B_j(p|qkl)B_j(q|kl)\Big). \tag{10.9}$$

10.3 Finding the sets $\mathcal{S}_n$

As many criteria as possible should be set to reduce the number of elements in the sets S_n. We must only make sure that $p_{gen}(i|j) \neq 0$ when the spawning probability $p_d(i|j) \neq 0$, otherwise the sampling will be erroneous. If $p_{gen} \neq 0$ and $p_d(i|j) = 0$ the spawning step will be rejected, and the sampling will be less than optimal but will still give the correct result. For two dimensional quantum dots, the following quantities are conserved in an interaction

$$m_p = m_k, \quad s_p = s_k \quad \text{(single excitation)}, \tag{10.10}$$

$$m_p + m_q = m_k + m_l, \quad s_p + s_q = s_k + s_l \quad \text{(double excitations)}\,, \tag{10.11}$$

where m_i and s_i is the magnetic quantum number and the spin of the i'th orbital. We will use these conservation rules and outline a simple procedure to find the sets S_n.

In the case of a single excitation, assume that we have annihilated an orbital φ_k, then $\mathcal{S}_0$ is calculated

$$\mathcal{S}_0^+ = \{\varphi_i, m_i = m_k, s_i = s_k\}, \tag{10.12}$$

$$\mathcal{S}_0 = \mathcal{S}_0^+ \backslash \mathcal{T}, \tag{10.13}$$

where $\mathcal{T}$ is the set of all occupied orbitals in $|D_j\rangle$ and $\mathcal{S}_0^+ \backslash \mathcal{T}$ is the set $\mathcal{S}_0^+$ where all elements which are in $\mathcal{T}$ are removed.

In the case of a double excitation, assume that we have drawn two occupied orbitals φ_k, φ_l with opposite spins. Then we have the restrictions that

$$\begin{aligned} &\text{If } s_k \neq s_l \Rightarrow \\ &\quad |m_k + m_l - m_q| \leq R, \end{aligned} \tag{10.14}$$

where R is the absolute value of the largest m in our basis. The smallest and the largest allowed magnetic quantum number m_{MIN}, m_{MAX} can be found

$$\begin{aligned} &\text{If } s_k \neq s_l \Rightarrow, \\ &\quad m_{MIN} = \max(-R, m_k + m_l - R), \end{aligned} \tag{10.15}$$

$$m_{MAX} = \min(R, m_k + m_l + R). \tag{10.16}$$

In the case where the two orbitals k, l have the same spins, it is possible to improve Eq. (10.14) slightly

$$\begin{aligned} &\text{if } s_k = s_l, \\ &\Rightarrow |m_k + m_l - m_p| < R, \end{aligned} \tag{10.17}$$

which gives

$$\begin{aligned} &\text{If } s_k = s_l \Rightarrow \\ &\quad m_{MIN} = \max(-R + 1, m_k + m_l - R), \end{aligned} \tag{10.18}$$

$$m_{MAX} = \min(R - 1, m_k + m_l + R). \tag{10.19}$$

The set $\mathcal{S}_1$ is now defined as

$$\mathcal{S}_1^+ = \left\{\varphi_i; m_i \in [m_{MIN}, m_{MAX}], s_i \in \{s_k, s_l\}\right\}, \tag{10.20}$$

$$\mathcal{S}_1 = S_1^+ \backslash \mathcal{T}, \tag{10.21}$$

and the set $\mathcal{S}_2$ is defined as

$$\mathcal{S}_2^+ = \{\varphi_i, m_i = m_k + m_l - m_q, s_i = s_k + s_l - s_q\}, \tag{10.22}$$

$$\mathcal{S}_2 = \mathcal{S}_2^+ \backslash \{\mathcal{T}, \varphi_q\}. \tag{10.23}$$

If either $\mathcal{S}_1 = 0$ or $\mathcal{S}_2 = 0$, the spawning event is rejected. The sets $\mathcal{S}_3$ and $\mathcal{S}_4$ are calculated in the same way by interchanging p and q in the equations above.

Excited state: $\|nm\rangle$	Suggestion probability$/(1-p_s) = B_0(nm01)$
$\|\varphi_2\varphi_5\rangle$	1/18
$\|\varphi_3\varphi_4\rangle$	1/18
$\|\varphi_6\varphi_{11}\rangle$	2/18
$\|\varphi_7\varphi_{10}\rangle$	2/18
$\|\varphi_8\varphi_9\rangle$	2/18
$\|\varphi_{12}\varphi_{19}\rangle$	2/18
$\|\varphi_{13}\varphi_{18}\rangle$	2/18
$\|\varphi_{14}\varphi_{17}\rangle$	1/18
$\|\varphi_{15}\varphi_{16}\rangle$	1/18
$\|\varphi_2\varphi_{17}\rangle$	1/18
$\|\varphi_3\varphi_{16}\rangle$	1/18
$\|\varphi_4\varphi_{15}\rangle$	1/18
$\|\varphi_5\varphi_{14}\rangle$	1/18

Table 10.1: Example of the suggestion probabilities of double excitations from the reference determinant $|01\rangle$ which is described by the probability distribution $B_0(nm01)$ (see: Eq. (10.1)). The system is a two particle quantum dot with four shells in the basis, and the spin orbitals φ_i are enumerated as in Fig. 4.1. Only the states where the total spin and magnetic quantum number are conserved is included. The probabilities $B_0(nm01)$ are calculated using the rules of the last section. Note that the suggestion probability of a state has no connection to how physically important it is.

Chapter 11

Storing and accessing the Coulomb matrix elements

The Coulomb matrix elements (CME)

$$v_{ijkl} \equiv \langle \varphi_i \varphi_j | \hat{V} | \varphi_k \varphi_l \rangle_{AS}, \tag{11.1}$$

must be available every time the spawning probability is calculated. It is therefore critical for the speed of the code to be able to access them quickly. Calculation of the CME on the fly is very inefficient. Storage of the CME in an array is much faster but it is not straight forward to find an efficient way of addressing and accessing them.

One way is to store all nonzero matrix elements in one array and the corresponding occupations $ijkl$ in a second array. The address of the matrix element can then be obtained by finding the index of the corresponding occupations numbers in the occupations array. This method has the advantage that only the nonzero elements are stored and the drawback that all occupations must be stored as well and that a numerically expensive search must be performed on the occupations array.

In this section we will show how it is possible to map each of the nonzero matrix elements to a unique address. The advantages are that the expensive search and the storage of the occupations array is omitted.

11.1 Indexing scheme

In this section we will refer to the states $|\varphi_i \varphi_j\rangle$ as the spinless wave functions, meaning that any of the coulomb matrix elements can have 16 different spin configurations. Only 6 of these are permitted, namely those where the total spin is conserved. In addition, the matrix elements are unchanged if the spin projection of the states is polarized, reducing the number of unique elements to 3. We will enumerate the three allowed pairs of spin configurations in the following way

$$n_s(ijkl) = 0 \text{ if } (s_i, s_j, s_k, s_l) \in \{(\uparrow,\downarrow,\downarrow,\uparrow),(\downarrow,\uparrow,\uparrow,\downarrow)\}, \tag{11.2}$$

$$n_s(ijkl) = 1 \text{ if } (s_i, s_j, s_k, s_l) \in \{(\uparrow,\downarrow,\uparrow,\downarrow),(\downarrow,\uparrow,\downarrow,\uparrow)\}, \tag{11.3}$$

$$n_s(ijkl) = 2 \text{ if } (s_i, s_j, s_k, s_l) \in \{(\uparrow,\uparrow,\uparrow,\uparrow),(\downarrow,\downarrow,\downarrow,\downarrow)\}. \tag{11.4}$$

When we want to specify the spin configuration of the orbitals we will denote the matrix element

$$v_{ijkl}^{n_s(ijkl)}. \tag{11.5}$$

From the definition

$$v_{ijkl} = \int d\mathbf{x}d\mathbf{y}\psi_i^\dagger(\mathbf{x})\psi_j^\dagger(\mathbf{y})\left[\psi_l(\mathbf{y})\psi_k(\mathbf{x}) - \psi_k(\mathbf{y})\psi_l(\mathbf{x})\right], \tag{11.6}$$

it is clear that

$$v_{ijkl} = v_{jilk} = v_{klij} = v_{lkji} = -v_{jikl} = -v_{ijlk} = -v_{klji} = -v_{lkij}. \tag{11.7}$$

From the above symmetries we see that most matrix elements can be represented in multiple ways. For numerical reasons we want to construct a set where each element is represented only once. First we define the mapping $f(ijkl)$ which is a reordering of the indices $ijkl$ where only three operations are allowed to permute the indices:

- Changing the positions of the two first indices with the two last.
- Interchanging the two first indices.
- Interchanging the two last indices.

The reordered indices must fulfill one of the three equations below:

$$\begin{aligned}
&\text{Case I:}\\
&\quad i \le j \le k \le l.\\
&\text{Case II:}\\
&\quad i < j > k \le l \text{ where } i < k.\\
&\text{Case III:}\\
&\quad i < j > k < l \text{ where } i = k,\ j \le l.
\end{aligned} \tag{11.8}$$

It can be shown that

$$\begin{aligned}
&f(ijkl) = f(jilk) = f(klij) = f(lkji)\\
&= f(jikl) = f(ijlk) = f(klji) = f(lkij),
\end{aligned} \tag{11.9}$$

meaning that each unique element given the symmetries of Eq. (11.7) is represented exactly once in the set of $v_{ijkl\epsilon F}$ where F is the set of all possible combinations $f(ijkl)$. We define $p(ijkl)$ to be the number of permutations that are necessary to arrange the indices in the desired order. Any matrix element can now be written

$$v_{ijkl} = (-1)^{p(ijkl)} v_{f(ijkl)}^{n_s(f(ijkl))}, \tag{11.10}$$

where we have included the spin configuration in the superscript.

11.2 Mapping of the indices to a pointer

It is possible make a mapping of the indices $(ijkl)$ to memory addresses in such a way that none of the matrix elements of Eq. (11.10) are represented more than once, and we will now explain how this is done.

We store all elements with an address that is stored as (in C++ notation)

$$\texttt{(*(P[n\_S] + n\_G)+n\_I)}, \tag{11.11}$$

where `P` is an array of double pointers [1] and `n_S`, `n_G` and `n_I` are non-negative integers. First `n_S` is calculated according to Eq. (11.2)-(11.4)

$$\texttt{n\_S} \leftarrow n_s(ijkl). \tag{11.12}$$

Next we find $(i'j'k'l') \leftarrow f(ijkl)$ according to Eq. (11.8)). The indices j', k', l' (case I,II of Eq. (11.8)) or j', l' (case III of Eq. (11.8)) are mapped to the integer `n_G`, where the set of all possible `n_G` consists of all integers in an interval that start at 0. We will show how to find `N_G` later in this chapter.

Next we find n_I which is the number of orbitals φ_m with $m < i'$ and a magnetic quantum number

$$m_{i'} = m_{k'} + m_{l'} - m_{j'}. \tag{11.13}$$

Note that `n_I` can be all integers between 0 and j' in case I, between 0 and k' in case II and between 0 and l' in case III.

Since both `N_G` and `N_I` always occupy all values in an interval beginning at 0, all addresses of (11.11) corresponds to exactly one matrix element $v^{n_s(f(ijkl))}_{f(ijkl)}$ which means that this is a space efficient way of storing the CME.

11.3 Graphical representation of model spaces

The method of mapping the indices j, k, l or j, l to `n_G` is inspired by a method that is outlined by Ref. [12], where it is demonstrated how one can map general determinants to an index. A short introduction to this method follows.

Assume a three particle state $|n_0 n_1 n_2\rangle$ where $n_0 < n_1 < n_2$ and $n_i \in [0, 5]$. First we note that all possible determinants can be represented by one path in Fig. 11.1. All paths that start in the upper left circle and end in the lower left circle, and follow the direction of the arrows through the circles, represents a determinant. The occupation number n_i is equal to the value of n at the circle where one moves from the row marked (n_i) to the next row, and the index can be found by summing the arrow weights (the number above the arrows) along the path. In this case it is easy to check that all paths corresponds to a unique index and that $n_0 < n_1 < n_2$. As an example, the determinant $|124\rangle$ is represented by the path marked up by the double headed arrows in Fig. 11.1. The index of the path is found by summing the arrow weights and as we see from the figure the index is 1 + 1 + 4 = 6.

We need two rules to construct the tables Fig. 11.1. First note that the circled numbers represents the total number of paths leading from the upper left circle. To find this number

[1] In C++ this array is declared as `double*** P`.

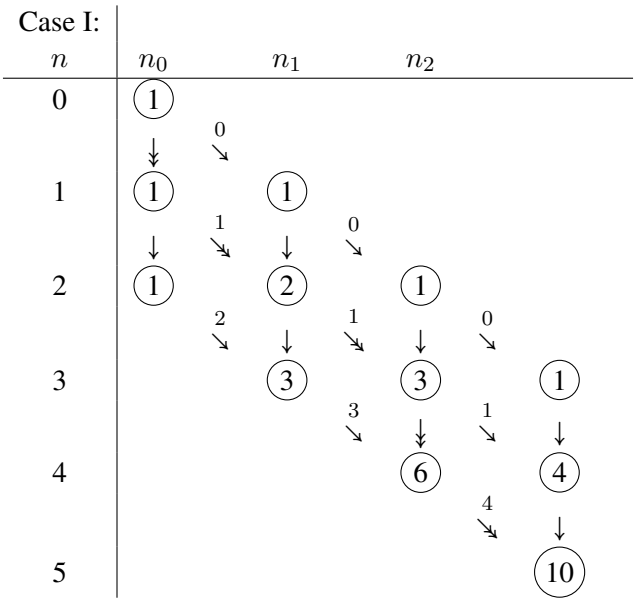


Figure 11.1: Each path that can be made by following the arrows from the upper left corner to the lower right corner represents a three particle state $|n_0 n_1 n_2\rangle$ where $n_0 < n_1 < n_2$ and $n_i \in [0, 5]$. The occupation number n_i is equal to the value of n at the circle where one moves from the row marked (n_i) to the next row. Each such path can be mapped to an index, and these indices occupy all values in an interval that starts on zero. The index can be found by summing the numbers above the arrows along the path, and all paths corresponds to a unique index. For example, $|124\rangle$ is represented by the path marked up by the double headed arrows, and the index is 1 + 1 + 4 = 6.

one simply adds the numbers circles from which one can move to the given circle. The numbers above the arrows is the difference between the circled number at the start and at the end of the arrow, and appears when one moves from one row to the next in the table.

In simulations, we would initialize the matrix of the arrow weights

$$M_{GRMS} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 3 & 4 \end{pmatrix}. \tag{11.14}$$

The index of a determinant (path) $|n_0 n_1 n_2\rangle$ can then be calculated

$$M_{GRMS}(n_0, 0) + M_{GRMS}(n_1 - 1, 1) + M_{GRMS}(n_2 - 2, 2). \tag{11.15}$$

11.4 The arrow weight matrices

In our case we need to make tables and arrow weight matrices for the three cases of Eq. (11.8)

Case I: All paths $j \le k \le l$, $i \le j$ are represented in the Fig. 11.2. The arrow weights are marked as $M^I_{i,j}$. By following the rules introduced in the last section, it is easy

to verify that

$$M^I_{0,i} = 0,\ M^I_{0,i} = M^I_{0,i-1} + 1, \tag{11.16}$$

$$M^I_{i,0} = M^I_{i-1,0} + M^I_{i,j-1} \text{ for } i \neq 0,\ j \neq 0, \tag{11.17}$$

$$N_I = M^I_{N_0-1,0} + M^I_{N_0-1,1} + M^I_{N_0-1,2}, \tag{11.18}$$

where N_I is the number of possible paths. The index `N_G` can be calculated by summing the matrix elements

$$\texttt{N\_G} = M^I_{j,0} + M^I_{k,1} + M^I_{l,2}. \tag{11.19}$$

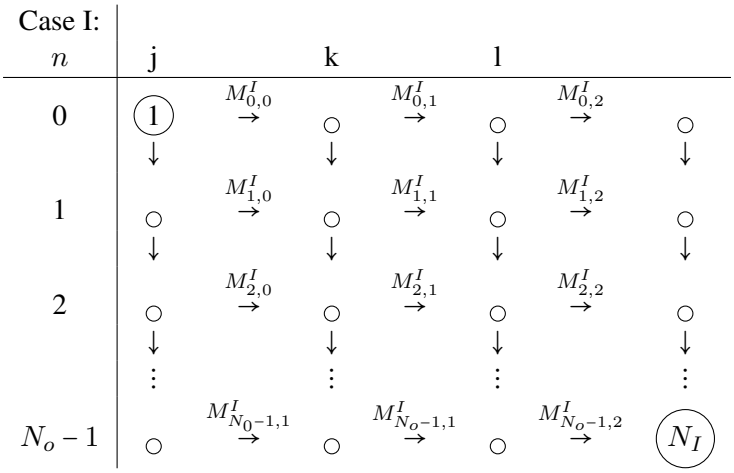


Figure 11.2: All paths $j \leq k \leq l$, $i \leq j$ are represented here. See the explanation in the text.

Case II: Figure 11.3 contains all paths $j > k \leq l$, $i < k$. The first row ($n = 0$) is empty since $i < k$. In this case the arrow weight matrix can be constructed by using the following rules:

$$M^{II}_{0,i} = M^{II}_{1,i} = 0, \tag{11.20}$$

$$M^{II}_{i,o} = i - N_o - 1, \text{ for } i > 1, \tag{11.21}$$

$$M^{II}_{2,1} = M^{II}_{2,2} = N_o - 1, \tag{11.22}$$

$$M^{II}_{i,1} = N_o - 1 - i \text{ for } i \in [3, N_o - 2], \tag{11.23}$$

$$M^{II}_{i,2} = M^{II}_{i-1,2} + M^{II}_{i,1} \text{ for } i \in [3, N_o - 2], \tag{11.24}$$

$$M^{II}_{N_o-1,2} = M^{II}_{N_o-2,2} + M^{II}_{N_o-2,1} + 1. \tag{11.25}$$

The index `N_G` is calculated

$$\texttt{N\_G} = M^{II}_{j,0} + M^{II}_{k,1} + M^{II}_{l,2} + N_I, \tag{11.26}$$

N_I is added to start the indexing at the last index if case I. Total number of paths is

$$N_{II} = M^{II}_{0,N_0-1} + M^{II}_{1,N_0-2} + M^{II}_{2,N_o-1} + 1. \tag{11.27}$$

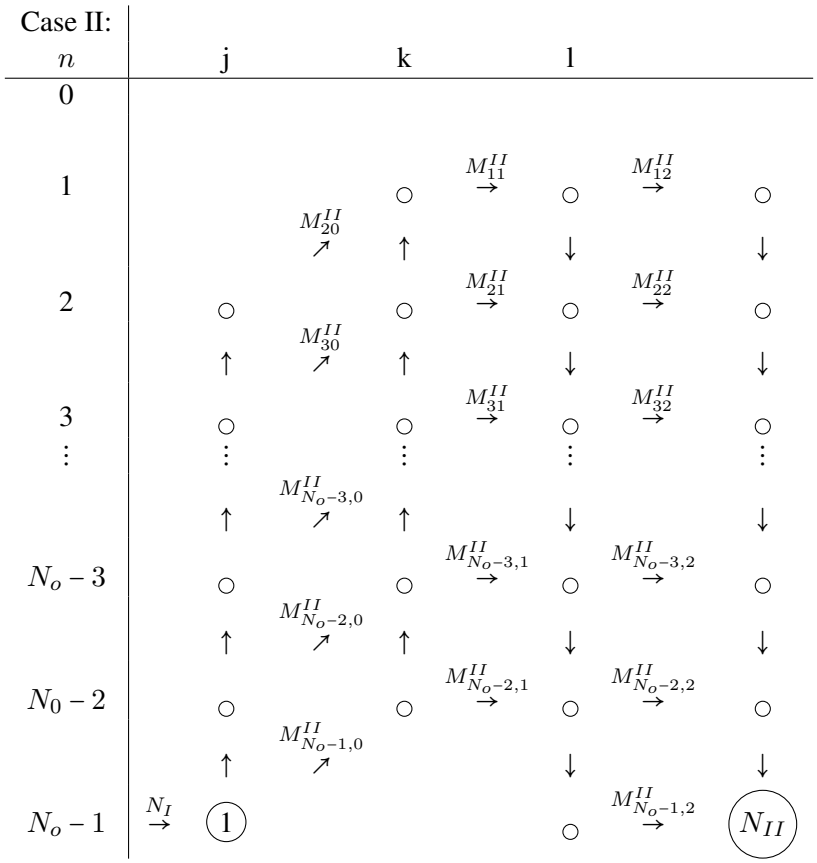


Figure 11.3: All paths $j > k \leq l$, $i < k$ are represented here. See the explanation in the text.

Case III: Figure 11.4 contains all paths $j \leq l$, $i < j$. The first row (n = 0) is empty since $i = k < j$. The arrow weight matrix can be constructed in the same way as $M^I_{i,j}$. The index N_G can be calculated using the tho first columns of M^I and subtracting 1 from j and l since both i, j must be larger than zero (see the table below)

$$\texttt{N\_G} = M^I_{j-1,0} + M^I_{l-1,1} + N_I + N_{II}, \tag{11.28}$$

where N_I and N_{II} is added to start the indexing at the last index if case II.

11.5 Memory requirements and numerical speed

The memory used to store the CME increases rapidly as a function of the basis size, but he memory requirements are still acceptable. For example, for a basis with 28 shells, there are 57×10^6 CME and 33×10^6 pointers which requires about 1.2GB of memory. The array weight matrices are very small, and with 28 shells the largest of these have the dimensions 406×3.

In our implementation of the algorithm, a total of 6 if-tests and a little less than 4 array

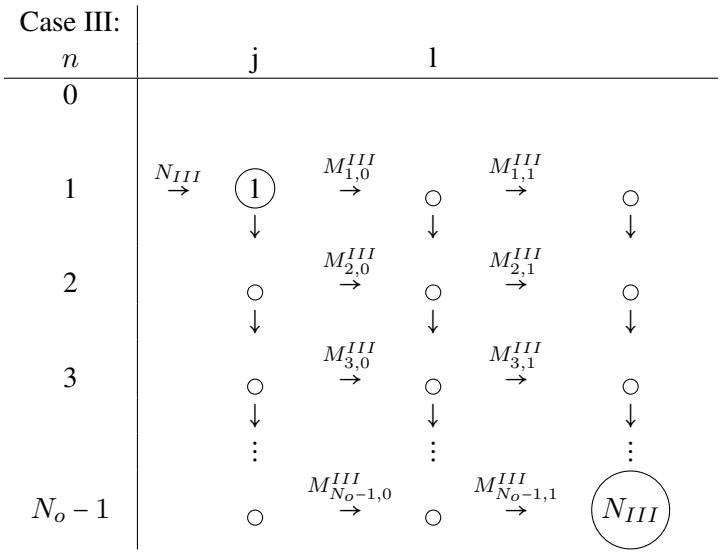


Figure 11.4: All paths $j \leq l$, $i < j$ are represented here. See the explanation in the text.

lookup on average (in the small weight matrices) are necessary to retrieve the address of a CME. This is the same for all systems regardless of the size of the basis.

11.6 Generalisation to other systems

It should be straight forward to generalize this method to other systems.The only requirement is that the matrix elements have the same symmerties (see: Eq. (11.7)) with regard to the spatial part of the single particle wave functions. Other quantum numbers should in general be possible to account for by redefining the array n_I.

Part IV

Data analysis

A large amount of data is collected during an FCIQMC simulation. The calculated observables are based on statistical measures, like the generational estimator and the projected estimator which we introduced in the last chapter. These are only interesting if we can find a reliable measure of the precision of the results. In the first chapter, we introduce a numerically efficient way of estimating the statistical error. This method was discussed in an article by Flyvbjerg and Pedersen (1989) [5], and we will simply refer to it as Flyvbjerg-Pedersen analysis or blocking analysis. We will first discuss this method in a general context, and then how in can be used to calculate the statistical error of the generational and the projected estimator.

We will then discuss curve fitting. In section 5.3, we introduced a formula that can be used to extrapolate the energies of finite Hilbert spaces to the limit of an infinite basis. But to do so, we need to fit a parametrized function to a set of datapoints. Least square fitting is an efficient method of curve fitting, and we will introduce this method in the second chapter in this part.

Chapter 12

Statistical analysis

A large amount of data is generated during a simulation, and observables are calculated on the basis of mean values of different data sets. These estimates are only interesting if we can find a reliable measure of the precision of the results. The standard way of doing this is to calculate the statistical error ϵ, which is defined as the standard deviation of the sample mean for a given number of samples. In this section we will show how the statistical error can be estimated using Flyvbjerg-Petersen analysis[5], and also discuss complications that arise when we calculate the statistical error of the projected energy.

12.1 The statistical error

Assume that we have drawn a number of samples $\{x_i, x_2, \ldots, x_n\}$ from the statistical distribution $p(x)$, and that we want to estimate the statistical error ϵ of the mean

$$\overline{f} = \frac{1}{n}\sum_{i=1}^{n} f_i, \quad f_i \equiv f(x_i). \tag{12.1}$$

We further assume that the sampling is ergodic, such that $\lim_{n\to\infty} \overline{f}$ is equal to the ensamble average

$$\langle f \rangle = \int dx\, p(x) f(x). \tag{12.2}$$

The mean $\overline{f}$ is a fluctuating quantity for real data sets where $n < \infty$, and the statistical error ϵ is defined as the standard deviation of $\overline{f}$. We first calculate the variance

$$\epsilon^2 = \langle \overline{f}^2 \rangle - \langle \overline{f} \rangle^2. \tag{12.3}$$

By inserting Eq. (12.1) into Eq. (12.3) it is straight forward to show that

$$\epsilon^2 = \langle [\frac{1}{n}\sum_{i=1}^{n} f_i]^2 \rangle - \langle \frac{1}{n}\sum_{i=1}^{n} f_i \rangle^2 = \frac{1}{n^2}\sum_{i,j=1}^{n} \gamma_{ij}, \quad \gamma_{ij} = \langle f_i f_j \rangle - \langle f_i \rangle \langle f_j \rangle. \tag{12.4}$$

Further manipulation of this expression give

$$\epsilon^2 = \frac{1}{n^2}\left[\sum_{i=1}^{n}\gamma_{ii} + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gamma_{ij}\right] = \frac{1}{n^2}\left[n\gamma_0 + 2\sum_{t=1}^{n-1}(n-t)\gamma_t\right], \tag{12.5}$$

here γ_t is the correlation function

$$\gamma_t = \gamma_{ij},\ t = |i - j|, \tag{12.6}$$

where the correlation only depends on the "time" t between the samples. Now we can write

$$\epsilon^2 = \frac{1}{n}\left[\gamma_0 + 2\sum_{t=1}^{n-1}(1 - t/n)\gamma_t\right]. \tag{12.7}$$

For uncorrelated data sets $\gamma_t = 0$ for $t \neq 0$ and the formula becomes very simple, and the expression for the statistical error can be reduced to the simple form

$$\epsilon = \frac{\gamma_0}{\sqrt{n}}, \tag{12.8}$$

where γ_0 is the standard deviation of f which can be estimated by taking the standard deviation of the data set $\{f_0, f_1, \ldots, f_n\}$.

12.2 Flyvbjerg-Petersen analysis

The samples from concurrent iterations in an FCIQMC simulation are heavily correlated. This is due to the fact that the distribution of walkers fluctuates slowly around an equilibrium distribution, and therefore we can not use the simple expression of Eq. (12.8). We will now introduce the Flyvbjerg-Petersen analysis[5], which is a method where the statistical error can be found without explicitly calculating the correlation function γ_t and at a low computational cost.

Assume that the data set $\{f_1, f_2, f_3, \ldots, f_n\}$ is transformed into a new data set that is half the size $\{f_1^{(2)}, f_2^{(2)}, f_3^{(2)}, \ldots, f_{n/2}^{(2)}\}$ where

$$f_i^{(2)} = \frac{1}{2}\left(f_{2i-1} + f_{2i}\right). \tag{12.9}$$

The key point is that the mean and the statistical error are invariant under this transformation such that

$$\overline{f} = \overline{f^{(2)}},\ \epsilon = \epsilon^{(2)}. \tag{12.10}$$

When we do this transformation repeatedly, we get the data sets

$$\left\{f_1^{(2^m)}, f_2^{(2^m)}, f_3^{(2^m)}, f_{n/2^m}^{(2^m)}\right\},\ m = 1, 2, 3, 4, \ldots. \tag{12.11}$$

and it is proved by induction that all of these data sets have the same mean and statistical error. As we increase m, the correlation between the samples $f_i^{(2^m)}$ and $f_{i+1}^{(2^m)}$ will become smaller. And therefore, as we see from Eq. (12.7),

$$\epsilon^{(2^m)} = \sigma^{(2^m)}/\sqrt{n/2^m} \to \epsilon \text{ as } m \text{ becomes large.} \tag{12.12}$$

We now see that we can find the error ϵ by calculating a series of values $\epsilon^{(2^m)}$ with increasing m until it converges to ϵ. The estimated error will then increase until the point where it remains fixed within statistical fluctuations. This procedure can be performed by writing

a small computer program that repeatedly transforms the initial data set according to Eq. (12.10) and calculates $\epsilon^{(2^m)}$.

Note that the statistical error of ϵ^{2^m} can be estimated using the formula

$$\epsilon^{2^m} \approx \frac{\sigma^{2^m}}{\sqrt{n/2^m}} \left(1 \pm \frac{1}{\sqrt{2(n/2^m))}} \right). \tag{12.13}$$

See [5] for a justification of this formula.

12.3 Flyvbjerg-Pedersen analysis for FCIQMC

To calculate the statistical error of the shift S, we store its value each iteration, and use blocking analysis to calculate the error after the simulation has finished. Since we only store one sample each iteration, the amount of data that needs to be stored is small.

It is somewhat more complicated to calculate the statistical error of the projected energy. This is due to the fact that the projected energy is a function of two stochastic variables. Recall that the projected energy can be written

$$\langle E_p \rangle = \frac{\sum_{j \in \mathcal{C}} H_{0j} \langle n_j \rangle}{\langle n_0 \rangle}, \tag{12.14}$$

where $\mathcal{C}$ is the set of the indices of all determinants that are connected to the reference determinant D_0

$$\mathcal{C} = \left\{ j \, ; \, \langle D_0 | \hat{H} | D_j \rangle \neq 0 \right\}. \tag{12.15}$$

Each iteration i, we save the two quantities

$$E_i \equiv \sum_{j \in \mathcal{C}} H_{0j} n_j^{(i)}, \text{ and } G_i \equiv n_0^{(i)}. \tag{12.16}$$

The projected energy can be written as a function of these quantities

$$\langle E_p \rangle = \frac{\sum_{i=1}^{N} E_i / N}{\sum_{i=1}^{N} G_i / N} = \frac{\sum_{i=1}^{N} E_i}{\sum_{i=1}^{N} G_i}. \tag{12.17}$$

However, we can not use the blocking algorithm on $\langle E_P \rangle$ because there is no simple way to split up the fraction of the sums into blocks. We have instead sorted the individual stochastic variables in blocks $E_i^{(n)}$ and $G_i^{(n)}$

$$E_i^{(n)} \equiv \frac{1}{n} \sum_{k=n(i-1)+1}^{ni} E_i, \tag{12.18}$$

$$G_i^{(n)} \equiv \frac{1}{n} \sum_{k=n(i-1)+1}^{ni} G_i, \tag{12.19}$$

$$\text{with } i \in [1, N/n], \tag{12.20}$$

such that the projected energy can be written

$$\langle E_P \rangle \approx \frac{\sum_{i=1}^{M} E_i^{(n)}}{\sum_{i=1}^{M} G_i^{(n)}} = \frac{1}{M} \sum_{i=1}^{M} \frac{E_i^{(n)}}{G_i^{(n)}} + \xi_n, \quad M = N/n. \tag{12.21}$$

and the sample variance

$$\mathrm{Var}\left[\frac{\sum_{i=1}^{M} E_i^{(n)}}{\sum_{i=1}^{M} G_i^{(n)}}\right] = \mathrm{Var}\left[\frac{1}{M}\sum_{i=1}^{M}\frac{E_i^{(n)}}{G_i^{(n)}} + \xi_n\right]$$
$$= \mathrm{Var}\left[\frac{1}{M}\sum_{i=1}^{M}\frac{E_i^{(n)}}{G_i^{(n)}}\right] + \mathrm{Var}\left[\xi_n\right] + 2\mathrm{Cov}\left[\frac{1}{M}\sum_{i=1}^{M}\frac{E_i^{(n)}}{G_i^{(n)}}, \xi_n\right]. \quad (12.22)$$

In the case that the statistical fluctuations of ξ_n are much smaller than the statistical fluctuations of the sample mean, as we see from the above equation, the statistical error can be estimated by doing blocking analysis on the dataset $\left\{E_i^{(n)}/G_i^{(n)}\right\}_{i=1}^{N/n}$ threating $E_i^{(n)}/G_i^{(n)}$ as a single stochastic variable. We will now find an expression for ξ_n, and show that it in fact can be neglected.

The error ξ_n can be found by considering the Taylor expansion of the projected estimator. We let μ_E be the sample mean of $\{E_i\}$ and μ_G be the sample mean of $\{G_i\}$. The multivariate Taylor expansion of the fraction $E_i^{(n)}/G_i^{(n)}$ around the point (μ_E, μ_G) is

$$T\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = T_0\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] + T_1\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] + T_2\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] + \ldots \quad (12.23)$$

The zeroth, first and second order terms are

$$T_0\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{\mu_E}{\mu_G}, \quad (12.24)$$

$$T_1\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{1}{\mu_G}(E_i^{(n)} - \mu_E) - \frac{\mu_E}{\mu_G^2}(G_i^{(n)} - \mu_G), \quad (12.25)$$

$$T_2\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{\mu_E}{\mu_G^3}(G_i^{(n)} - \mu_G)^2 - \frac{1}{2\mu_G^2}(G_i^{(n)} - \mu_G)(E_i^{(n)} - \mu_E). \quad (12.26)$$

We want to find the Taylor expansion of the sum $(1/M)\sum_{i=1}^{M}(E_i^{(n)}/G_i^{(n)})$. This is a multivariate Taylor expansion $\tilde{T}$ in the variables $(E_1^{(n)}, \ldots, E_M^{(n)}, G_1^{(n)}, \ldots, G_M^{(n)})$, but since each term of the sum only depends on one pair $E_i^{(n)}, G_i^{(n)}$, it can be written as the sum

$$\tilde{T}\left[\frac{1}{M}\sum_{i=1}^{M}\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{1}{M}\sum_{i=1}^{M} T\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right]. \quad (12.27)$$

The zeroth order term is

$$\frac{1}{M}\sum_{i=1}^{M} T_0\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{\mu_E}{\mu_G}, \quad (12.28)$$

which is exactly the projected energy. The first order contribution disappears since

$$\frac{1}{M}\sum_{i=1}^{M} T_1\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{1}{\mu_G}\left(\frac{1}{M}\sum_{i=1}^{M} E_i^{(n)} - \mu_E\right) - \frac{\mu_E}{\mu_G^2}\left(\frac{1}{M}\sum_{i=1}^{M} G_i^{(n)} - \mu_G\right) = 0. \quad (12.29)$$

This means that the leading term in the error ξ_n is the second order term

$$\frac{1}{M}\sum_{i=1}^{M} T_2\left[\frac{E_i^{(n)}}{G_i^{(n)}}\right] = \frac{\mu_E}{\mu_G^3}\frac{1}{M}\sum_{i=1}^{M}(G_i^{(n)}-\mu_G)^2 - \frac{1}{2\mu_G^2}\frac{1}{M}\sum_{i=1}^{M}(G_i^{(n)}-\mu_G)(E_i^{(n)}-\mu_E)$$
$$= \frac{\mu_E}{\mu_G^3}\mathrm{Var}(G^{(n)}) - \frac{1}{2\mu_G^2}\mathrm{Cov}(G^{(n)},E^{(n)}), \tag{12.30}$$

where $\mathrm{Var}(G^{(n)})$ is the variance of the data set $\{G_1^{(n)},\ldots,G_M^{(n)}\}_{i=1}^N$ and $\mathrm{Cov}(E^{(n)},G^{(n)})$ is the covariance of the data sets $\{G_1^{(n)},\ldots,G_M^{(n)}\}_{i=1}^N$ and $\{E_1^{(n)},\ldots,E_M^{(n)}\}_{i=1}^N$. We have now shown that ξ_n has a leading term

$$\xi_n \approx \frac{\mu_E}{\mu_G^3}\mathrm{Var}(G^{(n)}) - \frac{1}{2\mu_G^2}\mathrm{Cov}(G^{(n)},E^{(n)}), \tag{12.31}$$

which, for the systems we have treated, is a very small number. Note that typical values for the means are $\mu_G \sim 10^3$-10^5 and $\mu_E/\mu_G \sim 10^{-1}$-10. The fluctuations of ξ_n are expected to be much smaller than the fluctuations of the projected estimator and can therefore be neglected when we calculate the statistical error. As we can see from Fig. 12.2, ξ_n falls off as a function of n which means that it is favourable to use a large n. We therefore leave the block size at one, and increase n instead. This is analogous to increasing the block size when ξ_n is negligible, but in the case of a large ξ_n we would see a convergence of the statistical error in terms of this variable as well.

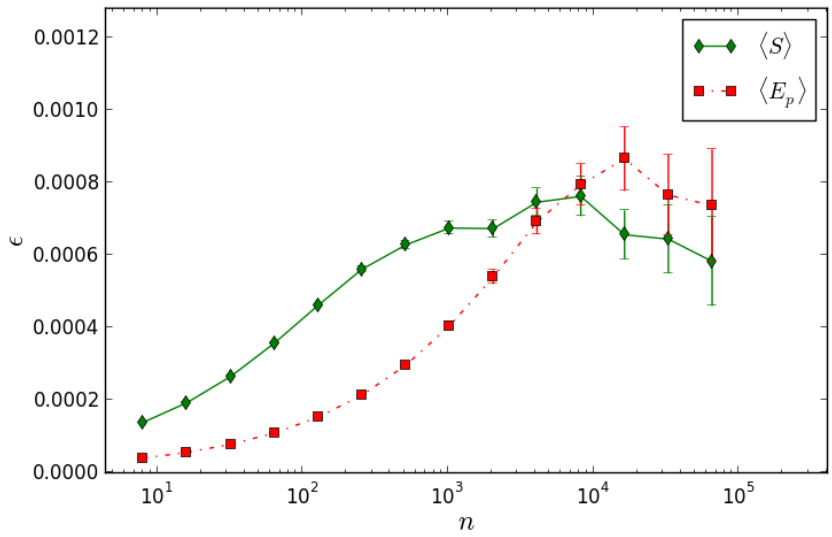


Figure 12.1: Blocking analysis of the of the projected estimator $\langle E_P\rangle$ and the generational estimator $\langle S\rangle$. We see convergence at ~ 1 mili-Hartree for both estimators. The blocksize n is doubled between each data point.

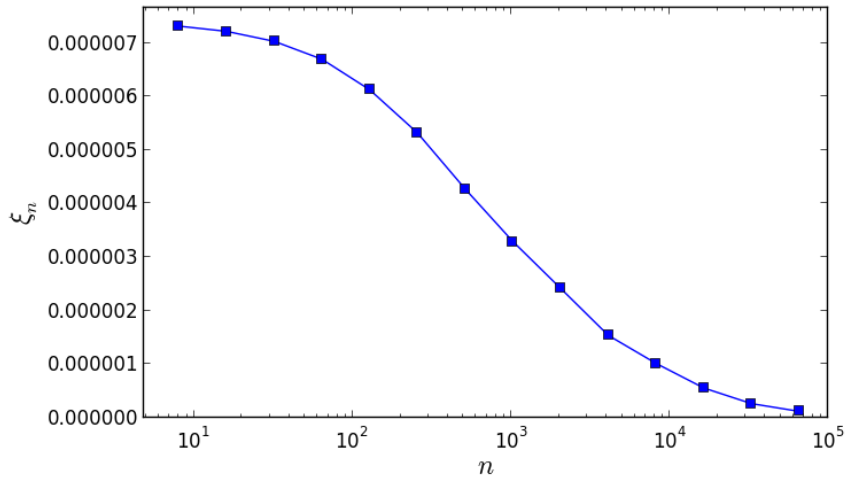


Figure 12.2: The error ξ_n as a function of n, where n is the blocksize. Both of these are defined in Eq. (12.21). The error is in units of Hartree, and is very small compared to the projected energy which is close to 20 Hartree.

Chapter 13

Curve fitting

We will here introduce the least square method which is a method of fitting parametrized functions to data points. This section is to a large degree based on the note on least square methods by Richter (1995) [28] and the article on the Levenberg-Marquardt algorithm by Marquardt (1963) [17].

13.1 The least square method

Assume that we have a set of points $\{(x_i, y_i)\}_{i=1}^{n}$, and that we want to fit to the function $y(x, \mathbf{a})$, $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ to these points with respect to the parameters a_i. A common optimization method is the least squares method, where the optimal values for a_i are found by minimizing the function

$$M(\mathbf{a}) = \sum_{i=1}^{n} \frac{[y(x, \mathbf{a}) - y_i]^2}{\sigma_i^2}, \tag{13.1}$$

where σ_i is the standard deviation of the value y_i. The minima is found at the point $\mathbf{a}_0$ where

$$\left[\frac{\delta M(\mathbf{a})}{\delta a_i}\right]_{\mathbf{a}_0} = 0 \; \forall \; i \in [1, m]\,. \tag{13.2}$$

We will first look at the linear least square problem where $y(x, \mathbf{a})$ can be written as a linear function. This is the starting point for nonlinear least square problems as well since all functions can be approximated as a first degree Taylor polynomial close to a minima. The linear $y(x, \mathbf{a})$ is written

$$y(x_j, \mathbf{a}) = \sum_{i=1}^{m} a_i f_i(x_j) = \sum_i a_i F_{ji}, \tag{13.3}$$

which can be rewritten as the matrix equation

$$\mathbf{y} = \mathbf{F}\mathbf{a}. \tag{13.4}$$

Eq. (13.1) can also be written as a matrix equation

$$M(\mathbf{a}) = (\mathbf{b} - \mathbf{X}\mathbf{a})^T (\mathbf{b} - \mathbf{X}\mathbf{a})\,, \tag{13.5}$$

where $\mathbf{b}$ is a vector with the components $b_i = y_i/\sigma_i$ and $\mathbf{X}$ is a matrix $X_{ij} = f_j(x_i)/\sigma_i$. The minima is found when Eq. (13.2) is fulfilled. Derivation with respect to a_i now yields

$$\begin{aligned} 0 &= 2\mathbf{X}^T(\mathbf{b} - \mathbf{X}\mathbf{a}_0) \\ &\Rightarrow \mathbf{X}^T\mathbf{X}\mathbf{a}_0 = \mathbf{X}^T\mathbf{b} \\ &\Rightarrow \mathbf{a}_0 = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\mathbf{b} = \mathbf{C}\mathbf{X}^T\mathbf{b}. \end{aligned} \tag{13.6}$$

This problem can be solved efficiently on a computer using standard methods. See for example [8] for a detailed description.

The nonlinear problem is typically solved by using an iterative algorithm that searches for the minima of $M(\mathbf{a})$ in the parameter space of $\mathbf{a}$. One class of such methods are the gradient methods, where one repeatedly steps in the direction of the negative gradient $\mathbf{a} \leftarrow \mathbf{a} - k\nabla_{\mathbf{a}}M(\mathbf{a})$, until the minima is found. Another class of iterative methods are the Gauss-Newton methods, where the strategy is to expand the nonlinear function as a first order Taylor polynomial and solve it as a linear least square problem. The procedure is repeated until convergence. The most popular least squares algorithms, like the Levenberg-Marquardt method [17], combines the individual strengths of these algorithms. Several highly optimized implementations of the Levenberg-Marquardt algorithm are available as for example the MinPack library, or its Python wrapper `curve_fit` in the SciPy library.

13.2 Error analyses of the least square fit

In this section we will only look at the nonlinear problem. More specifically, we want to calculate the error of the extrapolated function $y(x, \mathbf{a})$ at a point x based on the standard deviations σ_i of the data.

When we get sufficiently close to the minima $\mathbf{a}_0$, the $y(x, \mathbf{a})$ and $M(\mathbf{a})$ can be written as first order Taylor expansions

$$y(x, \mathbf{a}) = y(x, \mathbf{a}_0) + (\mathbf{a} - \mathbf{a}_0)^T\mathbf{d}(x, \mathbf{a}_0), \tag{13.7}$$

$$M(\mathbf{a}) = M(\mathbf{a}_0) + (\mathbf{a} - \mathbf{a}_0)^T\mathbf{H}(\mathbf{a}_0)(\mathbf{a} - \mathbf{a}_0), \tag{13.8}$$

where $\mathbf{H}(\mathbf{a})$ has elements H_{ij} and $\mathbf{d}(x, \mathbf{a})$ has elements d_i

$$d_i = \frac{\delta y(x, \mathbf{a})}{\delta a_i}, \; H_{ij} = \frac{\delta^2 M(\mathbf{a})}{\delta a_i \delta a_j}. \tag{13.9}$$

If we now substitute $f(x) \to d(x, \mathbf{a}_0)$ in Eq. (13.3) we get

$$X_{ij} = \frac{1}{\sigma_i}\frac{\delta y(x_i)}{\delta a_j}. \tag{13.10}$$

Now it is easy to see that we end up with the same expression for the minima as in the linear case

$$\mathbf{a} = \mathbf{C}\mathbf{X}^T\mathbf{b}. \tag{13.11}$$

An infinitesimal change in $\mathbf{a}_0$ is written

$$\partial\mathbf{a} = \mathbf{C}\mathbf{X}^T\partial\mathbf{b}. \tag{13.12}$$

The covariance matrix σ_a^2 with elements $\sigma_{a_{ij}}$ is

$$\begin{aligned}\sigma_a^2 &= \langle \partial\mathbf{a}\partial\mathbf{a}\rangle = \langle \mathbf{C}(\mathbf{X}^T\partial\mathbf{b})(\mathbf{X}^T\partial\mathbf{b})^T C\rangle \\ &= \mathbf{C}\mathbf{X}^T\langle\partial\mathbf{b}\partial\mathbf{b}^T\rangle\mathbf{X}\mathbf{C}^T. \end{aligned} \tag{13.13}$$

The matrix $\langle\partial\mathbf{b}\partial\mathbf{b}^T\rangle$ has the elements $\langle(\partial y_i/\sigma_i)(\partial y_j/\sigma_j)\rangle$. Here $\{y_i\}$ are assumed to be statistically independent variables, which means that

$$\langle(\partial y_i/\sigma_i)(\partial y_j/\sigma_j)\rangle = \delta_{ij} \tag{13.14}$$

$$\Rightarrow \langle\partial\mathbf{b}\partial\mathbf{b}^T\rangle = 1, \tag{13.15}$$

and

$$\sigma_a^2 = \mathbf{C}\mathbf{X}^T\mathbf{X}\mathbf{C}^T = \mathbf{C}\mathbf{C}^{-1}\mathbf{C}^T = \mathbf{C}. \tag{13.16}$$

This result allows us to calculate the covariance matrix $\sigma_y^2(x,x',\mathbf{a}) = \langle\partial y(x,\mathbf{a})\partial y(x',\mathbf{a})\rangle$ which in the special case that $x = x'$ is the variance of $y(x,\mathbf{a})$.

$$\begin{aligned}\sigma_y^2(x,x',\mathbf{a}) &= \langle\partial y(x,\mathbf{a})\partial y(x',\mathbf{a})\rangle \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m}\langle\frac{\delta y(x,\mathbf{a})}{\delta a_i}\frac{\delta y(x',\mathbf{a})}{\delta a_j}\partial a_i\partial a_j\rangle \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m}\langle\partial a_i\partial a_j\rangle\frac{\delta y(x,\mathbf{a})}{\delta a_i}\frac{\delta y(x',\mathbf{a})}{\delta a_j} \\ &= \mathbf{d}(x',\mathbf{a})^T\mathbf{C}\mathbf{d}(x,\mathbf{a}), \end{aligned} \tag{13.17}$$

which means that the standard deviation of $y(x,\mathbf{a})$ can be written

$$\sigma_y^2(x,x,\mathbf{a}) = \sqrt{\mathbf{d}(x,\mathbf{a})^T\mathbf{C}\mathbf{d}(x,\mathbf{a})}. \tag{13.18}$$

13.3 Error analysis of $E(R)$

We will use Eq. 13.18 from the last section to find the estimated error of the parametrized energy $E(R)$ defined in Eq. (5.11) at $R\to\infty$. The error vector $\mathbf{d}$ with the elements d_i is

$$d_i = \left[\frac{\delta E}{\delta a_i}\right]_{a=a_0}. \tag{13.19}$$

For the parametrized function for the error $E(R)$ we find the elements d_i:

$$\left[\frac{\delta E(R)}{\delta a}\right]_{\mathbf{a}=\mathbf{a}_0} = 1, \tag{13.20}$$

$$\left[\frac{\delta E(R)}{\delta b}\right]_{\mathbf{a}=\mathbf{a}_0} = -b_0\sum_{r=1}^{R}(N+r)r^{-c_0}, \tag{13.21}$$

$$\left[\frac{\delta E(R)}{\delta c}\right]_{\mathbf{a}=\mathbf{a}_0} = -b_0\sum_{r=1}^{R}\log(c_0)(N+r)r^{-c_0}. \tag{13.22}$$

<++> The error is always converging at a large R. We already know that $\sum_{r=1}^{R} r^{-(c_0-1)}$ converges for $c > 2$. The sum $\sum_{r=1}^{R}\log(r)r^{-c_0}$ will also converge since it is bounded by $\sum_{r=1}^{R} r^{-(c_0-1)}$.

Part V

Implementation of the FCIQMC algorithm

In this part we aim to give an overview of our implementation of FCIQMC. The code is written in C++, and is tailored to calculate energies of the stationary states of 2-dimensional open shell quantum dots. It is organized in separate interchangeable modules, meaning that it relatively easily can be modified to tackle other systems such as 3-dimensional quantum dots, molecules or atomic nuclei, and is fully parallelized using a hybrid approach which combines multithreading and parallelization with MPI.

The main ideas behind the algorithm is more or less explained in part III, but there are several aspects of the implementation that we want to discuss further such as the numerical representation of the determinants and the state vectors, the code structure, parallelization, optimizations etc.

In the first three chapters we give an overview of our implementation. First we write about the numerical representation of the state vector followed by a description of how we have implemented the basic (sequential) algorithm and how we have implemented the parallel algorithm. After reading these chapters, one should have a general understanding of how our implementation works. In the fourth chapter we look closer at the code. We present the class structure and provide details about the different classes. This chapter should give a good understanding of how the code is structured and what responsibilities of the different classes are. In the fifth chapter we have presented some benchmarks and discuss the efficiency of the program. We have also looked at different run time parameters, and how they should be tuned to optimize the performance of the code. The sixth and last chapter contains a short discussion of how the code should be modified in we want to simulate different systems than the two dimensional quantum dots.

Chapter 14

Numerical representation of the determinants, the basis and the state vector

14.1 The determinants

A large part of the of the code is devoted to manipulating, sorting, reading from and transferring information about determinants and their populations. It is therefore crucial for the efficiency of the code that we find a smart way of representing them numerically, which is both space efficient and computationally fast. We have chosen to store the determinants as binary strings, $"b_{\varphi_0}, b_{\varphi_1}, \ldots, b_{\varphi_{2M}}"$, with the same number of elements as the number of spin orbitals $2M$ that are accessible in the basis. The bit b_{φ_i} here represents the i'th orbital, and is set to 1 if the orbital is occupied and 0 otherwise. As an example, the state

$$|D_i\rangle = |\varphi_0, \varphi_1, \varphi_3, \varphi_5\rangle,\ \varphi_i \in \{\varphi_0, \varphi_1, \varphi_2, \ldots, \varphi_{2M}\} \tag{14.1}$$

would be represented as the binary string

$$|D_i\rangle \rightarrow "b_{\varphi_0} b_{\varphi_1} b_{\varphi_2} b_{\varphi_3} \ldots b_{\varphi_{2M}}" = "1\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0 \ldots 0" \tag{14.2}$$

Note that in the direct product basis, the number of possible combinations of $"b_{\varphi_0}, \ldots, b_{\varphi_{2M}}"$ is exactly the number of determinants in the basis, which means that this is the most space efficient way of storing the determinants. We have chosen to use the C++ container class `sdt::bitset` to store the binary strings. This class is optimized to be space efficient, and each element occupies only one bit of memory. Note that `bool`, which probably is the most used type for storing bits, usually occupies 8 bits memory per bit of information. Low memory usage is an important consideration when optimizing the code for parallel computations since we want to minimize the communication overhead.

For numerical reasons that will be made clear in the next chapter, we want to keep the determinants ordered. We have therefore defined the operator $<$ as

$$\begin{aligned} &|\varphi_{a_1}\varphi_{a_2}, \ldots, \varphi_{a_{N_P}}\rangle < |\varphi_{b_1}\varphi_{b_2}, \ldots, \varphi_{b_{N_P}}\rangle \text{ only if} \\ &2^{2M-a_1} + 2^{2M-a_2} + \cdots + 2^{2M-a_{N_P}} < 2^{2M-b_1} + 2^{2M-b_2} + \cdots + 2^{2M-b_{N_P}}, \end{aligned} \tag{14.3}$$

and we define the determinants $\{|D_{n_1}\rangle, |D_{n_2}\rangle, \dots\}$ to be sorted when $|D_{n_i}\rangle < |D_{n_{i+1}}\rangle$. In the rest of this thesis, when we say that a determinant is greater or lesser than another determinant or that a set of determinants are sorted, we are referring to this equation.

14.2 The single particle basis

We keep $2M$ spin orbitals in our basis, each identified by its index $i \in [0, 2M-1]$. Each spin orbital φ_i is uniquely represented by the triplet (m, s, n), where m is the magnetic quantum number, s is the spin and n is the principal quantum number. Given the index i, the triplet (m, s, n) is stored in the following way:

(i): φ_i has spin $\uparrow$ if i is odd and $\downarrow$ if i is even.

(ii): The quantum numbers n and m are stored in two arrays of length M.

```
int *pi_m = new int[M];
int *pi_n = new int[M];
```

The magnetic quantum number of orbital i is stored as the floor$(i/2)$'th element in `pi_m`, and the principal quantum number is stored as the floor$(i/2)$'th element in `pi_n`.

14.3 The state vector

The state vector is represented by the signed population of walkers that populate different determinants. Assume that N_D is the number of populated determinants. We store the determinants as the N_D first elements of a `bitset` array.

```
bitset<N> *pb_determinants = new bitset<N>[i_ndmax];
```

And the signed population of walkers is stored as the N_D first elements of a `long` array.

```
long *pl_population = new long[i_nwmax];
```

Here the population of `pb_determinants[i]` is stored as `pl_population[i]`.

The number of populated determinants N_D will vary from one iteration to the next, but usually only a fraction of the total number of determinants in the basis are populated. So every iteration these arrays have to be rearranged, such that the populated determinants always are stored as the N_D first elements.

Chapter 15

A single FCIQMC iteration

The aim of this chapter is to explain how the sequential FCIQMC algorithm has been implemented. We will do this by going through a single iteration of the algorithm, which includes the kill/clone, spawn and annihilation step.

Assume that we start with a population of walkers $\left\{n_{i_0}, n_{i_1}, \ldots, n_{i_{N_D}}\right\}$ where n_i is the signed population of the determinant $|D_i\rangle$. As we already have discussed, the population is represented by two vectors

$$\texttt{pb\_determinant[m]} \leftarrow |D_{i_m}\rangle, \tag{15.1}$$

$$\texttt{pl\_population[m]} \leftarrow n_{i_m}. \tag{15.2}$$

We further assume that the determinants are indexed and sorted such that $|D_{i_m}\rangle < |D_{i_n}\rangle$ if $m < n$. During an FCIQMC iteration, we loop through all determinants

$$|D_\alpha\rangle,\ \alpha = i_1, i_2, \ldots, i_{N_D}, \tag{15.3}$$

where each determinant is constructed from the orbitals $\left\{\varphi_{\alpha_1}, \ldots, \varphi_{\alpha_{N_P}}\right\}$

$$|D_\alpha\rangle = |\varphi_{\alpha_1}, \ldots, \varphi_{\alpha_{N_P}}\rangle. \tag{15.4}$$

A: The kill/clone and spawn steps:

For each element we perform the following steps:

1: Calculate the killing/cloning probability:

We first calculate and store the diagonal matrix element $\langle D_\alpha|\hat{H}|D_\alpha\rangle$

$$\texttt{p\_dp} \leftarrow \langle D_\alpha|\hat{H}|D_\alpha\rangle, \tag{15.5}$$

which is given by Eq. (15.6)

$$\langle D_\alpha|\hat{H}|D_\alpha\rangle = \sum_{i=1}^{N_P}\langle\varphi_{\alpha_i}|\hat{H}_{HO}|\varphi_{\alpha_i}\rangle + \frac{1}{2}\sum_{ij=1}^{N_P}\langle\varphi_{\alpha_i}\varphi_{\alpha_j}|\hat{V}|\varphi_{\alpha_i}\varphi_{\alpha_j}\rangle_{AS}. \tag{15.6}$$

The kill/clone probability is now calculated according to Eq. (8.11).

```
d_pd = d_dt*(d_pd-dS);
```

Here `d_dt`$\leftarrow \tau$ is the timestep and `d_dS`$\leftarrow S$ is the shift.

2: Loop over all walkers on $|D_\alpha\rangle$:

We then enter a loop where we do the kill/clone and spawn step for all walkers on the current determinant $|D_\alpha\rangle$. We let

$$\texttt{i\_loopw} \leftarrow |n_\alpha|, \tag{15.7}$$

and enter the loop:

```
for (; i_loopw>0; i_loopw--)
```

- *2a: Perform the spawning step:*

  We attempt to spawn walkers on a connected determinant $|D_\beta\rangle$. As we discussed in chapter 10, we let $|D_\beta\rangle$ be a single excitation of $|D_\alpha\rangle$ with probability d_s and a double excitation with probability $(1 - d_s)$. We draw a random uniform $\chi \in [0, 1]$ and if $\chi < d_s$ we try to spawn walkers on a determinant

  $$|D_\beta\rangle = (-1)^P a_p^\dagger a_k |D_\alpha\rangle, \tag{15.8}$$

  and else we try to spawn walkers on a determinant

  $$|D_\beta\rangle = (-1)^P a_p^\dagger a_q^\dagger a_k a_l |D_\alpha\rangle, \tag{15.9}$$

  where k, l, p, q are sampled according to the rules that we set up in chapter 10, and P is the number of permutations that are necessary to rearrange the annihilation and creation operators. We spawn $n_{spawned}$ new walkers, where

  $$n_{spawned} = \text{floor}\left(|p_d(\beta, \alpha)| + 1\right). \tag{15.10}$$

  and $p_d(\beta, \alpha)$ is calculated according to Eq. (8.12)

  $$p_d(\beta|\alpha) = \tau \langle D_\alpha|\hat{H}|D_\beta\rangle / p_{gen}(\beta|\alpha). \tag{15.11}$$

  Here $p_{gen}(\beta|\alpha)$ is the sampling probability of determinant $|D_\beta\rangle$ from determinant $|D_\alpha\rangle$ which we also discussed in chapter 10. The off diagonal matrix element $\langle D_\alpha|\hat{H}|D_\beta\rangle$ is calculated according to Eq. (4.29)

  $$\langle D_\alpha|\hat{H}\{a_p^\dagger a_k\}|D_\alpha\rangle = \sum_{j=1}^{N_P} \langle \varphi_p \varphi_{\alpha_j}|\hat{V}|\varphi_k \varphi_{\alpha_j}\rangle_{AS}, \tag{15.12}$$

  in the case of a single excitation and Eq. (4.30)

  $$\langle D_\alpha|\hat{H}\{a_p^\dagger a_q^\dagger a_k a_l\}|D_\alpha\rangle = \langle \varphi_p \varphi_q|\hat{V}|\varphi_k \varphi_l\rangle_{AS}, \tag{15.13}$$

  in the case of a double excitation. If a spawning event is successful, we store the newly spawned walkers on the arrays: `pb_new` and `ppl_new`. The first is an $1 \times n_{max}$ `bitset` array which stores the determinant $|D_\beta\rangle$, and the second is a $2 \times n_{max}$ long array that stores the signed number of new walkers, and the integer n_{max} is a preset value that

sets the size of the array. The newly spawned walkers and whether their parents were initiators, are stored according to the following rules:

$$\texttt{pb\_new[l\_nnew]} \leftarrow |D_\beta\rangle \tag{15.14}$$

if $|D_\alpha\rangle$ is an initiator: (15.15)

$$\texttt{ppl\_new[0][l\_nnew]} \leftarrow n_{spawned},$$
$$\texttt{ppl\_new[1][l\_nnew]} \leftarrow 0,$$

else if $|D_\alpha\rangle$ is a non-initiator:

$$\texttt{ppl\_new[0][l\_nnew]} \leftarrow 0,$$
$$\texttt{ppl\_new[1][l\_nnew]} \leftarrow n_{spawned}, \tag{15.16}$$

$$\texttt{l\_nnew} \leftarrow \texttt{l\_nnew} + 1. \tag{15.17}$$

Here `l_nnew` is an integer that we use to count the number of newly spawned walkers. This parameter is set to zero at the beginning of each FCIQMC iteration.

Note that according to the spawning rules of i-FCIQMC, we need to know if the parent of a walker is an initiator, the total number of non- initiators trying to spawn on the same determinant and whether $|D_\beta\rangle$ was previously populated before we can say if the spawning event was successful. Consequently, we do not know whether a spawning event was successful before after an iteration.

- *2b: Perform the kill/clone step:*

  We try to kill or clone a walker with the probablity $\texttt{d\_pd} \leftarrow P_d(\alpha|\alpha)$. Remember that `d_pd` is already calculated and stored, and is equal for all walkers on the current determinant. We now attempt to kill or spawn walkers according to the following rules:

  first draw a random uniform $\chi \in [0, 1]$ (15.18)

  if $\texttt{d\_pd} > \chi$:

  $$\texttt{pl\_population}[\alpha] -= \text{sign}(n_\alpha) * \text{floor}(|\texttt{d\_pd}| + 1), \tag{15.19}$$

  else if $|\texttt{d\_pd}| > \chi$:

  $$\texttt{pl\_population}[\alpha] += \text{sign}(n_\alpha) * \text{floor}(|\texttt{d\_pd}| + 1). \tag{15.20}$$

  where n_α is the signed population on $|D_\alpha\rangle$.

B: The annihilation step:

When we have run through the previous steps for all determinants, we must annihilate all pairs of walkers of opposite signs that populate the same determinant. We have chosen to do this in the following way: We first sort the arrays that contain the newly spawned walkers `pb_new, ppl_new`. The determinants array are sorted in ascending order, and the population arrays are reordered in exactly the same way. Note that these arrays might contain equal determinants, in which case we sum the populations and remove the duplicate. For example, it might happen that

$$\texttt{pb\_new[i] = pb\_new[j]}, \tag{15.21}$$

and in this case we sum

$$\texttt{ppl\_new[0][i] += ppl\_new[0][j]}, \tag{15.22}$$

$$\texttt{ppl\_new[1][i] += ppl\_new[1][j]}, \tag{15.23}$$

remove `pb_new[j]`, `pb_new[0][j]` and `pb_new[1][j]` from the simulation and subtract one from `l_nnew` (remember that `l_nnew` is the total number of spawning events). After this step we end up with a sorted determinants array

$$\texttt{pb\_new[i]} < \texttt{pb\_new[i+1]} \text{ for all } i \in [0, \texttt{l\_nnew}-2], \tag{15.24}$$

and where `ppl_new[0][i]` and `ppl_new[0][i]` contain all walkers spawned on `pb_new[i]` from initiators and non- initiators respectively.

The next step is now to merge the arrays of old and newly spawned walkers. We make use of temporary arrays `pb_temp` and `pl_temp`, and copy both the old and new determinants to these arrays. This must be done according to the rules of the initiator adaption of FCIQMC. This part of the algorithm is most easily understood by reading the C++ code which we have pasted below. Here `l_ndet` is the number of elements in the arrays that stores the old determinants and `l_nnew` is the number of elements in the sorted list of newly spawned walkers.

```
long i = 0, j = 0, n = 0, l_temp;
while ( (i<l_ndet) && (j<l_nnew) )
{
  if < ... pb_determinants[i] == pb_new[j] ... >
  {
    pb_temp[n] = pb_determinants[i]; //old determinants
    pl_temp[n] = pl_population[i];
    // here all attempts to spawn is accepted since the
    // determinant was previously populated
    pl_temp[n] += ppl_new[j][0];
    pl_temp[n] += ppl_new[j][1];

    i++;
    j++;
  }
  else if < ... pb_determinants[i] > pb_new[j] ... >
    {
      l_temp = 0;
      // initiators are always allowed to spawn
      l_temp += ppl_new[j][0];
      // non initiators are only allowed to spawn if
      // the absolute value of the sum of new walkers
      // is larger than one
      if ( abs(ppl_new[j][1]) > 1 )
      {
        l_temp += ppl_new[j][1];
      }
      pb_temp[n] = pb_new[j];
      pl_temp[n] = l_temp;
      j++;
    }
```

```
  else  \\( < ... pb_determinants[i] < pb_new[j] ... > )
  {
    //old determinants
    pb_temp[n] = pb_determinants[i];
    pl_temp[n] = pl_population[i];
    i++;
  }
  // if the population is larger than 0, accept the determiant
  if ( pl_temp[n] != 0 )
  {
    n++;
  }
}
```

Now, either all old walkers or all new walkers are copied to the temporary arrays. The next step is to copy the rest of the walkers:

```
if (i==l_ndet)
{

  while (j<l_nnew) // all old walkers are already copied
  {
    pb_temp[n] = pb_new[j];
    pl_temp[n] = ppl_new[j][0];
    if ( abs(ppl_new[j][1])>1 )
    {
      pl_temp[n] += ppl_new[j][1];
    }
    j++;
    if ( pl_temp[n] != 0 )
    {
      n++;
    }
  }
}
else if ( j==l_nnew )
{
  while ( i<l_ndet) // all new walker are already copied
  {
    pb_temp[n] = pb_determinants[i];
    pl_temp[n] = pl_population[i];
    i++;
    if ( pl_temp[n] != 0 )
    {
      n++;
    }
  }
}
```

Now, all determinants are contained in the n first elements of the temporary arrays. We finish by swapping the pointers of the temporary arrays and the populations and determinants arrays, and reset the number of walkers and determinants.

```
//reset number of determinants
l_ndet = n;
l_nnew = 0;

//Pointer swapping
bitset<N> *pb_sswap = pb_determinants;
pb_determinants = pb_temp;
pb_temp = pb_swap;

long *pl_swap = pl_population;
pl_population = pl_temp;
pl_temp = pl_swap;
```

C: Varying the shift:

The last step is to vary the shift S to stabilize the populatios. We assume that the old number of walkers is already stored as `l_oldnumwalkers`. The new number of walkers `l_numwalkers` can be calculated:

```
l_numwalkers = 0;
for (int i=0; i<l_ndet; i++)
  l_numwalkers += abs(pl_population[i]);
```

The new shift is now calculated according to Eq. (8.16).

Chapter 16

Parallelization

16.1 Introduction

Parallelization allows simulations on multicore computers or clusters of interconnected computers or nodes. There are two main models of parallel computing, the shared memory model and the distributed memory model, which are based on different underlying memory architectures. Shared memory programs communicate by changing shared memory variables and distributed memory programs communicate via message passing. On multicore computers, we can choose whether to use a shared or a distributed memory model. On computational clusters, where the memory is only accessible to the cores on each node, a distributed memory model must be chosen. When we implemented FCIQMC, we found that it was necessary to use a composite approach combining the shared and the distributed memory models. For simulations with a large set of basis functions, the amount of memory needed to store the Coulomb matrix elements is so large that at least some of the cores have to share memory. Distributed memory parallelization was implemented because we wanted the code to run on computational clusters with many nodes.

FCIQMC is in principle a highly paralellizable algorithm. The most time consuming part of the algorithm is the clone/kill and spawn steps which must be calculated for a large number $\sim 10^5$ – 10^8 of walkers. These calculations are independent of each other and can therefore be run in parallel without any communication between the processes. The annihilation step does however include a lot of communication and therefore induces a significant communication overhead. In this chapter we will discuss these issues along with other details of the parallelization of algorithm.

16.2 Numerical libraries

We have used the library OpenMP[21] which is an interface for shared memory parallelization. More specifically, it is an implementation of multithreading which is a method where several lightweight processes or threads are started and allocated to different cores.

OpenMP has been implemented using a coarse grained model where the threads only share a minimum of objects and variables, as for example the Coulomb matrix elements and the list of walkers. This was done to minimize the number of locks and mutex'es (critical and atomic clauses in OpenMP) that are necessary to avoid conflicting read and write operations to the same memory locations.

MPI is an abbreviation for Message Passing Interface and is a library specification which provides an interface for synchronization and communication between processes with distributed memory. The MPI standard defines the syntax and the semantics of a number of library routines and has several implementations, among others OpenMPI[7] which we have used in our code.

16.3 The distribution of walkers on the MPI tasks

Due to the annihilation step in the FCIQMC algorithm, we need a fast method to search through the population and find all walkers that reside on any given determinant, and this becomes more difficult when we run the code with many MPI tasks. This problem can be solved by distributing the determinants in a specific way which we will now discuss.

We assume that the determinants are enumerated such that

$$|D_i\rangle < |D_{i+1}\rangle, \tag{16.1}$$

where the operator $<$ is defined in Eq. (14.3), and that the walkers are distrubuted on multiple MPI tasks which are enumerated $1, 2, 3, \ldots$. The determinants $\{|D_{i_1}\rangle, |D_{i_2}\rangle, \ldots\}$, are sorted and distributed such that only walkers that populates determinants $|D_i\rangle$ with i in some interval $[M(n-1)+1, M(n)]$ populates MPI task n. The array

$$\mathbf{V} = (|D_{M(1)}\rangle, |D_{M(2)}\rangle, \ldots, |D_{n_D}\rangle), \tag{16.2}$$

now keeps the information about where a determinant, if it is populated, is stored. To make this information accessible to all processes we distribute $\mathbf{V}$ to the MPI tasks.

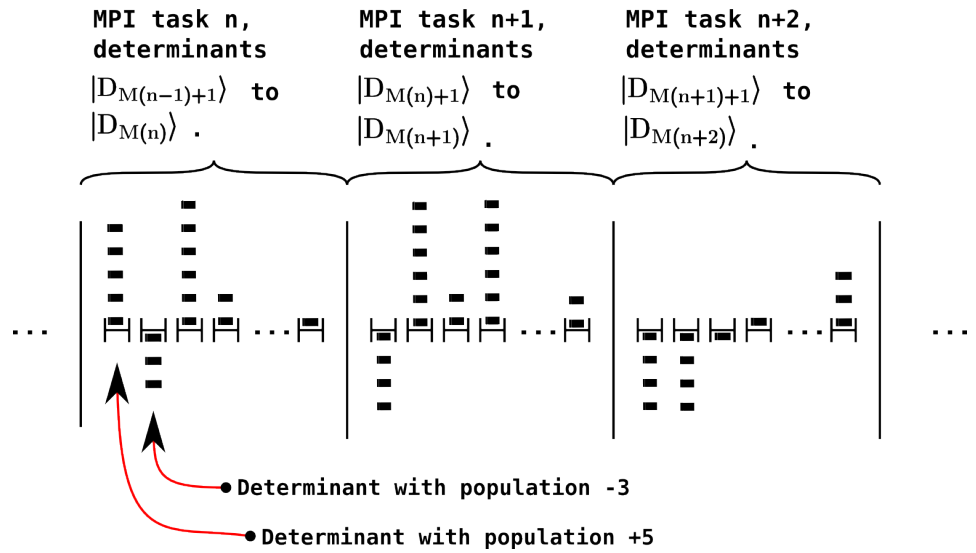


Figure 16.1: This figure shows schematically how the determinants are distributed on the MPI tasks. The determinants are sorted such that all determinants on MPI task i are larger than $|D_{M(i-1)}\rangle$ and smaller than or equal to $|D_{M(i)}\rangle$.

16.4 Load balancing

The process of distributing the walkers such that the workload is the same on all threads is referred to as load balancing. All threads have to wait for the last thread to finish at the end of an iteration, which means that the numerical efficiency will suffer if the computational load is not evenly distributed. The load balancing of the OpenMP threads and the MPI tasks follow the same principles, and in this section we will refer to both the MPI tasks and the threads as processes.

Each process i receives a number $N_W^{(i)}$ of walkers that populates a number $N_D^{(i)}$ of determinants. Ideally, the walkers should be distributed in such a way that all processes spend the same time processing the walkers. However, this is not straight forward since the CPU time needed to process one walker is dependent on many different factors. We have assumed that the computation time for a set of $N_W^{(i)}$ walkers living on $N_D^{(i)}$ determinants is

$$t_{CPU}(N_W^{(i)}, N_D^{(i)}) \propto k_W N_W^{(i)} + k_D N_D^{(i)} + const, \tag{16.3}$$

where k_W, k_D are implementation and system dependent parameters. We think this is a reasonable assumption since the number of calls to the most costly functions, which are the functions that calculate the kill/clone and spawn probabilities, are linearly dependent of $N_D^{(i)}$ and $N_W^{(i)}$. According to Eq. (16.3), the walkers should ideally be distributed to the processes $i =, 1, 2, 3, \ldots, N_{procs}$ in such a way that

$$N_W^{(a)} k_W + N_D^{(a)} k_D = \sum_{i=1}^{N_{procs}} (N_W^{(i)} k_W + N_D^{(i)} k_D)/N_{procs}, \; \forall a \in [1, N_{procs}]. \tag{16.4}$$

The constants k_W and k_D are found empirically. In practice we set $k_W = 1$ and optimize k_D. This parameter is system dependent, and the optimal value should ideally be found for each new simulation.

16.5 The program flow

In this section, we will look at the work flow and how the walkers are administrated by the parallelized program. Together with the last chapter where we described how the sequential algorithm is implemented, this section should give a more or less complete picture of how we have implemented the FCIQMC algorithm. We do not provide programming details or code examples here, but these can be found in the next chapter where we discuss the C++ code.

In figure 16.2, one iteration of the code is schematically illustrated in four steps A,B,C and D, and these steps are explained in detail below.

A: We start with a certain number of walkers on each MPI task. The walkers populate the determinants $|D_{i_1}\rangle, |D_{i_2}\rangle, \ldots$, which are sorted and distributed on the different MPI tasks. The MPI tasks are load balanced in accordance with Eq. 16.4.

   The first step is, on each MPI task, to distribute the walkers to the threads in such a way that they are load balanced.

B: Each thread performs the clone/kill and spawn steps on its list of walkers by using the sequential algorithm.

C: After the previous step, all MPI tasks now have two lists of walkers. The first is the old list of walkers minus the killed walkers and plus the cloned walkers. These walkers are situated on the correct MPI task since the cloned walkers populates the same determinants as their parents. The second list consists of the newly spawned walkers which populates different determinants than their parents. These walkers must be sent to the MPI task where their resident determinants are listed, sorted and merged with the old list of walkers.

   As we discussed in the last chapter, the newly spawned walkers are stored in the arrays `pb_new` and `pl_new`. And it is the elements of these arrays that must be redistributed. After the redistribution, the walkers are merged with the old populations and determinants arrays in the same way as we described in the last chapter [1] .

D: The number of walkers and determinants on the different MPI tasks may now have changed. The MPI tasks are load balanced by redistributing the determinants. Because of the way the determinants are sorted, only the first/last determinants are transmitted, and only to or from the last/next MPI task.

Except for the additional steps that must be taken to redistribute the newly spawned walkers and load balance the nodes and the threads, the parallel algorithm is more or less identical to the sequential algorithm.

[1] We discussed this in the last chapter under point B (the annihilation step).

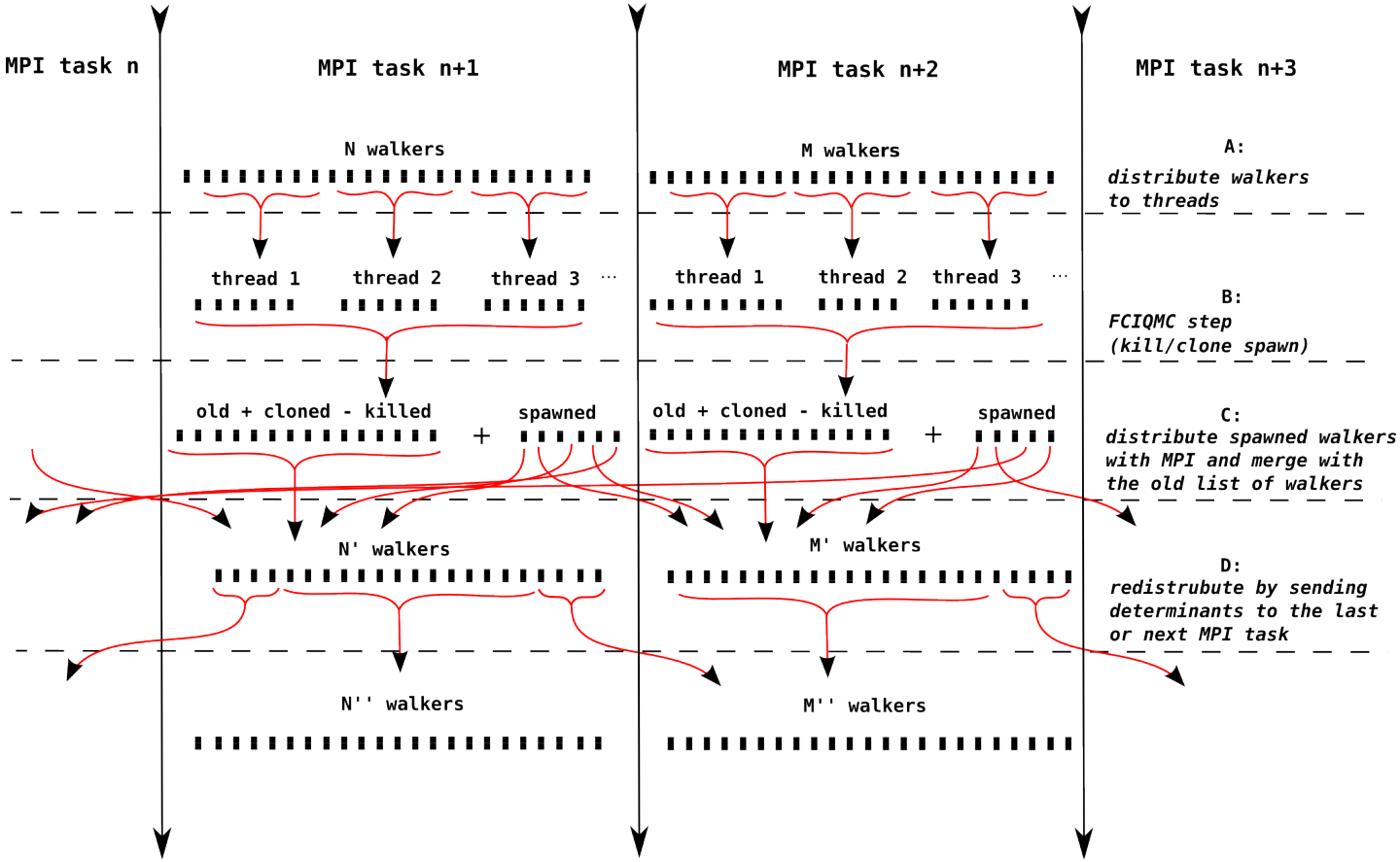


Figure 16.2: This figure is explained in the text in section 16.5

Chapter 17

Organization of the code and the classes

In this chapter we will look closer at the C++ code, and show how the code is structured. Note that with exception of the main function, there is no part of the code that is not part of a class. In this chapter each of the classes has its own section where we describe the functionality of the class, what classes it instantiates and so on.

17.1 Overview

The code is organized in the following classes

- `runSimulation`: This class contains the main loop and coordinates the other classes.
- `walkerContainerClass`: Keeps informations about the population of walkers.
- `sortWalkers`: Sorts the new walkers and merges them with the list of old walkers.
- `loadBalanceThreads`: Distributes the walkers to the threads.
- `walkerDistribution`: Distributes the walkers to the MPI tasks.
- `walkerPropagator`: Runs through a list of walkers and performes the killing cloning and spawning steps.
- `hamiltonianElements`: Calculates the hamiltonian elements $\langle D_i|\hat{H}|D_j\rangle$.
- `libGRIE`: Tabulates and stores the Coulomb matrix elements.
- `inputVars`: Reads and stores the preset run parameters.
- `initSimulation`: Initializes the basis and some run time parameters.

In addition, the library `OpenFCI` [15] are used to calculate the Coulomb integrals, and the library `CRandomMersenne` [6] to generate random numbers. The random number generator that we have used is an implementation of the Mersenne Twister algorithm [18] which is considered to be a fast algorithm that produces random numbers of high quality.

The simulator is wrapped in a python code that compiles the C++ code, starts simulations, organizes, analyzes and plots the output data etc.

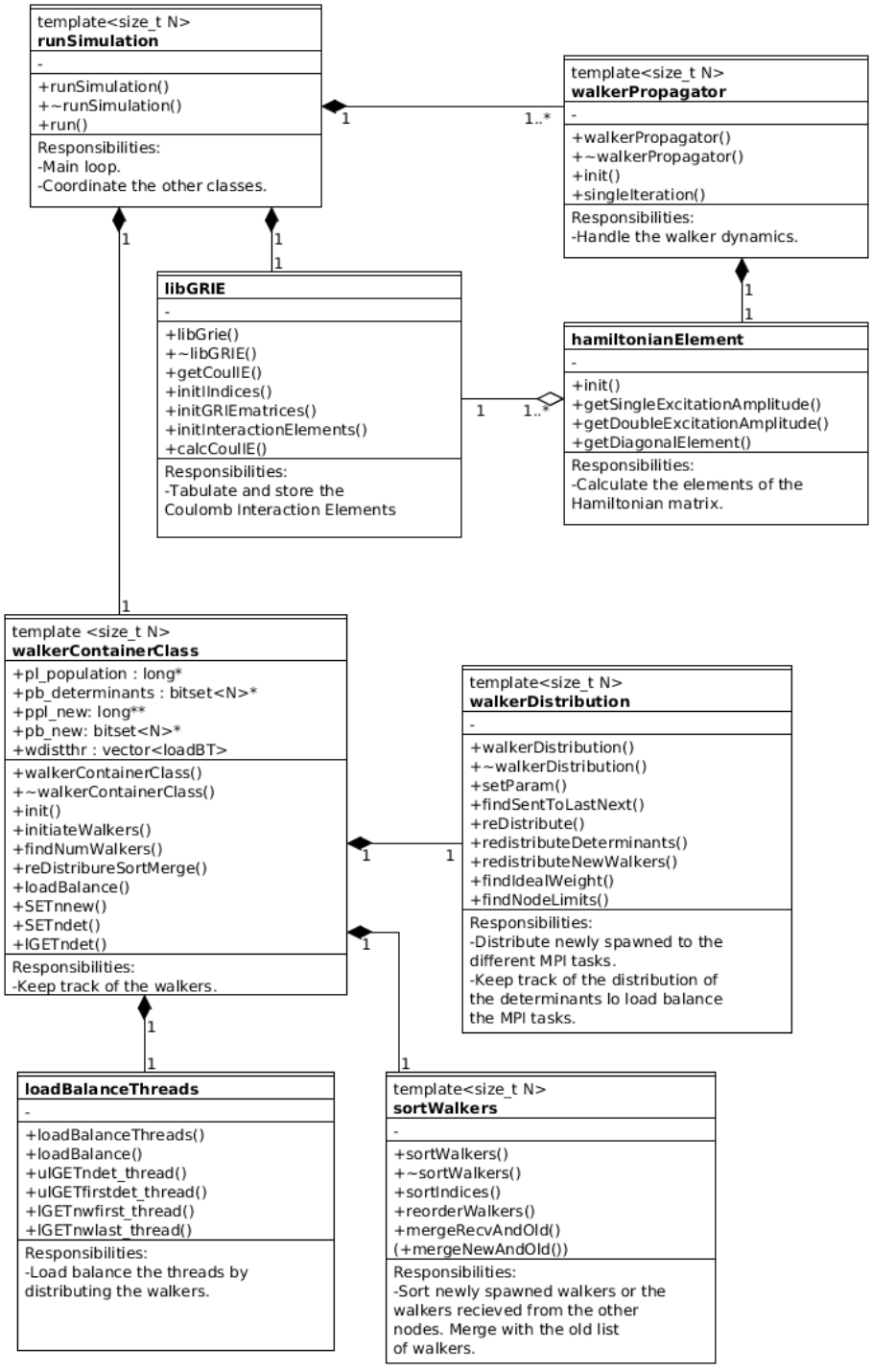


Figure 17.1: UML Class Diagram. Some classes are not included like `initSimulation`, `CRandomMersenne`, `OpenFCI` and `inputVars`.

17.2 The `initSimulation` class

This class initiates the `libGrie` class which contains the Coulomb matrix elements and the arrays `pi_m` and `pi_n` which contains the quantum numbers of the single particle orbitals. It also sets the seed [1] of the random number generator and some other run time parameters.

17.3 The `runSimulation` class

This class coordinates the other classes and contains the main loop of the program. The classes `walkerContainerClass` and `libGRIE` are instantiated before we enter the main loop, and are shared between all threads. We then enter the function `run` which can be summarized as follows:

(I): A number `i_numthreads` of threads are spawned and the program enters the parallel section.

- (A): The class `walkerPropagator` are instantiated within each thread. Each of these classes recieves a pointer to the `libGRIE` object to be able to access the shared Coulomb matrix elements.
- (B): Each thread allocates memory for intermediate storage of newly spawned walkers. This is done to prevent data races among the threads.
- (C): The program enters the main loop
  - (i): Each thread calls the `walkerPropagator::singleStep()` function to do a single FCIQMC iteration.
  - (ii): The newly spawned walkers are written to the shared arrays `ppd_new` and `pb_new` of the class `walkerContainerClass` .
  - (iii): The program enters a sequential section where only one of the threads does work
    - (a): The `walkerContainerClass` object redistributes the new walkers to the different MPI task. It then loadbalance the MPI tasks, and the threads of the individual tasks.
    - (b): Statistics are collected from all MPI tasks and written to an output file.

17.4 The `walkerContainerClass` class

This class holds four public arrays and one public vector that stores all information about the current population of walkers, the newly spawned walkers and the load balancing of the threads. It instantiates the classes `walkerDistribution` and `sortWalkers`, which sorts the walkers, redistributes the walkers on the different MPI tasks and merge the list of old and new walkers.

The state vector is represented by two arrays. The first is a `bitset` array which stores the populated determinants, and the second is a `long` array which stores their populations.

[1] The random number generator is initialized with an integer that is called a seed. The sequence of random numbers that are produced by the random number generator depends on the value of the seed.

```
bitset<N>* pb_determinants = new bitset<N>[i_ndmax];
long* pl_population = new long[i_nwmax];
```

The variable `l_ndet` tells us how many different determinants that have a non zero population. These determinants are always stored in the `l_ndet` first entries of `pb_determinants` and `pl_population`.

The newly spawned walkers are stored on the two arrays

```
bitset<N>* pb_new = new bitset<N>[i_maxnspawn];
long** ppl_new =  new long[i_maxnspawn];
```

Assume a spawning event where |n| walkers are spawned on the determinant $|D_j\rangle$ from the determinant $|D_i\rangle$, and that this is the `i_nnew`'th spawning event. The sign of `n` can be positive or negative depending of the sign of the spawned walkers. The information about the spawning event is stored in the following way:

`pb_new[i_nnew]` $\leftarrow |D_j\rangle$.

If $|D_i\rangle$ is an initiator:

`ppl_new[i_nnew][0]` $\leftarrow$ `n`

`ppl_new[i_nnew][1]` $\leftarrow 0$

If $|D_i\rangle$ is a non-initiator:

`ppl_new[i_nnew][0]` $\leftarrow 0$

`ppl_new[i_nnew][1]` $\leftarrow$ `n`

These arrays must be available when the new list of walkers are merged with the old list of walkers. Often, a walker is spawned on a determinant that is listed on a different MPI task, and there is no way of finding out if the spawning event was successful until the new walkers are redistributed.

The loadbalancing of the threads are calculated by the class `loadBalanceThreads` and the information about how the walkers are distributed is stored on the vector

```
vector<loadBalThr> wdistthr;
```

which has the same number of elements as the number of OpenMP threads. `wdistthr[i]` contains the information about which walkers that are processed by thread `i` and is an instance of the struct

```
struct loadBalThr
{
  unsigned long first;
  unsigned long ndet;
  long nwfirst_thrd;
  long nwlast_thrd;
};
```

where `first` is the index of the first determinant to be processed, and `ndet` is the number of determinants. The walkers on the first and the last determinant on a thread can be split between two threads. `nwfirst_thrd` and `nwlast_thrd` is the number of walkers on the first and the last determinant that should be processed by the current thread. `wdistthr` is reset every time the function `loadBalance` is called.

17.5 The `sortWalkers` class

This class sorts the spawned walkers and merges the lists of new and old walkers.

The functor `LessThan` finds the smallest of two determinants. This is a class where the operator `()` is overloaded with function that compares two determinants given their indices.

```
class LessThan
{
  private:
    bitset<N> b_tmp;
    unsigned int l_tmp;
    bitset<N>* pb_dets;
  public:
    LessThan (bitset<N>* pb_dets) { this->pb_dets = pb_dets; }
    bool operator()
      (const unsigned long lhs, const unsigned long rhs);
};
```

The comparison of the determinants can be efficiently performed by using bit operations. The next example shows how the `xor` operator (represented by `^` in C++) can be used to find the "smallest" of the two determinants `b_lhs` and `b_rhs`.

```
bool operator()(const unsigned long lhs, const unsigned long rhs);
{
  b_tmp = (pb_dets[rhs]^pb_dets[lhs]);
  l_tmp = b_tmp._Find_first();
  if (l_tmp==N)
    return false;
  else if (pb_dets[rhs].TEST(l_tmp))
    return false;
  else
    return true;
}
```

Here `_Find_first()` returns either the index of the first bit equal to 1, or `N` if all bits are 0, where `N` is the template parameter of the `bitset` objects which equals the number of spin orbitals in the basis. The function `lhs.test(k)` returns true if the `k`'th bit of `lhs`is set, and false otherwise. The class `LessThan` is instantiated with a pointer to the first first determinant in `pb_determinant` or `pb_new`. This allows us to test if determinant `i` is smaller than determinant `j` using the following notation

```
LessThan lessthan(&pb_determinant[0]);
bool b_lessthan = lessthan(i,j);
```

Instead of sorting the bitset array directly, we now sort an array `ul_index` with ordered indices in the range $[1,\ldots,N_D]$, where N_D is the number of determinants that are to be sorted. The order of he sorted indices tells us how to reorder the determinants afterwards. This is faster and more practical for many reasons. Most importantly, when `N` is large it is much more time consuming to sort the `bitset` array directly simply because more data needs to be moved around.

The function `orderIndices` initialise `ul_index` array and sort it by using the C++ Standard Template Library function `std::sort` [2].

```
for (unsigned long i=0; i<n; i++)
        ul_index[i] = i;
std::sort(&ul_index[0], &ul_index[n], LessThan(pb_new) );
```

Note that we pass the functor `LessThan` as an argument. This is a method to overload the operator < with the operator `()` of the `LessThan` class.

The function `reorderWalkers` reorder the newly spawned determinants after the array `ul_index` is sorted. Determinants that is represented more than once are merged and the their populations are summed. To do the reordering of the arrays efficiently, we copy the determinants array and the populations array to an intermediate array and swap pointers.

The function `mergeNewAndOld` merges the old list of determinants and a list of sorted walkers according rules of initiator-FCIQMC. We use intermediate arrays and pointer swapping in this function as well.

17.6 The `walkerDistribution` class

This class take care of the distribution of walkers and determinants on the different MPI tasks. Most of the MPI communication is handled here.

As discussed earlier, the determinants are sorted and distributed on the different MPI tasks. Every iteration, the newly spawned walkers are sent to the MPI task where their host determinant is listed. To know where to sent the walkers, each MPI task must be aware of which determinants that are listed on which MPI task. This information is, as we discussed in section 16.3, contained in the array

```
bitset<N>* pb_lastdet = new bitset<N>[i_numthreads];
```

which contains the greatest determinant on each of the MPI tasks.

After the spawning step, each new walker is sorted according to which determinant it populates. When the walkers are sorted, we only need find the first and the last determinant of the interval that we want to transmit to each of the MPI tasks. For this purpose we use the STL binary search function `std::lower_bound`. Similar to what we did in the last section, we overload `std::lower_bound` with a functor that defines the < operator.

This class also redistributes the determinants when the MPI tasks are load balanced.

17.7 The `loadBalanceThreads` class

This class is responsible for the load balancing of the threads. The function `loadBalance` finds the distribution of walkers according to Eq. (16.4).

[2] `std::sort` is an implementation of the so called introsort algorithm [19], which is a fast sorting algorithm with a worst case scaling of $\mathcal{O}(N\log(N))$.

17.8 The `walkerPropagator` class

This class performs a single FCIQMC step, and is instantiated on all threads. Each object of this class receives the list of walkers that is distributed to the threads by the `loadbalance` class. It calculates the spawning, killing and cloning step according to the FCIQMC algorithm.

The function `singleStep` runs through the list of determinants. For each determinant i the following steps are performed:

(i:) Call the class `hamiltonianElements` and calculate the diagonal matrix element $\langle D_i|\hat{H}|D_i\rangle$.

(ii:) Call the function `updatePojectedEnergy` to accumulate the projected energy if $|D_0\rangle$ is connected to $|D_i\rangle$.

(iii:) Run through the list of all walkers j on determinant i

(a) Assume that the preset probability of a single excitation is d_s.

If $\mathcal{U} < d_s$ try to do a single excitation
Else try to do a double excitation.

where $\mathcal{U}$ is a random uniform $\in (0,1)$. Then call `sampleProjector` which draws the new determinant $a_p^\dagger a_r|D_i\rangle$ or $a_p^\dagger a_q^\dagger a_r a_s|D_i\rangle$ and calculates the suggestion probability, and `trySingleExcitation` or `tryDoubleExcitation` which calculates the Coulomb matrix elements and the excitation amplitude, "rolls the dice" and spawns new walkers.

(b) Try to kill or clone a walker. Draw a random uniform $\mathcal{U} \in [0,1]$ and calculate $d_p = -\tau(\langle D_i|\hat{H}|D_i\rangle - S)$.

If $\langle D_i|\hat{H}|D_i\rangle < 0$ then remove the walker if $|\langle D_i|\hat{H}|D_i\rangle| > \mathcal{U}$.
Else if $\langle D_i|\hat{H}|D_i\rangle > 0$ then add a walker if $\langle D_i|\hat{H}|D_i\rangle > \mathcal{U}$.

Adding or removing a walker is done by adding to or subtracting from the i'th element of the populations array.

We will take a closer look at the function `sampleProjector`. This function draw a random determinant that is connected to D_i, and calculate the suggestion probability. For a single excitation we call

```
double walkerPropagator<N>::sampleProjector(
    const int iact_w,
    int &ia1,
    int &ic1)
```

Where `iact_w` is the index of the determinant, and `ia1`,`ic1` is changed to the indices of the annihilated and created orbitals. For a double excitation the function is overloaded with two more indices `ia2` and `ic2`.

First we pick a random orbital to be annihilated and store it as ai1

```
ia1 = o_random.IRandomX(1,inum_part);
//find the position of the i'th set bit
i = b_slater_active._Find_first();
for (j=1;j<ia1;j++)
  i = b_slater_active._Find_next(i);
ia1 = i;
```

where `_Find_first()` and `_Find_next(i)` are highly optimized functions of the bitset class that finds the first bit or the first bit after the `i`'th bit that is set. Then we construct a bitset `b_slatertemp` which contain all orbitals that preserve the total magnetic quantum number and the spin

```
//find the magnetic quantum number of the new state
ima = pi_m[ia1/2];
//the spin must be the same as that of the annihil. orb
if (ia1%2) //ic1 is spin down (1)
  b_slatertemp = pbamc_dwn[ima+ir]&~b_slater_active;
else //ic1 is spin up (0)
  b_slatertemp = pbamc_up[ima+ir]&~b_slater_active;
```

The bitsets `pbamc_up[ima+ir]` / `pbamc_dwn[ima+ir]` contains all orbitals with a magnetic quantum number `ima` and spin up / down. These bitsets are initialized when the class is instantiated. `b_slater_active` is the active determinant $|D_i\rangle$ and `pi_m` is the array containing the magnetic quantum number of the orbitals.

We count the number of orbitals in this set. This number must be known to calculate the suggestion probability.

```
idim_S5 = b_slatertemp.count();
if (idim_S5==0) {return -1;}
```

Then we pick a random orbital from this set.

```
//pick random from S5
ic1 = o_random.IRandomX(1,idim_S5);
//find the position of the i'th set bit
i = b_slatertemp._Find_first();
for (j=1;j<ic1;j++)
  i = b_slatertemp._Find_next(i);
ic1 = i;
return static_cast<double>(inum_part*idim_S5);
```

The suggestion probability for the single excitations is the return value divided by the predeterminad probability of a single excitation p_s.

The same strategy is used to find the doubly excited determinants and their suggestion probability.

17.9 The `hamiltonianElements` class

This class calculates the elements of the Hamiltonian matrix, and is instantiated by the `walkerPropagator` class. It communicates with the `libGRIE` class that holds the Coulomb matrix elements.

17.10 The `libGRIE` class

This class tabulates the Coulomb matrix elements using the indexing scheme that is described in section 11. This class is only instantiated once on each MPI task.

The reason why the `libGRIE` object is shared among the threads is that the amount of memory needed to store the Coulomb matrix elements can become several GB for large R. If each thread where to store its own `libGRIE` object the memory footprint of the program would become very large.

The class variables are not changed after the initialization. This means that several threads can access the class methods simultaniously without risking the occurrence of race conditions. Thus, sharing an object of the class between the threads will not slow down the code.

17.11 The `inputVars` class

This class reads runtime parameters from a text file and stores them as class variables. Instead of passing the runtime parameters around in the program, we pass an object of this class.

Chapter 18

Benchmarking the Code

In this chapter we have tested the efficiency of the code, both sequentially and with OpenMP and MPI, and which effect different run time parameters have on the execution time.

18.1 Hotspots by CPU usage

The hotspots are the most CPU intensive parts of the code. These are the parts of the code that we would gain the most from optimizing. To find the hotspots we have performed a tests on a single workstation. MPI was not used in this test, but the scaling with MPI is reviewed separately later in this chapter.

(i:) `class walkerPropagator` **38% of total CPU time** of which

68% was spent in the function `sampleProjector`. which calculates the generational probabilities p_{gen} for the single and the double excitations.

(ii:) `class CRandomMersenne` **30% of total CPU time**. This is the random number generator generating random uniforms.

(iii:) `class libGRIE` **20% of total CPU time** of which

close to **100%** was spent in the function `getCoulIE`. This function finds the Coulomb matrix elements given the indices (p, q, r, s) of the annihilated and created orbitals.

(iv:) `class hamiltonianElements` **7% of total CPU time** of which

94% was spent in the function `getSingleExcitationAmplitude` which calculates the amplitudes $\langle D_i|\hat{H}\left\{a_p^\dagger a_r\right\}|D_i\rangle$.

6% was spent in the function `getDiagonalElement` which calculates $\langle D_i|\hat{H}|D_i\rangle$

(v:) The remaining CPU time was mainly spent waiting because the threads had different workloads. The sortWalkers class and the `loadBalanceThreads` class spent less than **1%** of the total run time.

As we see, the calculation of the generational probabilities p_{gen}, the generation of random uniforms and accessing the Coulomb matrix elements is the most time consuming part of the code.

The `sampleProjector` method is believed to be reasonably fast, and it is difficult for us too see how this function could be further optimized. It is however clear that the overall performance of the code would improve much from such optimizations, and alternative sampling methods might be an interesting subject for future studies. The random number generator is a highly optimized implementation of the well known Mersenne-Twister algorithm, which we believe to be a good choice for our implementation. Although faster algorithms exists, this one does not compromise on the quality of the random numbers. We have also invested much effort in optimizing the method `getCoulIE` that returns the Coulomb matrix elements, and we do not know any other method that is faster than ours for this purpose.

18.2 Scaling with openMP and MPI

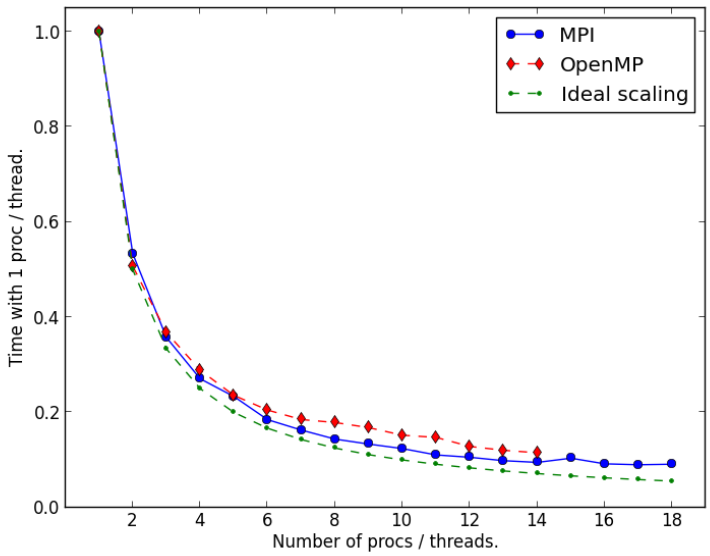
Because of the serial sections of the code, the computational efficiency will only increase up to a certain number of nodes and cores. The total run time can not be less that the execution time of the serial part of the code, and the performance gain of adding an extra node falls rapidly close to this limit. This relationship is described by Amdahls law which states that if the serial part of the code spends s of the total run time, the speedup with n processes is $n/(1 + s[n - 1])$ which converges to $1/s$ for large n.

The walkers on a single determinant can be split between threads but not between MPI tasks. This means that the best possible speedup for a system with N_W walkers in total and n_m walkers on the most populated determinant is of order N_W/n_m. If we increase the number of MPI tasks beyond roof$[N_W/n_m]$, we will not see any speedup. This means that a simulation will always be limited by the most populated MPI task with n_m walkers, and that simulations with many walkers on few determinants should be run with as many threads as possible per MPI task.

The speedup as a function of the number of OpenMP threads and as a function of MPI tasks is shown in Fig. 18.1. This plot shows that the MPI implementation is efficient and indicates that the communication overhead is small. But even if the system which we have simulated here scales well with MPI, this result does not necessarily apply to all systems. There are many factors which affect the amount of communication between the nodes and the communication overhead. For example, we would expect less communication with systems that are weakly correlated since the spawning probabilities on average are smaller (Hamiltonian matrix closer to diagonal form). Likewise, if the fluctuations in the distribution is small, we would expect less communication since the determinants would have to be redistributed less often. These characteristics are system dependent and difficult to predict.

Among the tunable parameters which affect the amount of communication is the time step τ, the determinant load parameter k_D and the redistribution threshold. The time step affects the walker dynamics, and there exists an optimal parameter from a sampling perspective. But when we run the code on many nodes, the time step is proportional with the spawning rate, and should therefore be kept low to minimize the communication overhead. The optimal values for τ, k_D and the redistribution threshold are system dependent, and

should ideally be optimized for each new system.



(a)

Figure 18.1: (MPI) Wall time of simulations with a different number processes or MPI tasks. All MPI tasks run on different nodes to maximize the communication overhead. (OpenMP) CPU time of simulations with a different number of cores. All runs performed on a single node. (Ideal scaling) The best possible parallel scaling is n^{-1}, where n is the number of MPI tasks or threads.

18.3 Scaling with the number of walkers

As we have argued earlier (see section 16.4), we would expect the run time of the part of the code that handles the kill/clone and spawn steps to scale linearly with the number of walkers and the number of determinants. The scaling of the part that handles sorting, annihilation and redistribution of walkers is more difficult to predict, but is expected to have a worst case scaling of $\mathcal{O}(N\log(N))$ [1] with both the number of newly spawned determinants and the total number of determinants. However, the dependency of the number of determinants on the number of walkers is system dependent and not generally known. Furthermore, this part of the code is not parallelized with OpenMP, and could therefore have a large negative impact on the overall run time.

Our experience from simulations is that given that the number of nodes are kept constant, the computation time scales close to linearly with the number of walkers. This is illustrated in Fig 18.2 where we have plotted the run time of simulations of different systems as a function of the number of walkers.

[1] The sort algorithm has a worst case scaling of $\mathcal{O}(N\log(N))$ where N is the number of determinants that are to be sorted. We use C++ Standard Library function `std::sort` which is an implementation of the introsort algorithm [19].

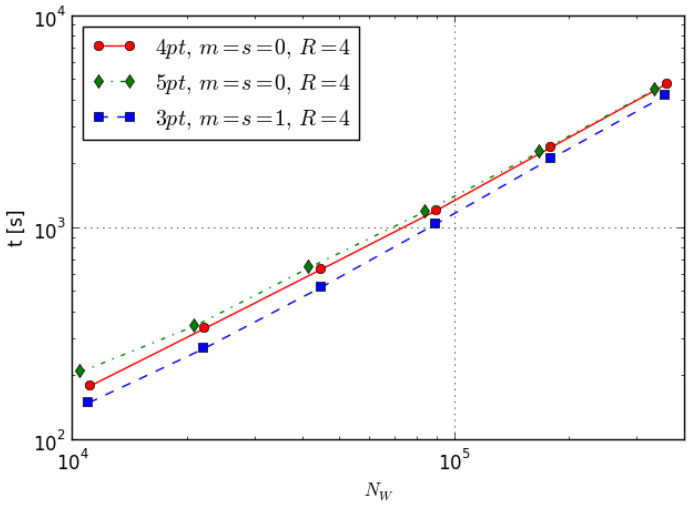


Figure 18.2: Scaling of the algorithm when the number of walkers are increased for systems with a different number of particles N_P, number of shells $R+1$, spin s and magnetic quantum number m. All systems have an oscillator frequency $\omega = 1$ and are run with the same number of iterations. As we see from this figure, the run time t seems to have a linear dependency of the number of walkers.

18.4 The determinant load parameter and the redistribution threshold

The CPU time of simulations with different values for the determinant load parameter k_D is illustrated in Fig. 18.3. The optimal value for this parameter differs from system to system, thus it should ideally be tuned for each new system. But as this is a very time consuming task we have used a fixed value $k_D = 1.4$ in most simulations.

When it comes to the redistribution threshold, we have found that using a value $\sim 10^{-2}$ is efficient for the systems that we have simulated. We have used a redistribution threshold of 0.03 for all simulations.

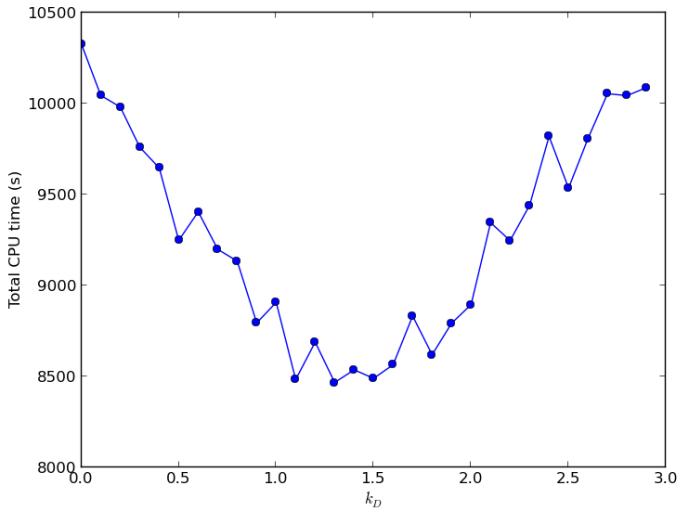


Figure 18.3: Different simulations where all runtime parameters are the same except the load weight parameter k_D. This parameter determines how the nodes are load balanced as defined in Eq. (16.4). As we see, the CPU time is at its minima at $k_D \approx 1.4$. This number is however system dependent, and ideally each new system should be minimized with respect to k_D to optimize the numerical speed.

Chapter 19

Modifying the code to simulate other systems

Our code is tailored to simulate two dimensional quantum dots, but it is straight forward to modify it to handle other systems like atoms, molecules or nuclei. In this short chapter we are looking closer on how this can be done.

Assume that we want to apply the FCIQMC code to an arbitrary system. To do so, we need to know the Hamiltonian matrix in the basis of the determinants $\{|D_i\rangle\}$ which are constructed from the single particle wave functions $\{|\varphi_i\rangle\}$. In addition we need to keep track of the quantum numbers $n, m, s, \ldots$ of the single particle wave functions. To be able to sample the spawning step efficiently, we must also know which quantities that are conserved in an interaction $\langle D_i|\hat{H}|D_j\rangle$. These properties differs from one quantum mechanical system to the next, and the code has to be modified if the quantum numbers or the conservation rules are different from the two dimensional parabolic quantum dots which we have simulated.

Only a few functions and classes will have to be modified or rewritten to simulate different many body systems. As we already have discussed, we store the quantum numbers on arrays `pi_n` and `pi_m`. If the new system have additional quantum numbers, more such arrays would have to be introduced, and the classes `walkerPropagator` and `hamiltonianElement` would have to undergo some small changes. In addition we would have to change the following classes and functions:

1. The function `walkerPropagator::sampleProjector`.

   This function samples the suggestion probabilities (see: chapter 10) and has to be rewritten if the new system has different symmetries. Generally, only determinants $|D_i\rangle$, $|D_j\rangle$ where certain quantities are conserved are connected, and only connected determinants should be sampled when we try to spawn new walkers.

2. The class `hamiltonianElement`.

   This class has to be rewritten to calculate the Hamiltonian matrix elements of the new system.

3. The class `libGRIE`.

   This class must be replaced with a class that stores or calculates the Coulomb matrix elements (or mode generally the interaction matrix elements) of the new system.

4. The class `initSimulation`.

   This class initiates the arrays `pi_m` and `pi_n` which contains the quantum numbers and has to be rewritten.

All the classes that administrates the walkers (`walkerContainerClass`, `loadBalanceThreads`, `sortWalkers` and `walkerDistribution`), and the `runSimulation` class would be left unchanged.

Part VI

Results

In this part we discuss the results of a series of simulations with the FCIQMC algorithm and the optimized algorithm i-FCIQMC. We test the algorithm, discuss its advantages and disadvantages and compare it to other numerical methods, and we have also made a few predictions for different open shell systems.

The first chapter is devoted to testing and validating the program, and we show that our code reproduces known results with a high accuracy. In the next two chapters, we look at how the code performs with different systems with a varying degree of correlation and a different number of particles. In the fourth and fifth chapter we look closer at different methods to optimize the algorithm.More specifically, we study the convergence of the results of simulations with i-FCIQMC, with both the plain harmonic oscillator basis and with a Hartree-Fock basis. In the sixth chapter we calculate the extrapolated energies of different open shell systems and compare the results to Diffusion Monte Carlo (DMC) resuls. We also discuss the validity of the extrapolation formula and discrepancies between our results and the DMC energies.

At last we summarize and discuss our results together with what we believe are interesting furure prospects.

Chapter 20

Validating the code

20.1 Introduction

To validate the code we have found some cases where we know the "correct" outcome of a simulation. In some cases there exist closed form expressions which provides us with the exact result and in other cases we have compared with values from other numerical methods which ought to give the same outcome. We know that the FCIQMC energy should converge to the Full Configuration Interaction (FCI) energy within a given Hilbert space. An obvious way to validate the code is therefore to compare our results with FCI energies. We have compared our energies to the results of Refs. Kvaal 2006 [15], Rontani *et. al.* (2006) [29] and Olsen (2012) [20] which have performed FCI simulations with the same Hilbert spaces as we study. In the case of the two particle quantum dot, the closed form expression of the energy for certain values of the oscillator frequency ω is known [37]. For example, the two particle $\omega = 1$ quantum dot has a ground state energy of exactly 3 Hartree (H). Because of the relatively small dimensionality of the Hilbert spaces of this system, we can do simulations with a high number of shells and a high accuracy.

The situation is different for the initiator adaption of the algorithm, i-FCIQMC. We do not even know the closed form expression of the underlying projection operator, and consequently it is difficult to find any good test cases. The only method we have found to test our implementation is to check that the energies actually converge to the FCIQMC value in the limit of a high number of walkers and a low initiator limit.

20.2 FCIQMC

20.2.1 A simple system, two electrons in two shells

The first test to validate the code was to compare with the FCI ground state energy of the two particle system in two shells with an oscillator frequency $\omega = 1$. In two shells, the single particle basis consists of only six spin orbitals

$$\{\varphi_1, \varphi_2, \varphi_3, \ldots, \varphi_6\}, \tag{20.1}$$

and the Hilbert space is spanned by the three determinants which have a total spin and magnetic quantum number equal to zero

$$\mathscr{H} = \{|\varphi_1, \varphi_2\rangle, |\varphi_3, \varphi_6\rangle, |\varphi_4, \varphi_5\rangle\} = \{|1\rangle, |2\rangle, |3\rangle\}. \tag{20.2}$$

The Hamiltonian matrix is a 3×3 matrix with the elements

$$H_{i,j} = \langle i|\hat{H}|j\rangle, \quad i,j \in [1,3], \tag{20.3}$$

The numerical value of the matrix elements is generated with `OpenFCI`, and we find the ground state energy to be $3.15233\ldots H$, a result which is reproduced with our code.

20.2.2 Comparing FCIQMC energies and FCI energies

The next test was to compare FCIQMC energies to the results of Refs. [15, 20, 29] which have used FCI to calculate ground state energies of parabolic quantum dots. In Tab. 20.1, the results of a small series FCIQMC runs are listed and compared with FCI results. The FCI results was in most cases reproduced with a small error in the range of 0.6 mH or less. The exception is the simulations of the 5 particle quantum dots with $R = 5$, where the FCI energies are significantly lower. Note that the results still agree to within 1 mH, and that the FCI results for the same system are reproduced with $R = 7$. When the differences are so small, it is difficult to discuss them further without an error estimate for the FCI results.

All Refs. have used the Lanczos algorithm, which is an iterative procedure where a tridiagonal matrix which is similar to the Hamiltonian matrix is constructed. The eigenvalues of this matrix are expected to converge to the energy spectrum of the Hamiltonian. When the difference between the estimated ground state energy from one iteration to the next is smaller than some preset value δ_k, the simulation is assumed to have converged. But δ_k does not set an upper bound for the error. Olsen(2012) [20] demonstrated that for some systems the energy seems to converge to one value before it suddenly drops and converges to the "real" ground state value. As an example, for the four particle quantum dot with $m = s = 0$ and $\omega = 1/\sqrt{6}$, Refs. [15, 29] found a ground state energy of approximately $0.6581H$ while the "true" value is $0.0016H$ lower at $0.6566H$. In this case, a δ_k smaller than $10^{-7}H$ was needed for the simulation to converge to the lower value (which was the same groundstate energy as the FCIQMC algorithm found for this system). The convergence criteria for the energy δ_k is only provided by Olsen [20], which used a value of $10^{-6}H/\omega^2$ in most simulations. Although it is possible to estimate the Lanczos error [8], only Olsen has discussed this subject, and none of the references have included the errors of their results.

N	ω	m	$2s$	$R = 5$ $\langle S\rangle$	$R = 5$ $\langle E_p\rangle$	$R = 5$ FCI	$R = 7$ $\langle S\rangle$	$R = 7$ $\langle E_p\rangle$	$R = 7$ FCI
2	1	0	0	3.0140(6)	3.0136(1)	3.01363 (†)	3.0091(4)	3.0091(1)	3.00924 (†)
2	1/4	0	0	0.9334(2)	0.9335(1)	0.933399 (†)	0.9323(2)	0.9323(1)	0.932331 (†)
3	1/16	1	1	0.69029(4)	0.69025(8)	0.690300 (†)	0.69008(5)	0.69021(9)	0.690159 (†)
3	1/16	0	3	0.69087(3)	0.69089(2)	0.690893 (†)	0.69080(3)	0.69079(2)	0.690789 (†)
4	1/36	0	0	0.65672(5)	0.6562(4)	0.656630 (⋆)	0.65546(5)	0.6557(6)	0.655454 (⋆)
4	1/36	0	2	0.65674(4)	0.6563(3)	0.65669 (‡)	0.65551(5)	0.6550(5)	0.65550 (‡)
5	1/4	0	5	5.2887(1)	5.2886(1)	5.28773 (†)	5.2825(1)	5.2825(1)	5.28248 (†)
5	1/16	0	5	1.84080(4)	1.8409(1)	1.83971 (†)	1.83132(4)	1.8314(2)	1.83141 (†)

Table 20.1: Here and m, s and ω are the magnetic quantum number, the spin and the oscillator frequency, and $R+1$ the number of shells. $\langle S\rangle$ and $\langle E_p\rangle$ are the statistical and generational estimator, and energies are in units of Hartree. The FCI energies are provided by [15] (†), [29] (‡) and [20] ($\ast$). Note that the number of digits of the FCI energies does not reflect the accuracy of the results.

20.2.3 Testing FCIQMC energies against analytical results

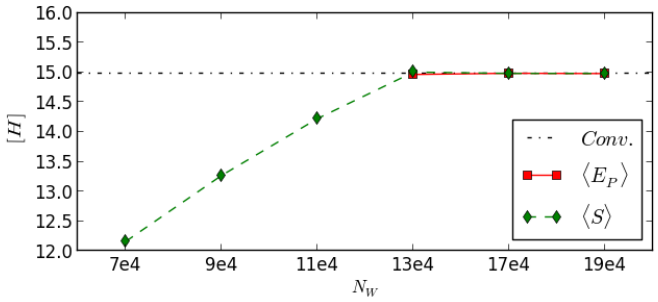
In the article by Taut [37], it is shown that the ground state energy of the two particle quantum dot with $\omega = 1$ is exactly $3H$. We have performed a series of simulations with $R = 5, 10, 15, 20, 25$ and 30 to check whether the energy converges towards the exact result. The energy is improved every time we increase the number of shells, and with $R = 30$ the energy is approximately $2mH$ too high. Also note that, as will be discussed later, the extrapolated energy (using Eq. (5.11)) yields an energy $3.0000(1)H$. The results are listed in Tab. 20.2 and illustrated in Fig. 25.2.

R	$\langle S \rangle$	$\langle E_P \rangle$
5	3.0133(6)	3.00136(1)
10	3.0059(6)	3.0063(2)
15	3.0037(7)	3.0036(2)
20	3.0028(8)	3.0028(2)
25	3.0019(4)	3.0021(1)
30	3.0016(3)	3.0018(1)

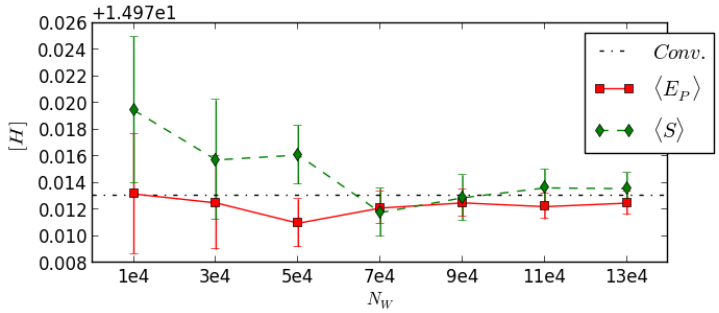
Table 20.2: Simulations with two particle quantum dots with spin and magnetic quantum number $m = s = 0$ and different number of shells $R + 1$ in the basis. All listed energies have units of Hartree. The exact result in the limit $R \to \infty$ is 3 Hartrees.

20.3 i-FCIQMC

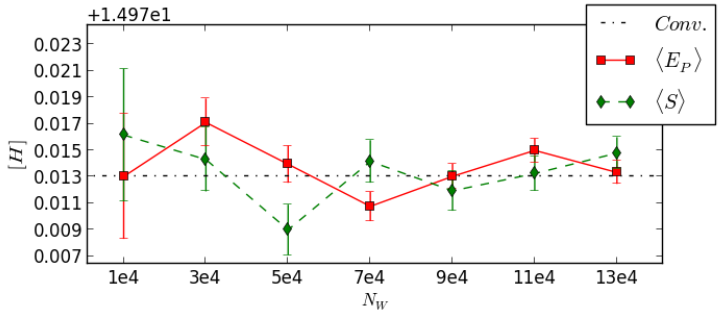
To ensure that the i-FCIQMC energy has converged, a series of simulations must be run with an increasing number of walkers. When both $\langle S \rangle$ and $\langle E_P \rangle$ are converged to the same energy, we can assume that the energy has converged to the FCIQMC value. Based on the results of Ref. [4], we would also expect this to happen with a number of walkers that is smaller that N_C. We have observed this behaviour for all i-FCIQMC simuations that we have performed. As an example we will look at the 5 particle quantum dot with $\omega = 1$ and $2s = m = 1$ in 9 shells. We have used the FCIQMC result with 5×10^5 particles as a benchmark for the energy. The FCI space consists of 6.5×10^5 determinants and the critical number of walkers is approximately 1.3×10^5. As we see from Fig. 20.1, FCIQMC must be run with at least the critical number of walkers to converge while the i-FCIQMC energies converges at a much lower number of walkers. Already at 10^4 walkers the i-FCIQMC energy is within a few mH from the FCIQMC result, and as the number of walkers is increased the errorbars gets smaller.



(a) $N_I = 1$



(b) $N_I = 2$



(c) $N_I = 3$

Figure 20.1: Simulations for systems with 5 particles, 9 shells, magnetic quantum number and spin $m = s = 1$ and a Hilbert space with a dimensionality $\dim(\mathscr{H}) = 6.5 \times 10^5$. The critical number of walkers is $N_C \approx 1.3 \times 10^5$. Here $\langle S \rangle$ is the generational estimator and $\langle E_P \rangle$ is the projected estimator while *conv* is the "exact" FCI value at $14.9830(5)H$. Here we have used results from an FCIQMC simulation with 5×10^5 walkers as the exact energy. All simulations are run with 2×10^6 iterations.

Chapter 21

The statistical estimators

In this section we will discuss the behaviour of the generational estimator $\langle S \rangle$ and the projected estimator $\langle E_P \rangle$. We have included results from simulations of different 2 particle systems with 8 shells and different oscillator frequencies ω. Although these systems have a very small Hilbert space with only 104 determinants in the basis, the statistical estimators show the same general behaviour as we have seen for other systems. We have run 4 simulations with 10^4 walkers which is well above the critical number of walkers that is necessary for the energies to converge. The value for ω is set to different values, 10.0, 1.0, 0.1 and 0.01, otherwise all parameters are the same. The results are listed in Tab. 21.1 and as we see, both the projected estimator $\langle E_p \rangle$ and the generational estimator $\langle S \rangle$ converges to the FCI value.

First, notice that the relative error (measured as a fraction of the total energy) of both estimators gets larger for smaller ω. This is a general trend in our simulations, and might be explained as a consequence of the degree of correlation in the systems. For high ω the system is weakly correlated, and excitations will be less probable. In this case, most of the walkers will presumably be concentrated in a smaller part of the Hilbert space. In a strongly correlated system, the walkers will presumably be more spread out in the Hilbert space, and one can say (with a little sloppiness) that we are sampling a larger number of determinants with the same number of walkers.

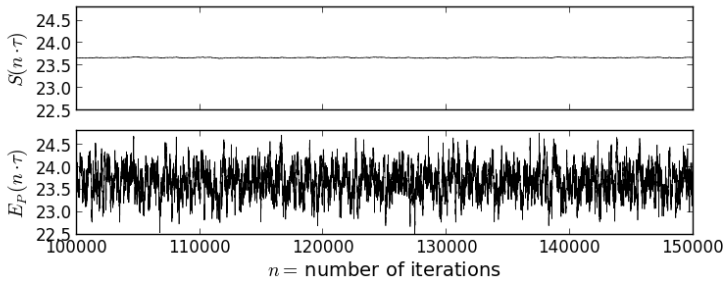
From Tab. 21.1 we also see that the projected estimator $\langle E_p \rangle$ gives the best energy estimate for large ω, while the generational estimator $\langle S \rangle$ does for small ω. This is illustrated in Fig. 21.1 where the fluctuations E_p gradually become larger than the fluctuations of S as we decrease ω. There may be several reasons why the statistical estimators have a different dependence of the oscillator frequency ω. First, the projected estimator does only depend on walkers that populates determinants that are connected to the reference determinant $|D_0\rangle$ while the generational estimator depends on the total population. This means that for most systems, the projected estimator are calculated on basis of a subset of the population. (This does not apply to the two particle system where all determinants are connected to $|D_0\rangle$). Second, the general estimator is sensitive to the total number of walkers while the projected estimator is sensitive to the shape of the distribution. And third, the projected estimator is extra sensitive to the number of walkers n_0 on the reference determinant $|D_0\rangle$ since it appears in the denominator in the expression for the projected energy. If n_0 gets to small, we would expect the projected estimator to have a large error. It should be commented that systems where the $|D_0\rangle$ component of the ground state is small, may require a very large

population to acquire a sufficiently large n_0.

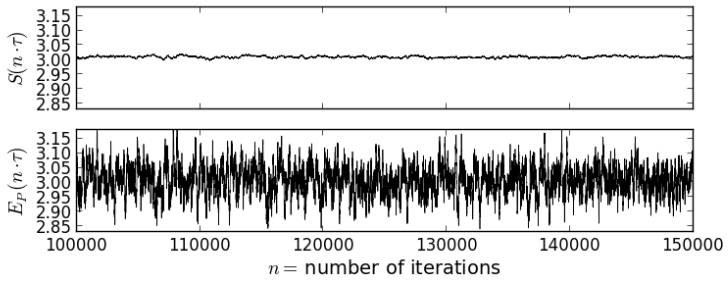
ω	$\langle n_0 \rangle / \langle N_W \rangle$	$\varepsilon_{\langle S \rangle} / \varepsilon_{\langle E_p \rangle}$	FCI [20]	$\langle S \rangle$	$\langle E_p \rangle$
10.0	0.69	20	–	23.6772(40)	23.6771(2)
1.0	0.47	6	3.0092637...	3.0094(6)	3.0092(1)
0.1	0.37	2.5	–	0.4413(10)	0.4412(5)
0.01	0.15	0.5	0.073835...	0.07387(3)	0.07380(6)

Table 21.1: The ratio of the population on the reference determinant to the total population $\langle n_0 \rangle / \langle N_W \rangle$ and ratio of the error of the generational estimator to the statistical estimator $\varepsilon_{\langle S \rangle} / \varepsilon_{\langle E_p \rangle}$ increase as we decrease the oscillator frequency ω. Thus the generational estimator will give a better estimate of the energy than the projected estimator when ω is small. FCI energies were only available for ω = 1 and 0.1. All energies are in units of Hartree.

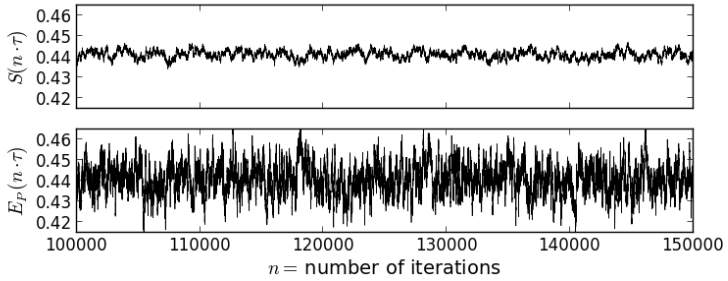
Having two different estimates for the energy is valuable as it heightens our confidence in the results. In fact, in the case that the estimators does not agree we can assume that the simulations are not converged. The estimators are only, as we can see from their definitions, expected to give the same result in the case that the distribution of walkers resembles the ground state of the given Hilbert space.



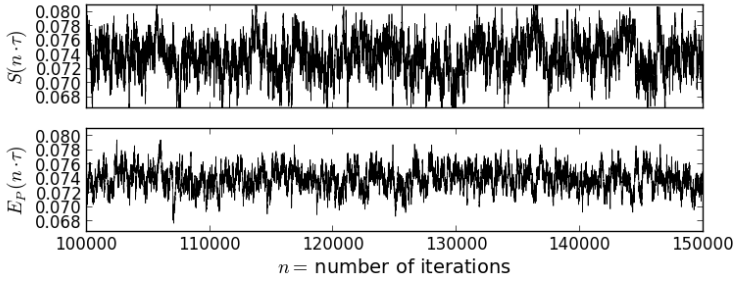
(a) ω = 10



(b) ω = 1



(c) ω = 0.1



(d) ω = 0.01

Figure 21.1: The instantaneous value of the projected energy $E_p(\tau)$ and the shift $S(\tau)$ during 5×10^4 iterations for simulations with different oscillator frequencies ω. All energies are in units of Hartree. The fluctuations in $E_p(\tau)$ becomes larger relative to the fluctuations in $S(\tau)$ as we decrease ω. All simulations are with two particle quantum dots with spin and magnetic quantum number $s = m = 0$ and with eight shells in the basis.

Chapter 22

The scaling of the algorithm

In this chapter we will discuss how the algorithm scales with the basis size, the oscillator frequency (which is linked to the degree of correlation), the number of walkers and also the number of particles.

22.1 The critical number of walkers

The number of walkers N_C that is necessary for the energy to converge is a system and basis dependent parameter which we will refer to as the critical number of walkers. The strength of the FCIQMC algorithm is that N_C often is much smaller than the number of determinants in the basis $\dim(\mathscr{H})$. The algorithm is only efficient for systems where the $N_C/\dim(\mathscr{H})$ ratio is small, and it is therefore of interest to study the dependency of this ratio on different parameters.

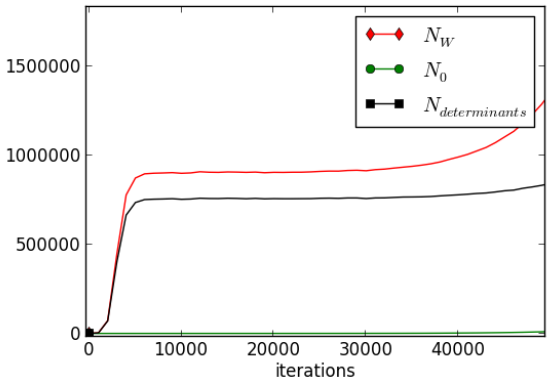
As demonstrated in Fig. 22.1 and described in section 8.5, the critical number of walkers can be found by performing a simulation with a fixed shift S close to the ground state value. In the beginning of the simulation, the spawning events will be much more frequent than the annihilation events and the population will grow rapidly. But when the number of walkers is large enough, the annihilation events will become more probable, and the population will eventually reach a plateau.

If the number of walkers is smaller than N_C, the long time average of the distribution of walkers will generally not converge to the ground state distribution. The annihilation between positive and negative walkers will be inefficient, and as we discussed in section 8.5, annihilation is neccessary for convergence to the ground state.

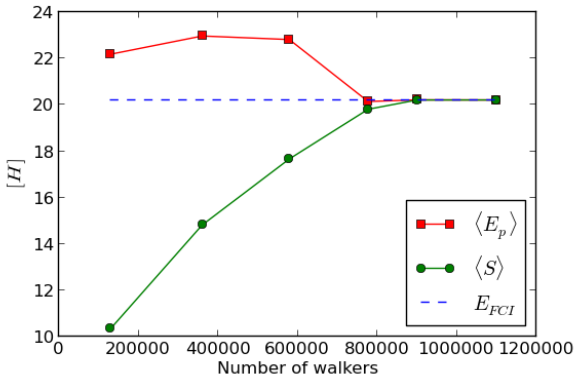
As illustrated in Fig. 22.2, the critical number of walkers is both dependent on the dimensionality of the basis and the oscillator frequency ω. When the basis is truncated on a higher shell number R, or when the oscillator frequency ω is decreased, the critical number of walkers N_C increases. This is not very surprising since the walkers presumably are spread out on more determinants in both cases, either because more determinants are available in the basis or because the correlations between the electrons are stronger.

22.2 Scaling of the FCIQMC algorithm

We have found that the critical number of walkers N_C has a linear relationship to the dimensionality of the Hilbert space $\dim(\mathscr{H})$. This is illustrated in Fig. 22.3(a). This figure also



(a) FCIQMC simulation with constant $S \approx E_0$

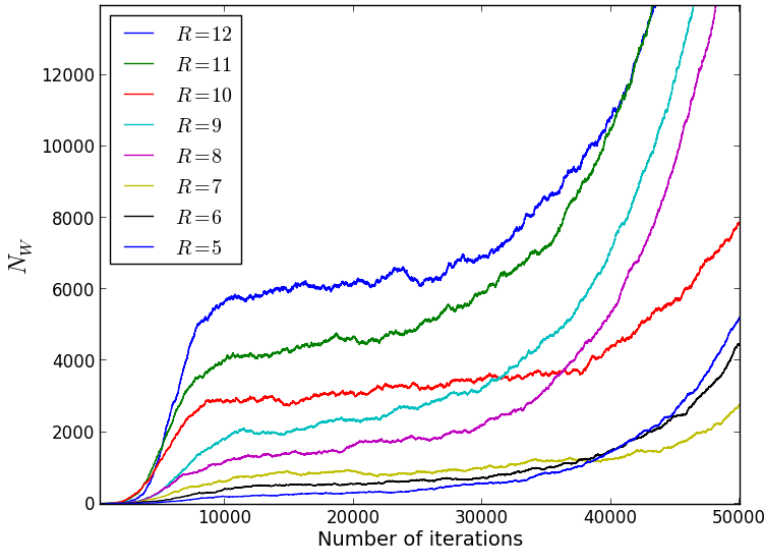


(b) FCIQMC energies.

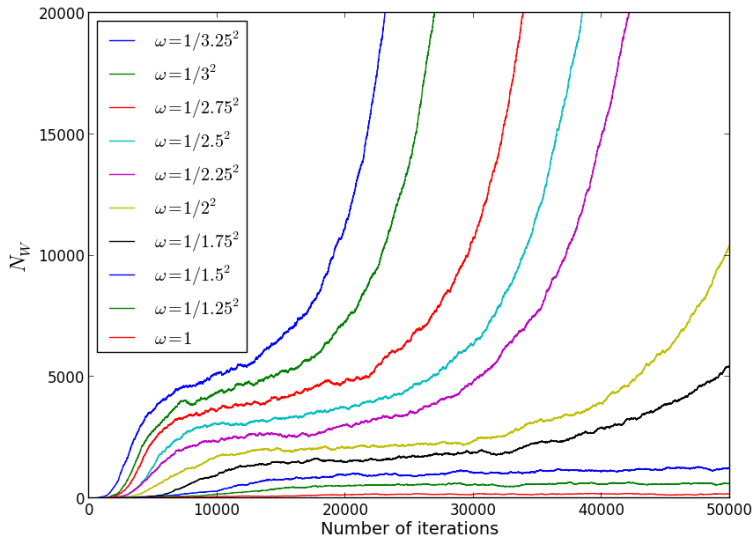
Figure 22.1: FCIQMC simulations of 6 particle quantum dots with $m = 2s = 0$ and an oscillator frequency $\omega = 1$. In plot (a) we see that the number of walkers N_W reaches a plateau at $N_W \approx 9 \times 10^5$ which is the critical number of walkers N_C. The population growth is still exponential when $N_W > N_C$, but with a smaller exponent. As plot (b) shows, the FCIQMC ($N_I = 1$) energy does not converge before the number of walkers is close to N_C. The energies are in units of Hartree (H)

shows that N_C increase faster as a function of $\dim(\mathscr{H})$ for systems with a low oscillator frequency ω. As an example, the three particle quantum dot with $\omega = 0.25$ requires a N_C which is close to 20 times as large as for the $\omega = 1$ quantum dot.

The slope of N_C as a function of $\dim(\mathscr{H})$ for a given system is a measure of the efficiency of FCIQMC. Since the run time of a simulation is roughly proportional to the number of walkers, it is a direct measure of the computational cost. Fig. 22.3(b) shows the numerical value of the slope for different quantum dots with the same ω. As this plot illustrates, some of the systems scales more favourably than others. The extremes are the 3 and 4 particle quantum dots, where the slopes differs by two orders of magnitude. It is interesting to notice that the four particle system is a difficult system for the Lanczos algorithm (FCI) as well. This is discussed by Ref. [20] and in section 20.2.2.



(a) Simulations with different R.

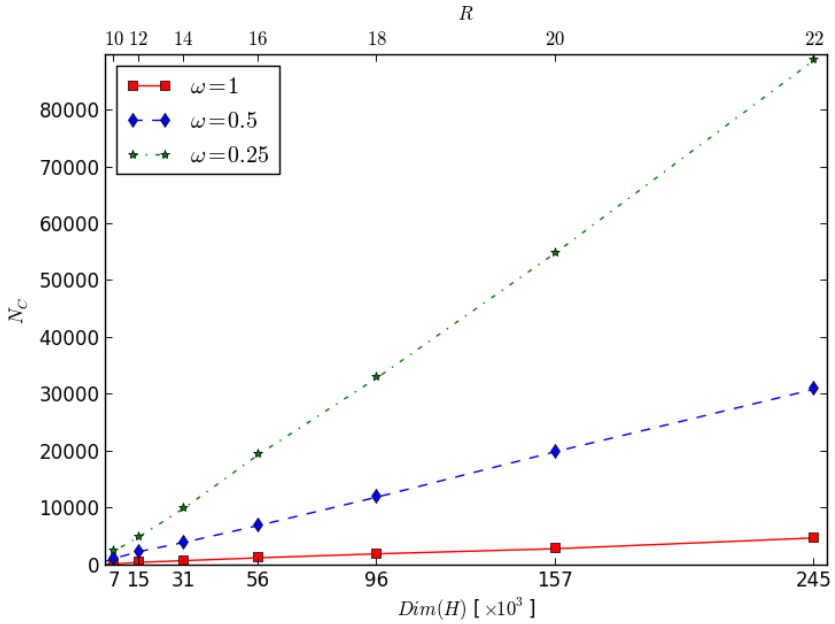


(b) Simulations with different ω.

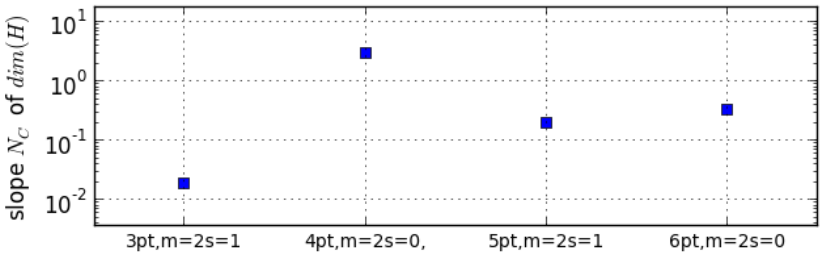
Figure 22.2: Simulations of 3 particle quantum dots with a fixed shift $S \approx E_0$ and a spin and magnetic quantum number $m = 2s = 1$. The critical number of walkers N_C is marked by a plateau in the number of walkers. N_C increase when the shell number R is increased or when the oscillator frequency ω is decreased. The reason that the population is less smooth here than in Fig. 22.1 is that N_C is smaller and consequently the statistical fluctuations become larger.

22.3 Concluding remarks

We have observed that the critical number of walkers is approximately linearly dependent of the basis size $N_C \approx s\dim(\mathscr{H})$. The slope s is a system dependent parameter which increases when ω is decreased. Based on our results we can not see any relationship between s and the number of particles in the system. This is a weak result since we only have taken the slope of N_C for four systems. As we discussed in section 18.3 the run time is approximately linear with the number of walkers. This means that the algorithm scales as $\mathcal{O}(\dim(\mathscr{H}))$.



(a) N_C as a function of $\dim(\mathscr{H})$ for 3 particle quantum dots with $m = 2s = 1$.



(b) The slope of N_C as a function of $\dim(\mathscr{H})$. $\omega = 1$ for all quantum dots.

Figure 22.3: (a): Simulations of 3 particle quantum dot with a magnetic and spin quantum number $m = 2s = 1$ with different oscillator frequencies ω. The critical number of walkers N_C was calculated with a fixed shift $S \approx E_0$. The critical number of walkers is plotted against the dimensionality of the Hilbert space, and as we see there is a linear relationship, but with a different slope for different ω. (b): The slope of N_C as a function of $\dim(\mathscr{H})$ is a measure of how easily the system is threated by the FCIQMC algorithm. The slopes have been calculated with a fixed shift S which is approximately 1% higher that the ground state energy.

Chapter 23

Convergence of the i-FCIQMC energies

Simulations with the initiator adaption of FCIQMC converges with a number of walkers N_W less than the critical number of walkers N_C, but at the cost of introducing an initiator error. As we discussed in chapter 9, the initiator error will disappear when the number of walkers N_W is large enough. By running a series of simulation with an increasing number of walkers, we would therefore expect the energy to converge as the initiator error becomes small. This provides us with a simple procedure to find the unbiased energy.

As an example, we will look at simulations of a five particle quantum dot with spin and magnetic quantum numbers $m = 2s = 1$ and with 24 shells in the basis. The Hilbert space of this system has a dimensionality $\dim(\mathscr{H}) = 3.5 \times 10^9$, and the critical number of walkers is $N_C \sim 3 \times 10^8$ assumed that N_C scales linearly with $\dim(\mathscr{H})$. We have run one series of simulations with an initiator threshold [1] $N_I = 4$ and one with $N_I = 12$, and with a varying number of walkers in the range $N_W \in [5 \times 10^4, 2 \times 10^6]$. The results are plotted in Fig. 23.1.

At a low number of walkers the initiator error is large, but eventually the energies seems to have converged within the statistical fluctiations. This happened at $N_W = 4 \times 10^4$ for the $N_I = 4$ simulations and at $N_W = 8 \times 10^4$ for the $N_I = 12$ simulations. This is between two and three orders of magnitude less than N_C, and demonstrates that i-FCIQMC is a very efficient optimization for this system.

With $N_I = 4$ and $N_W \leq 4 \times 10^4$, the distribution of walkers did not equilibriate. This could be observed by monitoring the population of the reference determinant n_0 which fluctuated around zero. We believe that the reason is that the density of walkers was too low for efficient annihilation, similar to what happens in a FCIQMC simulation when $N_W < N_C$. This did not happen with $N_I = 12$, in which case n_0 was stable already at $N_W = 2{\times}10^4$. The simulations converged at a lower number of walkers when we used a high N_I, but at the cost of a large initiator error. In fact, both with $N_I = 4$ and $N_I = 12$ approximately 8×10^5 walkers was required before the energy could be established with an error smaller than one mili-Hartree.

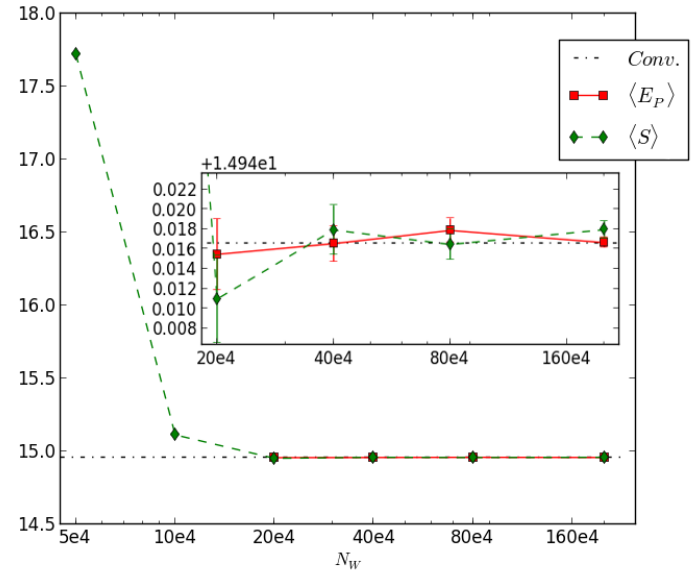
As we have demonstrated here, the i-FCIQMC energies converge when we increase the number of walkers. In our simulations we have assumed that the energies are converged when the following criteria are fulfilled:

[1] Remember that the initiator threshold N_I is the population limit where a determinant is added to the initiator space. This was discussed in chapter 9.

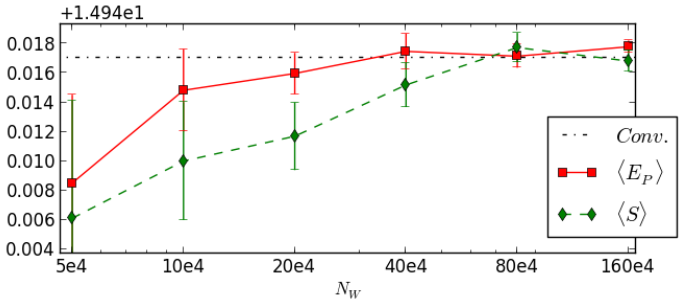
(i): The statistical estimators $\langle S \rangle$ and $\langle E_P \rangle$ yields the same result.

(ii): The statistical estimators are constant within the statistical error when we increase the number of walkers.

We have chosen to use small values for N_I less than or equal to four in all simulations. The reason is twofold. First, using a large N_I is an uncontrolled approximation, and we do not know how it affects our results. Second, our experience is that even though the simulations converge at a lower number of walkers with a large N_I, the initiator error may be large, and the number of walkers that is needed for the energies to converge to the FCI energy may be the same or even larger.



(a) $N_I = 4$



(b) $N_I = 12$

Figure 23.1: Simulations of a five particle quantum dot with 24 shells in the basis and with magnetic and spin quantum numbers $m = 2s = 1$. The vertical axis are in units of Hartree. All runs are performed with the same number of iterations (2×10^6). For this system the dimension of the Hilbert space is $\dim(\mathscr{H}) = 3.5 \times 10^9$ and the critical number of walkers is $N_C \sim 3 \times 10^8$. We have plotted the generational estimator $\langle S \rangle$ and the projected estimator $\langle E_P \rangle$ as a function of the number of walkers N_W with different initiator thresholds N_I. $Conv$ is the best i-FCIQMC result. Note that the energies converge at $N_W << N_C$.

Chapter 24

Simulations with a Hartree Fock basis

As we have seen, FCIQMC and i-FCIQMC enable us to calculate the FCI energies of systems which are far out of the scope of conventional diagonalization methods like the Lanczos algorithm. But we have also seen that the critical number of walkers together with the numerical cost, increases very fast as we increase the number of shells or the number of particles. In an attempt to improve the convergence we have looked at the effect of using a Hartree- Fock (HF) basis instead of the plain harmonic oscillator (HO) basis. We have run tests with a six particle $R = 9, \omega = 1, m = s = 0$ quantum dot with both the HF basis and the HO basis to compare the results. Since the HF basis is simply a unitary transformation of the HO basis, we would expect to get the exact same energies.

The first test was to find the critical number of walkers N_C. Surprisingly, N_C was a little higher with the HF basis than with the HO basis, with $N_C \approx 8 \times 10^6$ and $N_C \approx 9 \times 10^6$ respectively. This indicates that the plain FCIQMC algorithm would require approximately the same number of walkers to converge with both the HF basis and the HO basis.

The next test was to run two series of i-FCIQMC simulations with identical run time parameters, one with the HF basis and one with the HO basis. The results are plotted in Figs. 24.1 and 24.2 and clearly shows that the statistical errors becomes much smaller with the HF basis. Figure 24.2 shows that the statistical error of the projected energy is almost one order of magnitude smaller with the HF basis than with the HO basis. In fact, since the numerical error is proportional to

$$\epsilon \propto n^{-1/2}, \tag{24.1}$$

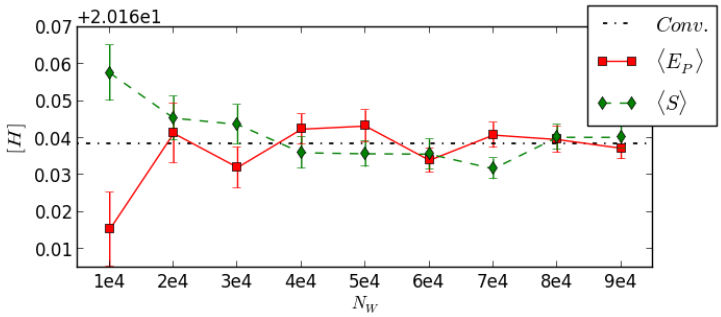
where n is the number of iterations, we would need 100 times as many iterations to reduce the error with one order of magnitude. And in this sense, implementing the HF basis is a large optimization.

The total run time of these simulations were relatively small. As an example, the total run time with $N_W = 5 \times 10^4$ on 32 cores was approximately 19 hours (36 minutes wall time). The energy of this simulation were converged and the projected estimator had a statistical error of 0.4 mili-Hartree (H). As a comparison, Olsen [20] have run simulations with the exact same system, but with a HO basis with effective interactions [1] . This simulation spent

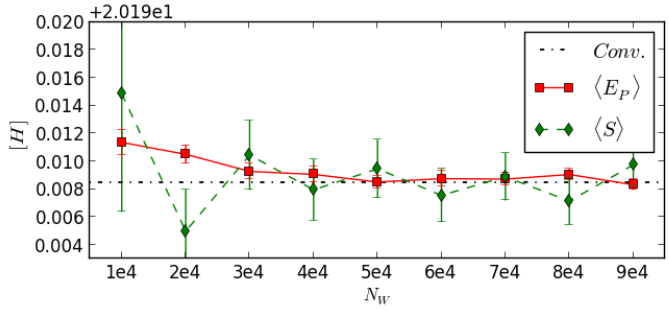
[1] The HO basis with effective interaction is a transformed HO basis, and was shown by Olsen [20] to speed up the convergence of the Lanczos algorithm considerably. Effective interactions are explained by Ref. [15].

a total run time of 13500 hours. Although the FCI simulation maybe would converge faster with a HF basis, this results clearly demonstrates that i-FCIQMC is much more efficient for this system.

Note that we have only performed simulations with a HF basis on this particular system, and we have yet to test how it works with other systems.

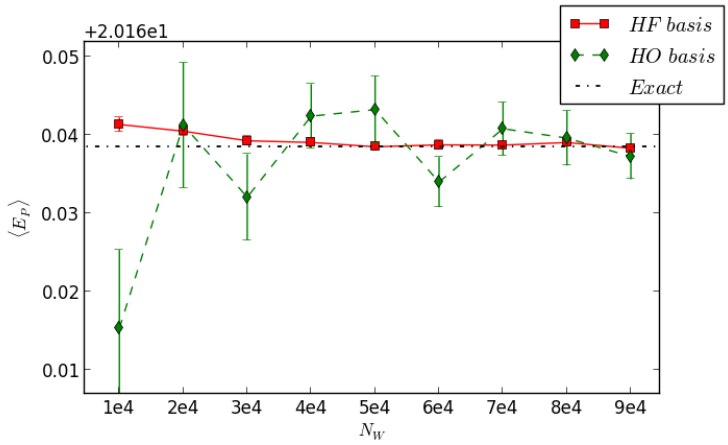


(a) HO basis.

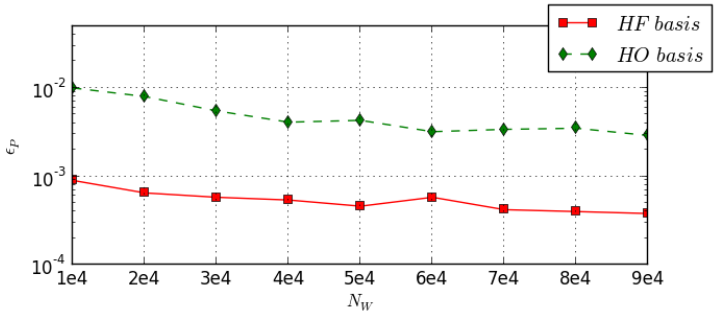


(b) HF basis.

Figure 24.1: These plots shows the convergence of the projected estimator $\langle E_P \rangle$ and the generational estimator $\langle S \rangle$ for simulations with six particles, $m = s = 0$, an oscillator frequency $\omega = 1$, an initiator threshold $N_I = 3$ and a basis truncated at ten shells. The exact value ($Conv.$) is taken to be the energy with the smallest error. Note that the y-axis of the lower the plot are zoomed in to a smaller scale. The simulations with the HF basis have a much smaller statistical error than the simulations with the HO basis.



(a) Convergence of E_P



(b) The statistical error of $\langle E_P \rangle$

Figure 24.2: These plots shows the convergence of the projected energy for simulations with six particles, $m = s = 0$ an oscillator frequency $\omega = 1$, an initiator threshold $N_I = 3$ and a basis truncated at ten shells. The exact value is taken to be the energy with the smallest error. We have run the simulations both with the harmonic oscillator (HO) basis and the Hartree Fock (HF) basis. The filtered Hilbert space of this systems has 2.3×10^7 determinants and the critical number of walkers is $\sim 8 \times 10^6$ for the HO basis and $\sim 9 \times 10^6$ for the HF basis. We see that the error is almost one order smaller for the simulations performed with the HF basis.

Chapter 25

Extrapolated energies

25.1 Introduction

The FCIQMC energies set an upper bound to the ground state energy, but are limited by the basis set incompleteness of the FCI spaces. We have used the results of Ref. [16] and section 5.3 to extrapolate the energies to the limit of a complete basis. Extrapolation of the energy is common practice in quantum chemistry [12], and provides a systematic way of improving the energies.

In this section, we will calculate a set of extrapolated energies and compare these to Diffusion Monte Carlo (DMC) energies which are not affected by a basis incompleteness error. We use the extrapolation formula Eq. (5.11)

$$E(R) \approx a - b \sum_{r=1}^{R} (N + r) r^{-c}, \tag{25.1}$$

where N is the number of particles, $R + 1$ is the number of shells and $a, b, c \in \mathbb{R}$ are constants. A curve fit of the FCIQMC values to $E(R)$ yields the optimal parameters a, b, c. The extrapolated energy is taken at the limit $R \to \infty$, and the error is estimated using Eqs. (13.18) and (13.22) assuming that Eq. (25.1) is exact.

As we discussed earlier, the extrapolation formula has an uncontrolled error $\propto \nu(R)$ where $\nu(R)$ is an unknown function. $\nu(R)$ is however assumed to fall off quickly as a function of R. This is justified by the results of Kvaal [16] together with general considerations about the form of the function. As we will see in this section, this assumption seems to hold well for the systems that we have threated. To minimize $\nu(R)$ and reduce the extrapolation error, we have only used energies calculated with $R \geq 5$.

25.2 Testing the extrapolation formula

Fig. 25.1 shows the fitted curves of the extrapolation formula to energies from a series of simulations. All energies are calculated with i-FCIQMC with an initiator threshold $N_I = 4$ and a different number of walkers N_W to assure that the energies are properly converged. The correspondence between the data points and the fitted curves are very good, indicating that Eq. (5.11) applies well to the direct product spaces and that $\nu(R)$ indeed is small. This impression is strengthened by the numerical values of the extrapolated energies of

the 2 and 6 particle quantum dots with an oscillator frequency $\omega = 1$ which shows a good correspondence with the results of Refs. [24, 20, 37]. The extrapolated energy of the two particle quantum dot is $3.0000(1)$ Hartree (H) which is very close to the exact result of $3H$ [37]. The extrapolated energy for the 6 particle quantum dot is $20.160(1)H$ whereas the DMC result is $20.1597(2)H$ and the coupled cluster [1] result is 20.1582 [24]. The convergence of the energy for the two and six particle systems are illustrated in Fig. 25.2.

25.2.1 Open shell results for $\omega = 1/1.89^2$

We have run a series of simulations in an attempt to reproduce the results of Pederiva *et. al.* (2000) [22, 23]. Pederiva *et. al.* have run DMC simulations with single and multi determinant wave functions on open shell systems. These results are as far as we know not reproduced in any other articles, and it is therefore interesting on its own to find out whether FCIQMC yields the same results. Even if the DMC calculation minimize the energy for a given set of nodes, the nodes imposed by the guiding wave function is an uncontrolled approximation. FCIQMC on the other hand is free of approximations and is systematically improvable, but is limited by a much higher computational cost. As we see from Tab. 25.2, the results for the two particle quantum dot is in good agreement with the DMC results. The three and five particle results are however significantly lower. We also see that the lowest FCIQMC value is lower than the DMC values for the three and four particle systems, and since our energies are variational, this indicates that the DMC values are too high.

N_p	m	$2s$	$E(\infty)$	Lowest FCIQMC	DMC (Ref. [22, 23])
2	0	0	1.0219(2)	1.0222(1)	1.02164(1)
3	1	1	2.2314(3)	2.2326(1)	2.2339(1)
4	0	0	3.7138(3)	3.71508(2)	(*)
5	1	1	5.5302(2)	5.5327(3)	5.5338(1)

Table 25.1: Open shell results with a different number of particles N_P and with an oscillator frequency $\omega = 1/1.89^2$. The three and the five particle results are significantly lower than the DMC results. Since the lowest FCIQMC energies are variational, this suggests that the DMC results are too high for these systems. The result marked with $(*)$ is not available. We found that there exists an erratum [23] to the article by Pederiva *et. al.* [22] where they correct some of the energies. Here they write that the energies of the four particle states with $m = s = 0$ and $m = 0, 2s = 2$ are nearly degenerated, and that the $m = 0, 2s = 2$ state is the ground state with an energy $3.7145(1)H$. The energy of the $m = s = 0$ state in the first article was not correct. We unfortunately found out too late to do simulations with the $m = 0, 2s = 2$ state.

25.2.2 Open shell results for $\omega = 1$

We have also included the open shell results of the $\omega = 1$ simulations for up to six particles. Only the closed shell results (with two and six particles) have been found in published articles [24, 20, 37]. And as we already have discussed, these results correspond very well with our FCIQMC energies.

[1] The coupled cluster energies was obtained with CCSD(T) and a basis with effective interactions. We have not written about the coupled cluster method in this thesis, but a good introduction is provided in Ref. [33].

N_P	m	$2s$	$E(\infty)$	Lowest FCIQMC	DMC (Ref. [24])
2	0	0	3.0000(3)	3.0018(1)	3.00000(3)
3	1	1	6.3564(2)	6.3596(3)	
4	0	0	10.2635(5)	10.2684(4)	
5	1	1	14.9489(6)	14.9570(7)	
6	0	0	20.160(1)	20.171(2)	20.1597(2)

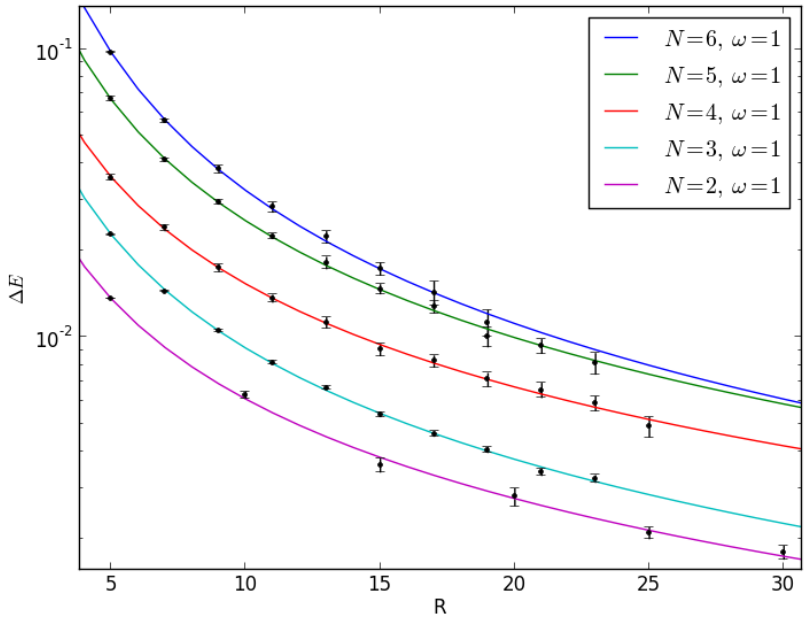
Table 25.2: Open shell results with a different number of particles N_P and with an oscillator frequency $\omega = 1$. We have only found published energies for the closed shell results. These results are reproduced by FCIQMC.

25.3 Summary and Comments

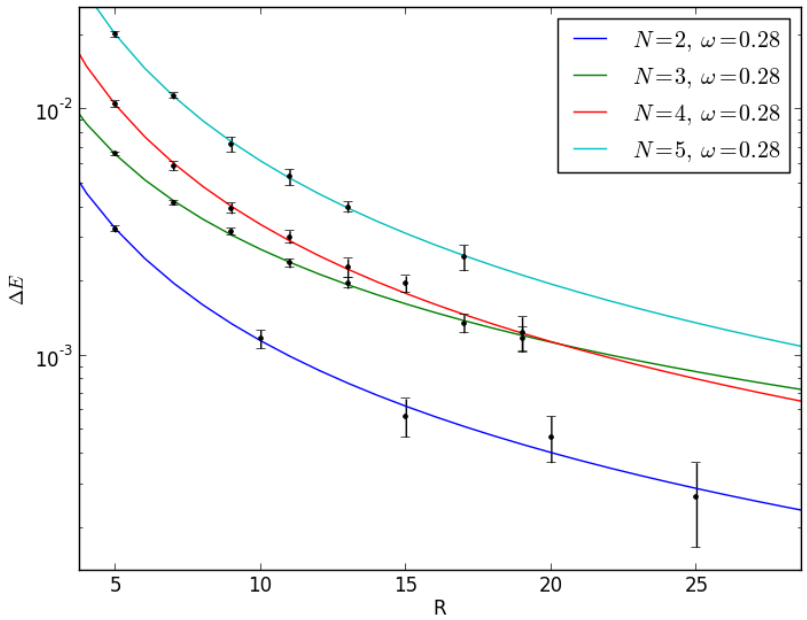
This chapter contains two main results that we want to emphasize. The first concerns the validity of the extrapolation formula (5.11) and the second concerns the discrepancies between the DMC energies and the FCIQMC energies for the open shell quantum dots.

As we discussed in section 5.3, Kvaal [16] has derived a closed form expression for the basis incompleteness error. This expression has however an unknown factor $(1 + \nu(R))$ where $\nu(R)$ is assumed to be negligible, and is only proved to be valid for the energy cut spaces. Kvaal has used FCI calculations to show that his formula applies well to various three, four and five particle quantum dots. But due to the high numerical cost of the FCI method, he was only able to study these systems with "small" FCI spaces. The FCIQMC method has a much lower numerical cost, and we were therefore able to test Kvaal's formula with much larger FCI spaces. Our results indicate that Kvaal's formula applies well to the direct product spaces, and that the error $\nu(R)$ is negligible for the FCI spaces that we have studied.

Next, we want to comment on the differences between the FCIQMC and the DMC results. Our results indicate that some of the open shell DMC results of Pederiva *et. al.* (2000) [22] are to high. These are the three and five particle results for the $\omega = 1/1.89^2$ quanum dots, for which we have seen that the DMC energies are significantly higher than our FCIQMC energies. Furthermore, the FCIQMC energies are variational which indicates that the DMC energies are too high. The most important source of error in a DMC calculation is the fixed node error, and it is therefore reasonable to assume that this is the reason why the DMC energies are too high. This would suggest that the nodes need to be optimized either by using a multi determinant trial wave function or by finding a more optimal single particle basis. Note that Pederiva *et. al.* [22] have used single determinant wave functions for the three and five particle systems, and a single particle basis which is optimized using Density Functional Theory calculations.

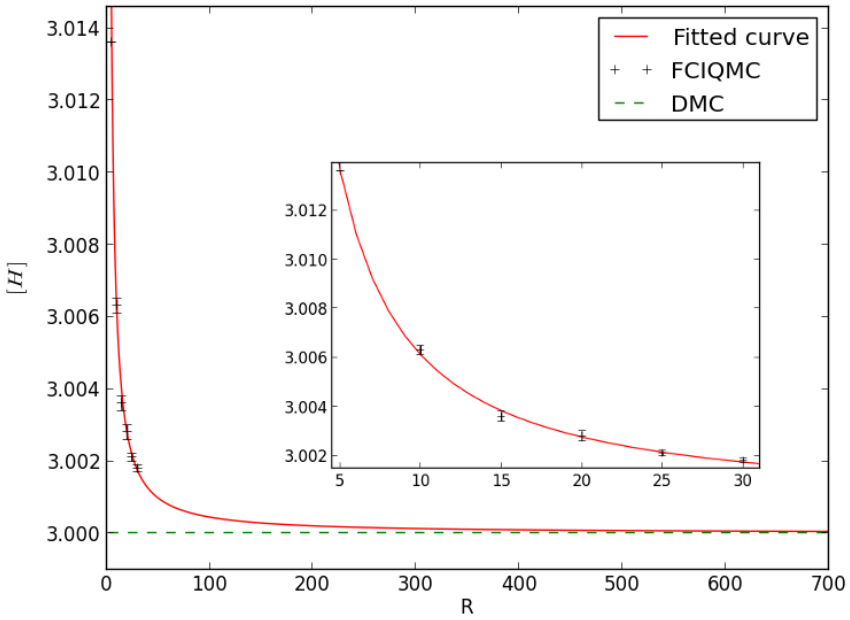


(a) $\omega = 1$.

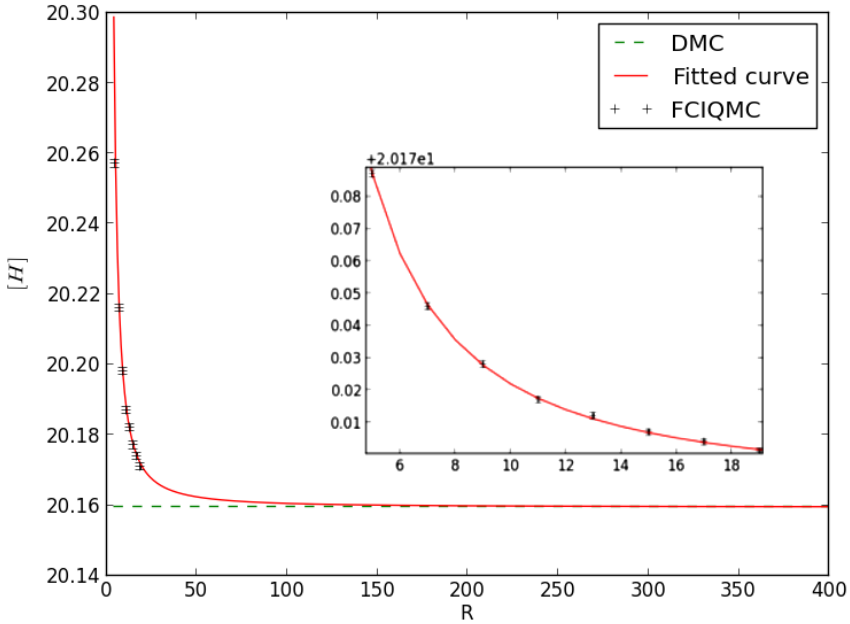


(b) $\omega = 1/1.89^2 \approx 0.28$.

Figure 25.1: Plots of the parametrized error $\Delta E = E(R) - E(\infty)$ where $E(R)$ is calculated according to the extrapolation formula Eq. (5.11). $R + 1$ is the number of shells in the basis, ω is the oscillator frequency and the error are in units of Hartree. The datapoints to which the curves are fitted are plotted with errorbars. These are the FCIQMC energy minus the extrapolated energy at infinity $E(\infty)$. As we see, the correspondence between the parametrized energy $E(R)$ and the FCIQMC energies is very good.

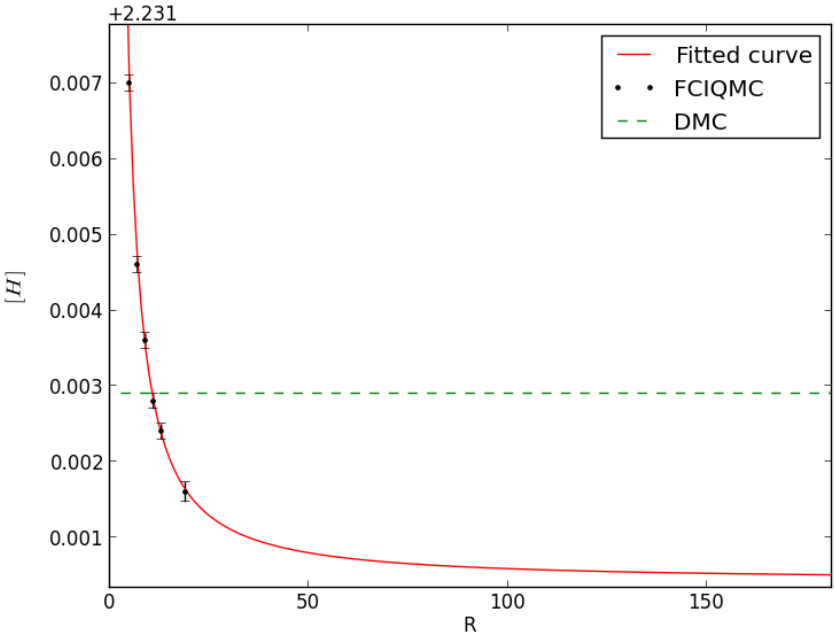


(a) 2 particle results, $\omega = 1$.

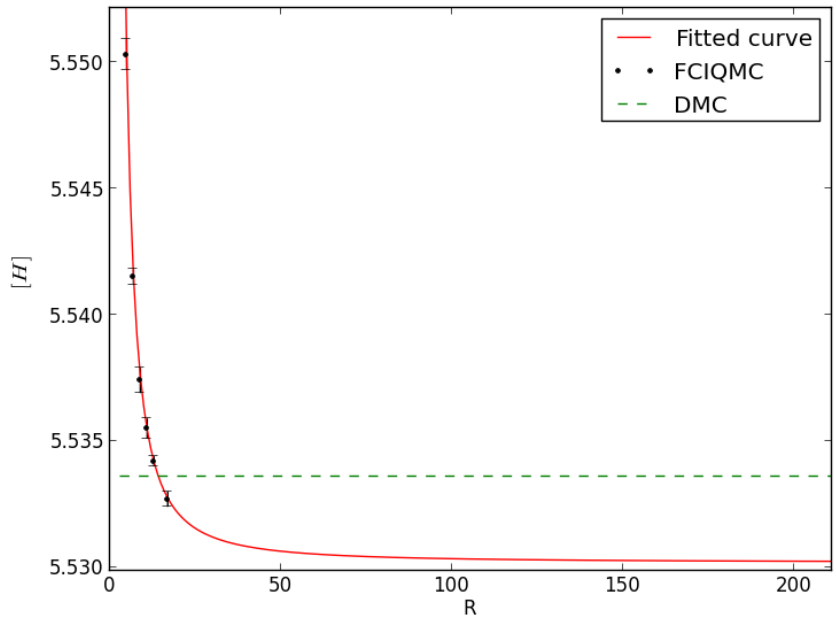


(b) 6 particle results, $\omega = 1$.

Figure 25.2: These plots shows the convergence of the FCIQMC energies as we increase the number of shells $R + 1$. The lines labelled "DMC" are the Diffusion Monte Carlo energies of Ref. [24], and the lines labelled "Fitted curve" are calculated according to the extrapolation formula Eq. (5.11). For $R \to \infty$ the parametrized functions converge to $3.0000(1)H$ for two particles and $20.160(1)H$ for six particles. The exact results are $3H$ in the two particle case and in the six particle case the best value is the DMC result at $20.1597(2)H$.



(a) 3 particle results, $\omega = 1/1.89^2$, $2s = m = 1$.



(b) 5 particle results, $\omega = 1/1.89^2$, $2s = m = 1$.

Figure 25.3: These plots shows the convergence of the FCIQMC energies as we increase the number of shells $R + 1$. The lines labelled "DMC" are the Diffusion Monte Carlo energies of Ref. [22, 23], and the lines labelled "Fitted curve" are calculated according to the extrapolation formula Eq. (5.11). FCIQMC converges to lower energies than DMC [22, 23] which are $2.2339H$ for the three particle quantum dot and $5.5338(1)H$ for the five particle quantum dot. The data points represent the FCIQMC energies, and since these are variational the DMC energies appears to be too high.

Chapter 26

Conclusions and perspectives

We will in this chapter sum up our main achievements and results, and discuss prospects for further work.

26.1 Summation and achievements

Our initial motivation was to implement the FCIQMC algorithm and apply it to quantum dots. We have written a fully parallelized C++ code from scratch which is scaling well up to several hundred cores. This may be seen as our main achievement, and has been the part of this project in which we have invested the most work. In this part we have validated the code by demonstrating that it reproduces FCI energies, and by showing that the i-FCIQMC energies converge to the FCIQMC energies when the number of walkers is sufficiently large. We have also applied the algorithm on a new physical system, namely the two dimensional quantum dots, which to our knowledge has not been studied with FCIQMC before.

We have found that the efficiency of the algorithm is highly system dependent. We used the ratio of the critical number of walkers N_C to the number of determinants in the basis $\dim(H)$ as a measure of the efficiency of the algorithm. This is a valid measure since the run time of the simulations is roughly proportional to the number of walkers. The algorithm was shown to scale as $s \dim(\mathscr{H})$ where s is a system dependent parameter. We have found that s is larger for systems with a low oscillator frequency ω (strong correlations), but we were not able to see any systematic relationship to the number of particles in the system.

We have also tested our implementation of the i-FCIQMC algorithm, and demonstrated that for the five particle system with $m = 2s = 1$ and 24 shells in the basis, the algorithm converged within mili-Hartree precision with less than 10^6 walkers while the critical number of walkers N_C was more than two orders of magnitude larger. We also did some runs with a Hartree- Fock (HF) basis for the six particle quantum dot with $m = s = 0$ and ten shells. Although the critical number of walkers did not decrease, the statistical error was measured to be almost one order of magnitude smaller with the HF basis than with the HO basis.

At last, we estimated FCI energies in the limit of an infinite basis $E(\infty)$ by fitting the extrapolation formula Eq. 5.3 to FCIQMC energies. This formula was derived by Kvaal [16], but was only shown to be correct for energy cut spaces. Our results indicate that it applies to direct product spaces as well. The parametrized curves show a seemingly perfect fit to the FCIQMC energies up to a high number of shells and the extrapolated energies reproduce Diffusion Monte Carlo and Coupled Cluster results for the closed shell quantum

dots [24, 23]. However, the Diffusion Monte Carlo results for open shell quantum dots with $\omega = 1/1.89^2$ [22, 23] was higher than the FCIQMC energies. Since the FCIQMC energies are variational, this indicates that the DMC results are too high.

26.2 Ideas and prospects

As we saw in the last part, implementing the Hartree-Fock(HF) basis resulted in a much faster convergence of the FCIQMC energies. The first step to improve the performance of the program would therefore be to do simulations with an improved basis. In addition to the HF method, it is also possible to optimize the basis using more refined methods like Density Functional Theory calculations [12].

As the FCIQMC algorithm is a relatively new algorithm, it has only been applied to a limited range of systems. And since this algorithm has different strengths and weaknesses compared to other many-body methods, it might be possible to gain new insights by applying it to new systems. To my knowledge, FCIQMC has been applied to a range of atoms and molecules (see for example [3, 4]), the homogeneous electron gas [34] and a range of solids [2], but not to nuclear systems.

One application of the algorithm that could be interesting is to study three-body forces. The number of Hamiltonian matrix elements increases drastically when three body forces are included, and as a consequence, the numerical cost of methods like FCI or Coupled Cluster blows up. Assumably, with the FCIQMC method, it might be possible to include three-body forces without a large increase in the numerical cost. The reason is that the three-body interactions are less probable than the one or two body interactions. From a sampling perspective, this means that triple excitations should be sampled less often than the single and double excitations. So even if the three body forces are numerically more expensive to sample, they will not necessarily add very much to the overall numerical cost. Note that this is the same as we do when we sample the one-body and two-body interactions, and choose a high probability (p_s) for the single excitations (see chapter 10).

26.3 Concluding remarks

As we have seen, the FCIQMC algorithm enables us to calculate FCI energies for systems with Hilbert spaces that are far out of the scope of "brute force" diagonalization algorithms like the Lanczos algorithm. But as we saw in the last chapter, the energy converges very slowly as a function of the basis size. And although we can do simulations with large Hilbert spaces, the exact ground state can only be estimated by extrapolating the energies to the complete basis. FCIQMC is a good complement to the Lanczos algorithm, which we believe is more practical for "small" Hilbert spaces as it automatically calculates a spectrum of excited energies. Both of these algorithms provide good benchmarks for other many body methods that operate in a Slater determinant basis since they yield unbiased energies within a given Hilbert space.

We would also like to comment on the development of FCIQMC and related algorithms. As the algorithm is relatively new, we currently see a lot of development and new applications of the algorithm. See for example the article by Booth *et. al.* (2013) [2] where the algorithm is applied to a range of solids. Recently, several new algorithms have appeared which also do stochastic sampling of discrete Hilbert spaces with walkers. Examples are the

stochastic extension of the Lanczos algorithm Shimizu *et. al.* (2013) [35], and stochastic coupled cluster Thom (2010) [38]. This is an exciting field where there currently is many new ideas and a lot of development happening.

Part VII

Appendix

Appendix A

Running the code

The code is wrapped in a python `runFCIQMC.py` script that generates a headerfile and a `.ini` file with run time parameters. Template parameters must be available at compile time. Therefore the bitset template parameter `N`, is written to the headerfile as a precompilator definition.

The following parameters must be set before each simulation

- `i_r`: The number of shells in the basis -1.
- `i_maxndets`: The maximum number of determinants that can be stored on each MPI task.
- `d_dt`: The time step τ.
- `d_lambda`: The interaction strength $\lambda = 1/\omega^2$.
- `i_ranseed`:The random seed. If this parameter is set to -1, the random seed is set equal to the system time.
- `i_limit_nw`: When the number of walkers reaches this limit, we start variyng the shift parameter to control the number of walkers.
- `i_num_loops`: The total number of FCIQMC loops.
- `i_n_start_coll_e`: The number of iterations where we start to collect energy.
- `d_ds`: The initial energy shift S.
- `i_initiatorlimit`: When the population of walkers on a determinant reaches this limit, the determinant is added to the initiator space.
- `s_initial_state`: The reference determinant and the initial state in our simulations. Ex: $|\varphi_0, \varphi_2\rangle$ =`'0,2'`.
- `s_frozen_orbs`: The frozen spin orbitals of the CAS. Ex: If the spin osbitals φ_1, φ_2 are frizen, we set this variable to `'1,2'`.
- `i_active_orbs`: The number of the highest spin orbital in the CAS space.
- `b_writeodata`: Set to `True` to write the simulation data to file.

- •: `d_detweight`: The load weight parameter. Find the best value to optimize the load balancing.
- •: `d_redistlimit`: The maximal relative difference between the load weight of two MPI tasks before we redistribute the determinants.
- •: `i_numthreads`: The number of threads. Overridden by the enviromental variable `OMP_SET_NUM_THREADS`.

The following example shows how the runtime parameters are set

```
#example run script runscript6particles.py
import runFCIMC as r
from math import sqrt

r.b_writeodata = True #write output data to file

r.i_maxndets = 1e7 #size of determinants arrays
r.i_num_loops = 5e4 #number of loops
r.i_n_start_coll_e = 1e4 #start collecting energy

r.d_detweight = 1.4 # load balancing parameter
r.d_redistlimit = 0.04 # relative load weight difference before
    load balancing of MPI tasks

r.delta_t = 0.001 #timestep
r.d_ds = 27 #initial energy shift

r.d_lambda    = 1 # interaction strength (= 1/sqrt(omega))
r.i_r       = 10 # number of shells-1 in the basis
r.s_initial_state = '0,1,2,3,4,5'

r.i_initiatorlimit  = 4 #i-FCIQMC initiarot limit
r.i_active_orbs = 11 #active orbitals in the CAS 0,1,2,3,..,11
r.frozen_orbs = '0,1' #frozen orbitals in the CAS

#the name of the simulations (and the outputfiles)
simulationtag = '6particles_example'
#simulation tag set to yyyy_dd_mm_ss by defaulth in not specified
r.geninifiles(simulationtag)
```

`.ini` file and the header file can is created when we call the function `geninifiles`. The code can now be run from the command line with the following commands.

```
user@host:$ python runscript6particles.py
user@host:$ make run
```

The outputdata will be stored to the file `6_particles_example.dat`.

Bibliography

[1] J. B. ANDERSON, *A random-walk simulation of the schrödinger equation:* H_3^+, The Journal of Chemical Physics, 63 (1975), p. 1499.

[2] G. H. BOOTH, A. GRUNEIS, G. KRESSE, AND A. ALAVI, *Towards an exact description of electronic wavefunctions in real solids*, Nature, 493 (2013), p. 365.

[3] G. H. BOOTH, A. J. W. THOM, AND A. ALAVI, *Fermion monte carlo without fixed nodes: A game of life, death, and annihilation in slater determinant space*, The Journal of Chemical Physics, 131 (2009), p. 054106.

[4] D. CLELAND, G. H. BOOTH, AND A. ALAVI, *Communications: Survival of the fittest: Accelerating convergence in full configuration-interaction quantum monte carlo*, The Journal of Chemical Physics, 132 (2010), p. 041103.

[5] H. FLYVBJERG AND H. G. PETERSEN, *Error estimates on averages of correlated data*, The Journal of Chemical Physics, 91 (1989), p. 461.

[6] A. FOG, *[software]* `CRandomMersenne` *version 2.01. 2010-08-03*, 2013. A C++ library to calculate random numbers using the Mersenne Twister algorithm. `http://www.agner.org/random`.

[7] E. GABRIEL, G. E. FAGG, G. BOSILCA, T. ANGSKUN, J. J. DONGARRA, J. M. SQUYRES, V. SAHAY, P. KAMBADUR, B. BARRETT, A. LUMSDAINE, R. H. CASTAIN, D. J. DANIEL, R. L. GRAHAM, AND T. S. WOODALL, *Open MPI: Goals, concept, and design of a next generation MPI implementation*, in Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004, p. 97.

[8] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*, The Johns Hopkins University Press, 3rd ed., Oct. 1996.

[9] E. GROSS, E. RUNGE, AND O. HEINONEN, *Many-particle theory*, A. Hilger, 1991.

[10] B. L. HAMMOND, W. A. LESTER, AND P. J. REYNOLDS, *Monte Carlo Methods in Ab Initio Quantum Chemistry*, World Scientific, 1994.

[11] A. HARJU, *Variational monte carlo for interacting electrons in quantum dots*, Journal of Low Temperature Physics, 140 (2005), p. 181.

[12] T. HELGAKER, P. JØRGENSEN, AND J. OLSEN, *Molecular Electronic Structure Theory*, John Wiley & Sons, LTD, Chichester, 2000.

[13] M. H. KOLODRUBETZ, J. S. SPENCER, B. K. CLARK, AND W. M. FOULKES, *The effect of quantization on the full configuration interaction quantum monte carlo sign problem*, The Journal of Chemical Physics, 138 (2013), p. 024110.

[14] S. KVAAL, *Analysis of many-body methods for quantum dots*, PhD thesis, University of Oslo, Norway, 2008.

[15] S. KVAAL, *Open source FCI code for quantum dots and effective interactions*. arXiv:0810.2644v1, 2008.

[16] S. KVAAL, *Harmonic oscillator eigenfunction expansions, quantum dots, and effective interactions*, Physical Review B, 80 (2009), p. 045321.

[17] D. W. MARQUARDT, *An Algorithm for Least-Squares Estimation of Nonlinear Parameters*, SIAM Journal on Applied Mathematics, 11 (1963), p. 431.

[18] M. MATSUMOTO AND T. NISHIMURA, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation, 8 (1998), p. 3.

[19] D. MUSSER, *Introspective sorting and selection algorithms*, Software Practice and Experience, 27 (1997), p. 983.

[20] V. K. B. OLSEN, *Full configuration interaction simulation of quantum dots*, Master's thesis, University of Oslo, 2012.

[21] OPENMP ARCHITECTURE REVIEW BOARD, *OpenMP Application Program Interface Version 3.1*, 2011.

[22] F. PEDERIVA, C. J. UMRIGAR, AND E. LIPPARINI, *Diffusion monte carlo study of circular quantum dots*, Physical Review B, 62 (2000), p. 8120.

[23] ——, *Erratum: Diffusion monte carlo study of circular quantum dots [physical review b 62, 8120 (2000)]*, Physical Review B, 68 (2003), p. 089901.

[24] M. PEDERSEN LOHNE, G. HAGEN, M. HJORTH-JENSEN, S. KVAAL, AND F. PEDERIVA, *Ab initio computation of the energies of circular quantum dots*, Physical Review B, 84 (2011), p. 115302.

[25] M. A. REED, J. N. RANDALL, R. J. AGGARWAL, R. J. MATYI, T. M. MOORE, AND A. E. WETSEL, *Observation of discrete electronic states in a zero-dimensional semiconductor nanostructure*, Physical Review Letters, 60 (1988), p. 535.

[26] S. M. REIMANN AND M. MANNINEN, *Electronic structure of quantum dots*, Reviews of Modern Physics, 74 (2002), p. 1283.

[27] P. J. REYNOLDS, D. M. CEPERLEY, B. J. ALDER, AND W. A. LESTER, *Fixed-node quantum monte carlo for molecules.*, The Journal of Chemical Physics, 77 (1982), p. 5593.

[28] P. RICHTER, *Estimating errors in least-squares fitting*, The Telecommunications and Data Acquisition Progress Report 42-122, (1995), p. 107.

[29] M. RONTANI, C. CAVAZZONI, D. BELLUCCI, AND G. GOLDONI, *Full configuration interaction approach to the few-electron problem in artificial atoms*, The Journal of Chemical Physics, 124 (2006), p. 124102.

[30] C. C. J. ROOTHAAN, *New developments in molecular orbital theory*, Rev. Mod. Phys., 23 (1951), p. 69.

[31] J. J. SAKURAI, *Modern Quantum Mechanics (Revised Edition)*, Addison Wesley, 2 ed., Sept. 1993.

[32] M. SCHLOSSHAUER, *Decoherence: And the Quantum-To-Classical Transition*, The Frontiers Collection, Springer, 2007.

[33] I. SHAVITT AND R. BARTLETT, *Many-Body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*, Cambridge Molecular Science, Cambridge University Press, 2009.

[34] J. J. SHEPHERD, G. H. BOOTH, A. GRUNEIS, AND A. ALAVI, *Full configuration interaction perspective on the homogeneous electron gas*, Physical Review B, 85 (2012), p. 081104.

[35] N. SHIMIZU, T. MIZUSAKI, AND K. KANEKO, *Stochastic extension of the lanczos method for nuclear shell-model calculations with variational monte carlo method*, Physics Letters B, 723 (2013), p. 251.

[36] J. S. SPENCER, N. S. BLUNT, AND W. M. FOULKES, *The sign problem and population dynamics in the full configuration interaction quantum monte carlo method*, The Journal of Chemical Physics, 136 (2012), p. 054110.

[37] M. TAUT, *Two electrons in an external oscillator potential: Particular analytic solutions of a coulomb correlation problem*, Physical Review A, 48 (1993), p. 3561.

[38] A. J. W. THOM, *Stochastic coupled cluster theory*, Phys. Rev. Lett., 105 (2010), p. 263004.

[39] C. J. UMRIGAR, M. P. NIGHTINGALE, AND K. J. RUNGE, *A diffusion Monte Carlo algorithm with very small time-step errors*, The Journal of Chemical Physics, 99 (1993), p. 2865.