Coupled Cluster Studies in Computational Chemistry

by

Ole Tobias B. Norli

THESIS
for the degree of
MASTER OF SCIENCE

(Master i Computational Physics)

Faculty of Mathematics and Natural Sciences
University of Oslo

August 2014

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Abstract

In this thesis we explore the Coupled Cluster method in Quantum Chemistry. We have implemented an effective Coupled Cluster Singles and Doubles code. We also explore deviations from the true ground state. For this purpose we have implemented a Coupled Cluster Singles, Doubles and Triples code. Our results are in agreement with theory that Coupled Cluster converge to the ground state when including more excitations and improving the basis set.

Our code performance is approaching the level of the best performing software available. Further continuations of already implemented optimizations are proposed to help development of more effective Coupled Cluster code.

Acknowledgement

I would like to acknowledge my supervisor Morten H. Jensen. You are the best supervisor I could ask for, thank you. Also thank you to Diako Darian.

Contents

Chapter 1

Introduction

Quantum Chemistry is a field of research where quantum mechanics is used to describe the behaviour of atoms and molecules. This can be used to model for example chemical reactions. A good understanding of chemical reactions is vitally important in several fields, from materials science to life science and medicine, with a huge potential for industrial applications. To develop accurate many-body methods which allow us to reproduce and predict properties of atoms and molecules is thus extremely important for scientific progress in a wide range of scientific fields, from basic research to industrial applications.

In 2013 Martin Karplus and Michael Levitt were awarded the Nobel Prize in chemistry for their development of multiscale models for complex chemical systems. Their work focused on Molecular Dynamics (MD) simulations of large chemical reactions. An important breakthrough in their work was combing higher and lower accuracy methods to provide an accurate and computationally efficient model. The active parts of the molecule were described with high accuracy, while the inactive parts were described with less accurate methods.

In this thesis we will focus on a high accuracy method in quantum chemistry. We will study the Coupled Cluster method. The Coupled Cluster method is one of the highly successful so-called first-principle methods (or *ab initio* methods) and was introduced in the late 1950s by Coester and Kümmel within the context of nuclear physics. It was introduced in quantum chemistry in the 1960s by Cizek and Paldus. It is considered to be a highly accurate many-body method. In the 1980s through 1990s several computational chemists predicted Coupled Cluster would be the method of choice for most calculations in quantum chemistry today. However the method applied in the absolute majority of publications is Density Functional Theory (DFT).

The main reason DFT is so popular is the computational affordability.

DFT can model much larger systems than Coupled Cluster, and in much less time. Therefore we will focus much of our attention on implementing an optimized Coupled Cluster code.

In this thesis we will develop computational chemistry methods based on quantum mechanics. These are called ab initio quantum chemistry methods. We will implement Hartree Fock (HF) theory, Coupled Cluster Singles and Doubles (CCSD) and Coupled Cluster Singles, Doubles and Triples (CCSDT) from scratch. We will design parallel algorithms for HF and CCSD. Our algorithms will focus on effective memory distribution and high performance. Calculations will be performed on the Abel supercomputing cluster of the University of Oslo. In particular the CCSD implementation will be greatly optimized. We will also present an extremely optimized algorithm for transformation of the four index integrals involved in post-HF methods.

We have benchmarked our performance and results against existing software. Since our implementation is made from scratch we will also propose further optimizations in great detail. The proposed optimizations combine positive features from our implementation and existing software developed by others. The main purpose will be working towards a more computationally affordable CCSD implementation. One that can also run calculations on larger molecules.

We will not present an optimized CCSDT implementation. CCSDT is implemented to better study the limitations on accuracy in CCSD. The Coupled Cluster method in theory only contains two errors. These are a limited basis set and a truncation of excitations included. With CCSDT implemented we will be able to study both these errors.

The thesis is structured for a good presentation of theory, code development and results. Chapter 2 describes the basics of the system we will study. The theoretical derivation of the Hartree Fock method using Gaussian Type Orbitals is given in chapters 3 and 4.

Chapter 5 contains a derivation of the CCSD method. In Chapter 6 we provide the factorized and implementation ready CCSD equations.

Chapter 7 contains information about the general programming principles we will apply in our implementation. This includes information on parallel programming and external libraries in use. Chapter 8 discusses our actual serial and parallel implementation of HF theory and CCSD.

Chapter 9 is an implementation guide to the CCSDT method. We will not derive the equations for this method. Multiple references are included and the equations are presented in an implementation ready form. Our actual implementation of CCSDT is plain and simple, as presented in this chapter.

In chapter 10 we present benchmark calculations to validate our implementation. Chapter 11 presents new results and chapter 12 states our conclusions. In chapter 13 we propose future prospects.

All code developed is freely available on github. Please see Ref.[92].

Chapter 2

Definition of Hamiltonian

In this chapter we present the Hamiltonian, with some basic definitions, for the systems we want to study in this thesis, namely various atoms and molecules (with an emphasis on molecules) using first principle theories. We are mainly interested in the ground state of atoms and molecules, and we aim at solving the time-independent Schrödringer equation

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle, \tag{2.1}$$

where $\mathbf{H}$ is the Hamiltonian of the system, Ψ the given eigenstate function and E the corresponding eigenenergy or simply energy of the system.

2.1 Hamiltonian

The full Hamiltonian for such atoms and molecules is well defined, it reads

$$\begin{aligned}\mathbf{H} = &-\sum_{A}^{nuc}\frac{1}{2m_A}\nabla_A^2 - \sum_{i}^{E}\frac{1}{2}\nabla_i^2 - \sum_{A}^{nuc}\sum_{i}^{E}\frac{Z_A}{|r_i - R_A|} \\ &+\sum_{i>j}^{E}\frac{1}{|r_i - r_j|} + \sum_{A>B}^{nuc}\frac{1}{|R_A - R_B|}.\end{aligned} \tag{2.2}$$

The various terms represent the kinetic and potential energy terms for the electrons and the nucleus (in case of atoms) or nuclei in case of molecules. Here R_A is the position of a given nucleus, r_i is the position of electron i, m_A is the mass ratio of a given given nucleus with the electron mass and Z_A is the charge of that specific nucleus.

2.2 The Born-Oppenheimer approximation

Throughout this thesis we will employ a Hamiltonian where the Born-Oppenheimer approximation is used. In this approximation we neglect the nuclear kinetic energy, since the time it takes for a nucleus to move is large compared to the time it takes for the electrons to obtain their ground state configuration. This means we can solve the equations first with the nucleus or the nuclei at fixed positions. When the nucleus (or nuclei in case of molecules) is (are) at a fixed position the kinetic energy term becomes zero. We neglect also contributions from nuclear forces since their energy scales are in the giga-electronvolt domain. We will refer to such a Hamiltonian as the electronic Hamiltonian, $\mathbf{H}_e$, and it reads

$$\mathbf{H}_e = -\sum_{i}^{E} \frac{1}{2}\nabla_i^2 - \sum_{A}^{nuc}\sum_{i}^{E} \frac{Z_A}{|r_i - R_A|} + \sum_{i>j}^{E} \frac{1}{|r_i - r_j|} + \sum_{A>B}^{nuc} \frac{1}{|R_A - R_B|}. \quad (2.3)$$

The term $\frac{1}{|R_A - R_B|}$ has no electrons in it, but it is often included in the electronic Hamiltonian. With nuclei at fixed positions this term reduces to a constant value. Using the electronic Hamiltonian we can then find the potential energy of the nuclei.

In this thesis we will thus only be working with the electronic Hamiltonian, and in future chapters we will just call it $\mathbf{H}$.

2.3 Comments on the Wavefunction

In this thesis we will represent the many-particle state function (or just wavefunction, Ψ) by single-particle basis function that solve the Hartree-Fock equations. These equations will be derived in chapter 3. The Hartree-Fock method represents an approximation to the solution of the full Schrödinger equation.

In practical terms, it is an algorithm which allows us to rewrite the above-mentioned many-particle Schrödinger equation in terms of coupled single-particle equations. It represents perhaps the simplest approach to the full many-body problem and provides a so-called self-consistently solved basis of orthogonal single-particle wavefucntions. We will call these single-particle states for spin orbitals hereafter. These basis functions are in turn used as input to so-called post Hartree-Fock methods like coupled-cluster theory.

In the Hartree-Fock approximation we assume that the many-body wavefunction can be written as a function of single electron wavefunctions. Each electron will occupy its own spin orbital. Since we are aiming at the ground state, the occupied spin orbitals are the ones with the lowest energy from the solution of the Hartree-Fock equations.

Figure 2.1 is an illustration of this. The state $|\Psi_0\rangle$ has six electrons. Two electrons occupy the lowest orbital in energy, one with spin up and one with

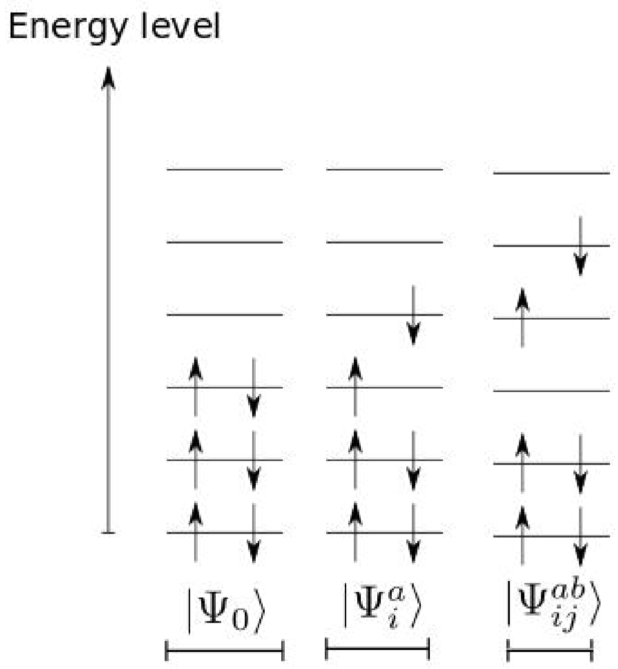


Figure 2.1: Illustration of electrons occupying orbitals to construct a wavefunction.

spin down. Two electrons occupy the second lowest orbital in energy, and the last two electrons occupy the third lowest energy level. This forms then an ansatz for the ground state. The next wavefunction has a configuration where one electron can be excited to a higher energetic orbital. This is labeled as $|\Psi_i^a\rangle$.

We here use a standard quantum chemistry notation, where occupied spin orbitals are labeled by the letters $i, j, k, \ldots$ and unoccupied orbitals are labeled as $a, b, c, \ldots$. The occupied spin orbitals serve to define the ansatz for the ground state wave function, in our case this will be a so-called Slater determinant since our particles (the electrons) are fermions and need to obey the requirement that the total wavefunction is antisymmetric in space. A generic spin orbital is labeled by the letters $p, q, r, \ldots$. Any electron can be excited to any of the higher orbitals. The state $|\Psi_{ij}^{ab}\rangle$ represents two electrons excited to higher energetic orbitals.

To find the true electronic wavefunction we would need a perfect description of the orbitals, and a linear combination of all the different possible excitations. There is an unlimited number of possible excitations, but some are more likely than others. In Coupled Cluster theory, to be discussed later, we will include some of these excited states.

Chapter 3

Hartree Fock

In this chapter we will discuss the Hartree-Fock (HF) method. We will derive the HF equations. For the most part we will limit ourselves to deal with a spin restricted HF (RHF) method, with closed shells and a single Slater determinant, Ref. [90], approximation to the wavefunction. However a spin unrestricted version has also been implemented and will be discussed briefly.

Much of the material discussed here is based on a series of summer lectures series from the Sherill Group, see for example Ref. [1], but see also Ref. [2] for further details. Additional references include the recent Master of Science theses from the Computational Physics group at the University of Oslo, see the theses of S. A. Dragly [3], H. M. Eiding [4] and M. H. Mobarhan [5]. This chapter is also closely related to the following chapter on an optimal basis for atoms and molecules, the so-called Gaussian Type Orbitals (GTO).

3.1 Introduction

Since our main focus is on molecules, in our exposition of the HF method we will describe how to approximate the Schrödringer equation for an arbitrary molecule. Our equation in atomic units reads

$$\mathbf{H}|\Psi\rangle = E|\Psi|\rangle, \tag{3.1}$$

where our electronic Hamiltonian, $\mathbf{H}$, is defined

$$\mathbf{H} = -\sum_i \frac{1}{2}\nabla_i^2 - \sum_{iA} \frac{Z_A}{r_{iA}} + \sum_{i>j} \frac{1}{r_{ij}} + \sum_{AB} \frac{Z_A Z_B}{R_{AB}}. \tag{3.2}$$

Here Z_A is the atomic number of nucleus A (with charge in atomic units) and r_{iA} is the distance from nucleus A to electron i. To simplify notation we will introduce two new operators, $\mathbf{h}(i)$ and $\mathbf{v}(i,j)$. These are defined by

$$\mathbf{h}(i) = -\frac{1}{2}\nabla_i^2 - \sum_A \frac{Z_A}{r_{iA}}, \tag{3.3}$$

which defines the one-body (or single-particle) electron Hamiltonian and

$$\mathbf{v}(i,j) = \frac{1}{r_{ij}}, \tag{3.4}$$

which is called the two electron part of our Hamiltonian. The quantity r_{ij} is defined as the distance between electron i and electron j. $r_{ij} = |r_i - r_j|$. This quantity has the following symmetry $r_{ij} = r_{ji}$, and has the constraint that $i \neq j$. We also introduce a shorthand notation for the nucleus-nucleus repulsion, namely,

$$V_{NN} = \sum_{AB} \frac{Z_A Z_B}{r_{AB}}. \tag{3.5}$$

This leaves Eq. (3.1) as

$$\left(\sum_i \mathbf{h}(i) + \frac{1}{2} \sum_{ij} \mathbf{v}(i,j) + V_{NN} \right) |\Psi(R)\rangle = E|\Psi(R)\rangle. \tag{3.6}$$

Here R is a vector of Cartesian coordinates (x, y, z) and spin for the different electrons. We have also included a factor $\frac{1}{2}$ since we removed the constraint $i > j$ from the sum.

3.2 Slater Determinant

The first assumption made in HF theory is that the wavefunction, $\Psi(R)$, can be written as a single Slater determinant. A Slater determinant is defined as

$$\Psi_T(R) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(x_1) & \psi_2(x_1) & \psi_3(x_1) & \dots & \psi_N(x_1) \\ \psi_1(x_2) & \psi_2(x_2) & \psi_3(x_2) & \dots & \psi_N(x_2) \\ \psi_1(x_3) & \psi_2(x_3) & \psi_3(x_3) & \dots & \psi_N(x_3) \\ \dots & \dots & \dots & \dots & \dots \\ \psi_1(x_N) & \psi_2(x_N) & \psi_3(x_N) & \dots & \psi_N(x_N) \end{vmatrix}. \tag{3.7}$$

Here N is the number of electron and x_i denotes the x, y and z coordinates for a single electron, i = 1, 2, ... The subscript T in Ψ_T indicates that this is a trial wavefunction, and not the exact one. The factor $\frac{1}{\sqrt{N!}}$ is a normalization factor.

An orbital is the wavefunction for a single electron. An atomic orbital is the wavefunction for a single electron in an atom. A molecular orbital is the wavefunction of a single electron in a molecule.

A spacial orbital is an orbital that describes the position of an electron. A spin orbital describes the position and the spin of an electron. Each spacial orbital has two spin orbitals, since electrons are fermions with spin up or spin down. The quantity ψ_i represents a molecular spin orbital, in case of molecules.

There are a few properties that make a Slater determinant an attractive trial wavefunction. First, it is antisymmetric, which means a change in sign upon interchanging two particles. Second it incorporates the Pauli Exclusion Principle, whose consequence states that two identical fermions cannot occupy the same state simultaneously.

For our purposes we approximate Ψ with a single Slater determinant. A single Slater determinant is a so called independent particle approximation. This will be discussed later in more depth.

Another shorthand notation for $\Psi_T(R)$ we will use soon is

$$|\Psi_T(R)\rangle = |ijkl\ldots\rangle, \tag{3.8}$$

where index i, j, k, l, ... refer to a molecular spin orbital.

3.3 The Energy Expression

We will now find an expression for the energy with this wavefunction. The energy can be found by rewriting Eq. (3.1), namely.

$$\begin{aligned}
E^{HF} &= \langle\Psi_T|\mathbf{H}|\Psi_T\rangle \\
&= \langle\Psi_T|\left(\sum_i \mathbf{h}(i) + \frac{1}{2}\sum_{ij}\mathbf{v}(i,j) + V_{NN}\right)|\Psi_T\rangle \\
&= \langle\Psi_T|\sum_i \mathbf{h}(i)|\Psi_T|\rangle + \frac{1}{2}\langle\Psi_T|\sum_{ij}\mathbf{v}(i,j)|\Psi_T\rangle + \langle\Psi_T|V_{NN}|\Psi_T\rangle. \qquad (3.9)
\end{aligned}$$

Here we have labeled the energy as E^{HF} in order to stress that it is the Hartree Fock energy we are aiming at. We also split up the equations into three parts. The easiest one comes from the nucleus-nucleus repulsion,

$$\langle\Psi_T|V_{NN}|\Psi_T\rangle = V_{NN}\langle\Psi_T|\Psi_T\rangle = \sum_{AB}\frac{Z_A Z_B}{r_{AB}}. \tag{3.10}$$

This will be a constant number. For the other two terms we use the attributes of the Slater determinant to simplify. We also insert the alternative notation noted in Eq. (3.8) and have

$$\langle \Psi_T | \mathbf{h}(i) | \Psi_T \rangle = \langle ijkl \ldots | \mathbf{h}(i) | ijkl \ldots \rangle. \tag{3.11}$$

The operator $\mathbf{h}(i)$ acts only on one orbital at the time, namely orbital i. The properties of the Slater determinant are such that this simplifies to

$$\langle ijkl \ldots | \mathbf{h}(i) | ijkl \ldots \rangle = \langle i | \mathbf{h} | i \rangle. \tag{3.12}$$

The expression for the two-electron operator simplifies to

$$\langle ijkl \ldots | \mathbf{v}(i,j) | ijkl \ldots \rangle = \langle ij || ij \rangle. \tag{3.13}$$

Notice that only two electrons are involved since we only have a two-body operator at most in our Hamiltonian. Here $\langle ij||ij\rangle$ is a shorthand for the double bar integral, defined as

$$\langle ij || ij \rangle = \langle ij | ij \rangle - \langle ij | ji \rangle. \tag{3.14}$$

with x_i being the coordinates and spin of electron i. Inserting this into Eq. (3.9) gives us

$$E^{HF} = \langle i | \mathbf{h} | i \rangle + \frac{1}{2} \left(\langle ij | ij \rangle - \langle ij | ji \rangle \right) + V_{NN}, \tag{3.15}$$

with $\langle ij|ij\rangle$ being defined as

$$\langle ij | ij \rangle = \int dx_1 \int dx_2 \psi_i^*(x_1) \psi_j^*(x_2) \frac{1}{r_{12}} \psi_i(x_1) \psi_j(x_2). \tag{3.16}$$

Note that ψ_i^* and ψ_i takes the same electron as input. Another notation frequently used in quantum chemistry is

$$\langle ij | ij \rangle = [ii|jj], \tag{3.17}$$

or in the case of general spin orbitals p, q, r, s as

$$\langle pq | rs \rangle = [pr|qs]. \tag{3.18}$$

The two-body interaction has several symmetries that we can utilize to improve the performance of our codes. One symmetry is given by the relation

$$\langle pq | rs \rangle = \langle qp | sr \rangle. \tag{3.19}$$

We will use real orbitals. This provides four more symmetries, namely

$$\langle pq | rs \rangle = \langle rq | ps \rangle = \langle ps | rq \rangle = \langle rs | pq \rangle. \tag{3.20}$$

These four symmetries can also be applied to Eq. (3.19) which means we have in total eight symmetries..

3.4 The Hartree Fock Equations

To find the lowest possible energy we must find the molecular orbitals that produce this energy. When finding a minima in such an equation we employ the method of Lagrangian multipliers. The method is described in detail in Ref. [56]. Here we will simply present the equations for our system, and give some brief arguments why these terms are present in the equation

$$\mathcal{L}[\{\psi_i\}] = E^{HF}[\{\psi_i\}] - \sum_{ij} \epsilon_{ij} \left(\langle i|j\rangle - \delta_{ij} \right). \tag{3.21}$$

Here $\mathcal{L}$ is a functional of the set of ψ_i. The aim is to find the minimum of this functional. The set of single-particle orbitals ψ_i will be varied in order to find this minimum. The condition $\langle i|j\rangle - \delta_{ij}$ is a constraint we impose to ensure the molecular spin orbitals remain orthonormal, even when we vary them. That is we require

$$\langle i|j\rangle = \delta_{ij}. \tag{3.22}$$

The quantity ϵ_{ij} are the undetermined Lagrange multipliers. The variation in our orbitals can be described as

$$\psi_i \to \psi_i + \delta\psi_i. \tag{3.23}$$

We want to find the minimum of the functional $\mathcal{L}$. This means its derivative must be equal to zero, that is

$$\delta\mathcal{L} = \delta E^{HF}[\{\psi_i\}] - \sum_{ij} \epsilon_{ij} \delta\langle i|j\rangle = 0. \tag{3.24}$$

We then insert Eq. (3.23) into the two terms in this equation, starting with the final term and obtain

$$\delta\langle i|j\rangle = \langle \delta i|j\rangle + \langle i|\delta j\rangle + \langle \delta i|\delta j\rangle. \tag{3.25}$$

where δi represent the variation of a single-particle orbital. With

$$\delta\langle i|j\rangle \approx \langle \delta i|j\rangle + \langle i|\delta j\rangle. \tag{3.26}$$

we find the variation in energy δE^{HF} as

$$
\begin{aligned}
\delta E^{HF} =& \sum_i \left(\langle \delta i|\mathbf{h}|i\rangle + \langle i|\mathbf{h}|\delta i\rangle\right) + \frac{1}{2}\sum_{ij}(\langle \delta ij|ij\rangle + \langle i\delta j|ij\rangle + \langle ij|\delta ij\rangle \\
&+ \langle ij|i\delta j\rangle - \langle \delta ij|ji\rangle - \langle i\delta j|ji\rangle - \langle ij|\delta ji\rangle - \langle ij|j\delta i\rangle) \\
=& \sum_i(\langle \delta i|\mathbf{h}|i\rangle + \langle i|\mathbf{h}|\delta i\rangle) \\
&+ \sum_{ij}(\langle \delta ij|ij\rangle + \langle i\delta j|ij\rangle - \langle \delta ij|ji\rangle - \langle i\delta j|ji\rangle). \qquad (3.27)
\end{aligned}
$$

Here we used the symmetries defined in Eq. (3.19). We insert this in Eq. (3.24) and get

$$
\begin{aligned}
0 =& \sum_i(\langle \delta i|\mathbf{h}|i\rangle + \langle i|\mathbf{h}|\delta i\rangle) - \sum_{ij}\epsilon_{ij}(\langle \delta i|j\rangle) + \langle i|\delta j\rangle) \\
&+ \sum_{ij}(\langle \delta ij|ij\rangle + \langle i\delta j|ij\rangle - \langle \delta ij|ji\rangle - \langle i\delta j|ji\rangle). \qquad (3.28)
\end{aligned}
$$

We now examine the term $\sum_{ij}\epsilon_{ij}\langle\psi_i|\delta\psi_j\rangle$. We will specifically take its complex conjugate twice, resulting in

$$
\sum_{ij}\epsilon_{ij}\langle i|\delta j\rangle = \left[\sum_{ij}\epsilon^*_{ij}\left(\langle i|\delta j\rangle\right)^*\right]^*. \qquad (3.29)
$$

The complex conjugate of the inner product interchanges the bra and the ket states. We insert this and then interchange the indeces i and j. We can make this interchange because we are summing over all possible indices i and j, resulting in

$$
\left[\sum_{ij}\epsilon^*_{ij}\left(\langle i|\delta j\rangle\right)^*\right]^* = \left[\sum_{ij}\epsilon^*_{ij}\langle \delta j|i\rangle\right]^* = \left[\sum_{ij}\epsilon^*_{ji}\langle \delta i|j\rangle\right]^*. \qquad (3.30)
$$

We will assume ϵ_{ij} is part of a hermitian matrix where $\epsilon^*_{ji} = \epsilon_{ij}$. We have then

$$
\left[\sum_{ij}\epsilon^*_{ji}\langle \delta i|j\rangle\right]^* = \left[\sum_{ij}\epsilon_{ij}\langle \delta i|j\rangle\right]^*. \qquad (3.31)
$$

The content inside the parenthesis is the same as the other term involving ϵ_{ij} in Eq. (3.28). We have just shown that the two terms are the complex conjugate of each other. This will hold true for all terms in Eq. (3.28). One term in the equation is the complex conjugate of another. We will mark this in our equation as $+c.c.$, where this represents the complex conjugate of every single term remaining in Eq. (3.28). We have then

$$0 = \sum_i \langle \delta i|\mathbf{h}|i\rangle - \sum_{ij} \epsilon_{ij}\langle \delta i|j\rangle) + \sum_{ij} (\langle \delta ij|ij\rangle - \langle \delta ij|ji\rangle) + c.c. \tag{3.32}$$

This equation can be rewritten using the definition of the inner product and drawing the sum over i and $\delta\psi_i^*(x_1)$ outside a parenthesis, resulting in

$$\begin{aligned} 0 = &\sum_i \int dx_1 \delta\psi_i^*(x_1)\Big[\mathbf{h}(x_1)\psi_i(x_1) + \sum_j \psi_i(x_1)\int dx_2 \frac{1}{r_{12}}\psi_j^*(x_2)\psi_j(x_2) \\ &- \sum_j \psi_j(x_1)\int dx_2 \frac{1}{r_{12}}\psi_j^*(x_2)\psi_i(x_2) - \sum_j \epsilon_{ij}\psi_j(x_1)\Big] + c.c. \end{aligned} \tag{3.33}$$

We should be able to insert any reasonable set of ψ_i into this equation and find a minimum of the Lagrangian. This means that the terms inside the bracket are the ones that should be zero. If the content of the brackets are zero, then the complex conjugate of this will also be zero. This may not hold if the content inside the brackets are purely imaginary. However we will not be dealing with such a situation.

We can thus put the content inside the bracket equal to zero, and set $\psi_j^*(x_2)\psi_j(x_2) = |\psi_j(x_2)|^2$, yielding

$$\begin{aligned} 0 = &\mathbf{h}(x_1)\psi_i(x_1) + \sum_j \psi_i(x_1)\int dx_2 \frac{1}{r_{12}}|\psi_j(x_2)|^2 \\ &- \sum_j \psi_j(x_1)\int dx_2 \frac{1}{r_{12}}\psi_j^*(x_2)\psi_i(x_2) - \sum_j \epsilon_{ij}\psi_j(x_1). \end{aligned} \tag{3.34}$$

We can rewrite the latter as

$$\begin{aligned} \sum_j \epsilon_{ij}\psi_j(x_1) = &\mathbf{h}(x_1)\psi_i(x_1) + \sum_j \left[\int dx_2 \frac{1}{r_{12}}|\psi_j(x_2)|^2\right]\psi_i(x_1) \\ &- \sum_j \left[\int dx_2 \frac{1}{r_{12}}\psi_j^*(x_2)\psi_i(x_2)\right]\psi_j(x_1). \end{aligned} \tag{3.35}$$

It is common to define two operators, **J** and **K**, to make this equation more compact. These operators are defined as

$$\mathbf{J}_j(x_1) \equiv \int dx_2 \frac{1}{r_{12}}|\psi_j(x_2)|^2. \tag{3.36}$$

and

$$\mathbf{K}_j(x_1)\psi_i(x_1) \equiv \left[\int dx_2 \frac{1}{r_{12}} \psi_j^*(x_2)\psi_i(x_2)\right] \psi_j(x_1). \tag{3.37}$$

Using these definitions in Eq. (3.35) results in

$$\sum_j \epsilon_{ij}\psi_j(x_1) = \left[\mathbf{h}(x_1) + \sum_j \mathbf{J}_j(x_1) - \sum_j \mathbf{K}_j(x_1)\right] \psi_i(x_1). \tag{3.38}$$

The content of the brackets on the right hand side of the equation will be defined as the Fock operator, namely

$$\mathbf{F}(x_1) \equiv \mathbf{h}(x_1) + \sum_j \mathbf{J}_j(x_1) - \sum_j \mathbf{K}_j(x_1). \tag{3.39}$$

Since we require that our single-particle orbitals should be orthonormal, ϵ is a diagonal matrix, that is $\epsilon_{ij} = \delta_{ij} \times \epsilon_i$. This means we can remove the sum over j. The only term to survive on the left hand side of Eq. (3.38) is thus given by $i = j$,

$$\mathbf{F}(x_1)\psi_i(x_1) = \epsilon_i\psi_i(x_1). \tag{3.40}$$

The term ϵ_i becomes the eigenvalues of the Fock operator. This means we have an eigenvalue problem. The operator $\mathbf{F}$ is defined in terms of ψ_i, but to find ψ_i we need the operator $\mathbf{F}$. This is a circular problem, which can be solved iteratively.

3.5 Restricted Hartree Fock

The Hartree-Fock-Roothan method, Ref. [91], is one way of solving the HF equations when the spin is restricted (that is all spin orbitals are occupied, resulting in a total spin and angular momentum equal to zero). What we do is to choose a basis set of predefined functions that will be our guess for the atomic orbitals, ϕ_μ. These will be discussed in the next chapter. For the present discussion we simply state that the basis functions we choose are usually not orthonormal. We want the atomic orbitals to define our molecular orbitals ψ_i,

$$\psi_i(x_1) = \sum_\mu^M C_{i\mu}\phi_\mu(x_1). \tag{3.41}$$

The lefthand side here is a molecular orbital, whereas the righthand side involves atomic orbitals ϕ. We have M such atomic orbitals. Inserting these into the Fock equation, Eq. (3.40), we arrive at

$$\mathbf{F}(x_1)\sum_{\mu}^{M} C_{i\mu}\phi_\mu(x_1) = \epsilon_i \sum_{\mu}^{M} C_{i\mu}\phi_\mu(x_1). \tag{3.42}$$

We then multiply by $\phi_v^*(x_1)$ and integrate both sides. We also pull the sum over μ and $C_{i\mu}$ outside the integral, resulting in

$$\sum_{\mu}^{M} C_{i\mu} \int dx_1 \phi_v^*(x_1)\mathbf{F}(x_1)\phi_\mu(x_1) = \epsilon_i \sum_{\mu}^{M} C_{i\mu} \int dx_1 \phi_v^*(x_1)\phi_\mu(x_1). \tag{3.43}$$

The righthand side is again not equal to $\delta_{v\mu}$ since the basis functions are usually not orthonormal. It can however be represented as a matrix element $C_{r\mu}S_{\mu v}$, where S is known as the overlap. The integral on the left handside is equivalent to a matrix element $F_{\mu v}$,

$$\sum_{\mu}^{M} F_{\mu v}C_{\mu i} = \epsilon_i \sum_{\mu}^{M} S_{\mu v}C_{\mu i}. \tag{3.44}$$

Here we have defined the matrix element $F_{\mu v}$ to be equal to

$$F_{\mu v}(x_1) = \int dx_1 \phi_v^*(x_1)\mathbf{F}(x_1)\phi_\mu(x_1), \tag{3.45}$$

and S is defined as

$$S_{\mu v} = \int dx_1 \phi_\mu^*(x_1)\phi_v(x_1). \tag{3.46}$$

On matrix form the equation becomes

$$FC = SC\epsilon, \tag{3.47}$$

where ϵ is a diagonal matrix. This equation is a matrix problem, and matrix problems are generally well suited to be handled on computers.

We should also mention briefly that ϵ_i physically comes to represent how much energy is required to remove an electron out of orbital i. The highest occupied molecular orbital (HOMO) will then be the energy required to remove the most loosely bound electron from say an atom. This defines the simplest possible approximation to the ionization energy, according to Koopmans Theorem', [6]. Koopmans Theorem' only works for spin restricted HF.

In Eq. (3.45), the quantity S becomes the overlap between basis functions, and does not change during iterations. The quantity C becomes a coefficient, and changes in each iteration. This applies to the Fock matrix elements as well.

Now we would like to make use of our spin restriction to simplify things even further. The quantity **F** was defined as

$$\mathbf{F}(x_1) = \mathbf{h}(x_1) + \sum_j^N \mathbf{J}_j(x_1) - \sum_j^N \mathbf{K}_j(x_1), \tag{3.48}$$

with **J** and **K** defined in Eqs. (3.36) and (3.37). Our molecular spin orbitals have two possible spin orientations, either spin up or spin down. We want to restrict spin so that total spin is zero. We also want specifically each spacial orbital to be occupied by one spin up and one spin down particle. This means in total that half the electrons will have spin up and the other half spin down.

We will use this spin restriction to simplify our operators **J** and **K**. If we look at the definition of **J** first,

$$\mathbf{J}_j(x_1) = \int dx_2 \frac{1}{r_{12}} |\psi_j(x_2)|^2. \tag{3.49}$$

we notice that **J** depends only on the spin orbital j. Orbital j obviously has the same spin orientation as itself. This means we can add a factor 2 in front of **J** in Eq. (3.48) and only sum over $\frac{N}{2}$, resulting in

$$\mathbf{K}_j(x_1)\psi_i(x_1) = \left[\int dx_2 \frac{1}{r_{12}} \psi_j^*(x_2)\psi_i(x_2)\right] \psi_j(x_1). \tag{3.50}$$

The quantity **K** depends however on orbitals i and j. If orbital i has its spin orientation defined, then the integral will be equal to zero whenever orbital j does not have the same spin orientation. This occurs half the time. We can still restrict the sum to only go over $\frac{N}{2}$ and add a factor 2, but must also add a $\frac{1}{2}$ for this reason. These two cancels out, and results in Eq. (3.48) for the spin restricted case to be equal to

$$\mathbf{F}(x_1) = \mathbf{h}(x_1) + 2\sum_j^{\frac{N}{2}} \mathbf{J}_j(x_1) - \sum_j^{\frac{N}{2}} \mathbf{K}_j(x_1). \tag{3.51}$$

We also insert the molecular orbitals as a linear combination of atomic orbitals in the matrix element $F_{\mu v}$, giving

$$F_{\mu v} = h_{\mu v} + \sum_j^{\frac{N}{2}} \sum_{rs}^{M} C_{rj} C_{sj}^* \left(2\langle \mu r|vs\rangle - \langle \mu s|vr\rangle\right), \tag{3.52}$$

with

$$h_{\mu v} = \int dx_1 \phi_\mu^*(x_1) \mathbf{h} \phi_v(x_1), \tag{3.53}$$

and

$$\langle \mu v | rs \rangle = \int dx_1 \int dx_2 \phi_\mu^*(x_1)\phi_v^*(x_2)\frac{1}{r_{12}}\phi_r(x_1)\phi_s(x_1). \tag{3.54}$$

The Fock matrix elements are now defined by atomic orbitals. To get implementation ready equations we must define these atomic orbitals, and solve the integrals using them. This will be done in the next chapter. We note the sum over $\frac{N}{2}$ and M, where N is the number of electrons and M is the number of basis functions.

3.6 Unrestricted Hartree Fock

The Hartree-Fock equations can also be solved without restricting each spin orbital to be occupied by two electrons. In this section we will derive the Pople-Sesbet equations, Ref. [2]. These equations achieves exactly this.

In Unrestricted Hartree-Fock (UHF) case we can define two sets of molecular spin orbitals. One set of occupied orbitals with spin up, $\{\psi_i^\alpha\}$, and another set of occupied orbitals with spin down, $\{\psi_i^\beta\}$. The total set of molecular spin orbitals contains both of these, that is

$$\{\psi_i\} = \begin{cases} \{\psi_j^\alpha\} \\ \{\psi_j^\beta\} \end{cases} \tag{3.55}$$

Inserted into Eq. (3.40), we obtain

$$\mathbf{F}^\alpha \psi_i^\alpha(x_1) = \epsilon_i^\alpha \psi_i^\alpha(x_1), \tag{3.56}$$

and

$$\mathbf{F}^\beta \psi_i^\beta(x_1) = \epsilon_i^\beta \psi_i^\beta(x_1). \tag{3.57}$$

We applied the spin restriction in the definition of $\mathbf{F}$. This expression will be different, otherwise our equations remain the same, that is

$$F^\alpha C^\alpha = SC^\alpha \epsilon^\alpha, \tag{3.58}$$

and

$$F^\beta C^\beta = SC^\beta \epsilon^\beta. \tag{3.59}$$

The Fock operator is defined in Eq. (3.48) and depends on $\mathbf{h}$, $\mathbf{J}$ and $\mathbf{K}$. We used our spin approximation in the expression for $\mathbf{J}$ and $\mathbf{K}$

$$\mathbf{J}_j(x_1) = \int dx_2 \frac{1}{r_{12}} |\psi_j(x_2)|^2. \tag{3.60}$$

The quantity $\mathbf{J}$ integrates over spin orbital j. This orbital can have spin up or spin down. This will be independent of the spin orientation of orbital i, meaning that we can split this expression in two terms, summing thereby over the occupied up spin orbitals and the occupied down spin orbitals separately.

The quantity $\mathbf{K}$ from Eq. (3.37) on the other hand involved spin orbitals i and j. This will follow the same argument as for the spin restricted case, resulting in the expression for the Fock matrix elements to be

$$F^{\alpha}_{\mu v} = h_{\mu v} + \sum_{j}^{N_\alpha} \sum_{rs}^{M} C^{\alpha}_{rj} \left(C^{\alpha}_{sj}\right)^{*} (\langle \mu r|vs\rangle - \mu r|sv\rangle) + \sum_{j}^{N_\beta} \sum_{rs}^{M} C^{\beta}_{rj} \left(C^{\beta}_{sj}\right)^{*} \langle \mu r|vs\rangle. \tag{3.61}$$

and

$$F^{\beta}_{\mu v} = h_{\mu v} + \sum_{j}^{N_\beta} \sum_{rs}^{M} C^{\beta}_{rj} \left(C^{\beta}_{sj}\right)^{*} (\langle \mu r|vs\rangle - \mu r|sv\rangle) + \sum_{j}^{N_\alpha} \sum_{rs}^{M} C^{\alpha}_{rj} \left(C^{\alpha}_{sj}\right)^{*} \langle \mu r|vs\rangle. \tag{3.62}$$

As in the restricted Hartree-Fock case we have the Fock matrix defined by atomic orbitals. Now it is time to define the atomic orbitals. This is the aim of the next chapter.

Chapter 4

Gaussian Type Orbitals

In 1950 Boys, [59], proposed the use of Gaussian Type Orbitals (GTOs) in electronic structure theory. Years after his proposal the use of Gaussian Type Orbitals are now standard in computational chemistry. In this chapter we will define what GTOs are and examine in detail how to construct them. We will also look at the mathematical expressions required for solving the integrals left open in the previous chapter. In total we will present all the required programmable equations for calculating energies with Hartree Fock. The content exposed here follows closely the work of T. Helgaker, P. Jorgensen and J. Olsen, [7, 8, 9].

The reader may also find additional and useful information in the articles by McMurcie and Davidson [10] and Pople and Hehre [11].

4.1 Contracted GTOs

Two of the basic ingredients in the formalism exposed here are the so-called contracted and primitive GTOs. A contracted GTO is used to describe an atomic orbital and is defined as

$$\phi(x,y,z) = \sum_i N_i \chi_i(x,y,z). \tag{4.1}$$

Here ϕ_i represents a contracted GTO, N_i is its normalization constant and χ_i is a primitive GTO. A primitive GTO is defined as

$$\chi_i(x,y,z) = c_i x^m y^n z^o e^{-\alpha_i R^2}, \tag{4.2}$$

where x, y and z are Cartesian coordinates and $R^2 = x^2 + y^2 + z^2$. These coordinates represent the distance to a given nucleus, while m, n and o depend on the angular momentum of the orbital we wish to describe. When the primitives are defined with $x^m y^n z^o$ they are called Cartesian Gaussian functions. We will only be dealing with these kind of Gaussians and m, n and o take only integer values, that is

$$m + n + o = l, \tag{4.3}$$

where l is the total angular momentum. The parameters α_i and c_i are variational parameters.

A contracted GTO is a linear combination of primitive GTOs. The goal of making contracted GTOs is to mimic the behaviour of a Slater Type Orbital (STO). An STO is considered to resemble the true atomic orbitals. We will see later that GTOs allow us to perform calculations much faster. For this reason we want to use GTOs, but we want the behaviour of an STO. An STO is defined as

$$\Phi(r) = NR^m e^{-\alpha R}, \tag{4.4}$$

where N is the normalization constant, R is the distance from the electron to the nucleus, m depends on angular momentum and α is again a variational parameter.

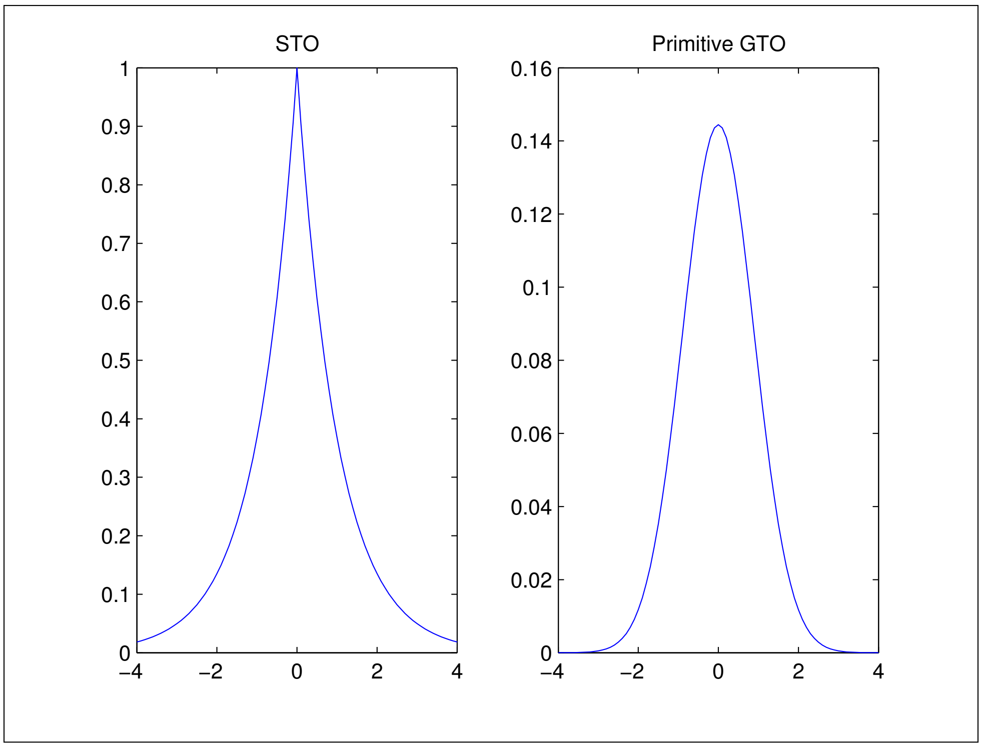


Figure 4.1: Illustration of the shape of a primitive GTO vs a STO

Figure 4.1 is an illustration of the shape of a primitive GTO side by side of an STO. We notice that they behave differently. We therefore make a linear combination of primitive GTOs, as is shown in figure 4.2.

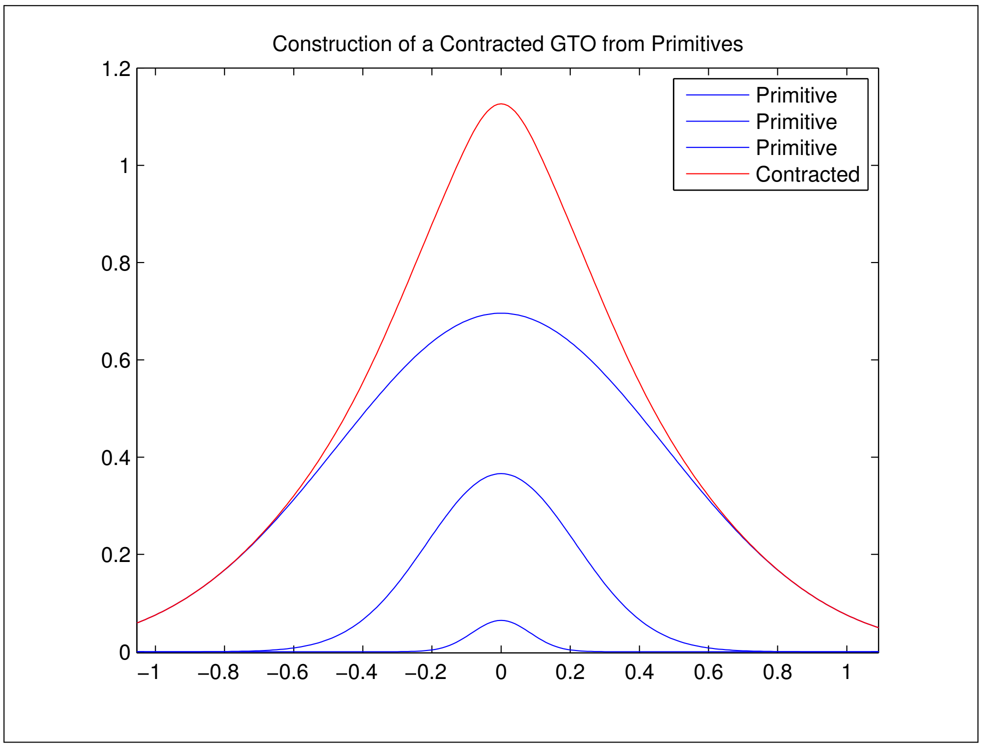


Figure 4.2: Illustration of the construction of a contracted GTO from three primitives

The contracted GTO is made up of three primitives. The problem with GTOs is that they fall off to quickly for increasing R compared to the STO. Increasing R is known as a long-range behaviour. Also when R goes to zero GTO and STO behave differently.

The problems of long and short range behaviour are reduced when going from a single primitive GTO to a contracted GTO of three primitives. In theory we can describe an STO with increasing accuracy by adding more primitive GTOs with different α_i. The parameters α_i control the width of the primitive GTO.

In figure 4.3 we have made a comparison of a contracted GTO made up of three primitives, versus a contracted GTO made up of six primitives. We see that the more primitives the better our GTO becomes relative to the

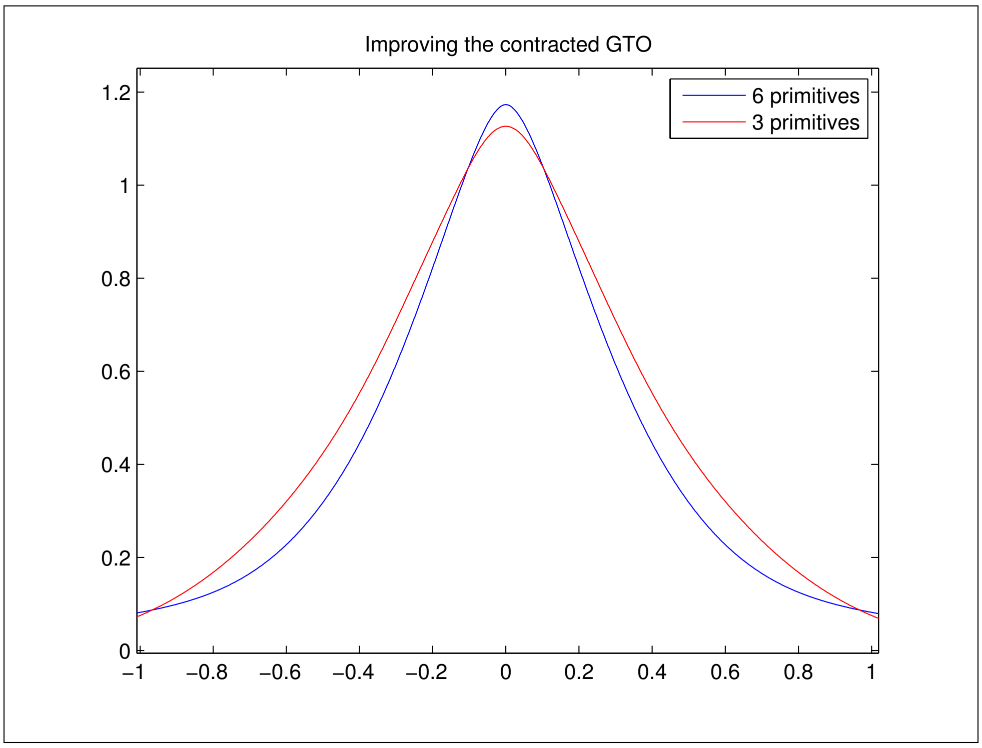


Figure 4.3: Illustration of how an increased number of primitives improve our contracted GTO

original STO it is meant to represent. As mentioned previously, an STO is considered to behave like a true atomic orbital (AO). We therefore have a good argument for using GTOs to describe AOs. However, this requires that we know how to get the right primitives.

4.2 Variational Principle

Constructing good primitives means defining α_i and c_i. These are variational parameters, and in theory we can use the variational principle. The variational principle states that

$$E_0 \leq \frac{\int \langle \psi_T | \mathbf{H} | \psi_T \rangle}{\int \langle \psi_T | \psi_T \rangle}. \tag{4.5}$$

This means that we can optimize our variational parameters, by minimizing the energy as a function of α_i and c_i. There is a huge computational cost attached to this, since the number of variational parameters scales quickly with increasing number of primitives.

4.3 EMSL

The software library EMSL [13, 14, 15] provides already calculated α and c. We will make use of these pre-computed parameters in our calculations.

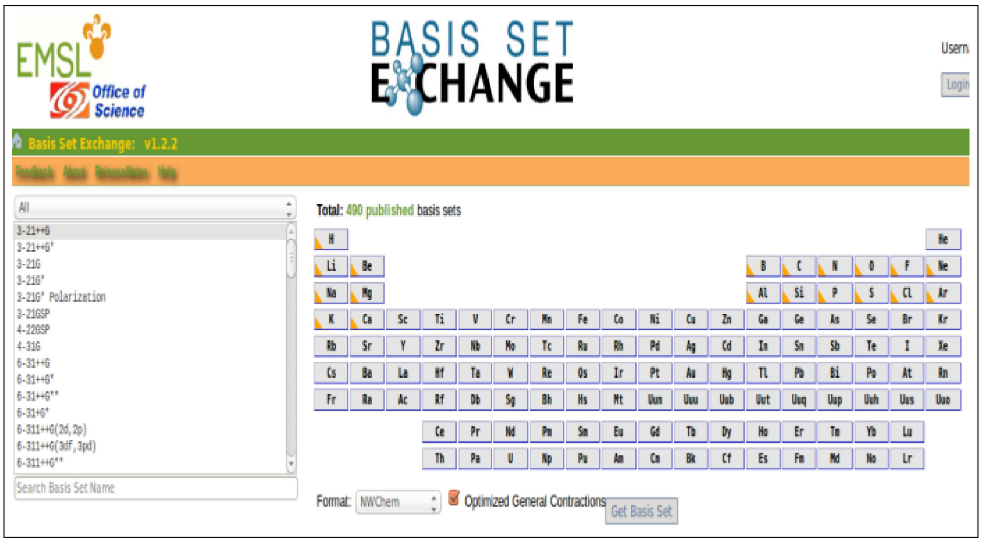


Figure 4.4: Front page of the EMSL website

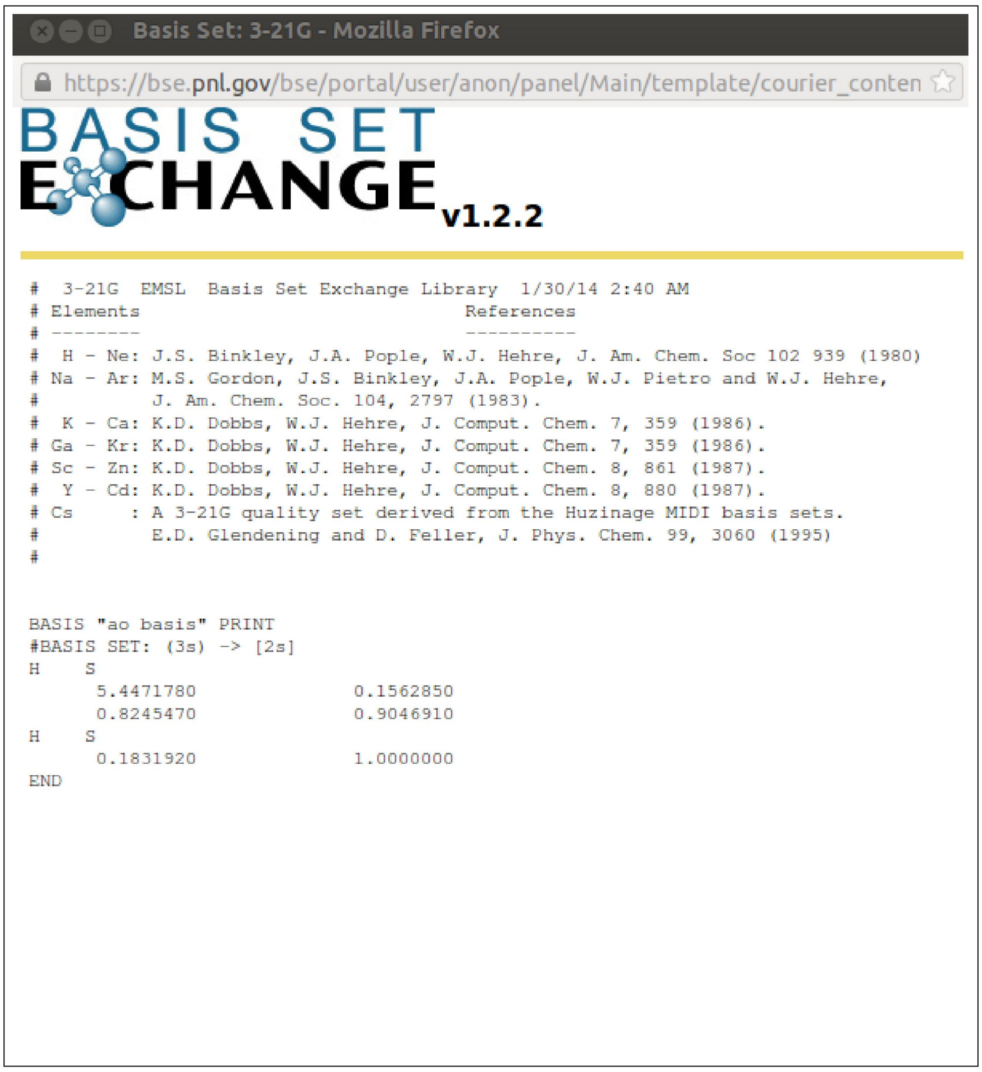
When entering the basis set exchange we must select two options. First what basis set, listed on the left in figure 4.4. Secondly, which atom(s) we will study. What the different basis sets represent will be explained later, but for now let us examine how to read data from EMSL. If we click on the 3-21G basis set and Hydrogen and then "Get Basis Set" we will get the basis set seen in figure 4.5.

The first line of interest is

BASIS "ao basis" PRINT

which means that this is a basis for atomic orbitals. Two lines down we see two letters, H and S. H means this is a basis for the hydrogen atom. S means this is an S orbital, which means that the angular momentum is 0 for all primitive GTOs that define this orbital. Angular momentum of 0 means that m, n and o are zero in Eq. (4.2).

The next two lines are filled with four numbers. Each line contains one α value (left number) and one c value (right value). Inserting these four numbers in Eq. (4.2) we obtain our first two primitive GTOs


Basis Set: 3-21G - Mozilla Firefox
https://bse.pnl.gov/bse/portal/user/anon/panel/Main/template/courier_conten
BASIS SET EXCHANGE v1.2.2

```
#  3-21G  EMSL  Basis Set Exchange Library  1/30/14 2:40 AM
# Elements                             References
# --------                             ----------
#  H - Ne: J.S. Binkley, J.A. Pople, W.J. Hehre, J. Am. Chem. Soc 102 939 (1980)
# Na - Ar: M.S. Gordon, J.S. Binkley, J.A. Pople, W.J. Pietro and W.J. Hehre,
#          J. Am. Chem. Soc. 104, 2797 (1983).
#  K - Ca: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 7, 359 (1986).
# Ga - Kr: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 7, 359 (1986).
# Sc - Zn: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 8, 861 (1987).
#  Y - Cd: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 8, 880 (1987).
# Cs     : A 3-21G quality set derived from the Huzinage MIDI basis sets.
#          E.D. Glendening and D. Feller, J. Phys. Chem. 99, 3060 (1995)
#


BASIS "ao basis" PRINT
#BASIS SET: (3s) -> [2s]
H    S
      5.4471780              0.1562850
      0.8245470              0.9046910
H    S
      0.1831920              1.0000000
END
```

Figure 4.5: The 3-21G basis set for Hydrogen

$$\chi_1(x,y,z) = 0.1562850 \times exp(-5.4471780 \times (r - R_H)^2), \tag{4.6}$$

and

$$\chi_2(x,y,z) = 0.9046910 \times exp(-0.8245470 \times (r - R_H)^2). \tag{4.7}$$

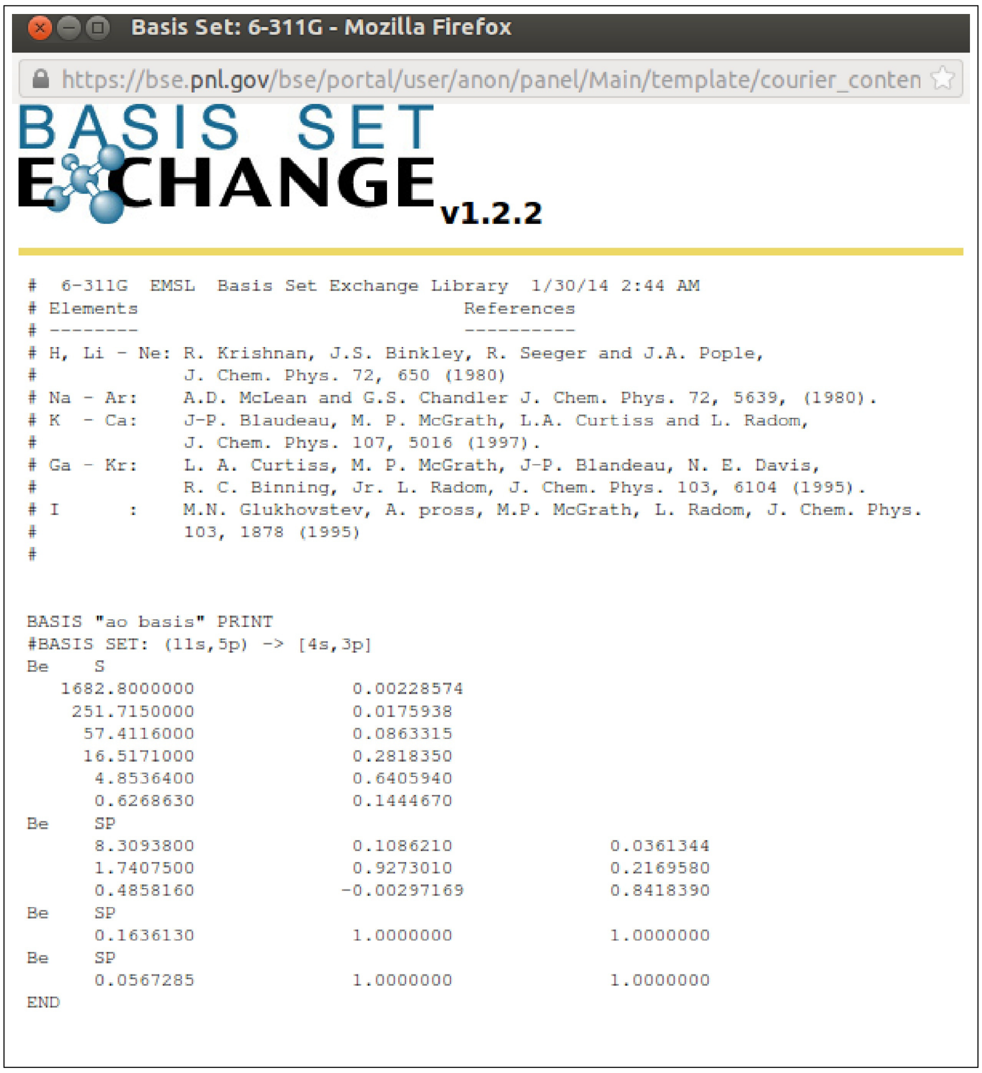
These can be combined to a contracted GTO which will represent the first atomic orbital.

$$\phi_1(x,y,z) = N_1\chi_1 + N_2\chi_2. \tag{4.8}$$

The next line represents another atomic orbital with angular momentum zero. However this contains only one primitive GTO,

$$\phi_2(x, y, z) = N_1 \times 0.1562850 \times exp(-5.4471780 \times (r - R_H)^2). \qquad (4.9)$$

It may seem confusing why a hydrogen atom with only one electron would need two atomic orbitals. This will be explained in a later section. Further notation from EMSL can be noted in figure 4.6.


Basis Set: 6-311G - Mozilla Firefox
https://bse.pnl.gov/bse/portal/user/anon/panel/Main/template/courier_conten
BASIS SET EXCHANGE v1.2.2

```
#  6-311G  EMSL  Basis Set Exchange Library  1/30/14 2:44 AM
# Elements                             References
# --------                             ----------
# H, Li - Ne: R. Krishnan, J.S. Binkley, R. Seeger and J.A. Pople,
#             J. Chem. Phys. 72, 650 (1980)
# Na - Ar:    A.D. McLean and G.S. Chandler J. Chem. Phys. 72, 5639, (1980).
# K  - Ca:    J-P. Blaudeau, M. P. McGrath, L.A. Curtiss and L. Radom,
#             J. Chem. Phys. 107, 5016 (1997).
# Ga - Kr:    L. A. Curtiss, M. P. McGrath, J-P. Blandeau, N. E. Davis,
#             R. C. Binning, Jr. L. Radom, J. Chem. Phys. 103, 6104 (1995).
# I      :    M.N. Glukhovstev, A. pross, M.P. McGrath, L. Radom, J. Chem. Phys.
#             103, 1878 (1995)
#


BASIS "ao basis" PRINT
#BASIS SET: (11s,5p) -> [4s,3p]
Be    S
   1682.8000000           0.00228574
    251.7150000           0.0175938
     57.4116000           0.0863315
     16.5171000           0.2818350
      4.8536400           0.6405940
      0.6268630           0.1444670
Be    SP
      8.3093800           0.1086210           0.0361344
      1.7407500           0.9273010           0.2169580
      0.4858160          -0.00297169          0.8418390
Be    SP
      0.1636130           1.0000000           1.0000000
Be    SP
      0.0567285           1.0000000           1.0000000
END
```

Figure 4.6: The 6-311G basis set for Beryllium

Figure 4.6 contains the data for Be from the 6-311G basis set. The first atomic orbital is an S orbital with angular momentum zero and a contracted

GTO of six primitive GTOs. These six primitives together actually construct the blue line plotted in figure 4.3.

The next orbital is marked as SP. This is a short notation for one S orbital and one P. The notation means that the left column represents the α values for both orbital S and P. The second column are c values for the S orbital, whereas the third column are c values for the P orbital. In this basis set the S and P orbital share α values. This is a common feature. The basis set is designed like this to allow for a more efficient implementation.

However, for now our interest is the notation on the EMSL website. The P orbital represents an angular momentum of 1, which means $m+n+o=1$. This can be achieved by either $m=1$, $n=1$ or $o=1$.

When we make our basis set all of these possibilities must be available. This means a P orbital is really 3 atomic orbitals, but all 3 orbitals have the same α_i and c_i. One of them will have $m=1$, $n=0$ and $o=0$. The second will have $m=0$, $n=1$ and $o=0$. The third will have $m=0$, $n=0$ and $o=1$. The 6-311G basis set therefore has a total of 13 orbitals for an atom like beryllium.

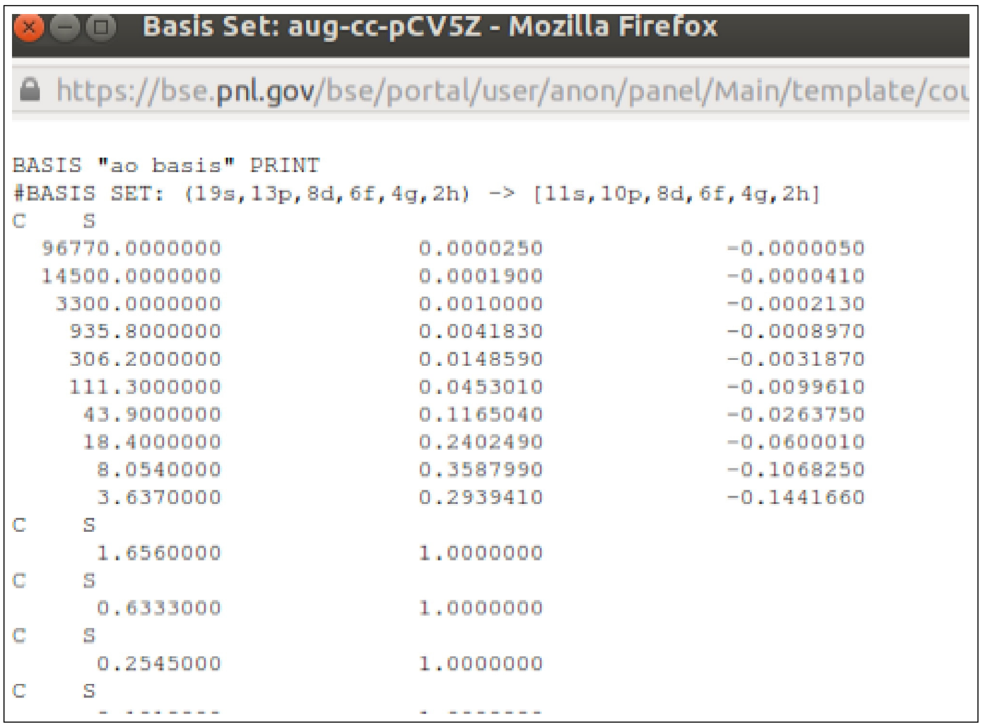
Different letters represent different angular momentum on EMSL. This means the different letters also represent a different number of atomic orbitals, as indicated in table here.

Letter	Ang mom	Nr of Orbitals
S	0	1
P	1	3
D	2	6
F	3	10
G	4	15
H	5	21

The number of orbitals is increasing because of the different ways to arrange m, n and o to achieve the given angular momentum. The general number of orbitals for an angular momentum l is $\frac{(l+1)(l+2)}{2}$. The next table represent the different ways of organizing m, n and o for the D orbital.

m	n	o
2	0	0
0	2	0
0	0	2
1	0	1
1	1	0
0	1	1

The same method can be applied to F, G, H and higher orbitals.


Basis Set: aug-cc-pCV5Z - Mozilla Firefox
https://bse.pnl.gov/bse/portal/user/anon/panel/Main/template/cou

```
BASIS "ao basis" PRINT
#BASIS SET: (19s,13p,8d,6f,4g,2h) -> [11s,10p,8d,6f,4g,2h]
C    S
  96770.0000000            0.0000250            -0.0000050
  14500.0000000            0.0001900            -0.0000410
   3300.0000000            0.0010000            -0.0002130
    935.8000000            0.0041830            -0.0008970
    306.2000000            0.0148590            -0.0031870
    111.3000000            0.0453010            -0.0099610
     43.9000000            0.1165040            -0.0263750
     18.4000000            0.2402490            -0.0600010
      8.0540000            0.3587990            -0.1068250
      3.6370000            0.2939410            -0.1441660
C    S
      1.6560000            1.0000000
C    S
      0.6333000            1.0000000
C    S
      0.2545000            1.0000000
C    S
```

Figure 4.7: Illustration of another possible EMSL notation

Figure 4.7 shows part of the aug-cc-pCV5Z basis set for carbon. The interesting part is that we now have three columns but only an S orbital represented. When this notation occurs it means there are two S orbitals with identical α values but different c values. Some basis sets lists seven or eight columns, but the same principle applies. The first column stands for the α values. The next one represents the c values, where each column contains c values for its own orbital.

4.4 Product of Gaussians

In this section we will derive the normalization constant. The normalization constant is defined such that the inner product is equal to 1, that is

$$|\langle\phi_i|\phi_i\rangle|^2 = 1, \tag{4.10}$$

with

$$\phi(x, y, z) = \sum_i N_i \chi_i(x, y, z), \tag{4.11}$$

where χ_i are the primitive GTOs. This means we can calculate the integral over contracted GTOs by first calculating the integral over two primitive GTOs

$$\chi_1 = c_1 x_A^i y_A^j z_A^k exp(-\alpha_1 \mathbf{r}_A^2), \tag{4.12}$$

and

$$\chi_2 = c_2 x_B^m y_B^n z_B^o exp(-\alpha_2 \mathbf{r}_B^2). \tag{4.13}$$

Here $\mathbf{r}_A = \mathbf{r} - \mathbf{A}$ and $\mathbf{A}$ is the position of nucleus A. The primitive χ_1 is part of a contracted GTO that describes an atomic orbital in nucleus A. Similar relations apply for $\mathbf{r}_B$. The product of χ_1 and χ_2 is well defined,

$$\chi_1\chi_2 = c_1 c_2 x_A^i x_B^m y_A^j y_B^n z_A^k z_B^o exp(-(\alpha_1 \mathbf{r}_A^2 + \alpha_2 \mathbf{r}_B^2)). \tag{4.14}$$

A key feature of Gaussian functions is that a product of two is equal to another Gaussian. This is shown by finding the "charge center" P

$$\mathbf{P} = \frac{\alpha_1 \mathbf{A} + \alpha_2 \mathbf{B}}{\alpha_1 + \alpha_2}. \tag{4.15}$$

Defining $\mathbf{r}_P = r - P$ we can rewrite the exponential term as

$$exp(-(\alpha_1 \mathbf{r}_A^2 + \alpha_2 \mathbf{r}_B^2)) = G_{IJ} exp(-\alpha_p \mathbf{r}_P), \tag{4.16}$$

where

$$G_{IJ} = exp(-\alpha_1 \alpha_2 \alpha_p^{-1} |\mathbf{A} - \mathbf{B}|^2), \tag{4.17}$$

is independent of r. We now realise that the product $\chi_1\chi_2$ can be separated in its three Cartesian coordinates, x, y and z with $\mathbf{r}_P = (x_P, y_P, z_P)$. This results in

$$\chi_1\chi_2 = G_{IJ} \chi_X \chi_Y \chi_Z, \tag{4.18}$$

where

$$\chi_X = x_A^i x_B^m exp(-\alpha_p x_P^2), \tag{4.19}$$

and similar for χ_Y and χ_Z. We now define

$$\Lambda_j(x_p, \alpha_p)exp(-\alpha_p x_p^2) = (\frac{\partial}{\partial P_x})^j exp(-\alpha_p x_p^2). \tag{4.20}$$

This relates to a Hermite polynomial H_j as such

$$\Lambda_j(x_p, \alpha_p) = \alpha_p^{j/2} H_j(\alpha_p^{1/2} x_P). \tag{4.21}$$

The purpose of this definition is to replace $x_A^i x_B^m$ with derivatives of P_x which can be placed outside an integral.

We wish to expand $x_A^i x_B^m$ as such:

$$x_A^i x_B^m = \sum_{N=0}^{i+m} E_N^{i,m} \Lambda(x_P, \alpha_p). \tag{4.22}$$

The first 5 Hermite polynomials are (see also [12]):

$$H_0(x) = 1. \tag{4.23}$$

$$H_1(x) = 2x. \tag{4.24}$$

$$H_2(x) = 4x^2 - 2. \tag{4.25}$$

$$H_3(x) = 8x^3 - 12x. \tag{4.26}$$

$$H_4(x) = 16x^4 - 48x^2 + 12. \tag{4.27}$$

We notice from these that the following recursion relation holds:

$$H_{N+1}(x) = 2xH_N(x) - 2NH_{N-1}(x). \tag{4.28}$$

The latter results in another recursion relation:

$$x_A \Lambda_N(x_P, \alpha_p) = N\Lambda_{N-1} + (P_x - A_x)\Lambda_N + \frac{1}{2\alpha_p}\Lambda_{N+1}. \tag{4.29}$$

We can combine Eq. (4.22) with Eq. (4.29) and find the following recursion relations for $E_N^{i,m}$.

$$E_N^{i+1,m} = \frac{1}{2\alpha_p} E_{N-1}^{i,m} + (P_x - A_x)E_N^{i,m} + (N+1)E_{N+1}^{i,m}, \tag{4.30}$$

and

$$E_N^{i,m+1} = \frac{1}{2\alpha_p} E_{N-1}^{i,m} + (P_x - B_x) E_N^{i,m} + (N+1) E_{N+1}^{i,m}, \tag{4.31}$$

where $E_0^{0,0} = 1$ and $E_N^{i,m} = 0$ for N $>$ i+m and N $<$ 0. We can use similar recursive relations for $y_A^j y_B^n$ and $z_A^k z_B^o$. We have

$$y_A^j y_B^n = \sum_{N=0}^{j+n} E_N^{j,n} \Lambda_N(y_p, \alpha_p), \tag{4.32}$$

and

$$z_A^k z_B^o = \sum_{N=0}^{k+o} E_N^{k,o} \Lambda_N(z_p, \alpha_p). \tag{4.33}$$

These equations are then used to rewrite Eq. (4.18) as

$$\chi_1 \chi_2 = G_{IJ} \sum_{N,M,L} E_N^{i,m} E_L^{j,n} E_M^{k,o} \Lambda_N(x_P) \Lambda_L(y_P) \Lambda_M(z_P) exp(-\alpha_p \mathbf{r}_p^2). \tag{4.34}$$

The integral over $\chi_1\chi_2$ can be calculated easily. We will label this integral $S_{1,2}$,

$$S_{1,2} = \int dr \chi_1 \chi_2 = \int dr G_{IJ} \chi_X \chi_Y \chi_Z = G_{IJ} S_{i,m} S_{j,n} S_{k,o}. \tag{4.35}$$

A general integral over an Hermitian Gaussian function is given as

$$\int dx \Lambda_N(x_P, \alpha_p) exp(-\alpha x_P^2) = \delta_{N,0} \left(\frac{\pi}{\alpha_p}\right)^{1/2}. \tag{4.36}$$

Integrating over the x, y and z coordinates separately and placing the derivatives outside the integral results in the following relations

$$S_{i,m} = \int dx \phi_X = \sum_{N=0}^{i+m} E_N^{i,m} \int \Lambda_N(x_p) dx, \tag{4.37}$$

and

$$S_{i,m} = \sum_{N=0}^{i+m} E_N^{i,m} \delta_{N,0} \left(\frac{\pi}{\alpha_p}\right)^{1/2}. \tag{4.38}$$

Here only one term will survive from this sum. When N = 0, we have

$$S_{i,m} = E_0^{i,m} \left(\frac{\pi}{\alpha_p}\right)^{1/2}. \tag{4.39}$$

We get similar results for $S_{j,n}$ and $S_{k,o}$, namely

$$S_{1,2} = G_{IJ} E_0^{i,m} E_0^{j,n} E_0^{k,o} \left(\frac{\pi}{\alpha_p} \right)^{3/2}. \tag{4.40}$$

It is very common to insert G_{IJ} into $E_0^{0,0}$. For the x coordinate we will get

$$E_0^{0,0} = G_{IJ,x} = exp\left(-\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |A_x - B_x|^2 \right), \tag{4.41}$$

with similar results for the y and z coordinates. Collecting all our results we have

$$S_{1,2} = E_0^{i,m} E_0^{j,n} E_0^{k,o} \left(\frac{\pi}{\alpha_p} \right)^{3/2}, \tag{4.42}$$

where we to repeat the use of these relations for E:

$$E_N^{i+1,m} = \frac{1}{2\alpha_p} E_{N-1}^{i,m} + (P_x - A_x) E_N^{i,m} + (N+1) E_{N+1}^{i,m}, \tag{4.43}$$

and

$$E_N^{i,m+1} = \frac{1}{2\alpha_p} E_{N-1}^{i,m} + (P_x - B_x) E_N^{i,m} + (N+1) E_{N+1}^{i,m}. \tag{4.44}$$

4.5 Normalization

The quantity S is known as the overlap between primitive GTOs. This means we have to calculate the inner product as

$$|\langle \phi_i | \phi_i \rangle|^2 = 1, \tag{4.45}$$

with

$$\phi_i = \sum_j N_j \chi_j. \tag{4.46}$$

We can now calculate the normalization constants for orbital ϕ_i. We have the same orbital on the bra and ket sides, meaning in this inner product the coordinates $\mathbf{A}$ and $\mathbf{B}$ will be identical, as well as c_i, α_i and the angular momentum for the primitives. With $\mathbf{A} = \mathbf{B}$ we get $E_0^{0,0} = 1$ and $\alpha_p = 2\alpha_i$.

We can now calculate the normalization constant for $\langle \chi_j | \chi_j \rangle$ for different angular momenta.

4.5.1 $\mathbf{l = 0}$

For $l = 0$ all primitive GTOs are

$$\chi_j = c_j exp(-\alpha_j \mathbf{r}^2). \tag{4.47}$$

The normalization constant can be calculated as

$$1 = N^2 c^2 S = N_j^2 c_j^2 \left(E_0^{0,0}\right)^3 \left(\frac{\pi}{2\alpha_j}\right)^{3/2}. \tag{4.48}$$

This results in

$$\Rightarrow N_{0,0,0} c = \left(\frac{2\alpha_j}{\pi}\right)^{3/4}. \tag{4.49}$$

Here the notation $N_{0,0,0}$ means it is the normalization constant for primitives with $m = 0$, $n = 0$ and $o = 0$.

4.5.2 $\mathbf{l = 1}$

For $l = 1$ all primitive GTOs will have one $E_0^{1,1}$ term and two $E_0^{0,0} = 1$ terms. We must now use the recursive relations to find the $E_0^{1,1}$ term. We have

$$E_0^{1,1} = 0 + 0 + E_1^{1,0} = \frac{1}{4\alpha_j}, \tag{4.50}$$

which results in a normalization constant

$$1 = N_j^2 c_j^2 4\alpha_j \left(\frac{\pi}{2\alpha_j}\right)^{3/2} \tag{4.51}$$

yielding

$$\Rightarrow N_{1,0,0} c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4} \sqrt{4\alpha_j}. \tag{4.52}$$

Here the notation $N_{1,0,0}$ means primitives with $(m, n, o) = (1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$.

4.5.3 $\mathbf{l = 2}$

For $l = 2$ we will have two possibilities. Either $S_{ij} = E_0^{2,2} E_0^{0,0} E_0^{0,0} \left(\frac{\pi}{2\alpha_j}\right)^{3/2}$, or $S_{ij} = E_0^{1,1} E_0^{1,1} E_0^{0,0} \left(\frac{\pi}{2\alpha_j}\right)^{3/2}$. This results in two different normalization constants for primitives with $l = 2$. The first case is

$$1 = N_j^2 c_j^2 \left(\frac{\pi}{2\alpha_j}\right)^{3/2} E_0^{2,2} E_0^{0,0} E_0^{0,0}, \tag{4.53}$$

with $E_0^{2,2}$

$$E_0^{2,2} = E_1^{1,2} = \frac{1}{4\alpha_j} E_0^{0,0} + 2E_2^{0,2} = \frac{3}{4\alpha_j} E_1^{0,1} = \frac{3}{16\alpha_j^2} \tag{4.54}$$

giving

$$\Rightarrow N_{2,0,0} c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4} \sqrt{\frac{16\alpha_j^2}{3}}. \tag{4.55}$$

The second case is

$$1 = N_j^2 c_j^2 \left(\frac{\pi}{2\alpha_j}\right)^{3/2} E_0^{1,1} E_0^{1,1} E_0^{0,0}. \tag{4.56}$$

where

$$E_0^{1,1} = \frac{1}{4\alpha_j} \tag{4.57}$$

with

$$\Rightarrow N_{1,1,0} c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4} \sqrt{16\alpha_j^2}. \tag{4.58}$$

4.5.4 l = 3

For $l = 3$ we have three possibilities, $N_{3,0,0}$, $N_{2,1,0}$ and $N_{1,1,1}$. First,$S_{ij} = E_0^{3,3} E_0^{0,0} E_0^{0,0} \left(\frac{\pi}{2\alpha_j}\right)^{3/2}$ with

$$E_0^{3,3} = E_1^{2,3} = \frac{1}{4\alpha_j} E_0^{1,3} + 2E_2^{1,3} = \frac{3}{4\alpha_j} E_1^{0,3} + 4E_2^{0,3} \tag{4.59}$$

$$\Rightarrow E_0^{3,3} = \frac{3}{16\alpha_j^2} E_0^{0,2} + \frac{6}{4\alpha_j} E_2^{0,2} + \frac{4}{4\alpha_j} E_1^{0,2} \tag{4.60}$$

$$\Rightarrow E_0^{3,3} = \frac{3}{16\alpha_j^2} E_1^{0,1} + \frac{6}{16\alpha_j^2} E_1^{0,1} + \frac{4}{16\alpha_j^2} E_0^{0,1} \tag{4.61}$$

$$\Rightarrow E_0^{3,3} = \frac{9}{64\alpha_j^3} \tag{4.62}$$

which gives

$$\Rightarrow N_{3,0,0}c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4}\sqrt{\frac{64\alpha_j^3}{9}}. \tag{4.63}$$

The second case is $S_{ij} = E_0^{2,2}E_0^{1,1}E_0^{0,0}\left(\frac{\pi}{2\alpha_j}\right)^{3/2}$, with

$$E_0^{2,2} = \frac{3}{16\alpha_j^2}. \tag{4.64}$$

and

$$E_0^{1,1} = \frac{1}{4\alpha_j}. \tag{4.65}$$

The result is

$$\Rightarrow N_{2,1,0}c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4}\sqrt{\frac{64\alpha_j^3}{3}} \tag{4.66}$$

Finally, $S_{ij} = E_0^{1,1}E_0^{1,1}E_0^{1,1}\left(\frac{\pi}{2\alpha_j}\right)^{3/2}$ with

$$E_0^{1,1} = \frac{1}{4\alpha_j} \tag{4.67}$$

resulting in

$$\Rightarrow N_{1,1,1}c_j = \left(\frac{2\alpha_j}{\pi}\right)^{3/4}\sqrt{64\alpha_j^3}. \tag{4.68}$$

4.5.5 Final normalization comments

We placed all the normalization constants next to the parameter c, and all the normalization constants are left as a function of α. This means in practice we can simply multiply this normalization in with the parameter c, and combine them.

The derivation of normalization factors for $l = 4, 5, \ldots$ are performed similarly.

4.6 Calculating Integrals for Hartree Fock

There were four integrals left untouched in the previous Hartree Fock chapter. All of these integrals involved AOs, which we will be approximating as GTOs. In this section we provide the analytical formula for solving these integrals, using GTOs. The solutions will provide insights into why GTOs are so popular in quantum chemistry.

4.6.1 Overlap

The overlap integrals are already solved during our quest to find the normalization constants. They are given as

$$S_{pq} = \sum_{I}^{p}\sum_{J}^{q} c_I c_J N_I N_J E_0^{i,m} E_0^{j,n} E_0^{k,o} \left(\frac{\pi}{\alpha_p}\right)^{3/2}, \tag{4.69}$$

with $\alpha_p = \alpha_I + \alpha_J$ and the sum over I and J sums over the primitives that define the contracted GTOs, p and q.

4.6.2 Kinetic Energy

The single-particle operator $\mathbf{h}$, as seen in Eq. (3.3), contained two terms. The kinetic energy part of this operator is calculated from the integral

$$-\frac{1}{2}\langle\phi_p|\nabla^2|\phi_r\rangle. \tag{4.70}$$

We again insert the primitives χ, and sum over them later, such that we need to calculate

$$-\frac{1}{2}\langle\chi_a|\nabla^2|\chi_b\rangle. \tag{4.71}$$

∇^2 acts on the right GTO, which can be split into its x, y and z components.

$$\chi_b = x^m y^n z^o e^{\alpha_b R^2} = \chi_{b,x}\chi_{b,y}\chi_{b,z}, \tag{4.72}$$

with

$$\chi_{b,x} = x^m e^{\alpha_b x^2}, \tag{4.73}$$

and similar for $\chi_{j,y}$ and $\chi_{j,z}$. We also define

$$\chi_a = x^i y^j z^k e^{\alpha_a R^2}. \tag{4.74}$$

Here ∇ can also be split into x, y and z components, with the mathematics of all three components being similar. We therefore just look at the x component

$$\begin{aligned}\nabla_x^2\chi_{b,x} =& \nabla_x^2\left[x^m e^{\alpha_b x^2}\right]\\ =& 4\alpha_b^2 x^{m+2}e^{-\alpha_b x^2} - 2\alpha_b(2m+1)x^m e^{-\alpha_b x^2}\\ &+ m(m-1)x^{m-2}e^{-\alpha_b x^2}.\end{aligned} \tag{4.75}$$

Here we have taken the derivative. We notice all the terms have $e^{-\alpha_b x^2}$ present, and a few constant terms. Also there is a change in the power of x. We can now insert the results from section 4.4, resulting in

$$\langle \chi_{a,x}|\nabla_x^2|\chi_{b,x}\rangle = 4\alpha_b^2 S_{i,m+2} - 2\alpha_b(2m+1)S_{i,m} + m(m-1)S_{i,m-2}. \quad (4.76)$$

For the y direction we will have the same result, except the index i will be replaced by j, and m will be replaced by n. For the z coordinate i will be replaced by k and m will be replaced by o. We notice the kinetic energy is simply a linear combination of quantities already calculated in the overlap. This will be used in the implementation.

4.6.3 Nuclei-Electron interaction

The second piece of the single particle operator, **h**, was a nuclei-electron interaction term. The integral to solve here is

$$\langle \phi_p| \sum_A \frac{Z_A}{r_{iA}} |\phi_r\rangle. \quad (4.77)$$

The sum over A and Z_A may be placed outside the integral. We can also insert the primitives and solve the integral based on them, and sum over all primitives later. We rename the distance from the electron to the nuclei r_C. This leaves

$$\langle \chi_a|\frac{1}{r_C}|\chi_b\rangle = \int dr_1 \chi_a^*(r_1)\frac{1}{r_C}\chi_b(r_1), \quad (4.78)$$

where $r_C = |r_1 - R_A|$. We first multiply χ_a^* with χ_b to get Ω_{ab}. This enables us to use results from section 4.4. This gives

$$\langle \chi_a|\frac{1}{r_C}|\chi_b\rangle = \int dr_1 \frac{\Omega_{ab}(r)}{r_C}. \quad (4.79)$$

We noted above that Ω_{ab} could be split into its x, y and z components

$$\Omega_{ab}(r) = \Omega_{im}(x)\Omega_{jn}(y)\Omega_{ko}(z), \quad (4.80)$$

with the results from section 4.4 staying the same, namely

$$\Omega_{im}(x) = \sum_{t=0}^{i+m} E_t^{i,m}\Lambda_t(x_P). \quad (4.81)$$

Combining the x, y and z directions we can write this as

$$\Omega_{ab}(r) = \sum_{tuv} E_{tuv}^{ab}\Lambda_{tuv}(r_p), \quad (4.82)$$

where $\Lambda_{tuv}(r_p)$ is defined as

$$\Lambda_{tuv}(r_p) = \frac{\partial^{t+u+v}}{\partial P_x^t \partial P_y^u \partial P_z^u} exp(-\alpha_p r_p^2), \quad (4.83)$$

as defined in section 4.4. We insert this into Eq. (4.79) and obtain

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \int dr_1 \frac{\sum_{tuv} E^{ab}_{tuv}\Lambda_{tuv}(r_p)}{r_C}. \tag{4.84}$$

We can pull the sum and E^{ab}_{tuv} outside the integral and get

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \sum_{tuv} E^{ab}_{tuv} \int dr_1 \frac{\Lambda_{tuv}(r_p)}{r_C}. \tag{4.85}$$

The next step is a substitution. We want to avoid having r_C in our integral and use

$$\frac{1}{r_C} = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} exp\left(-r_C^2 t^2\right) dt. \tag{4.86}$$

This is inserted into Eq. (4.85), alongside the definition of $\Lambda_{tuv}(r_p)$, and gives us

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \frac{1}{\sqrt{\pi}} \sum_{tuv} E^{ab}_{tuv} \frac{\partial^{t+u+v}}{\partial P_x^t \partial P_y^u \partial P_z^u} \int dr_1 exp(-\alpha_p r_p^2) \int_{-\infty}^{\infty} exp\left(-r_C^2 t^2\right) dt. \tag{4.87}$$

This is a rather large equation now, and we will focus on the integral part of it first, that is

$$V_p = \int dr_1 exp(-\alpha_p r_p^2) \int_{-\infty}^{\infty} exp\left(-r_C^2 t^2\right) dt. \tag{4.88}$$

We first multiply together the exponentials, through the Gaussian product rule.

$$exp(-\alpha_p r_p^2) exp\left(-r_C^2 t^2\right) = exp\left(-\frac{\alpha_p t^2}{\alpha_p + t^2} R_{cP}^2\right) exp\left(-(\alpha_p + t^2) r_s^2\right). \tag{4.89}$$

We insert this into the integral over r and t and obtain

$$V_p = \int dr \int_{-\infty}^{\infty} dt \times exp\left(-(\alpha_p + t^2) r_s^2\right) exp\left(-\frac{\alpha_p t^2}{\alpha_p + t^2} R_{cP}^2\right). \tag{4.90}$$

The integral over r can now be solved analytically.

$$\int exp\left(-(\alpha_p + t^2) r_s^2\right) dr = \left(\frac{\pi}{\alpha_p + t^2}\right)^{3/2}. \tag{4.91}$$

This is inserted in V_p and we get

$$V_p = \int_{-\infty}^{\infty} exp\left(-\frac{\alpha_p t^2}{\alpha_p + t^2} R_{cP}^2\right)\left(\frac{\pi}{\alpha_p + t^2}\right)^{3/2} dt. \tag{4.92}$$

We now introduce u as

$$u^2 = \frac{t^2}{\alpha_p + t^2}. \tag{4.93}$$

Some rewriting results in

$$dt = \frac{1}{\alpha_p}\left(\frac{t^2}{u^2}\right)^{3/2} du. \tag{4.94}$$

We insert this into V_p and we obtain finally

$$V_p = \frac{2\pi^{3/2}}{\alpha_p}\int_0^1 exp(-\alpha_p R_{cP}^2 u^2) du. \tag{4.95}$$

Solving this integral is done using the Boys function discussed below.

The Boys Function

For the Boys function we have used additional references [38] and [39]. The Boys function is defined as

$$F_n(x) = \int_0^1 exp(-xt^2) t^{2n} dt. \tag{4.96}$$

This can be solved analytically if $x = 0$ and gives

$$F_n(0) = \int_0^1 t^{2n} dt = \frac{1}{2n+1}. \tag{4.97}$$

If x is small, we can Taylor expand around 0 and get

$$F_n(x) = \sum_k^{\infty} \frac{(-x)^k}{k!(2n+2k+1)}. \tag{4.98}$$

We cannot sum to infinity, so we must define some value M to make the sum finite, where M should be large.

If x is large, the exponential function will flatten out for increasing t, and will be approximately 0 for $t > 1$. This means we can make this approximation

$$F_n(x) \approx \int_0^{\infty} exp(-xt^2) t^{2n}. \tag{4.99}$$

This can be solved analytically and gives

$$F_n(x) = \frac{(2n+1)!!}{2^{n+1}}\sqrt{\frac{\pi}{x^{2n+1}}}. \tag{4.100}$$

Once we have calculated the Boys function for one n, we can use recurrence relations to find the others. These relations are defined as

$$F_{n-1}(x) = \frac{2xF_n(x) + exp(-x)}{2n-1}. \tag{4.101}$$

There is also an upward recurrence relation,

$$F_{n+1}(x) = \frac{(2n+1)F_n(x) - exp(-x)}{2x}, \tag{4.102}$$

but this is somewhat numerically unstable.

Defining R_{tuv}

The Boys function should be inserted into Eq. (4.95). In the equation we do not have any t^{2n} term, this is equivalent to having t^0, meaning $n = 0$.

$$V_p = \frac{2\pi}{\alpha_p}F_0(\alpha_p R_{cP}^2). \tag{4.103}$$

This should further be inserted into Eq. (4.87). We also place the constant $\frac{2\pi}{\alpha_p}$ outside the derivatives. We get

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \frac{1}{\sqrt{\pi}}\sum_{tuv} E_{tuv}^{ab}\frac{2\pi}{\alpha_p}\frac{\partial^{t+u+v}}{\partial P_x^t \partial P_y^u \partial P_z^u}F_0(\alpha_p R_{cP}^2). \tag{4.104}$$

We here define $R_{tuv}(\alpha_p, R_{cP})$.

$$R_{tuv}(\alpha_p, R_{cP}) = \frac{\partial^{t+u+v}}{\partial P_x^t \partial P_y^u \partial P_z^u}F_0(\alpha_p R_{cP}^2). \tag{4.105}$$

The quantity R_{tuv} is known as the Hermite Coulomb integrals. How to evaluate these will be shown in a later section. We now insert this into Eq. (4.104) and get

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \frac{1}{\sqrt{\pi}}\sum_{tuv} E_{tuv}^{ab}\frac{2\pi^{3/2}}{\alpha_p}R_{tuv}(\alpha_p, R_{cP}). \tag{4.106}$$

We combine terms and rewrite the equation as

$$\langle\chi_a|\frac{1}{r_C}|\chi_b\rangle = \frac{2\pi}{\alpha_p}\sum_{tuv} E_{tuv}^{ab}R_{tuv}(\alpha_p, R_{cP}). \tag{4.107}$$

This is a programmable expression once the equations for R_{tuv} have been defined.

4.6.4 Electron-Electron interaction

The final integral to solve is $\langle ab|cd\rangle$.

$$\langle ab|cd\rangle = \int dr_1 \int dr_2 \phi_a^*(r_1)\phi_b^*(r_2)\frac{1}{r_{12}}\phi_c(r_1)\phi_d(r_2). \tag{4.108}$$

This a similar situation to that of the nucleus-electron interaction, except that we now have two particle and thereby (for a three-dimensional case) in principle a six-dimensional integral. We can combine $\phi_a^*(r_1)$ with $\phi_c(r_1)$ and $\phi_b^*(r_2)$ with $\phi_d(r_2)$ and have

$$\langle ab|cd\rangle = \int dr_1 \int dr_2 \frac{\Omega_{ac}(r_1)\Omega_{bd}(r_2)}{r_{12}}. \tag{4.109}$$

Here we can insert Ω as we did in section 4.6.3.

$$\langle ab|cd\rangle = \sum_{tuv} E_{tuv}^{ac} \sum_{\tau\mu\theta} E_{\tau\mu\theta}^{bd} \int dr_1 \int dr_2 \frac{\Lambda_{tuv}(r_{1P})\Lambda_{\tau\mu\theta}(r_{2Q})}{r_{12}}. \tag{4.110}$$

We insert the definition of Λ as we did in Eq. (4.83).

$$\begin{aligned}\langle ab|cd\rangle =& \sum_{tuv} E_{tuv}^{ac} \sum_{\tau\mu\theta} E_{\tau\mu\theta}^{bd} \frac{\partial^{t+u+v}}{\partial P_x^t \partial P_y^u \partial P_z^v} \frac{\partial^{\tau+\mu+\theta}}{\partial Q_x^\tau \partial Q_y^\mu \partial Q_z^\theta} \int dr_1 \times \\ & \int dr_2 \frac{exp(-\alpha_p r_{1P}^2) exp(-\alpha_q r_{2Q}^2)}{r_{12}}.\end{aligned} \tag{4.111}$$

The terms inside the integral are similar to what we had for the nucleus-electron interaction, except we now have r_{12}. Recalling

$$r_{12} = \sqrt{(r_1 - r_2)^2}. \tag{4.112}$$

Taking the derivative with respect to r_2 will change the sign of the equation. We will take this derivative $\tau+\mu+\theta$ times. Using the same technique applied to the nucleus-electron interaction this can be shown to reduce to

$$\langle ab|cd\rangle = \frac{2\pi^{5/2}}{\alpha_p\alpha_q\sqrt{\alpha_p+\alpha_q}} \sum_{tuv} E_{tuv}^{ac} \sum_{\tau\mu\theta} E_{\tau\mu\theta}^{bd} (-1)^{\tau+\mu+\theta} R_{t+\tau,u+\mu,v+\theta}(\alpha, R_{PQ}), \tag{4.113}$$

where α without any index is equal to

$$\alpha = \frac{\alpha_p\alpha_q}{\alpha_p+\alpha_q}. \tag{4.114}$$

4.6.5 Calculating R_{tuv}

The quantity R_{tuv} is in use in both the nucleus-electron interaction and the electron-electron interaction. Here we will find programmable equations for $R_{tuv}(a, A)$

$$R_{tuv}(a, A) = \frac{\partial^{t+u+v}}{\partial A_x^t \partial A_y^u \partial A_z^v} F_0(a \times A^2). \tag{4.115}$$

To obtain practical equations we introduce the auxiliary hermite integrals

$$R_{tuv}^n(a, A) = (-2a)^n \frac{\partial^{t+u+v}}{\partial A_x^t \partial A_y^u \partial A_z^v} F_n(a \times A^2). \tag{4.116}$$

We first get a starting value when $t = u = v = 0$,

$$R_{000}^n(a, A) = (-2a)^n F_n(a \times A^2). \tag{4.117}$$

From this we use the following recurrence relations which has been proven in [8], namely

$$R_{t+1,u,v}^n = tR_{t-1,u,v}^{n+1} + A_x R_{tuv}^{n+1}, \tag{4.118}$$

$$R_{t,u+1,v}^n = uR_{t,u-1,v}^{n+1} + A_y R_{tuv}^{n+1}, \tag{4.119}$$

and

$$R_{t,u,v+1}^n = tR_{t,u,v-1}^{n+1} + A_z R_{tuv}^{n+1}. \tag{4.120}$$

Since we do not have any $(-2a)^n$ term in any of our equations for the nucleus-electron or the electron-electron interactions, we make a small tweak to these equations when we use the auxiliary hermite integrals,

$$\langle \chi_a | \frac{1}{r_C} | \chi_b \rangle = \frac{2\pi}{\alpha_p} \sum_{tuv} E_{tuv}^{ab} R_{tuv}^0(\alpha_p, R_{cP}), \tag{4.121}$$

and

$$\langle ab|cd \rangle = \frac{2\pi^{5/2}}{\alpha_p \alpha_q \sqrt{\alpha_p + \alpha_q}} \sum_{tuv} E_{tuv}^{ac} \sum_{\tau\mu\theta} E_{\tau\mu\theta}^{bd} \, (-1)^{\tau+\mu+\theta} \, R_{t+\tau,u+\mu,v+\theta}^0(\alpha, R_{PQ}). \tag{4.122}$$

4.7 Choosing Basis Set

We now have plenty of programmable equations for solving the HF equations, and we will pull them all together in the implementation chapter. But before we can use any of them we must choose some basis functions. We will use basis sets from EMSL. In this section we will discuss the differences between the different basis sets.

There are several basis sets in EMSL. Some of these are 6-31G, 3-21G, 6-311++G**, STO-3G, cc-pVDZ etc. In general, Pople type basis sets are the ones with numbers inside the name, such as the 6-31G and 3-21G sets. The Pople type basis sets use cartesian GTOs, which are the same ones that we use. We can also work with STO-3G and other versions of this basis set.

It is possible to use basis sets designed for spherical GTOs with a program that use Cartesian GTOs. In this situation we usually get an energy slightly lower than if our program did actually use spherical GTOs. Also it is less effective in terms of program performance. For these reasons we will not make use of basis sets designed for spherical GTOs. These are basis sets such as cc-pVTZ, aug-cc-pVDZ etc.

4.7.1 STO-nG

First we look at the simplest, smallest basis set, the STO-nG family. These are single-zeta basis sets. Up till now we have usually talked about contracted GTOs and Atomic Orbitals (AOs) as the same thing, and for our purposes they are the same.

However, strictly speaking, the terminology of Atomic Orbitals in this context are the solutions of the HF equations for one atom. In this scenario we have no molecular orbitals since we have no molecule. An atomic orbital would then be a linear combination of contracted GTOs. Early on in the literature the terms atomic orbital meant basis function. The term basis function was introduced somewhat later where appropriate. This distinction is more important if we are constructing new basis sets. Since we will be citing old articles we will, with exception of this section, use the term atomic orbital as used in the old context.

A single zeta basis set means we have just enough contracted GTOs to contain the electrons of a neutral charged atom and retain its spherical symmetry.

The STO part of the name signifies that we are trying to mimic an STO, but we are still using GTOs. An STO-nG basis set will have n primitives for each contracted GTO.

For the hydrogen atom with only one electron, we will only have one contracted GTO to describe the 1s orbital. For the atoms Li to Ne STO-nG will have five contracted GTOs to describe the orbitals 1s, 2s, 2p$_x$ 2p$_y$ and 2p$_z$. All these orbitals need to be described since we want to retain the spherical symmetry of the atom.

4.7.2 Double Zeta Basis Sets

STO-nG is a small and poor basis set, regardless of the value of n. It should never be used, unless we are performing program tests. This simplifies things further, since we are left with the option of using the Pople style basis sets. These are the ones named 4-31G, 6-31G and so on.

The smallest Pople style basis sets are known as double zeta. The smallest of these are 3-21G. This was originally called STO-3-21G, but the STO part is omitted. 3-21G is a basis set with one contracted GTO for the core orbitals. This GTO consists of 3 primitives, hence the number 3 in the first position.

The numbers 2 and 1 mean that we have two contracted GTOs to represent the valence orbitals. These contracted GTOs consist of 2 and 1 primitives, respectively. The name double zeta comes from the fact that the valence orbitals are now represented by two contracted GTOs.

The set 6-31G is another example of a double zeta Pople basis set. Here we have six primitives for the contracted GTO describing the core orbitals, and two contracted GTOs describing the valence orbitals. These have 3 and 1 primitives, respectively.

For Li to Ne the valence orbitals will be 2s, 2p$_x$, 2p$_y$ and 2p$_z$. The core orbital will be 1s. This makes in total nine contracted GTOs. Figure 4.8 illustrates the two contracted GTOs in the 6-31G basis set that both illustrate the 2p$_x$ orbital in the Beryllium atom. Our solver can use both of these in the linear combination to make the molecular orbitals.

4.7.3 Tripple Zeta Basis Sets

Another Pople basis set is the 6-311G basis set. Here we have three numbers after the dash, meaning the valence electrons will be represented by three contracted GTOs.

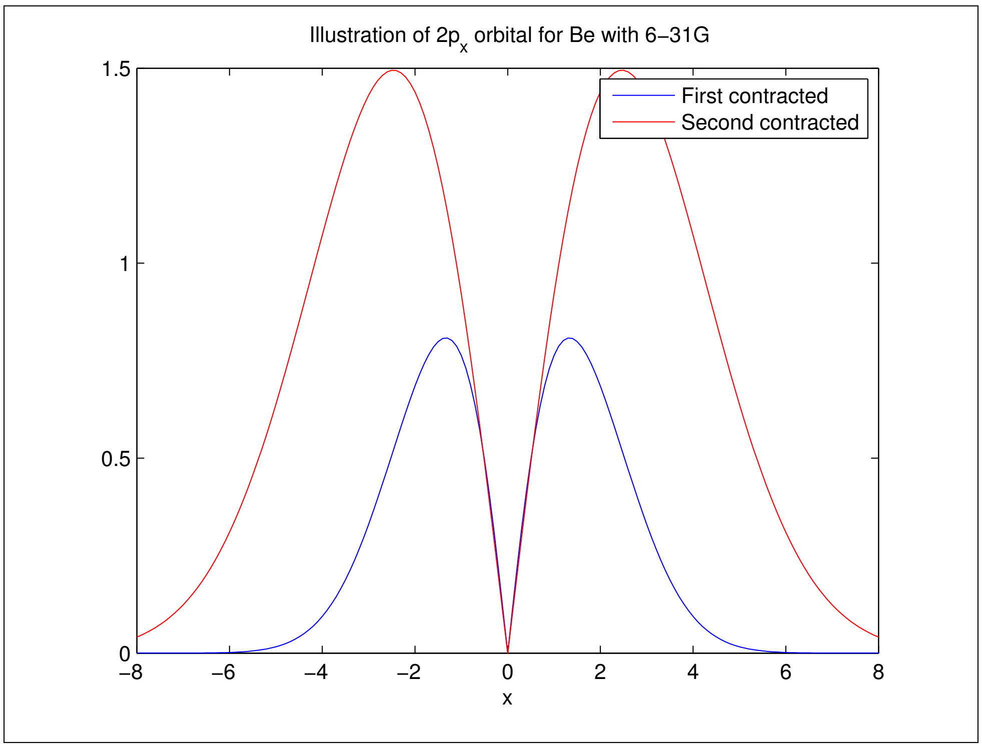


Figure 4.8: 2p$_x$ Basis Functions for Be with the 6-31G Basis Set

The core orbitals will have one contracted, that consists of six primitives. The valence electrons will have three contracted GTOs, where each of them consist of 3, 1 and 1 primitives. Figure 4.9 illustrates the three contracted GTOs in the 6-311G basis set that share quantum numbers with the 2p$_x$ orbital. Our solver can use all these three GTOs in the linear combination to make the molecular orbitals.

4.7.4 Polarized Basis Set

Polarized basis sets are usually marked by a star. The 6-311G basis set, with added polarization functions, is called 6-311G*. The difference here is that we add orbitals of higher angular momentum. If an atom has electrons where the P orbital contains the valence electrons, the P orbital has $l = 1$. Adding polarization will mean adding D orbitals, with $l = 2$.

4.7.5 Diffuse Basis Set

We can also choose a diffuse basis set. The largest basis sets are always diffuse, and the smaller diffuse basis sets are usually marked by a $+$. For

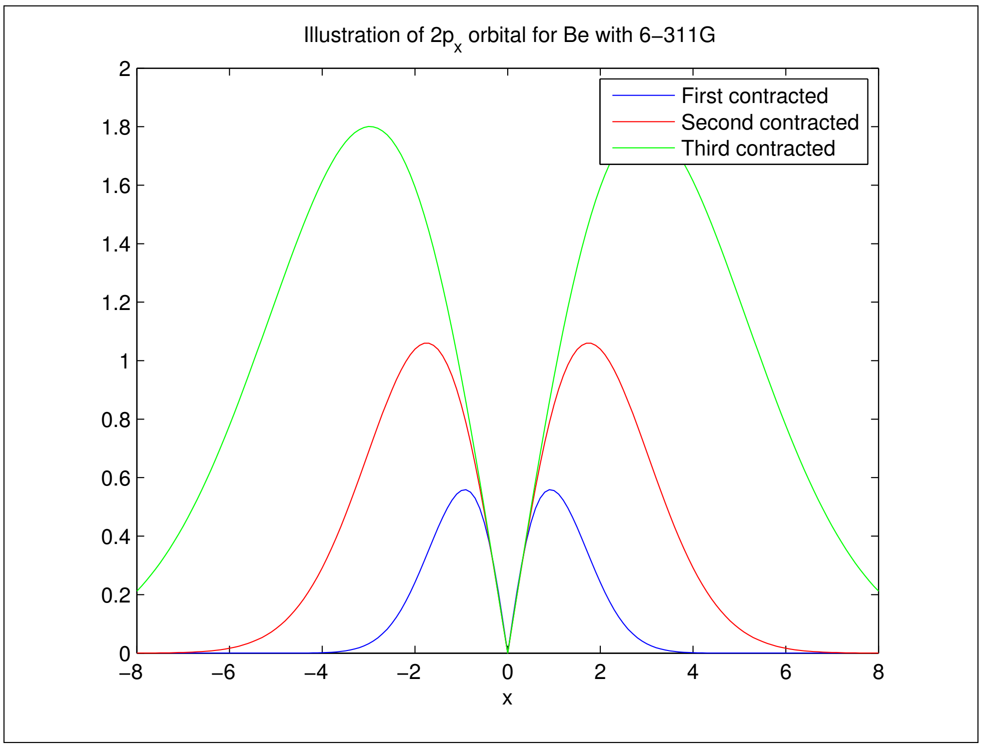


Figure 4.9: 2p$_x$ Basis Functions for Be with the 6-311G Basis Set

example, adding diffuse functions to a 6-311G basis set will make 6-311+G.

Adding diffuse functions means that we add another set of contracted GTOs. If we have a basis set of S and P orbitals, we add one more contracted GTO for the S and P orbitals. If our basis set is also polarized there are also D orbitals present. However a diffuse basis set does not add another D orbital.

Figure 4.10 illustrates the additional GTOs available for the linear combination. We can combine also diffuse and polarized basis functions. If we add diffuse and polarization functions to the 6-311G basis set we get the 6-311+G* basis set.

4.7.6 Reasons for Larger Basis Set

The main reason for a larger basis set is the ability for our HF solver to respond to a changing molecular environment. The electrons are likely to behave differently in an atom compared to in a larger molecule. We do con-

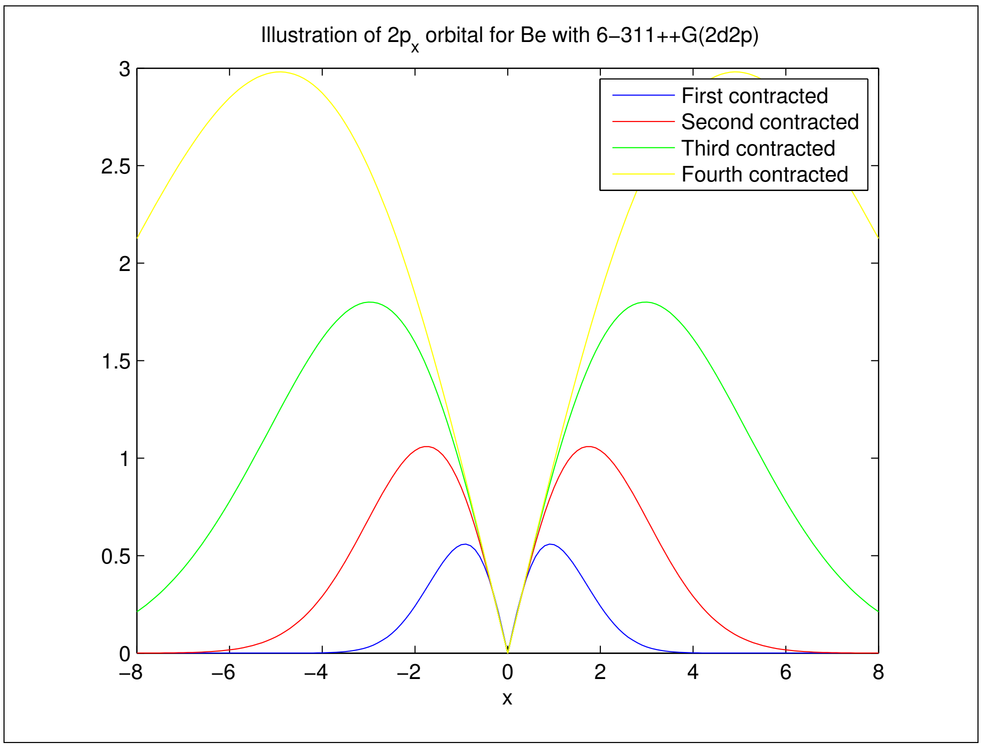


Figure 4.10: 2p$_x$ Basis Functions for Be with the 6-311++G(2d,2p) Basis Set

struct molecular orbitals as a linear combination of contracted GTOs. Since we minimize the energy with respect to this linear combination, having more contracted GTOs should produce a better result.

By adding diffuse functions, we make it possible that the electrons are pulled further away from their positions in a single atom. Double and triple zeta also helps with this. This is illustrated in figures 4.8, 4.9 and 4.10.

By introducing polarization functions, we make it possible for electrons to redistribute themselves in our molecule.

All this helps our HF solver in finding the best suited wavefunction, for any system.

4.8 HF Limitations

However if we add too many orbitals, some of them will get coefficients of zero in the linear combination. Once this happens we have reached a converged basis set. Increasing the size of the basis set at this point will not improve our results further.

When we have a converged basis set we still do not have the exact wavefunction. This is because we used a single Slater determinant. In a single Slater determinant the electrons feel a mean interaction with the other electrons. This is an approximation.

We can imagine two electrons in close proximity. Two electrons side by side will feel a strong and repulsive interaction, possibly even temporarily pushing both electrons into orbitals previously unoccupied. HF theory does not account for this at all. As such the energy cannot converge to the exact ground state energy. It can only converge to the so called Hartree Fock Limit.

Polarized basis sets include orbitals of higher angular momentum than the electrons occupy in a single atom. With a single Slater determinant, polarized basis sets are not as effective as they could be. If we go back to the example of the two electrons in close proximity, we stated that it was possible that they were temporarily excited into orbitals previously unoccupied. To describe this we need a description of these orbitals that are unoccupied. This is where the polarized basis sets are important. Coupled cluster theory, unlike HF theory, accounts for this situation. As such it is likely to see more basis functions required for the CCSD energy to be converged relative to HF, since Coupler Cluster can make better use of the polarized basis set.

4.9 DIIS

Direct Inversion of the Iterative Subspace is a method to help the convergence of our HF solution. This method can potentially reduce the number of iterations required for self consistency dramatically. This content is explained in detail in Refs. [66, 67, 68].

DIIS is a method to reduce the number of iterations required to solve any iterative problem. In HF theory this means updating the Fock matrix elements in between iterations. In our implementation we define an error given by

$$\Delta p = FPS - SPF. \tag{4.123}$$

The definition of the error is somewhat optional. We used the density matrix P defined as

$$P_{ij} = \sum_{m}^{N} C_{im} C_{jm}. \tag{4.124}$$

However the error can also be defined differently. Another option is simply using the current and the prior Fock matrix,

$$\Delta p_k = F_k - F_{k-1}. \tag{4.125}$$

In DIIS we want to make a linear combination of Fock matrices in prior iterations. We want

$$F = \sum_k^M c_k F_k. \tag{4.126}$$

Here we want to have $\sum_k^M c_k = 1$. We want to minimize the norm

$$\langle \Delta p | \Delta p \rangle = \sum_i^M \sum_j^M c_i^* c_j \langle \Delta p_i | \Delta p_j \rangle. \tag{4.127}$$

We define the overlap of the errors as a matrix B

$$B_{ij} = \langle \Delta p_i | \Delta p_j \rangle. \tag{4.128}$$

This matrix will be symmetric, since Δp is real. We can then use the method of Lagrangian multipliers, since we want to minimize the energy with the constraint that the sum of coefficients is 1, that is

$$\mathcal{L} = c^\dagger B c - \lambda \left(1 - \sum_i^M c_i \right). \tag{4.129}$$

We find the minimum of this Lagrangian by minimizing the coefficients c,

$$\frac{\partial \mathcal{L}}{\partial c_k} = 0. \tag{4.130}$$

This gives

$$\sum_i^M c_i B_{ik} + \sum_j^M c_j B_{kj} - \lambda = 0. \tag{4.131}$$

We can use the symmetry in B to combine the two sums and obtain

$$2 \sum_i^M c_i B_{ki} - \lambda = 0. \tag{4.132}$$

Since λ is a constant, we can place the factor two into λ and get

$$\sum_i^M c_i B_{ki} - \lambda = 0. \tag{4.133}$$

This results in Eq. (6) in [67], restated here

$$\begin{pmatrix} B_{11} & B_{12} & \dots & B_{1M} & -1 \\ B_{21} & B_{22} & \dots & B_{2M} & -1 \\ \dots & \dots & \dots & \dots & \dots \\ B_{M1} & B_{M2} & \dots & B_{MM}-1 & \\ -1 & -1 & \dots & -1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_M \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{pmatrix}$$

4.10 Four Index Integral, from AO to MO

The four index integral $\langle ij|kl\rangle$ is defined in terms of AOs. However, we want a procedure to recalculate these in terms of Molecular Orbitals (MOs). The MOs is defined as

$$\psi_a = \sum_i C_i^a \phi_i. \tag{4.134}$$

Our integral using MOs is describes as

$$\langle ab|cd\rangle = \int dr_1 \int dr_2 \psi_a^*(r_1)\psi_b^*(r_2)|\frac{1}{r_{12}}|\psi_c(r_1)\psi_d(r_2). \tag{4.135}$$

We insert the definition of an MO into this equation. We assume the coefficients, C_i^a, are real

$$\langle ab|cd\rangle = \sum_{ijkl} \int dr_1 \int dr_2 C_i^a \phi_i^*(r_1) C_j^b \phi_j^*(r_2)|\frac{1}{r_{12}}|C_k^c \phi_k(r_1) C_l^d \phi_l(r_2). \tag{4.136}$$

We then pull all the coefficients outside the integral.

$$\langle ab|cd\rangle = \sum_{ijkl} C_i^a C_j^b C_k^c C_l^d \int dr_1 \int dr_2 \phi_i^*(r_1)\phi_j^*(r_2)|\frac{1}{r_{12}}|\phi_k(r_1)\phi_l(r_2). \tag{4.137}$$

This is our two electron or four index integral based on AOs. We insert this and get

$$\langle ab|cd\rangle = \sum_{ijkl} C_i^a C_j^b C_k^c C_l^d \langle ij||kl\rangle. \tag{4.138}$$

We will make use of this equation when starting with the coupled cluster method. In coupled cluster theory the four index integral is the only term where molecular orbitals are involved. For this reason the four index molecular orbital integrals are usually just called MOs for short. Also, the four index atomic orbital integrals are named AOs for short in the context of coupled cluster.

Chapter 5

Coupled Cluster Singles and Doubles

The Coupled Cluster method is an important ab initio technique in computational chemistry. It is considered the most reliable and also computationally affordable method for solving the electronic Schrödinger equation. It was first introduced by Coester and Kümmel, [93, 94], within the context of the nuclear many-body problem during the late 1950s. Paldus and Cizek introduced the method to quantum chemists in the late 1960s, Ref. [95]. In this chapter we will look at the derivation of coupled cluster singles and doubles (CCSD), using the results from our Hartree-Fock calculations. The latter provides a self-consistent single-particle basis for methods like Coupled Cluster theory, normally called post Hartree-Fock methods.

This chapter is based on a book by Crawford and Schaeffer III, Ref. [16].

5.1 Creation and Annihilation operators

In Coupled Cluster (CC) theory we aim at solving the Schrödinger equation for the ground state, namely

$$\mathbf{H}|\Psi\rangle = E|\Psi\rangle. \tag{5.1}$$

From Hartree Fock calculations we have created an approximation to the true ground state $|\Psi\rangle_{HF}$ which contains molecular orbitals (MOs) in a Slater determinant. The Dirac notation provides a simple representation of this. In the Dirac notation only the diagonal terms in the slater determinant are listed. If $|\Psi_0\rangle$ has four electrons, its Dirac notation reads

$$|\Psi_0\rangle = |\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_l(r_4)\rangle. \tag{5.2}$$

Eq. (5.2) will be used to introduce a few new operators needed. The creation operator $\mathbf{a}_m^\dagger$ creates a new electron in orbital m

$$\mathbf{a}_m^\dagger|\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_l(r_4)\rangle = |\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_l(r_4), \psi_m(r_5)\rangle. \tag{5.3}$$

Also, the annihilation operator $\mathbf{a}_n$ destroys an electron in orbital n,

$$\mathbf{a}_n|\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_l(r_4), \psi_n(r_5)\rangle = |\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_l(r_4)\rangle. \tag{5.4}$$

These two operators working together can destroy one electron in orbital n, and create another in orbital m. The result is that one electron now occupies a different orbital

$$\mathbf{a}_m^\dagger\mathbf{a}_n|\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_n(r_4)\rangle = |\psi_i(r_1), \psi_j(r_2), \psi_k(r_3), \psi_m(r_4)\rangle. \tag{5.5}$$

These operators have a few interesting features. The annihilation operator acting on the vacuum state produces 0, that is

$$a_n|\rangle = 0. \tag{5.6}$$

Interchanging two rows in the Slater determinant introduces a change in the sign. Hence we have

$$\mathbf{a}_m^\dagger\mathbf{a}_n^\dagger|\rangle = |\psi_m, \psi_n\rangle = -|\psi_n, \psi_m\rangle = -\mathbf{a}_n^\dagger\mathbf{a}_m^\dagger|\rangle. \tag{5.7}$$

This means

$$\mathbf{a}_m^\dagger\mathbf{a}_n^\dagger + \mathbf{a}_n^\dagger\mathbf{a}_m^\dagger = 0. \tag{5.8}$$

The same applies to the annihilation operator

$$\mathbf{a}_m\mathbf{a}_n + \mathbf{a}_n\mathbf{a}_m = 0. \tag{5.9}$$

These are known as anti commutation relations. It can be shown that the anti commutation relation when mixing $\mathbf{a}$ and $\mathbf{a}^\dagger$ is

$$\mathbf{a}_m^\dagger\mathbf{a}_n + \mathbf{a}_n\mathbf{a}_m^\dagger = \delta_{mn}. \tag{5.10}$$

5.2 CCSD Wavefunction

The first step in coupled cluster is to rewrite the wavefunction as

$$|\Psi_{CC}\rangle \equiv e^{\mathbf{T}}|\Psi_{HF}\rangle. \tag{5.11}$$

Here $\mathbf{T}$ is known as the cluster operator. This includes all possible excitations. The operator $\mathbf{T}$ can be defined in terms of a one-orbital excitation operator, a two-orbital excitation operator and so on, that is

$$\mathbf{T} \equiv \mathbf{T}_1 + \mathbf{T}_2 + \mathbf{T}_3 + \mathbf{T}_4 \ldots. \tag{5.12}$$

In the Coupled Cluster Singles and Doubles approximations, labeled CCSD, only single excitations, $\mathbf{T}_1$, and double excitations, $\mathbf{T}_2$, are included

$$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2. \tag{5.13}$$

Other CC approaches include more terms. If $\mathbf{T}_3$ is included the method is called CCSDT and we include all three-particle-three-hole excitations, $\mathbf{T}_3$. With four-particle-four-hole excitations, CCSDTQ, we also include $\mathbf{T}_4$.

The one-particle-one-hole excitation operator $\mathbf{T}_1$ is defined using one creation and one annihilation operator, because we will have one single electron excited. Also defining $\mathbf{T}_1$ is an amplitude t_i^a and a summation over all possible excitations

$$\mathbf{T}_1 \equiv \sum_{a,i} t_i^a \mathbf{a}_a^\dagger \mathbf{a}_i. \tag{5.14}$$

Similarly, $\mathbf{T}_2$ is defined by two creation and two annihilation operators and an amplitude t_{ij}^{ab}

$$\mathbf{T}_2 \equiv \frac{1}{4} \sum_{a,b,i,j} t_{ij}^{ab} \mathbf{a}_a^\dagger \mathbf{a}_b^\dagger \mathbf{a}_i \mathbf{a}_j. \tag{5.15}$$

5.3 Derivation of Equations

This section contains the formal derivation of coupled cluster theory, starting from Eq. (5.1) and using the CCSD wavefunction as our first approximation to the excitation operator $\mathbf{T}$. We have

$$\mathbf{H} e^{\mathbf{T}} |\Psi\rangle_{HF} = E e^{\mathbf{T}} |\Psi\rangle_{HF}. \tag{5.16}$$

For this derivation $|\Psi\rangle_{HF}$ will be shortened to $|\Psi_0\rangle$. The energy is given by

$$E = \langle \Psi_0 | e^{-\mathbf{T}} \mathbf{H} e^{\mathbf{T}} | \Psi_0 \rangle. \tag{5.17}$$

We also assume an orthonormal basis, meaning

$$\langle \Psi_m | e^{-\mathbf{T}} \mathbf{H} e^{\mathbf{T}} | \Psi_0 \rangle = 0. \tag{5.18}$$

5.3.1 Baker-Campbell-Hausdorff formula

The Baker-Campbell-Hausdorff formula is used to expand $e^{-\mathbf{T}}\mathbf{H}e^{\mathbf{T}}$

$$\begin{aligned} e^{-\mathbf{T}}\mathbf{H}e^{\mathbf{T}} = \mathbf{H} + [\mathbf{H},\mathbf{T}] + \frac{1}{2}\left[[\mathbf{H},\mathbf{T}],\mathbf{T}\right] + \frac{1}{6}\left[[[\mathbf{H},\mathbf{T}],\mathbf{T}],\mathbf{T}\right] \\ +\frac{1}{24}\left[[[[\mathbf{H},\mathbf{T}],\mathbf{T}],\mathbf{T}],\mathbf{T}\right]\ldots. \end{aligned} \tag{5.19}$$

The operator $\mathbf{T}$ is expressed in terms of $\mathbf{a}^\dagger$ and $\mathbf{a}$. $\mathbf{H}$ contains a maximum of two orbital interactions. It can be shown that $\mathbf{H}$ can also be expressed in terms of these operators

$$\mathbf{H} = \sum_{a,i} h_{a,i}\mathbf{a}_a^\dagger\mathbf{a}_i + \frac{1}{4}\sum_{a,b,i,j}\langle ab||ij\rangle\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j. \tag{5.20}$$

Here $h_{a,i} = \langle\psi_a|\mathbf{h}|\psi_i\rangle$, with $\mathbf{h}$ the one-particle part of $\mathbf{H}$. This is the same Hamiltonian as before expressed slightly differently and will be discussed further later on. Equation (5.19) can be simplified using commutators, namely

$$[\mathbf{a}_a^\dagger\mathbf{a}_i, \mathbf{a}_b^\dagger\mathbf{a}_j] = \mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_b^\dagger\mathbf{a}_j - \mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_a^\dagger\mathbf{a}_i. \tag{5.21}$$

Using the anti commutator relations this commutator itself can be simplified to

$$[\mathbf{a}_a^\dagger\mathbf{a}_i, \mathbf{a}_b^\dagger\mathbf{a}_j] = \mathbf{a}_a^\dagger\delta_{ib}\mathbf{a}_j - \mathbf{a}_b^\dagger\delta_{ja}\mathbf{a}_i. \tag{5.22}$$

This simplification reduces the number of indices from four to three, replacing two operators with a Kronecker delta. Each nested commutator in Eq. (5.19) will reduce the number of indexes by one. The maximum number of creation/annihilation operators in $\mathbf{H}$ was four. This means that Eq. (5.19) will naturally truncate after exactly four terms, and we can remove the dots

$$\begin{aligned} e^{-\mathbf{T}}\mathbf{H}e^{\mathbf{T}} = \mathbf{H} + [\mathbf{H},\mathbf{T}] + \frac{1}{2}\left[[\mathbf{H},\mathbf{T}],\mathbf{T}\right] + \frac{1}{6}\left[[[\mathbf{H},[\mathbf{T}],\mathbf{T}],\mathbf{T}\right] \\ +\frac{1}{24}\left[[[[\mathbf{H},\mathbf{T}],\mathbf{T}],\mathbf{T}],\mathbf{T}\right]. \end{aligned} \tag{5.23}$$

5.3.2 Normal Order and Contractions

When deriving the CCSD equations it is common to introduce a concept called normal ordering of second quantized operators. This means that all creation operators are placed to the left and the annihilation operators to the right. The mathematics of swapping the order of creation and annihilation operators are well defined. To show this we define an example operator $\mathbf{O}$ and use the anti commutator relations

$$
\begin{aligned}
\mathbf{O} &= \mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger \qquad (5.24)\\
&= \delta_{ia}\mathbf{a}_j\mathbf{a}_b^\dagger - \mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_j\mathbf{a}_b^\dagger\\
&= \delta_{ia}\delta_{jb} - \delta_{ia}\mathbf{a}_b^\dagger\mathbf{a}_j - \delta_{jb}\mathbf{a}_a^\dagger\mathbf{a}_i + \mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_j\mathbf{a}_b^\dagger\\
&= \delta_{ia}\delta_{jb} - \delta_{ia}\mathbf{a}_b^\dagger\mathbf{a}_j + \delta_{ib}\mathbf{a}_a^\dagger\mathbf{a}_j - \delta_{jb}\mathbf{a}_a^\dagger\mathbf{a}_i - \mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j. \qquad (5.25)
\end{aligned}
$$

The final expression of $\mathbf{O}$ is in normal order since all creation operators are to the left and all annihilation operators to the right. Notice that we now have five terms, four of which have a reduced number of operators compared to our first definition of $\mathbf{O}$. Any combination of annihilation and creation operators can be expressed as a linear combination of normal ordered combinations of these operators.

The four terms with reduced number of operators arise from contractions between operators. A contraction between two operators $\mathbf{A}$ and $\mathbf{B}$ is defined as

$$\overbracket{\mathbf{A}\mathbf{B}} \equiv \mathbf{AB} - \{\mathbf{AB}\}_\nu\,. \qquad (5.26)$$

Here $\{\mathbf{AB}\}_\nu$ is the normal ordered form of $\mathbf{AB}$. $\overbracket{\mathbf{A}\mathbf{B}}$ is called the contraction between $\mathbf{A}$ and $\mathbf{B}$. As an example $\mathbf{A} = \mathbf{a}_i$ and $\mathbf{B} = \mathbf{a}_j$ will give a contraction of

$$\overbracket{\mathbf{a}_i\mathbf{a}_j} = \mathbf{a}_i\mathbf{a}_j - \{\mathbf{a}_i\mathbf{a}_j\}_\nu = \mathbf{a}_i\mathbf{a}_j - \mathbf{a}_i\mathbf{a}_j = 0. \qquad (5.27)$$

From the example above we see that the contraction of all annihilation operators will be zero since there will be no swapping of operators when creating the normal ordered form. The same will apply to contractions between operators formed from only creation operators. Also the contraction between already normal ordered operators will be zero.

However the contraction between different operators not in normal order will not be zero. The simplest example is one annihilation operator in front of one creation operator

$$\overbracket{\mathbf{a}_i\mathbf{a}_a^\dagger} = \mathbf{a}_i\mathbf{a}_a^\dagger - \left\{\mathbf{a}_i\mathbf{a}_a^\dagger\right\}_\nu = \mathbf{a}_i\mathbf{a}_a^\dagger + \mathbf{a}_a^\dagger\mathbf{a}_i = \delta_{ia}. \qquad (5.28)$$

Here we used Eq. (5.10).

5.3.3 Wick's Theorem

Wick's Theorem provides a schematic way of defining any string of annihilation and creation operators in terms of these contractions. A string of annihilation and creation operators can be defined as $ABC\ldots XYZ$ where A, B, C, X, Y, Z ... represent either a creation or an annihilation operator. Wick's Theorem reads

$$\begin{aligned}\mathbf{ABC}\ldots\mathbf{XYZ} = &\{\mathbf{ABC}\ldots\mathbf{XYZ}\}_\nu \\ &+ \sum_{singles}\{\contraction{}{\mathbf{A}}{}{\mathbf{B}}\mathbf{ABC}\ldots\mathbf{XYZ}\}_\nu \\ &+ \sum_{doubles}\{\contraction{}{\mathbf{A}}{\mathbf{B}}{\mathbf{C}}\contraction[2ex]{\mathbf{A}}{\mathbf{B}}{\mathbf{C}\ldots\mathbf{XY}}{\mathbf{Z}}\mathbf{ABC}\ldots\mathbf{XYZ}\}_\nu \\ &\ldots\end{aligned} \tag{5.29}$$

The right side of Eq. (5.29) should represent every possible contraction of $\mathbf{ABC}\ldots\mathbf{XYZ}$. To specify the notation we apply Wick's theorem as an example to the operator $\mathbf{O}$ defined in Eq. (5.24) and repeated here

$$\mathbf{O} = \mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger$$

Applying Wick's Theorem provides

$$\mathbf{O} = \{\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.30}$$

$$+ \{\contraction{}{\mathbf{a}_i}{}{\mathbf{a}_a^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.31}$$

$$+ \{\contraction{\mathbf{a}_i}{\mathbf{a}_a^\dagger}{}{\mathbf{a}_j}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.32}$$

$$+ \{\contraction{\mathbf{a}_i\mathbf{a}_a^\dagger}{\mathbf{a}_j}{}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.33}$$

$$+ \{\contraction{}{\mathbf{a}_i}{\mathbf{a}_a^\dagger}{\mathbf{a}_j}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.34}$$

$$+ \{\contraction{\mathbf{a}_i}{\mathbf{a}_a^\dagger}{\mathbf{a}_j}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.35}$$

$$+ \{\contraction{}{\mathbf{a}_i}{\mathbf{a}_a^\dagger\mathbf{a}_j}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \tag{5.36}$$

$$+ \{\contraction{}{\mathbf{a}_i}{}{\mathbf{a}_a^\dagger}\contraction{\mathbf{a}_i\mathbf{a}_a^\dagger}{\mathbf{a}_j}{}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu . \tag{5.37}$$

Equation (5.37) stems from the doubles summation. The other terms are from the singles summation. Equations (5.32), (5.34) and (5.35) are zero when using rules such as (5.27). This leaves the terms

$$\begin{aligned}\mathbf{O} = &\{\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu + \{\contraction{}{\mathbf{a}_i}{}{\mathbf{a}_a^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu + \{\contraction{\mathbf{a}_i\mathbf{a}_a^\dagger}{\mathbf{a}_j}{}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu \\ &+ \{\contraction{}{\mathbf{a}_i}{\mathbf{a}_a^\dagger\mathbf{a}_j}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu + \{\contraction{}{\mathbf{a}_i}{}{\mathbf{a}_a^\dagger}\contraction{\mathbf{a}_i\mathbf{a}_a^\dagger}{\mathbf{a}_j}{}{\mathbf{a}_b^\dagger}\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\}_\nu .\end{aligned} \tag{5.38}$$

These must be evaluated. Using Eq. (5.36) as an example we have

$$\{\wick{\c{\mathbf{a}}_i\mathbf{a}_a^\dagger\mathbf{a}_j\c{\mathbf{a}}_b^\dagger}\}_\nu = -\{\wick{\c{\mathbf{a}}_i\mathbf{a}_a^\dagger\c{\mathbf{a}}_b^\dagger}\mathbf{a}_j\}_\nu = \{\wick{\c{\mathbf{a}}_i\c{\mathbf{a}}_b^\dagger}\mathbf{a}_a^\dagger\mathbf{a}_j\}_\nu. \tag{5.39}$$

Remembering that $\{\wick{\c{\mathbf{a}}_i\c{\mathbf{a}}_b^\dagger}\}_\nu = \delta_{ib}$ then the terms in $\mathbf{O}$ reduce to

$$\mathbf{O} = \{\mathbf{a}_i\mathbf{a}_a^\dagger\mathbf{a}_j\mathbf{a}_b^\dagger\} + \delta_{ia}\{\mathbf{a}_j\mathbf{a}_b^\dagger\} + \delta_{jb}\{\mathbf{a}_i\mathbf{a}_a^\dagger\} + \delta_{ib}\{\mathbf{a}_a^\dagger\mathbf{a}_j\} + \delta_{ia}\delta_{ib}. \tag{5.40}$$

Using $\{\mathbf{a}\mathbf{a}^\dagger\} = -\mathbf{a}^\dagger\mathbf{a}$ and $\{\mathbf{a}^\dagger\mathbf{a}\} = \mathbf{a}^\dagger\mathbf{a}$ we get

$$\mathbf{O} = \mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j - \delta_{ia}\mathbf{a}_b^\dagger\mathbf{a}_j - \delta_{jb}\mathbf{a}_a^\dagger\mathbf{a}_i + \delta_{ib}\mathbf{a}_a^\dagger\mathbf{a}_j + \delta_{ia}\delta_{ib}, \tag{5.41}$$

which is identical to Eq. (5.24). The sign rules can sometimes be complicated, when there is more than one contraction present. Swapping two operators can fulfil the positioning for two contractions at once, as seen in example Eq. (5.42). This provides a minus sign which must not be neglected

$$\{\wick{\c1{\mathbf{a}}_i\c2{\mathbf{a}}_a^\dagger\c1{\mathbf{a}}_j\c2{\mathbf{a}}_b^\dagger}\}_\nu = -\{\wick{\c{\mathbf{a}}_i\c{\mathbf{a}}_a^\dagger\c{\mathbf{a}}_j\c{\mathbf{a}}_b^\dagger}\}. \tag{5.42}$$

5.3.4 Fermi Vacuum and Particle Holes

When using creation and annihilation operators it is common with a specific vacuum state, $|\rangle$. Equation (5.2) would commonly be represented as such

$$|\Psi_0\rangle = \mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_k^\dagger\mathbf{a}_l^\dagger|\rangle. \tag{5.43}$$

An excited state $|\Psi_m\rangle$ would for example be noted as the following

$$|\Psi_m\rangle = \mathbf{a}_m^\dagger\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_k^\dagger|\rangle. \tag{5.44}$$

The orbital m is occupied and the orbital l is not occupied. In our CCSD derivation we will not use this kind of notation. The Fermi Vacuum is introduced and is later defined as the Hartree-Fock result. However continuing this example the Fermi Vacuum could be defined as $|\Psi_0\rangle$, and the excited state would be

$$|\Psi_m\rangle = \mathbf{a}_m^\dagger\mathbf{a}_l|\Psi_0\rangle. \tag{5.45}$$

This creates a "hole state" in orbital l, since an occupied orbital is now unoccupied. It also creates a "particle state" in orbital m, since this is now occupied and was unoccupied in the Fermi Vacuum.

This definition will bring new features to Wick's theorem. The indices $a, b, c\dots$ will denote newly occupied orbitals, or particle/virtual states. The indices $i, j, k\dots$ will denote newly formed hole states, or single-particle states below the Fermi level. The operator $\mathbf{a}_i^\dagger$ can then be thought of as annihilating a hole. $\mathbf{a}_a$ can be thought of annihilating a particle. Likewise $\mathbf{a}_a^\dagger$ and $\mathbf{a}_i$

can be thought of as creating a particle or creating a hole.

This differs from the concept of $\mathbf{a}^\dagger$ always being a creation operator, since $\mathbf{a}_i^\dagger$ can be thought of as annihilating a hole state. This changes our Wick's Theorem calculations, since we still have the only terms not zero being those with one annihilation operator followed by one creation operator. There are only two possibilities of this happening.

$$\overbracket{\mathbf{a}_i^\dagger\mathbf{a}_j} = \mathbf{a}_i^\dagger\mathbf{a}_j - \{\mathbf{a}_i^\dagger\mathbf{a}_j\}_\nu = \mathbf{a}_i^\dagger\mathbf{a}_j + \mathbf{a}_j\mathbf{a}_i^\dagger = \delta_{ij}. \tag{5.46}$$

and

$$\overbracket{\mathbf{a}_a\mathbf{a}_b^\dagger} = \mathbf{a}_a\mathbf{a}_b^\dagger - \{\mathbf{a}_a\mathbf{a}_b^\dagger\}_\nu = \mathbf{a}_a\mathbf{a}_b^\dagger + \mathbf{a}_b^\dagger\mathbf{a}_a = \delta_{ab}. \tag{5.47}$$

Any other contraction will be 0 using rules analogous to Eq. (5.27)

5.3.5 Normal Ordered H

As noted in Eq. (5.20) **H** can be expressed in terms of creation and annihilation operators. This expression is known as the secound-quantized form of the electronic Hamiltionan and will be repeated here, but the indices will be changed because a, b, i, j have now been given a new meaning. We have

$$\mathbf{H} = \sum_{pq} < p|\mathbf{h}|q > \mathbf{a}_q^\dagger\mathbf{a}_p + \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r. \tag{5.48}$$

We now wish to use Wick's theorem on this operator to simplify. From the one-electron term we use:

$$\mathbf{a}_p^\dagger\mathbf{a}_q = \{\mathbf{a}_p^\dagger\mathbf{a}_q\} + \{\overbracket{\mathbf{a}_p^\dagger\mathbf{a}_q}\}. \tag{5.49}$$

Equation(5.46) states that $\{\overbracket{\mathbf{a}_p^\dagger\mathbf{a}_q}\}$ is not equal to zero only if both operators act on a hole state, then $\{\overbracket{\mathbf{a}_p^\dagger\mathbf{a}_q}\} = \delta_{pq}$. This means

$$\sum_{pq}\{\overbracket{\mathbf{a}_p^\dagger\mathbf{a}_q}\} = \sum_i\langle i|h|i\rangle. \tag{5.50}$$

Inserting this in **H** we get

$$\mathbf{H} = \sum_{pq} < p|\mathbf{h}|q > \{\mathbf{a}_p^\dagger\mathbf{a}_q\} + \sum_i\langle i|h|i\rangle + \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r. \tag{5.51}$$

Wick's theorem will also be applied to the two electron part, $\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r$. Included here are only the non-zero terms

$$
\begin{aligned}
\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r ={}& \{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} + \contraction{\{}{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{\mathbf{a}_s}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} + \contraction{\{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{}{\mathbf{a}_s}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} \\
&+ \contraction{\{}{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q\mathbf{a}_s}{\mathbf{a}_r}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} + \contraction{\{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{\mathbf{a}_s}{\mathbf{a}_r}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} + \contraction{\{}{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{\mathbf{a}_s}\contraction[2ex]{\{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{\mathbf{a}_s}{\mathbf{a}_r}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} \\
&+ \contraction{\{}{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q\mathbf{a}_s}{\mathbf{a}_r}\contraction[2ex]{\{\mathbf{a}^\dagger_p}{\mathbf{a}^\dagger_q}{\mathbf{a}_s}{\mathbf{a}_r}\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\}.
\end{aligned}
$$

These can be simplified using Eq. (5.46) and the rules for index swapping within a contraction noted in Eq. (5.39), resulting in

$$
\begin{aligned}
\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r ={}& \{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} + \delta_{pi}\delta_{ps}\{\mathbf{a}^\dagger_q\mathbf{a}_r\} \\
&+ \delta_{qi}\delta_{qs}\{\mathbf{a}^\dagger_p\mathbf{a}_r\} + \delta_{pi}\delta_{pr}\{\mathbf{a}^\dagger_q\mathbf{a}_s\} + \delta_{qi}\delta_{qr}\{\mathbf{a}^\dagger_p\mathbf{a}_s\} \\
&- \delta_{pi}\delta_{ps}\delta_{qj}\delta_{qr} + \delta_{pi}\delta_{pr}\delta_{qj}\delta_{qs}. \qquad (5.52)
\end{aligned}
$$

The two electron part can now be replaced, leading to

$$
\begin{aligned}
\frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r ={}& \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\{\mathbf{a}^\dagger_p\mathbf{a}^\dagger_q\mathbf{a}_s\mathbf{a}_r\} \\
&- \frac{1}{4}\sum_{iqr}\langle iq||ri\rangle\{\mathbf{a}^\dagger_q\mathbf{a}_r\} \\
&+ \frac{1}{4}\sum_{ipr}\langle pi||ri\rangle\{\mathbf{a}^\dagger_p\mathbf{a}_r\} \\
&+ \frac{1}{4}\sum_{iqs}\langle iq||is\rangle\{\mathbf{a}^\dagger_q\mathbf{a}_s\} \\
&- \frac{1}{4}\sum_{ips}\langle pi||is\rangle\{\mathbf{a}^\dagger_p\mathbf{a}_s\} \\
&- \frac{1}{4}\sum_{ij}\langle ij||ji\rangle \\
&+ \frac{1}{4}\sum_{ij}\langle ij||ij\rangle.
\end{aligned}
$$

From the symmetry in the single bar four index integrals it can be shown that these symmetries hold for the double bar integrals

$$
\langle pq||rs\rangle = \langle qp||sr\rangle = -\langle pq||sr\rangle = -\langle qp||rs\rangle, \qquad (5.53)
$$

and

$$
\langle pq||rs\rangle = \langle rs||pq\rangle, \qquad (5.54)
$$

giving us an eightfold symmetry. Using Eq. (5.53), reindexing terms and combining leaves the two electron part as the following

$$\begin{aligned}\frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle \mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r =& \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\} \\ &+\sum_{ipr}\langle pi||ri\rangle\{\mathbf{a}_p^\dagger\mathbf{a}_r\} \\ &+\frac{1}{2}\sum_{ij}\langle ij||ij\rangle.\end{aligned} \tag{5.55}$$

This can be inserted in Eq. (5.51) to yield

$$\begin{aligned}\mathbf{H} =& \sum_{pq} < p|\mathbf{h}|q > \{\mathbf{a}_p^\dagger\mathbf{a}_q\} + \sum_i\langle i|h|i\rangle + \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\} \\ &+\sum_{ipr}\langle pi||ri\rangle\{\mathbf{a}_p^\dagger\mathbf{a}_r\} + \frac{1}{2}\sum_{ij}\langle ij||ij\rangle.\end{aligned} \tag{5.56}$$

The first and fourth term on the right hand side are the normal ordered form of the Fock operator. If we also include the second term we have the HF energy

$$\mathbf{H} = \sum_{pq} f_{pq}\{\mathbf{a}_p^\dagger\mathbf{a}_q\} + \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\} + \langle\Psi_{HF}|\mathbf{H}|\Psi_{HF}\rangle. \tag{5.57}$$

We rename these terms and write

$$\mathbf{H} = \mathbf{F}_N + \mathbf{V}_N + \langle\Psi_{HF}|\mathbf{H}|\Psi_{HF}\rangle. \tag{5.58}$$

The normal ordered Hamiltonian is defined from this

$$\mathbf{H}_N \equiv \mathbf{H} - \langle\Psi_{HF}|\mathbf{H}|\Psi_{HF}\rangle = \mathbf{F}_N + \mathbf{V}_N. \tag{5.59}$$

5.3.6 CCSD Hamiltonian

The CCSD Hamiltonian is now defined as

$$\bar{H} \equiv e^{-\mathbf{T}}\mathbf{H}_N e^{\mathbf{T}}. \tag{5.60}$$

Using the CCSD cluster operator, $\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2$. This can be inserted in equation Eq. (5.23) and gives

$$\begin{aligned}\bar{H} =& \mathbf{H}_N + [\mathbf{H}_N, \mathbf{T}_1] + [\mathbf{H}_N, \mathbf{T}_2] + \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_1\right] \\ &+ \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_2\right] + \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_2], \mathbf{T}_1\right] + \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_2], \mathbf{T}_2\right] \dots\end{aligned} \tag{5.61}$$

The operators $\mathbf{T}_1$ and $\mathbf{T}_2$ commute and the full $\bar{H}$ reads

$$\begin{aligned}\bar{H} =& \mathbf{H}_N + [\mathbf{H}_N, \mathbf{T}_1] + [\mathbf{H}_N, \mathbf{T}_2] + \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_1\right] \\ &+ \left[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_2\right] + \frac{1}{2}\left[[\mathbf{H}_N, \mathbf{T}_2], \mathbf{T}_2\right] \\ &+ \frac{1}{6}\left[[[\mathbf{H}_N, [\mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_1\right] + \frac{1}{6}\left[[[\mathbf{H}_N, [\mathbf{T}_2], \mathbf{T}_2], \mathbf{T}_2\right] \\ &+ \frac{1}{2}\left[[[\mathbf{H}_N, [\mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_2\right] + \frac{1}{2}\left[[[\mathbf{H}_N, [\mathbf{T}_1], \mathbf{T}_2], \mathbf{T}_2\right] \\ &+ \frac{1}{24}\left[[[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_1\right] + \frac{1}{24}\left[[[[\mathbf{H}_N, \mathbf{T}_2], \mathbf{T}_2], \mathbf{T}_2], \mathbf{T}_2\right] \\ &+ \frac{1}{6}\left[[[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_2\right] + \frac{1}{6}\left[[[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_1], \mathbf{T}_2], \mathbf{T}_2\right] \\ &+ \frac{1}{4}\left[[[[\mathbf{H}_N, \mathbf{T}_1], \mathbf{T}_2], \mathbf{T}_2], \mathbf{T}_2\right].\end{aligned} \tag{5.62}$$

$\bar{H}$ will still analytically truncate after up to and including four nested commutators. When using $\mathbf{H}_N$ it is better to rewrite Eqs. (5.14) and (5.15) using contractions

$$\mathbf{T}_1 \equiv \sum_{ai} t_i^a \mathbf{a}_a^\dagger \mathbf{a}_i = \sum_{ai}\left(t_i^a\{\mathbf{a}_a^\dagger \mathbf{a}_i\} + \{\overbracket{\mathbf{a}_a^\dagger \mathbf{a}_i}\}\right) = \sum_{ai} t_i^a\{\mathbf{a}_a^\dagger \mathbf{a}_i\}. \tag{5.63}$$

Similarly for $\mathbf{T}_2$ we have

$$\mathbf{T}_2 = \frac{1}{4}\sum_{abij}\{\mathbf{a}_a^\dagger \mathbf{a}_b^\dagger \mathbf{a}_i \mathbf{a}_j\}. \tag{5.64}$$

The commutators can then be calculated, starting with $[\mathbf{H}_N, \mathbf{T}_1]$

$$[\mathbf{H}_N, \mathbf{T}_1] = \mathbf{H}_N \mathbf{T}_1 - \mathbf{T}_1 \mathbf{H}_N. \tag{5.65}$$

Using the definition of contractions on both these terms we can simplify this expression further

$$\begin{aligned}[\mathbf{H}_N, \mathbf{T}_1] &= \left(\{\mathbf{H}_N \mathbf{T}_1\} + \{\overbracket{\mathbf{H}_N \mathbf{T}_1}\}\right) - \left(\{\mathbf{T}_1 \mathbf{H}_N\} + \{\overbracket{\mathbf{T}_1 \mathbf{H}_N}\}\right) \\ &= \{\overbracket{\mathbf{H}_N \mathbf{T}_1}\} - \{\overbracket{\mathbf{T}_1 \mathbf{H}_N}\}.\end{aligned} \tag{5.66}$$

Equations. (5.46) and (5.47) explains the only terms that will not be 0 when calculating the contractions. $\{\overbrace{\mathbf{T}_1\mathbf{H}_N}\}$ will be 0 since $\mathbf{T}_1$ does not contain any creation or annihilation operator that when placed on the left creates a non-zero contraction when using Wick's Theorem. The same argument applies to $\mathbf{T}_2$

$$[\mathbf{H}_N, \mathbf{T}_1] = \{\overbrace{\mathbf{H}_N\mathbf{T}_1}\} = (\mathbf{H}_N\mathbf{T}_1)_C\,, \tag{5.67}$$

and

$$[\mathbf{H}_N, \mathbf{T}_2] = \{\overbrace{\mathbf{H}_N\mathbf{T}_2}\} = (\mathbf{H}_N\mathbf{T}_2)_C\,. \tag{5.68}$$

It then becomes clear that the only surviving terms when calculating all the commutators will be terms with $\mathbf{H}_N$ in the leftmost position.

A new notation is also introduced, $()_C$. This notations means that each cluster operator inside the parentheses should have at least one contraction each to $\mathbf{H}_N$ when applying Wick's Theorem. This holds for up to four cluster operators. The final form of $\bar{H}$ becomes

$$\begin{aligned}\bar{H} = \big(\mathbf{H}_N + \mathbf{H}_N\mathbf{T}_1 + \mathbf{H}_N\mathbf{T}_2 + \frac{1}{2}\mathbf{H}_N\mathbf{T}_1^2 + \frac{1}{2}\mathbf{H}_N\mathbf{T}_2^2 + \mathbf{H}_N\mathbf{T}_1\mathbf{T}_2 \\ + \frac{1}{6}\mathbf{H}_N\mathbf{T}_1^3 + \frac{1}{6}\mathbf{H}_N\mathbf{T}_2^3 + \frac{1}{2}\mathbf{H}_N\mathbf{T}_1^2\mathbf{T}_2 + \frac{1}{2}\mathbf{H}_N\mathbf{T}_1\mathbf{T}_2^2 \\ + \frac{1}{24}\mathbf{H}_N\mathbf{T}_1^4 + \frac{1}{24}\mathbf{H}_N\mathbf{T}_2^4 + \frac{1}{4}\mathbf{H}_N\mathbf{T}_1^2\mathbf{T}_2^2 + \frac{1}{6}\mathbf{H}_N\mathbf{T}_1^3\mathbf{T}_2 + \frac{1}{6}\mathbf{H}_N\mathbf{T}_1\mathbf{T}_2^3\big)_C.\end{aligned} \tag{5.69}$$

5.3.7 CCSD Energy

Using the definition of $\mathbf{H}_N$, Eq. (5.59), and $\bar{H}$, Eq. (5.69), we can now construct a programmable expression for the energy, namely

$$E_{CCSD} - E_0 = \langle\Psi_0|\bar{H}|\Psi_0\rangle. \tag{5.70}$$

The terms in Eq. (5.69) are here calculated separately and we have

$$\langle\Psi_0|\mathbf{H}_N|\Psi_0\rangle = 0. \tag{5.71}$$

From the construction of the normal ordered Hamiltonian this term will be 0. Furthermore, we have

$$\begin{aligned}\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_1)_C|\Psi_0\rangle &= \langle\Psi_0|\left((\mathbf{F}_N + \mathbf{V}_N)\mathbf{T}_1\right)_C|\Psi_0\rangle \\ &= \langle\Psi_0|(\mathbf{F}_N\mathbf{T}_1)_C|\Psi_0\rangle + \langle\Psi_0|(\mathbf{V}_N\mathbf{T}_1)_C|\Psi_0\rangle.\end{aligned} \tag{5.72}$$

where

$$(\mathbf{F}_N\mathbf{T}_1)_C = \sum_{pq}\sum_{ai} f_{pq}t_i^a\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\}. \tag{5.73}$$

Wick's Theorem is applied to simplify the expression. Only non zero terms are included and we get

$$\begin{aligned}\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\} = & \quad \{\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_a^\dagger\mathbf{a}_i\} + \{\mathbf{a}_p^\dagger\overbracket{\mathbf{a}_q\mathbf{a}_a^\dagger}\mathbf{a}_i\} \\ & + \{\underbracket{\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_a^\dagger\mathbf{a}_i}\} + \{\underbracket{\mathbf{a}_p^\dagger\overbracket{\mathbf{a}_q\mathbf{a}_a^\dagger}\mathbf{a}_i}\} \\ = & \quad \{\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_a^\dagger\mathbf{a}_i\} + \delta_{pi}\{\mathbf{a}_q\mathbf{a}_a^\dagger\} \\ & + \delta_{qa}\{\mathbf{a}_p^\dagger\mathbf{a}_i\} + \delta_{pi}\delta_{qa}.\end{aligned} \tag{5.74}$$

Inserting this gives

$$(\mathbf{F}_N\mathbf{T}_1)_C = \sum_{pq}\sum_{ai} f_{pq}t_i^a\left(\{\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_a^\dagger\mathbf{a}_i\} + \delta_{pi}\{\mathbf{a}_q\mathbf{a}_a^\dagger\} + \delta_{qa}\{\mathbf{a}_p^\dagger\mathbf{a}_i\} + \delta_{pi}\delta_{qa}\right). \tag{5.75}$$

When calculating $\langle\Psi_0|(\mathbf{F}_N\mathbf{T}_1)_C|\Psi_0\rangle$ only terms that solely consists of δ's will be non 0, since our basis is orthogonal and we have

$$\langle\Psi_0|(\mathbf{F}_N\mathbf{T}_1)_C|\Psi_0\rangle = \sum_{pq}\sum_{ai} f_{pq}t_i^a\delta_{pi}\delta_{qa} = \sum_{ai} f_{ai}t_i^a. \tag{5.76}$$

$\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_1)_C|\Psi_0\rangle$ must also be calculated, resulting in

$$(\mathbf{V}_N\mathbf{T}_1)_C = \frac{1}{4}\sum_{pqrs}\sum_{ai}\langle pq||rs\rangle t_i^a\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\}, \tag{5.77}$$

with

$$\begin{aligned}\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\} = & \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i\} + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\overbracket{\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger}\mathbf{a}_i\} \\ & + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\overbracket{\mathbf{a}_r\mathbf{a}_a^\dagger}\mathbf{a}_i\} + \{\mathbf{a}_p^\dagger\underbracket{\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i}\} \\ & + \{\underbracket{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i}\} + \{\underbracket{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\overbracket{\mathbf{a}_r\mathbf{a}_a^\dagger}\mathbf{a}_i}\} \\ & + \{\underbracket{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\overbracket{\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger}\mathbf{a}_i}\}.\end{aligned} \tag{5.78}$$

From the derivation of $\langle\Psi_0|(\mathbf{F}_N\mathbf{T}_1)_C|\Psi_0\rangle$ we noticed that the only term that survived was the term where every construction/annihilation operator was linked by a contraction. In this case we have no such terms. Hence the contribution from $\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_1)_C|\Psi_0\rangle$ will be 0.

Inserting Eq. (5.76) and $\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_1)_C|\Psi_0\rangle = 0$ into Eq. (5.72) gives

$$\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_1)_C|\Psi_0\rangle = \sum_{ai} f_{ai}t_i^a. \tag{5.79}$$

Here the contribution from $\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_2)_C|\Psi_0\rangle$ is calculated

$$\langle\Psi_0|(\mathbf{F}_N\mathbf{T}_2)_C|\Psi_0\rangle = \frac{1}{4}\sum_{pq}\sum_{abij} f_{pq}t_{ij}^{ab}\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\{\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j\}. \tag{5.80}$$

This is again a similar situation that will result in a zero contribution. The reason is that any two operators **A** and **B** that contain a different number of annihilation/creation operators will not create any fully contracted terms (terms that solely consists of δ's) when applying Wick's Theorem. This means because of orthogonality the contribution to E_{CCSD} from terms like this will always be 0.

The term $\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_2)_C|\Psi_0\rangle$ however has an equal number of operators. From this we will have a contribution

$$\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_2)_C|\Psi_0\rangle = \frac{1}{16}\sum_{pqrs}\sum_{abij}\langle pq||rs\rangle t_{ij}^{ab}\langle\Psi_0|\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\}|\Psi_0\rangle$$

Only the four terms that are fully contracted and non-zero are listed here

$$\begin{aligned}
\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j\} = {} & \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\} + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\} \\
& \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\} + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\} \\
= {} & \delta_{pi}\delta_{qj}\delta_{sb}\delta_{ra} - \delta_{pi}\delta_{gj}\delta_{rb}\delta_{sa} + \delta_{pj}\delta_{qi}\delta_{rb}\delta_{sa} - \delta_{pj}\delta_{qi}\delta_{ra}\delta_{sb}.
\end{aligned} \tag{5.81}$$

Inserting this provides

$$\begin{aligned}
\langle\Psi_0|(\mathbf{V}_N\mathbf{T}_2)_C|\Psi_0\rangle &= \frac{1}{16}\sum_{pqrs}\sum_{abij}\langle pq||rs\rangle t_{ij}^{ab}\langle\Psi_0|\delta_{pi}\delta_{qj}\delta_{sb}\delta_{ra} - \delta_{pi}\delta_{gj}\delta_{rb}\delta_{sa} \\
&+ \delta_{pj}\delta_{qi}\delta_{rb}\delta_{sa} - \delta_{pj}\delta_{qi}\delta_{ra}\delta_{sb}|\Psi_0\rangle \\
&= \frac{1}{16}\sum_{abij}(\langle ij||ab\rangle - \langle ij||ba\rangle + \langle ji||ba\rangle - \langle ji||ab\rangle)t_{ij}^{ab} \\
&= \frac{1}{4}\sum_{abij} t_{ij}^{ab}\langle ij||ab\rangle.
\end{aligned} \tag{5.82}$$

Here symmetry considerations about double bar integrals were used. This means that we have

$$\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_2)_C|\Psi_0\rangle = \frac{1}{4}\sum_{abij} t_{ij}^{ab}\langle ij||ab\rangle. \tag{5.83}$$

Next contribution from $\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_1^2)_C|\Psi_0\rangle$. From the expression of $\bar{H}$ there is a $\frac{1}{2}$ in front of this term,

$$\begin{aligned}\frac{1}{2}\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_1^2)_C|\Psi_0\rangle =& \frac{1}{8}\sum_{pqrs}\sum_{ai}\sum_{bj}\langle pq||rs\rangle t_i^a t_j^b \\ & \langle\Psi_0|\{\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\}|\Psi_0\rangle.\end{aligned} \tag{5.84}$$

Again Wick's Theorem is used. We note that from $\bar{H}$ there are four creation/annihilation operators. From each $\mathbf{T}_1$ there are two. Since we have two single excitation operators we have four. This means we will have non-zero terms, these are listed here

$$\begin{aligned}\{\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_i\mathbf{a}_j\}\{\mathbf{a}_a^\dagger\mathbf{a}_i\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\} =& \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_b^\dagger\mathbf{a}_j\} + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_b^\dagger\mathbf{a}_j\} \\ & \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_i\mathbf{a}_b^\dagger\mathbf{a}_j\} + \{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_a^\dagger\mathbf{a}_b^\dagger\mathbf{a}_j\mathbf{a}_i\} \\ =& \delta_{pi}\delta_{qj}\delta_{sb}\delta_{ra} - \delta_{pi}\delta_{gj}\delta_{rb}\delta_{sa} + \delta_{pj}\delta_{qi}\delta_{rb}\delta_{sa} - \delta_{pj}\delta_{qi}\delta_{ra}\delta_{sb}.\end{aligned} \tag{5.85}$$

This is a result we have seen before in Eq. (5.83), the only difference is the amplitudes and the factor $\frac{1}{2}$

$$\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_1^2)_C|\Psi_0\rangle = \frac{1}{2}\sum_{abij} t_i^a t_j^b\langle ij||ab\rangle. \tag{5.86}$$

The next term is $\langle\Psi_0|(\mathbf{H}_N\mathbf{T}_2^2)_C|\Psi_0\rangle$. $\mathbf{H}_N$ still only have four creation/annihilation operators. However now we have 8 in total from the cluster operators. This means we cannot have fully contracted terms, which means the entire contribution to E_{CCSD} will be zero.

This argument will hold true for every single remaining term. Meaning we now have an expression for the energy from Eqs. (5.79), (5.83) and (5.86)

$$E_{CCSD} = E_0 + \sum_{ai} f_{ai}t_i^a + \frac{1}{4}\sum_{abij}\langle ij||ab\rangle t_{ij}^{ab} + \frac{1}{2}\sum_{abij}\langle ij||ab\rangle t_i^a t_j^b. \tag{5.87}$$

Here all factors are known except for t_i^a and t_{ij}^{ab}. These must be determined.

5.3.8 t_i^a amplitudes

We can find expressions for t_i^a by calculating $\langle \Psi_i^a | \bar{H} | \Psi_0 \rangle = 0$. The notation Ψ_i^a means a state with one hole state and one orbital state. This will be an excited state and we did assume orthogonality. The mathematics of this excited state can be described as such

$$\langle \Psi_i^a | = \langle \Psi_0 | \mathbf{a}_i^\dagger \mathbf{a}_a. \tag{5.88}$$

A creation operator working to the left, on a bra, becomes an annihilation operator. $\langle \Psi_i^a | \bar{H} | \Psi_0 \rangle = 0$ can be solved in the same manner as we did for the energy. Starting with the first term $\langle \Psi_i^a | \mathbf{H}_N | \Psi_0 \rangle$.

$\langle \Psi_i^a | \mathbf{H}_N | \Psi_0 \rangle$

$$\begin{aligned} \langle \Psi_i^a | \mathbf{H}_N | \Psi_0 \rangle &= \sum_{pq} f_{pq} \langle \Psi_0 | \{ \mathbf{a}_i^\dagger \mathbf{a}_a \} \{ \mathbf{a}_p^\dagger \mathbf{a}_q \} | \Psi_0 \rangle \\ &+ \frac{1}{4} \sum_{pqrs} \langle pq || rs \rangle \langle \Psi_0 | \{ \mathbf{a}_i^\dagger \mathbf{a}_a \} \{ \mathbf{a}_p^\dagger \mathbf{a}_q^\dagger \mathbf{a}_r \mathbf{a}_s \} | \Psi_0 \rangle. \end{aligned} \tag{5.89}$$

Here the first term is from $\mathbf{F}_N$ and the second term from $\mathbf{V}_N$. The second term will be zero. Eq. (5.88) is inserted and Wick's Theorem applied.

$$\begin{aligned} \Rightarrow \langle \Psi_i^a | \mathbf{H}_N | \Psi_0 \rangle &= \sum_{pq} f_{pq} \langle \Psi_0 | \{ \mathbf{a}_i^\dagger \mathbf{a}_a \} \{ \mathbf{a}_p^\dagger \mathbf{a}_q \} | \Psi_0 \rangle \\ &= \sum_{pq} f_{pq} \langle \Psi_0 | \{ \mathbf{a}_i^\dagger \mathbf{a}_a \mathbf{a}_p^\dagger \mathbf{a}_q \} | \Psi_0 \rangle \\ &= \sum_{pq} f_{pq} \delta_{iq} \delta_{ap} \\ &= f_{ai}. \end{aligned} \tag{5.90}$$

$\langle \Psi_i^a | \mathbf{H}_N \mathbf{T}_1 | \Psi_0 \rangle$

The next term includes $(\mathbf{H}_N \mathbf{T}_1)_c = (\mathbf{F}_N \mathbf{T}_1)_c + (\mathbf{V}_N \mathbf{T}_1)_c$. The $()_c$ notation is here applied to specify that there must be at least one contraction reaching from $\mathbf{H}_N$ to $\mathbf{T}_1$.

$$
\begin{aligned}
\langle\Phi_i^a|(\mathbf{F}_N\mathbf{T}_1)_c|\Phi_0\rangle &= \sum_{pq}\sum_{jb} f_{pq}t_j^b\langle\Phi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_a\}(\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\})_c|\Phi_0\rangle \\
&= \sum_{pq}\sum_{jb} f_{pq}t_j^b\left(\{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_b^\dagger\mathbf{a}_j\} + \{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q\mathbf{a}_b^\dagger\mathbf{a}_j\}\right) \\
&= \sum_{pq}\sum_{jb} f_{pq}t_j^b\left(\delta_{iq}\delta_{ab}\delta_{pj} + \delta_{ij}\delta_{ap}\delta_{qb}\right) \\
&= -\sum_{j} f_{ji}t_j^a + \sum_{b} f_{ab}t_i^b, \qquad (5.91)
\end{aligned}
$$

and

$$
\begin{aligned}
\langle\Phi_i^a|(\mathbf{V}_N\mathbf{T}_1)_c|\Phi_0\rangle &= \frac{1}{4}\sum_{pqrs}\sum_{jb}\langle pq||rs\rangle t_j^b\langle\Phi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_a\}(\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\})_c|\Phi_0\rangle \\
&= \frac{1}{4}\sum_{pqrs}\sum_{jb}\langle pq||rs\rangle t_j^b(\{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_b^\dagger\mathbf{a}_j\} \\
&+ \{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_b^\dagger\mathbf{a}_j\} + \{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_b^\dagger\mathbf{a}_j\} \\
&+ \{\mathbf{a}_i^\dagger\mathbf{a}_a\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\mathbf{a}_b^\dagger\mathbf{a}_j\}) \\
&= \frac{1}{4}\sum_{pqrs}\sum_{jb}\langle pq||rs\rangle t_j^b(\delta_{pj}\delta_{qa}\delta_{rb}\delta_{si} \\
&+ \delta_{pa}\delta_{qj}\delta_{ri}\delta_{sb} - \delta_{pa}\delta_{qj}\delta_{rb}\delta_{si} - \delta_{pj}\delta_{qa}\delta_{ri}\delta_{sb}) \\
&= \sum_{jb}\langle ja||bi\rangle t_j^b. \qquad (5.92)
\end{aligned}
$$

In total the contribution to the amplitudes from $\langle\Psi_i^a|\mathbf{H}_N\mathbf{T}_1|\Psi_0\rangle$ is

$$
\langle\Psi_i^a|(\mathbf{H}_N\mathbf{T}_1)_c|\Psi_0\rangle = -\sum_{j} f_{ji}t_j^a + \sum_{b} f_{ab}t_i^b + \sum_{jb}\langle ja||bi\rangle t_j^b.
$$

$\frac{1}{2}\langle\Psi_i^a|(\mathbf{H}_N\mathbf{T}_1^2)_c|\Psi_0\rangle$

Still contributions from $(\mathbf{F}_N\mathbf{T}_1^2)_c$ and $(\mathbf{V}_N\mathbf{T}_1^2)_c$ are calculated individually. The number of steps in the calculation is now reduced since δ_{ij} is understood to be a result of a non zero contraction between indices i and j,

$$\begin{aligned}\frac{1}{2}\langle\Psi_i^a|(\mathbf{F}_N\mathbf{T}_1^2)_c|\Psi_0\rangle &= \frac{1}{2}\sum_{pq}\sum_{jb}\sum_{kc} f_{pq}t_j^b t_k^c\langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_a\}\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\}\{\mathbf{a}_c^\dagger\mathbf{a}_k\}|\Psi_0\rangle\\ &= \frac{1}{2}\sum_{pq}\sum_{jb}\sum_{kc} f_{pq}t_j^b t_k^c\left(-\delta_{pk}\delta_{qb}\delta_{ij}\delta_{ac}-\delta_{pj}\delta_{qc}\delta_{ik}\delta_{ab}\right)\\ &= -\sum_{kc} f_{kc}t_i^c t_k^a, \end{aligned}\tag{5.93}$$

and

$$\begin{aligned}\frac{1}{2}\langle\Psi_i^a|(\mathbf{V}_N\mathbf{T}_1^2)_c|\Psi_0\rangle &= \frac{1}{8}\sum_{pqrs}\sum_{jb}\sum_{kc}\langle pq||rs\rangle t_j^b t_k^c\langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_a\}\\ &\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_b^\dagger\mathbf{a}_j\}\{\mathbf{a}_c^\dagger\mathbf{a}_k\}|\Psi_0\rangle\\ &= \frac{1}{8}\sum_{pqrs}\sum_{jb}\sum_{kc}\langle pq||rs\rangle t_j^b t_k^c(\delta_{pa}\delta_{br}\delta_{sc}\delta_{qk}\delta_{ij}+\ldots\\ &= \sum_{jbs}\langle ja||bc\rangle t_j^b t_i^c-\sum_{jbk}\langle jk||bi\rangle t_j^b t_k^a. \end{aligned}\tag{5.94}$$

Total

The remaining terms are calculated similarly. For our purposes we list the result. A more complete derivation is available in Ref. [26].

$$\begin{aligned}0 =& f_{ai}+\sum_c f_{ac}t_i^c-\sum_k f_{ki}t_k^a+\sum_{kc}\langle ka||ci\rangle t_k^c+\sum_{kc} f_{kc}t_{ik}^{ac}+\frac{1}{2}\sum_{kcd}\langle ka||cd\rangle t_{ki}^{cd}\\ &-\frac{1}{2}\sum_{klc}\langle kl||ci\rangle t_{kl}^{ca}-\sum_{kc} f_{kc}t_i^c t_k^a-\sum_{klc}\langle kl||ci\rangle t_k^c t_l^a+\sum_{kcd}\langle ka||cd\rangle t_k^c t_i^d\\ &-\sum_{klcd}\langle kl||cd\rangle t_k^c t_i^d t_l^a+\sum_{klcd}\langle kl||cd\rangle t_k^c t_{li}^{da}\\ &-\frac{1}{2}\sum_{klcd}\langle kl||cd\rangle t_{ki}^{cd}t_l^a-\frac{1}{2}\sum_{klcd}\langle kl||cd\rangle t_{kl}^{ca}t_i^d. \end{aligned}\tag{5.95}$$

5.3.9 t_{ij}^{ab} amplitudes

The t_{ij}^{ab} amplitudes are generated in a similar fashion. These amplitudes are calculated by solving the equation

$$\langle\Psi_{ij}^{ab}|\bar{H}|\Psi_0\rangle = 0. \tag{5.96}$$

This holds true because of the orthogonality. We can describe the state $\langle\Psi_{ij}^{ab}|$ in terms of creation and annihilation operators as

$$\langle\Psi_{ij}^{ab}| = \langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}. \tag{5.97}$$

We can solve Eq. (5.96) starting with the contribution from $\mathbf{H}_N$

$$\begin{aligned}
\langle\Psi_{ij}^{ab}|\mathbf{F}_N + \mathbf{V}_N|\Psi_0\rangle &= \langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}\left(\mathbf{F}_N + \mathbf{V}_N\right)|\Psi_0\rangle \\
&= \sum_{pq} f_{pq}\langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}\left(\{\mathbf{a}_p^\dagger\mathbf{a}_q\}\right)|\Psi_0\rangle \\
&+ \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle\langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}\left(\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\right)|\Psi_0\rangle \\
&= \frac{1}{4}\sum_{pqrs}\langle pq||rs\rangle(\delta_{pa}\delta_{qb}\delta_{ri}\delta_{sj} - \\
&\delta_{pb}\delta_{qa}\delta_{ri}\delta_{sj} - \delta_{pa}\delta_{qb}\delta_{si}\delta_{rj} + \delta_{qa}\delta_{pb}\delta_{si}\delta_{rj}) \\
&= \sum_{pqrs}\delta_{pb}\delta_{qa}\delta_{ri}\delta_{sj} \\
&= \langle ab||ij\rangle.
\end{aligned} \tag{5.98}$$

The contribution from $\mathbf{F}_N$ will be zero. The contribution from $\mathbf{H}_N\mathbf{T}_1$ is more complicated and we need to evaluate

$$\langle\Psi_{ij}^{ab}|(\mathbf{F}_N + \mathbf{V}_N)\mathbf{T}_1|\Psi_0\rangle = \langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}\left(\mathbf{F}_N + \mathbf{V}_N\right)\mathbf{T}_1|\Psi_0\rangle. \tag{5.99}$$

These will be calculated individually. Here we will only calculate the contribution from $\mathbf{V}_N\mathbf{T}_1$,

$$\begin{aligned}
\langle\Psi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\}\mathbf{V}_N\mathbf{T}_1|\Psi_0\rangle &= \frac{1}{4}\sum_{pqrs}\sum_{kc}\langle pq||rs\rangle t_k^c\langle\Phi_0|\{\mathbf{a}_i^\dagger\mathbf{a}_j^\dagger\mathbf{a}_a\mathbf{a}_b\} \\
&\Big(\{\mathbf{a}_p^\dagger\mathbf{a}_q^\dagger\mathbf{a}_s\mathbf{a}_r\}\{\mathbf{a}_c^\dagger\mathbf{a}_k\}\Big)_c|\Phi_0\rangle \\
&= \frac{1}{4}\sum_{pqrs}\sum_{kc}\langle pq||rs\rangle t_k^c(\delta_{pa}\delta_{qb}\delta_{rc}\delta_{sj}\delta_{ik} \\
&- \delta_{pa}\delta_{qb}\delta_{rc}\delta_{si}\delta_{jk} - \delta_{pa}\delta_{qb}\delta_{rj}\delta_{sc}\delta_{ik} + \delta_{pa}\delta_{qb}\delta_{ri}\delta_{sc}\delta_{jk} \\
&+ \delta_{pa}\delta_{qk}\delta_{rj}\delta_{si}\delta_{bc} - \delta_{pa}\delta_{qk}\delta_{ri}\delta_{sj}\delta_{bc} + \delta_{pb}\delta_{qa}\delta_{rj}\delta_{sc}\delta_{ik} \\
&- \delta_{pb}\delta_{qa}\delta_{ri}\delta_{sc}\delta_{jk} + \delta_{pb}\delta_{qa}\delta_{rc}\delta_{si}\delta_{jk} + \delta_{pb}\delta_{qk}\delta_{ri}\delta_{sj}\delta_{ac} \\
&- \delta_{pb}\delta_{qk}\delta_{rj}\delta_{si}\delta_{ac} - \delta_{pb}\delta_{qa}\delta_{rc}\delta_{sj}\delta_{ik} + \delta_{pk}\delta_{qa}\delta_{ri}\delta_{sj}\delta_{bc} \\
&- \delta_{pk}\delta_{qb}\delta_{ri}\delta_{sj}\delta_{ac} - \delta_{pk}\delta_{qa}\delta_{rj}\delta_{si}\delta_{bc} + \delta_{pk}\delta_{qb}\delta_{rj}\delta_{si}\delta_{ac}) \\
&= \sum_c\left(\langle ab||cj\rangle t_i^c - \langle ab||ci\rangle t_j^c\right) \\
&+ \sum_k\Big(\langle ij||bk\rangle t_k^a - \langle ij||ak\rangle t_k^b\Big). \qquad (5.100)
\end{aligned}$$

The rest of Eq. (5.96) can be solved in a similar manner. This is a task testing stamina and determination. Here we will simply state the result, however there is a more complete derivation using "Feynman Diagrams" available in Appendix A.

Before we state the final result we must define the permutation operator $\mathbf{P}$

$$\mathbf{P}(ab)f(a,b) = f(a,b) - f(b,a). \qquad (5.101)$$

An example of this would be:

$$\mathbf{P}(ab)\sum_{abij} t_i^a t_j^b f_{ai} = \sum_{abij}\Big(t_i^a t_j^b f_{ai} - t_i^b t_j^a f_{bi}\Big). \qquad (5.102)$$

Using this definition and solving the rest of Eq. (5.96) the expression becomes the following:

$$
\begin{aligned}
0 =&\langle ab||ij\rangle + \mathbf{P}(ab)\sum_{c} f_{bc}t_{ij}^{ac} - \mathbf{P}(ij)\sum_{k} f_{kj}t_{ik}^{ab} + \frac{1}{2}\sum_{kl}\langle kl||ij\rangle t_{kl}^{ab} \\
&+ \frac{1}{2}\sum_{cd}\langle ab||cd\rangle t_{ij}^{cd} + \mathbf{P}(ij)\mathbf{P}(ab)\sum_{kc}\langle kb||cj\rangle t_{ik}^{ac} \\
&+ \mathbf{P}(ij)\sum_{c}\langle ab||cj\rangle t_{i}^{c} - \mathbf{P}(ab)\sum_{k}\langle kb||ij\rangle t_{k}^{a} \\
&+ \frac{1}{2}\mathbf{P}(ij)\mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{ik}^{ac}t_{lj}^{db} + \frac{1}{4}\sum_{klcd}\langle kl||cd\rangle t_{ij}^{cd}t_{kl}^{ab} \\
&- \frac{1}{2}\mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{ij}^{ac}t_{kl}^{bd} - \frac{1}{2}\mathbf{P}(ij)\sum_{klcd}\langle kl||cd\rangle t_{ik}^{ab}t_{jl}^{cd} \\
&+ \frac{1}{2}\mathbf{P}(ab)\sum_{kl}\langle kl||ij\rangle t_{k}^{a}t_{l}^{b} + \frac{1}{2}\mathbf{P}(ij)\sum_{cd}\langle ab||cd\rangle t_{i}^{c}t_{j}^{d} \\
&- \mathbf{P}(ij)\mathbf{P}(ab)\sum_{kc}\langle kb||ic\rangle t_{k}^{a}t_{j}^{c} + \mathbf{P}(ab)\sum_{kc} f_{kc}t_{k}^{a}t_{ij}^{bc} \\
&+ \mathbf{P}(ij)\sum_{kc} f_{kc}t_{i}^{c}t_{jk}^{ab} - \mathbf{P}(ij)\sum_{klc}\langle kl||ci\rangle t_{k}^{c}t_{lj}^{ab} \\
&+ \mathbf{P}(ab)\sum_{kcd}\langle ka||cd\rangle t_{k}^{c}t_{ij}^{db} + \mathbf{P}(ij)\mathbf{P}(ab)\sum_{kcd}\langle ak||dc\rangle t_{i}^{d}t_{jk}^{bc} \\
&+ \mathbf{P}(ij)\mathbf{P}(ab)\sum_{klc}\langle kl||ic\rangle t_{l}^{a}t_{jk}^{bc} + \frac{1}{2}\mathbf{P}(ij)\sum_{klc}\langle kl||cj\rangle t_{i}^{c}t_{kl}^{ab} \\
&- \frac{1}{2}\mathbf{P}(ab)\sum_{kcd}\langle kb||cd\rangle t_{k}^{a}t_{ij}^{cd} - \frac{1}{2}\mathbf{P}(ij)\mathbf{P}(ab)\sum_{kcd}\langle kb||cd\rangle t_{i}^{c}t_{k}^{a}t_{j}^{d} \\
&+ \frac{1}{2}\mathbf{P}(ij)\mathbf{P}(ab)\sum_{klc}\langle kl||cj\rangle t_{i}^{c}t_{k}^{a}t_{l}^{b} - \mathbf{P}(ij)\sum_{klcd}\langle kl||cd\rangle t_{k}^{c}t_{i}^{d}t_{lj}^{ab} \\
&- \mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{k}^{c}t_{l}^{a}t_{ij}^{db} + \frac{1}{4}\mathbf{P}(ij)\sum_{klcd}\langle kl||cd\rangle t_{i}^{c}t_{j}^{d}t_{kl}^{ab} \\
&+ \frac{1}{4}\mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{k}^{a}t_{l}^{b}t_{ij}^{cd} + \mathbf{P}(ij)\mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{i}^{c}t_{l}^{b}t_{kj}^{ad} \\
&+ \frac{1}{4}\mathbf{P}(ij)\mathbf{P}(ab)\sum_{klcd}\langle kl||cd\rangle t_{i}^{c}t_{k}^{a}t_{j}^{d}t_{l}^{b}.
\end{aligned} \tag{5.103}
$$

See Ref.[26] for the full derivation.

5.4 Introducing denominators

The expressions for t_i^a and t_{ij}^{ab} are complex and it is not easy to understand how to implement these equations effectively. The rest of this chapter and the next will be dedicated to simplifying Eqs. (5.95) and (5.103).

5.4.1 t_i^a

Eq. (5.95) should be rewritten considerably before it is programmable. Starting with a definition of D_i^a

$$D_i^a \equiv f_{ii} - f_{aa}, \tag{5.104}$$

and remembering Eq. (5.95) we know it starts like this

$$0 = f_{ai} + \sum_c f_{ac} t_i^c - \sum_k f_{ki} t_k^a + \sum_{kc} \langle ka||ci\rangle t_k^c + \sum_{kc} f_{kc} t_{ik}^{ac} + \dots. \tag{5.105}$$

The term $\sum_c f_{ac} t_i^c$ can be rewritten as

$$\sum_c f_{ac} t_i^c = f_{aa} t_i^a + \sum_c (1 - \delta_{ca}) f_{ac} t_i^c. \tag{5.106}$$

Doing the same with the term $\sum_k f_{ki} t_k^a$ and inserting in Eq. (5.105) we get

$$0 = f_{ai} + f_{aa} t_i^a + \sum_c (1 - \delta_{ca}) f_{ac} t_i^c - f_{ii} t_i^a - \sum_k (1 - \delta_{ki}) f_{ki} t_k^a + \dots$$

The two terms $f_{aa} t_i^a$ and $f_{ii} t_i^a$ are combined using the definition Eq. (5.104)

$$f_{aa} t_i^a - f_{ii} t_i^a = -D_i^a t_i^a. \tag{5.107}$$

This is inserted into Eq. (5.95), and moved to the other side of the equation resulting in

$$D_i^a t_i^a = f_{ai} + \sum_c (1 - \delta_{ca}) f_{ac} t_i^c - \sum_k (1 - \delta_{ki}) f_{ki} t_k^a + \dots \tag{5.108}$$

If we perform the same procedure for t_{ij}^{ab} we can solve this iteratively until self consistency is reached.

5.4.2 t_{ij}^{ab}

We also implement a denominator D_{ij}^{ab} in Eq. (5.103), namely

$$D_{ij}^{ab} \equiv f_{ii} + f_{jj} - f_{aa} - f_{bb} \tag{5.109}$$

Here we want to create the term $D_{ij}^{ab} t_{ij}^{ab}$. This is done with the same procedure with the two terms $\mathbf{P}(ab) \sum_c f_{bc} t_{ij}^{ac} - \mathbf{P}(ij) \sum_k f_{kj} t_{ik}^{ab}$ from Eq. (5.103). These two terms can be expressed in a different manner

$$\begin{aligned}\mathbf{P}(ab)\sum_c f_{bc}t_{ij}^{ac} - \mathbf{P}(ij)\sum_k f_{kj}t_{ik}^{ab} =& f_{aa}t_{ij}^{ab} + f_{bb}t_{ij}^{ab} + \mathbf{P}(ab)\sum_c (1-\delta_{bc})f_{bc}t_{ij}^{ac} \\ & - \mathbf{P}(ij)\sum_k (1-\delta_{kj})f_{kj}t_{ik}^{ab} - f_{ii}t_{ij}^{ab} - f_{jj}t_{ij}^{ab}.\end{aligned}$$

Then the four terms where no sums are present are combined into $D_{ij}^{ab}t_{ij}^{ab}$ and moved to the other side of the equation. This leaves another problem which we can solve iteratively, that is

$$D_{ij}^{ab}t_{ij}^{ab} = \langle ab||ij\rangle + +\mathbf{P}(ab)\sum_c (1-\delta_{bc})f_{bc}t_{ij}^{ac} - \dots$$

Here the three dots represent the rest of Eq. (5.103).

5.4.3 Initial guess

The initial guess from where to start the iterative process can be anything. However it is common to start an initial guess where all the amplitudes on the right side are 0. This gives us

$$t_i^a = \frac{f_{ai}}{D_i^a}. \tag{5.110}$$

and

$$t_{ij}^{ab} = \frac{\langle ab||ij\rangle}{D_{ij}^{ab}}. \tag{5.111}$$

However it is also common to simply guess $t_i^a = 0$. We will make this initial guess when benchmarking the number of iterations needed.

The iterative procedure is one where t_i^a and t_{ij}^{ab} are updated simultaneously, and in theory we have converged once these amplitudes stop changing. However in practice we define a convergence criteria. This is then compared to the change in energy each iteration with the newly updated amplitudes. We then define convergence to when the energy stop changing, which should for all intents and purposes be an equivalent criteria.

5.5 Variational Principle

The energy expression in CC contains the operator $e^{-\mathbf{T}}\mathbf{H}e^{\mathbf{T}}$, which is not Hermitian. This means the variational principle no longer applies. It is possible to use the variational principle with CC theory, but this implies a huge complication. This also means it is possible with coupled cluster theory to get energies lower than the true ground state energy.

Chapter 6

CCSD Factorization

In this chapter we will find an algorithm for a serial program to solve these equations using the RHF basis. A serial program is one which does not run in parallel. A lot of work has been done to reduce the number of calculations needed for each iteration. Most of this work involves the definition of intermediate variables that can be calculated separately. We can simplify the CCSD equations considerably through the use of these intermediates.

The factorization is based on the work of J. F. Stanton and J. Gauss and is listed in an article by E. Solomonik et al. [17]. However, at the time of this thesis there where some typos in the factorized equations, so we will repeat the calculations. Also [19] holds information on factorization. Additional information on symmetry aspects is found in an article by P. Carsky, [18]. H. Koch et al. also published a paper on CCSD using AOs in the equation, and partly avoiding the AO to MO transformation, [20]. This is an alternative to our solution.

One simplification we have already discussed is the introduction of D_i^a and D_{ij}^{ab}. These denominators are independent of the amplitudes, hence they can be calculated and stored outside any iterative procedure.

6.1 Constructing an algorithm

When constructing this CCSD serial algorithm we need to keep a few things in mind. First we wish to minimize the number of floating point operations per second (FLOPS). Second we want to utilize external linear algebra libraries to perform the calculations. Third we want the option to easily modify the algorithm to work in parallel. This third option brings up a few new concerns. One of which is that effective parallel programs are those that minimize the communication between nodes, in part by symmetry considerations. This issue will be dealt with in a later chapter in greater detail.

6.1.1 Inserting denominators

t_i^a

First we insert D_i^a into Eq. (5.90). We also insert our notation for $\langle ab||ij\rangle$ which is

$$I_{ij}^{ab} = \langle ab||ij\rangle. \tag{6.1}$$

This gives

$$\begin{aligned}
D_i^a t_i^a =& f_{ai} + \sum_{a\neq c} f_{ac}t_i^c - \sum_{i\neq k} f_{ki}t_k^a + \sum_{kc} I_{ka}^{ci}t_k^c + \sum_{kc} f_{kc}t_{ik}^{ac} \\
&+ \frac{1}{2}\sum_{kcd} I_{ka}^{cd}t_{ki}^{cd} - \frac{1}{2}\sum_{klc} I_{kl}^{ci}t_{kl}^{ca} - \sum_{kc} f_{kc}t_i^c t_k^a - \sum_{klc} I_{kl}^{ci}t_k^c t_l^a \\
&+ \sum_{kcd} I_{ka}^{cd}t_k^c t_i^d - \sum_{klcd} I_{kl}^{cd}t_k^c t_i^d t_l^a + \sum_{klcd} I_{kl}^{cd}t_k^c t_{li}^{da} \\
&- \frac{1}{2}\sum_{klcd} I_{kl}^{cd}t_{ki}^{cd}t_l^a - \frac{1}{2}\sum_{klcd} I_{kl}^{cd}t_{kl}^{ca}t_i^d.
\end{aligned} \tag{6.2}$$

The calculation of t_i^a scales as n^6.

t_{ij}^{ab}

We repeat the same procedure for the t_{ij}^{ab} amplitudes. We insert D_{ij}^{ab} and combine terms into the same sums and obtain

$$
\begin{aligned}
D_{ij}^{ab}t_{ij}^{ab} =& I_{ab}^{ij} + \frac{1}{2}\sum_{kl} I_{kl}^{ij}t_{kl}^{ab} + \frac{1}{2}\sum_{cd} I_{ab}^{cd}t_{ij}^{cd} + \frac{1}{4}\sum_{klcd} I_{kl}^{cd}t_{ij}^{cd}t_{kl}^{ab} \\
& - \sum_{k\neq j} f_{kj}t_{ik}^{ab} + \sum_{k\neq i} f_{ki}t_{jk}^{ab} + \sum_{c\neq b} f_{bc}t_{ij}^{ac} - \sum_{c\neq a} f_{ac}t_{ij}^{bc} \\
& + \mathbf{P}(ab)\{\sum_{kc} f_{kc}t_{k}^{a}t_{ij}^{bc} - \sum_{k} I_{kb}^{ij}t_{k}^{a} + \frac{1}{2}\sum_{kl} I_{kl}^{ij}t_{k}^{a}t_{l}^{b} \\
& + \sum_{kcd}(I_{ka}^{cd}t_{k}^{c}t_{ij}^{db} - \frac{1}{2}I_{kb}^{cd}t_{k}^{a}t_{ij}^{cd}) + \sum_{klcd} I_{kl}^{cd}(\frac{1}{4}t_{k}^{a}t_{l}^{b}t_{ij}^{cd} - t_{k}^{c}t_{l}^{a}t_{ij}^{db} - \frac{1}{2}t_{ij}^{ac}t_{kl}^{bd})\} \\
& + \mathbf{P}(ij)\{\sum_{c} I_{ab}^{cj}t_{i}^{c} + \frac{1}{2}\sum_{cd} I_{ab}^{cd}t_{i}^{c}t_{j}^{d} + \sum_{kc} f_{kc}t_{i}^{c}t_{jk}^{ab} \\
& + \sum_{klc}(\frac{1}{2}I_{kl}^{cj}t_{i}^{c}t_{kl}^{ab} - I_{kl}^{ci}t_{k}^{c}t_{lj}^{ab}) + \sum_{klcd} I_{kl}^{cd}(\frac{1}{4}t_{i}^{c}t_{j}^{d}t_{kl}^{ab} - t_{k}^{c}t_{i}^{d}t_{lj}^{ab} - \frac{1}{2}t_{ik}^{ab}t_{jl}^{cd})\} \\
& + \mathbf{P}(ab)\mathbf{P}(ij)\{\sum_{kc}(I_{kb}^{cj}t_{ik}^{ac} - I_{kb}^{ic}t_{k}^{a}t_{j}^{c}) + \sum_{kcd}(I_{ak}^{dc}t_{i}^{d}t_{jk}^{bc} - I_{kb}^{cd}t_{i}^{c}t_{k}^{a}t_{j}^{d}) \\
& + \sum_{klc}(\frac{1}{2}I_{kl}^{cj}t_{i}^{c}t_{k}^{a}t_{l}^{b} + I_{kl}^{ic}t_{l}^{a}t_{jk}^{bc}) + \sum_{klcd} I_{kl}^{cd}(t_{i}^{c}t_{l}^{b}t_{kj}^{ad} + \frac{1}{4}t_{i}^{c}t_{k}^{a}t_{j}^{d}t_{l}^{b} + \frac{1}{2}t_{ik}^{ac}t_{lj}^{db})\}.
\end{aligned}
\tag{6.3}
$$

Calculating this scales as n^8 and can go faster with the definition of intermediates. Much of the material presented is based on the work of John F. Stanton and Jurgen Gauss, Ref. [17].

6.1.2 $[W_1]$

We first factor out and rewrite the following terms that as they stand now are calculated for all a, b, i, j but only change when i or j changes

$$
\begin{aligned}
& \frac{1}{2}\sum_{kl} I_{kl}^{ij}t_{kl}^{ab} + \frac{1}{2}\mathbf{P}(ij)\sum_{klc} I_{kl}^{cj}t_{i}^{c}t_{kl}^{ab} + \mathbf{P}(ij)\frac{1}{4}\sum_{klcd} I_{kl}^{cd}t_{i}^{c}t_{j}^{d}t_{kl}^{ab} + \frac{1}{4}\sum_{klcd} I_{kl}^{cd}t_{ij}^{cd}t_{kl}^{ab} \\
=& \frac{1}{2}\sum_{kl} t_{kl}^{ab}\left[I_{kl}^{ij} + \sum_{c}\left(I_{kl}^{cj}t_{i}^{c} - I_{kl}^{ci}t_{j}^{c} + \frac{1}{2}\sum_{d} I_{kl}^{cd}(t_{i}^{c}t_{j}^{d} - t_{j}^{c}t_{i}^{d} + t_{ij}^{cd})\right)\right] \\
=& \frac{1}{2}\sum_{kl} t_{kl}^{ab}[W_1]_{ij}^{kl},
\end{aligned}
\tag{6.4}
$$

resulting in the new intermediate

$$
[W_1]_{ij}^{kl} = I_{kl}^{ij} + \sum_{c}\left(I_{kl}^{cj}t_{i}^{c} - I_{kl}^{ci}t_{j}^{c} + \frac{1}{2}\sum_{d} I_{kl}^{cd}(t_{i}^{c}t_{j}^{d} - t_{j}^{c}t_{i}^{d} + t_{ij}^{cd})\right). \tag{6.5}
$$

The calculation of this intermediate scales as n^6. The term $[W_1]$ appears once again in our equations. These terms can also be combined and factorized and give

$$\begin{aligned}
&\mathbf{P}(ab)\frac{1}{2}\sum_{kl} I^{ij}_{kl} t^a_k t^b_l + \frac{1}{2}\sum_{klc} \mathbf{P}(ij)\mathbf{P}(ab) I^{cj}_{kl} t^c_i t^a_k t^b_l \\
&+ \frac{1}{4}\mathbf{P}(ab)\mathbf{P}(ij)\sum_{klcd} I^{cd}_{kl} t^c_i t^a_k t^d_j t^b_l + \mathbf{P}(ab)\sum_{klcd} I^{cd}_{kl}\frac{1}{4} t^a_k t^b_l t^{cd}_{ij} \\
=&\frac{1}{2}\sum_{kl}(t^a_k t^b_l - t^b_k t^a_l)\left[I^{ij}_{kl} + \sum_c\left(\mathbf{P}(ij) I^{cj}_{kl} t^c_i + \frac{1}{2}\sum_d I^{cd}_{kl}(t^c_i t^d_j - t^c_j t^d_i + t^{cd}_{ij}\right)\right] \\
=&\frac{1}{2}\sum_{kl}(t^a_k t^b_l - t^b_k t^a_l)[W_1]^{kl}_{ij}.
\end{aligned} \tag{6.6}$$

6.1.3 $[W_2]$

Now we combine these terms

$$\begin{aligned}
&-\mathbf{P}(ab)\sum_k t^a_k I^{ij}_{kb} - \mathbf{P}(ab)\frac{1}{2}\sum_{kcd} I^{cd}_{kb} t^{cd}_{ij} t^a_k - \mathbf{P}(ab)\mathbf{P}(ij)\sum_{kc} t^a_k I^{ic}_{kb} t^c_j \\
&-\mathbf{P}(ab)\mathbf{P}(ij)\sum_{kcd} I^{cd}_{kb} t^c_i t^d_j t^a_k \\
=&-\mathbf{P}(ab)\sum_k t^a_k\left[I^{ij}_{kb} + \sum_c\left(I^{ic}_{kb} t^c_j - I^{jc}_{kb} t^c_i + \frac{1}{2}\sum_d I^{cd}_{kb}(t^{cd}_{ij} + t^c_i t^d_j - t^c_j t^d_i)\right)\right] \\
=&-\mathbf{P}(ab)\sum_k t^a_k [W_2]^{kb}_{ij} \\
=&-\mathbf{P}(ab)\sum_k t^b_k [W_2]^{ak}_{ij}.
\end{aligned} \tag{6.7}$$

We have now defined another intermediate.

$$[W_2]^{ak}_{ij} = I^{ij}_{ak} + \sum_c\left(I^{ic}_{ak} t^c_j - I^{jc}_{ak} t^c_i + \frac{1}{2}\sum_d I^{cd}_{ak}(t^{cd}_{ij} + t^c_i t^d_j - t^c_j t^d_i)\right). \tag{6.8}$$

This term scales as n^6.

6.1.4 $[W_3]$

Our next intermediate is defined by these two terms

$$\mathbf{P}(ab)\mathbf{P}(ij)\sum_{klc} I^{ic}_{kl} t^a_l t^{bc}_{jk} + \mathbf{P}(ab)\mathbf{P}(ij)\sum_{klcd} I^{cd}_{kl} t^c_i t^b_l t^{ad}_{kj}. \tag{6.9}$$

Since c and d are arbitrary indices we can relabel the second term. We can also use a trick with the permutation operators where

$$\mathbf{P}(ab)\mathbf{P}(ij)f(a,b,i,j) = -\mathbf{P}(ab)\mathbf{P}(ij)f(b,a,i,j), \tag{6.10}$$

and also the symmetry of I where

$$I_{kl}^{dc} = -I_{kl}^{cd}. \tag{6.11}$$

We pull the permutation operators outside a parenthesis in Eq. (6.9) and obtain

$$= \mathbf{P}(ab)\mathbf{P}(ij)\left(\sum_{klc} I_{kl}^{ic} t_l^a t_{jk}^{bc} + \sum_{klcd} I_{kl}^{cd} t_i^c t_l^a t_{kj}^{bc}\right). \tag{6.12}$$

Here we insert some more symmetry considerations to get

$$= \mathbf{P}(ab)\mathbf{P}(ij)\left[\sum_{klc} t_l^a \left(-I_{kl}^{ci} t_{jk}^{bc} - \sum_{d} I_{kl}^{cd} t_i^c t_{jk}^{bc}\right)\right]. \tag{6.13}$$

We factorize out as such

$$\begin{aligned} =& -\mathbf{P}(ab)\mathbf{P}(ij)\left[\sum_{klc} t_{jk}^{bc} t_l^a \left(I_{kl}^{ci} + \sum_{d} I_{kl}^{cd} t_i^c\right)\right] \\ =& -\mathbf{P}(ab)\mathbf{P}(ij)\sum_{klc} t_{jk}^{bc} t_l^a [W_3]_{ci}^{kl}. \end{aligned} \tag{6.14}$$

We then obtain the new intermediate

$$[W_3]_{ci}^{kl} = I_{kl}^{ci} + \sum_{d} I_{kl}^{cd} t_i^c. \tag{6.15}$$

This term scales as n^5.

6.1.5 $[F_1]$

Until now all intermediates have been four dimensional ones. Now we define our first two dimensional intermediate. This is defined from the terms

$$\begin{aligned}
&\mathbf{P}(ab)\sum_{kc} f_{kc}t_k^a t_{ij}^{bc} - \mathbf{P}(ab)\sum_{klcd} I_{kl}^{cd} t_k^c t_l^a t_{ij}^{db} \\
=&\mathbf{P}(ab)\sum_{kc} f_{kc}t_k^a t_{ij}^{bc} + \mathbf{P}(ab)\sum_{klcd} I_{kl}^{cd} t_l^d t_k^a t_{ij}^{bc} \\
=&\mathbf{P}(ab)\sum_{kc} t_k^a t_{ij}^{bc}\left[f_{kc} + \sum_{ld} I_{kl}^{cd} t_l^d\right] \\
=&\mathbf{P}(ab)\sum_{kc} t_k^a t_{ij}^{bc}[F_1]_k^c. \qquad (6.16)
\end{aligned}$$

giving

$$[F_1]_k^c = f_{kc} + \sum_{ld} I_{kl}^{cd} t_l^d. \qquad (6.17)$$

This intermediate is also repeated when combining the terms

$$\begin{aligned}
&\mathbf{P}(ij)\sum_{kc} f_{kc}t_i^c t_{jk}^{ab} - \mathbf{P}(ij)\sum_{klcd} I_{kl}^{cd} t_k^c t_i^d t_{lj}^{ab} \\
=&-\mathbf{P}(ij)\sum_{kc} f_{kc}t_i^c t_{kj}^{ab} - \mathbf{P}(ij)\sum_{klcd} I_{kl}^{cd} t_l^d t_i^c t_{kj}^{ab} \\
=&-\mathbf{P}(ij)\sum_{kc} t_i^c t_{kj}^{ab}\left[f_{kc} + \sum_{cd} I_{kl}^{cd} t_l^d\right] \\
=&-\mathbf{P}(ij)\sum_{kc} t_i^c t_{kj}^{ab}[F_1]_k^c. \qquad (6.18)
\end{aligned}$$

This term scales as n^4. Inserting the intermediates defined for now into the equations for t_{ij}^{ab} results in the following

$$
\begin{aligned}
D_{ij}^{ab}t_{ij}^{ab} =& I_{ab}^{ij} + \frac{1}{2}\sum_{kl}(t_k^a t_l^b - t_k^b t_l^a + t_{kl}^{ab})[W_1]_{ij}^{kl} + \frac{1}{2}\sum_{cd} I_{ab}^{cd} t_{ij}^{cd} \\
& - \sum_{k\neq j} f_{kj} t_{ik}^{ab} + \sum_{k\neq i} f_{ki} t_{jk}^{ab} - \sum_{c\neq b} f_{bc} t_{ij}^{ac} + \sum_{c\neq a} f_{ac} t_{ij}^{bc} \\
& + \mathbf{P}(ab)\{-\sum_{k} t_k^b [W_2]_{ij}^{ak} + \sum_{kc} t_k^a t_{ij}^{bc}[F_1]_k^c + \frac{1}{2}\sum_{kl} I_{kl}^{ij} t_k^a t_l^b \\
& + \sum_{kcd}(I_{ka}^{cd} t_k^c t_{ij}^{db}) + \sum_{klcd} I_{kl}^{cd}(\frac{1}{4}t_k^a t_l^b t_{ij}^{cd} - \frac{1}{2}t_{ij}^{ac}t_{kl}^{bd})\} \\
& + \mathbf{P}(ij)\{-\sum_{kc} t_i^c t_{kj}^{ab}[F_1]_k^c + \sum_{c} I_{ab}^{cj} t_i^c + \frac{1}{2}\sum_{cd} I_{ab}^{cd} t_i^c t_j^d \\
& + \sum_{klc}(-I_{kl}^{ci} t_k^c t_{lj}^{ab}) + \sum_{klcd} I_{kl}^{cd}(-\frac{1}{2}t_{ik}^{ab}t_{jl}^{cd})\} \\
& + \mathbf{P}(ab)\mathbf{P}(ij)\{\sum_{kc}(I_{kb}^{cj} t_{ik}^{ac}) + \sum_{kcd}(I_{ak}^{dc} t_i^d t_{jk}^{bc}) \\
& + \sum_{klc}(\frac{1}{2}I_{kl}^{cj} t_i^c t_k^a t_l^b - t_{jk}^{bc} t_l^a [W_3]_{ci}^{kl}) + \sum_{klcd} I_{kl}^{cd}(\frac{1}{4}t_i^c t_k^a t_j^d t_l^b + \frac{1}{2}t_{ik}^{ac}t_{lj}^{db})\}.
\end{aligned} \tag{6.19}
$$

6.1.6 $[F_2]$

We will now reuse the $[F_1]$ and combine it with a few other terms

$$
\begin{aligned}
& \sum_{k\neq i} f_{ki} t_{jk}^{ab} - \sum_{k\neq j} f_{kj} t_{ik}^{ab} - \mathbf{P}(ij)\sum_{kc} t_i^c t_{kj}^{ab}[F_1]_k^c \\
& - \mathbf{P}(ij)\sum_{klcd} I_{kl}^{cd}\frac{1}{2}t_{ik}^{ab}t_{jl}^{cd} - \mathbf{P}(ij)\sum_{klc} I_{kl}^{ci} t_k^c t_{lj}^{ab}.
\end{aligned} \tag{6.20}
$$

We notice that the two terms $\sum_{k\neq i} f_{ki} t_{jk}^{ab}$ and $\sum_{k\neq j} f_{kj} t_{ik}^{ab}$ can be written in terms of $\mathbf{P}(ij)$ and δ_{ki} and we get

$$
\begin{aligned}
\Rightarrow =&\mathbf{P}(ij)\left[\sum_k (1-\delta_{ki}) f_{ki} t_{jk}^{ab} - \sum_{kc} t_i^c t_{kj}^{ab} [F_1]_k^c - \sum_{klcd} I_{kl}^{cd} \frac{1}{2} t_{ik}^{ab} t_{jl}^{cd} - \sum_{klc} I_{kl}^{ci} t_k^c t_{lj}^{ab}\right] \\
=&\mathbf{P}(ij)\left[\sum_k (1-\delta_{ki}) f_{ki} t_{jk}^{ab} + \sum_{kc} t_i^c t_{jk}^{ab} [F_1]_k^c + \sum_{klcd} I_{kl}^{cd} \frac{1}{2} t_{jk}^{ab} t_{il}^{cd} + \sum_{klc} I_{kl}^{ic} t_l^c t_{jk}^{ab}\right] \\
=&\mathbf{P}(ij)\sum_k t_{jk}^{ab}\left[(1-\delta_{ki}) f_{ki} + \sum_c t_i^c [F_1]_k^c + \frac{1}{2}\sum_{lcd} I_{kl}^{cd} t_{il}^{cd} + \sum_{lc} I_{kl}^{ic} t_l^c\right] \\
=&\mathbf{P}(ij)\sum_k t_{jk}^{ab} [F_2]_i^k,
\end{aligned}
\tag{6.21}
$$

where $[F_2]$ is defined as

$$
[F_2]_i^k = (1-\delta_{ki}) f_{ki} + \sum_c \left[t_i^c [F_1]_k^c + \sum_l \left(I_{kl}^{ic} t_l^c + \frac{1}{2} \sum_d I_{kl}^{cd} t_{il}^{cd} \right) \right]. \tag{6.22}
$$

This term scales as n^5.

6.1.7 $[F_3]$

We now combine terms within the $\mathbf{P}(ab)$ operator.

$$
\begin{aligned}
&-\sum_{c\neq b} f_{bc} t_{ij}^{ac} + \sum_{c\neq a} f_{ac} t_{ij}^{bc} - \mathbf{P}(ab)\sum_{kc} t_k^a t_{ij}^{bc} [F_1]_k^c - \mathbf{P}(ab)\sum_{klcd} \frac{1}{2} I_{kl}^{cd} t_{ij}^{ac} t_{kl}^{bd} \\
&+ \mathbf{P}(ab)\sum_{kcd} I_{ka}^{cd} t_k^c t_{ij}^{db} \\
=&\mathbf{P}(ab)\left[+\sum_c (1-\delta_{ca}) f_{ac} t_{ij}^{bc} - \sum_{kc} t_k^a t_{ij}^{bc} [F_1]_k^c - \sum_{klcd} \frac{1}{2} I_{kl}^{cd} t_{ij}^{ac} t_{kl}^{bd} + \sum_{kcd} I_{ka}^{cd} t_k^c t_{ij}^{db}\right] \\
=&\mathbf{P}(ab)\left[+\sum_c (1-\delta_{ca}) f_{ac} t_{ij}^{bc} - \sum_{kc} t_k^a t_{ij}^{bc} [F_1]_k^c - \sum_{klcd} \frac{1}{2} I_{kl}^{cd} t_{ij}^{bc} t_{kl}^{ad} + \sum_{kcd} I_{ka}^{dc} t_k^d t_{ij}^{bc}\right] \\
=&\mathbf{P}(ab)\sum_c t_{ij}^{bc}\left[(1-\delta_{ca}) f_{ac} + \sum_k \left(-t_k^a [F_1]_k^c + \sum_d \left(I_{ka}^{dc} t_k^d - \sum_l \frac{1}{2} I_{kl}^{cd} t_{kl}^{ad} \right)\right)\right] \\
=&\mathbf{P}(ab)\sum_c t_{ij}^{bc} [F_3]_c^a.
\end{aligned}
\tag{6.23}
$$

We define $[F_3]$ from these terms as

$$
[F_3]_c^a = (1-\delta_{ca}) f_{ac} - \sum_k \left[t_k^a [F_1]_k^c + \sum_d \left(I_{ka}^{cd} t_k^d - \frac{1}{2} \sum_l I_{kl}^{cd} t_{kl}^{ad} \right) \right]. \tag{6.24}
$$

This term scales as n^5.

6.1.8 $[W_4]$

Now we combine the terms inside $\mathbf{P}(ab)\mathbf{P}(ij)$

$$\begin{aligned}
&\mathbf{P}(ab)\mathbf{P}(ij)\sum_{kc} I_{kb}^{cj} t_{ik}^{ac} + \mathbf{P}(ab)\mathbf{P}(ij)\sum_{kcd} I_{ak}^{dc} t_i^d t_{jk}^{bc} \\
&- \mathbf{P}(ab)\mathbf{P}(ij)\sum_{klc} t_{jk}^{bc} t_l^a [W_3]_{ci}^{kl} + \mathbf{P}(ab)\mathbf{P}(ij)\sum_{klcd} I_{kl}^{cd} \frac{1}{2} t_{ik}^{ac} t_{lj}^{db} \\
=&\mathbf{P}(ab)\mathbf{P}(ij) t_{jk}^{bc} \left[\sum_{kc} I_{ka}^{ci} + \sum_{kcd} I_{ak}^{dc} t_i^d - \sum_{klc} t_l^a [W_3]_{ci}^{kl} + \frac{1}{2}\sum_{klcd} I_{kl}^{cd} t_{lj}^{db}\right] \\
=&\mathbf{P}(ab)\mathbf{P}(ij)\sum_{kc} t_{jk}^{bc} \left[I_{ak}^{ic} + \sum_{d} I_{ak}^{dc} t_i^d - \sum_{l} t_l^a [W_3]_{ci}^{kl} + \frac{1}{2}\sum_{ld} I_{kl}^{cd} t_{il}^{ad}\right]. \quad (6.25)
\end{aligned}$$

These are defined as $[W_4]$ and is calculated as

$$[W_4]_{ic}^{ak} = I_{ak}^{ic} + \sum_{d} I_{ak}^{dc} t_i^d - \sum_{l} t_l^a [W_3]_{ci}^{kl} + \frac{1}{2}\sum_{ld} I_{kl}^{cd} t_{il}^{ad}. \quad (6.26)$$

In this formulation symmetries where used extensively, the term scales as n^6.

6.1.9 Inserting intermediates

Inserting Eqs. (6.17), (6.22), (6.24), (6.8), (6.15), (6.26) and (6.5) then gives the following equation

$$\begin{aligned}
D_{ij}^{ab} t_{ij}^{ab} =& I_{ab}^{ij} + \frac{1}{2}\sum_{kl}(t_{kl}^{ab} + t_k^a t_l^b - t_l^a t_k^b)[W_1]_{ij}^{kl} - \mathbf{P}(ab)\sum_{k} t_k^b [W_2]_{ij}^{ak} \\
&+ \mathbf{P}(ij)\sum_{k} t_{jk}^{ab} [F_2]_i^k + \frac{1}{2}\sum_{cd} I_{ab}^{cd} t_{ij}^{cd} + \mathbf{P}(ab)\sum_{c} t_{ij}^{bc} [F_3]_c^a \\
&+ \mathbf{P}(ij)\sum_{c} I_{ab}^{cj} t_i^c + \mathbf{P}(ij)\frac{1}{2}\sum_{cd} I_{ab}^{cd} t_i^c t_j^d + \mathbf{P}(ab)\mathbf{P}(ij)\sum_{kc} t_{jk}^{bc} [W_4]_{ic}^{ak}.
\end{aligned} \quad (6.27)$$

We can also define an intermediate τ_{ij}^{ab},

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b - t_j^a t_i^b. \quad (6.28)$$

This will reduce the number of calculations required, but not by any factor of n. Hence using this is a debate of speed versus memory. With this intermediate the equation for $\mathbf{T}_2$ amplitudes look like this

$$\begin{aligned}D_{ij}^{ab}t_{ij}^{ab} =& I_{ab}^{ij} + \frac{1}{2}\sum_{kl}\tau_{kl}^{ab}[W_1]_{ij}^{kl} - \mathbf{P}(ab)\sum_{k}t_{k}^{b}[W_2]_{ij}^{ak}\\ &+ \mathbf{P}(ij)\sum_{k}t_{jk}^{ab}[F_2]_{i}^{k} + \frac{1}{2}\sum_{cd}I_{ab}^{cd}\tau_{ij}^{cd} + \mathbf{P}(ab)\sum_{c}t_{ij}^{bc}[F_3]_{c}^{a}\\ &+ \mathbf{P}(ij)\sum_{c}I_{ab}^{cj}t_{i}^{c} + \mathbf{P}(ab)\mathbf{P}(ij)\sum_{kc}t_{jk}^{bc}[W_4]_{ic}^{ak}. \qquad (6.29)\end{aligned}$$

6.1.10 Inserting into t_i^a

If we combine the terms from t_i^a in the following manner

$$\begin{aligned}D_i^a t_i^a =& -\sum_k t_k^a\left[(1-\delta_{ki})f_{ki} + \sum_c t_i^c f_{kc} + \sum_{lc} I_{kl}^{ic}t_l^c + \sum_{lcd} I_{kl}^{cd}(\frac{1}{2}t_{il}^{cd} + t_l^d)\right]\\ &+ f_{ai} + \sum_{c\neq a} f_{ac}t_i^c + \sum_{kc} I_{ka}^{ci}t_k^c - \frac{1}{2}\sum_{klc} t_{kl}^{ca}\left[I_{kl}^{ci} + \sum_d I_{kl}^{cd}t_i^d\right]\\ &+ \sum_{kc} t_{ik}^{ac}\left[f_{kc} + \sum_{ld} I_{kl}^{cd}t_l^d\right] + \frac{1}{2}\sum_{kcd} I_{ka}^{cd}t_{ki}^{cd} + \sum_{kcd} I_{ka}^{cd}t_k^c t_i^d. \qquad (6.30)\end{aligned}$$

They reduce to

$$\begin{aligned}D_i^a t_i^a =& -\sum_k t_k^a[F_2]_i^k + f_{ai} + \sum_{c\neq a} f_{ac}t_i^c + \sum_{kc} I_{ka}^{ci}t_k^c - \frac{1}{2}\sum_{klc} t_{kl}^{ca}[W_3]_{kl}^{ic}\\ &+ \sum_{kc} t_{ik}^{ac}[F_1]_k^c + \frac{1}{2}\sum_{kcd} I_{ka}^{cd}t_{ki}^{cd} + \sum_{kcd} I_{ka}^{cd}t_k^c t_i^d. \qquad (6.31)\end{aligned}$$

This equation scales as n^6. The largest scaling factor throughout our algorithm now is n^6 whereas prior to intermediates it was n^8. This is significantly faster, but we do have eight (or seven if τ_{ij}^{ab} is excluded) intermediates which must be stored. It should be noted that $t_k^c t_i^d = \frac{1}{2}(t_k^c t_i^d - t_i^c t_k^d)$. This can be used to insert τ_{ij}^{ab} in the equations for t_i^a and the energy, resulting in

$$\begin{aligned}D_i^a t_i^a =& -\sum_k t_k^a[F_2]_i^k + f_{ai} + \sum_{c\neq a} f_{ac}t_i^c + \sum_{kc} I_{ka}^{ci}t_k^c - \frac{1}{2}\sum_{klc} t_{kl}^{ca}[W_3]_{kl}^{ic}\\ &+ \sum_{kc} t_{ik}^{ac}[F_1]_k^c + \frac{1}{2}\sum_{kcd} I_{ka}^{cd}\tau_{ki}^{cd}. \qquad (6.32)\end{aligned}$$

6.2 SSLRS

The science team of Scuseria, Scheiner, Lee, Rice and Schaefer are usually refered to as SSLRS. This team has developed some of the most efficient algorithms for our purposes, see for example Ref. [65]. They have defined quite different intermediates, and I would like to also present their algorithm in short for comparison. If the reader wants to develop his/her own CCSD algorithm this would be a good alternative. See also Ref. [64] for further details.

6.2.1 Description of algorithm

In the SSLRS algorithm the amplitude equations are defined with intermediates as follows

$$
\begin{aligned}
-D_i^a t_i^a = & - f_{ai} - \sum_{c \neq a} f_{ac} t_i^c + \sum_{k \neq i} f_{ki} t_k^a + \sum_{kc} f_{kc}(2t_{ik}^{ac} - \tau_{ik}^{ca}) + \sum_k g_i^k t_k^a \\
& - \sum_c g_c^a t_i^c - \sum_{klc} \left(2[D_1]_{li}^{ck} - [D_1]_{ki}^{cl} \right) t_l^c t_k^a \\
& - \sum_{kc} [2(D_{2A} - D_{2B}) + D_{2C}]_{ci}^{ka} - 2 \sum_k [D_1]_{ik}^{ak} \\
& + \sum_{kc} v_{ic}^{ka} t_k^c - \sum_{klc} v_{cl}^{ak} (2t_{ki}^{cl} - t_{ik}^{cl}) + \sum_{ljc} v_{ci}^{jl} (2t_{lj}^{ac} - t_{jl}^{ac}),
\end{aligned}
\tag{6.33}
$$

$$
-D_{ij}^{ab} t_{ij}^{ab} = v_{ij}^{ab} + J_{ij}^{ab} + J_{ji}^{ba} + S_{ij}^{ab} + S_{ji}^{ba}, \tag{6.34}
$$

and

$$
E_{CCSD} = E_{HF} + 2 \sum_{ia} f_{ia} t_a^i + \sum_{abij} v_{ij}^{ab} (2\tau_{ij}^{ab} + \tau_{ji}^{ab}). \tag{6.35}
$$

The minus signs on the left side of the equation are present due to the fact that SSLRS use a different definition of the denominators, noted here in the equations are our definitions. Their definitions is equal to ours except for this minus sign.

The term v_{ij}^{ab} is a short notation used by SSLRS for $\langle ab||ij \rangle$. It should be mentioned that the brackets surrounding for example $[D_1]$ is a notation where the brackets are a part of the variable. The rest of the intermediates are now defined as

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b, \tag{6.36}$$

$$J_{ij}^{ab} = \sum_{c\neq a} f_{ca} t_{ij}^{cb} - \sum_{k\neq i} f_{ik} t_{kj}^{ab} + \sum_{kc} f_{kc}(t_{ij}^{cb} t_k^b + t_{ik}^{ab} t_j^c) + \sum_c g_c^b t_{ij}^{ac} - \sum_k g_j^k t_{ik}^{ab}, \tag{6.37}$$

and

$$\begin{aligned}
S_{ij}^{ab} =& \frac{1}{2}[B_2]_{ij}^{ab} - [E_1^*]_{ij}^{ab} + [D_{2A}]_{ab}^{ij} + [F_{12}]_{ab}^{ij} \\
&+ \sum_{kc}([D_{2A}]_{kj}^{cb} - [D_{2B}]_{kj}^{cb} + 2[F_{12}]_{kj}^{cb} - [E_1^*]_{cj}^{kb})(t_{ik}^{ac} - \frac{1}{2}t_{ki}^{ac}) \\
&+ \sum_{kc}(\frac{1}{2}[D_{2C}]_{kj}^{cb} - v_{kj}^{bc} - [F_{11}]_{kj}^{bc} + [E_{11}]_{kj}^{bc})t_{ic}^{ak} \\
&+ \sum_{kc}(\frac{1}{2}[D_{2C}]_{ki}^{cb} - v_{ki}^{bc} - [F_{11}]_{ki}^{bc} + [E_{11}]_{ki}^{bc})t_{cj}^{ak} \\
&+ \sum_{cd}(\frac{1}{2}[D_2]_{ij}^{cd} + \frac{1}{2}v_{ij}^{cd} + \frac{1}{2}([E_1]_{ij}^{cd} + [E_1]_{ji}^{dc}))\tau_{cd}^{ab} \\
&+ \sum_{kc}\{([D_{2C}]_{ci}^{kb} - v_{ci}^{bk})t_j^c - ([D_{2A}]_{cj}^{kb} - [D_{2B}]_{cj}^{kb})t_i^c - ([D_1]_{ji}^{bk} + [F_2]_{ji}^{bk})\}t_k^a.
\end{aligned} \tag{6.38}$$

The definition of g depends on which index is where. Remembering **a** is an index indicating an occupied orbital and **i** is an index indicating a Fermi hole.

$$g_i^a = 2\sum_b [F_{11}]_{ib}^{ab} - \sum_b [F_{12}]_{ib}^{ba} - \sum_b [D_{2A}]_{ib}^{ba} + \sum_{bc}([D_1]_{bc}^{ic} - 2[D_1]_{cb}^{ib})t_c^a \tag{6.39}$$

$$g_a^i = \sum_c \{2([E_1]_{ic}^{ac} + [D_2]_{ic}^{ac}) - ([E_1]_{ic}^{ca} + [D_2]_{ic}^{ca})\}. \tag{6.40}$$

The rest of the intermediates in the SSLRS algorithm is now defined in order of appearance.

$$[D_1]_{ij}^{ab} = \sum_k v_{ik}^{ab} t_j^k. \tag{6.41}$$

$$[D_{2A}]_{ij}^{ab} = \sum_{kc} v_{ci}^{ka}(2t_{jc}^{ka} - t_{jc}^{ak}). \tag{6.42}$$

$$[D_{2B}]_{ij}^{ab} = \sum_{kc} v_{ci}^{ak} t_{jc}^{ka}. \tag{6.43}$$

$$[D_{2C}]_{ij}^{ab} = \sum_{kc} v_{ci}^{ak} t_{jc}^{ak}. \tag{6.44}$$

$$[B_2]_{ij}^{ab} = \sum_{kl} v_{kl}^{ab} t_{ij}^{kl}. \tag{6.45}$$

$$[E_1^*]_{ij}^{ab} = \sum_{k} v_{ij}^{ak} t_{k}^{b}. \tag{6.46}$$

$$[F_{12}]_{ij}^{ab} = \sum_{c} v_{ic}^{ab} t_{j}^{c}. \tag{6.47}$$

$$[F_{11}]_{ij}^{ab} = \sum_{c} v_{ic}^{ba} t_{j}^{c}. \tag{6.48}$$

$$[E_{11}]_{ij}^{ab} \sum_{c} v_{ij}^{cb} t_{c}^{a}. \tag{6.49}$$

$$[D_2]_{ij}^{ab} = \sum_{cd} v_{cd}^{ab} t_{ij}^{cd}. \tag{6.50}$$

$$[E_1]_{ij}^{ab} = \sum_{c} v_{ic}^{ab} t_{j}^{c}. \tag{6.51}$$

$$[F_2]_{ij}^{ab} = \sum_{cd} v_{cd}^{ab} \tau_{ij}^{cd}. \tag{6.52}$$

6.2.2 Scaling

Specific notice should be paid to $[D_{2A}]$, $[D_{2B}]$, $[D_{2C}]$ and $[F_2]$. These intermediates scale as with a factor of n^6, where n is the number of orbitals. However it is not necessary to loop over all orbitals with indices i, j, k referring to unoccupied orbitals. The indices a, b, c refer to occupied orbitals. The scaling then reduces to $n_v^3 n_o^3$. Overall Eqs. (6.33), (6.34) and (6.35) scale as

$$\frac{1}{2} n_v^4 n_o^2 + 7 n_v^3 n_o^3 + \left(\frac{1}{2} + \frac{1}{2}\right) n_v^2 n_o^4. \tag{6.53}$$

It should be noted that SSLRS does propose this algorithm with the purpose of not only low scaling, but also to avoid storage of a large number of variables. In their papers they do mention another algorithm with slightly better scaling, but they do argue the extra needed variable storage this requires is a worrying aspect. For this reason the SSLRS algorithm is presented like this in this thesis.

Another positive aspect of this algorithm is the emphasis on matrix operations. All the n^6 and n^5 scaling parts of the algorithm is designed so that matrix multiplication libraries can be used. These libraries are often specially designed to be efficient.

The SSLRS algorithm also scales as n^6, but the number of intermediates are much larger. For this reason we chose the prior algorithm.

6.3 TCE

As we have seen there are more than one way to factorize the CCSD equations. The best factorization actually depends on the system of interest. This has prompted the interest in the Tensor Contracted Engine, TCE.

TCE is an automated code generator for computational chemistry methods. Once the system of interest is known, the TCE aims to construct the optimal code. We will not go in detail on TCE, but additional information can be found in Ref.[57].

Chapter 7

Comments Prior to Implementation

In this chapter we discuss a few external libraries used in the implementation and how they work. Also we will discuss a few guiding principles we will apply to our implementation later. We will mainly discuss armadillo, MPI and general parallel programming. We will also mention OpenMP and external math libraries. The external math library we will use is Intel MKL. References are provided inside each section.

7.1 Armadillo

Armadillo [30] is a C++ linear algebra library. The library is designed to be similar to Matlab, Ref. [?], in syntax. It provides good speed relative to other libraries and makes it easy to utilize matrix or vector multiplications in an efficient way. The armadillo documentation is available in [30]. Armadillo is also available for other programming languages, but we will strictly focus on the C++ version.

7.1.1 Armadillo Types

Armadillo has its own objects. We will use four objects in armadillo. The first three are vector, matrix and cube. These are simply put one, two and three dimensional arrays defined by standard to contain numerical values of double precision. The last is a field. A field in armadillo is a two dimensional array that can contain other things than numerical values. A field can contain things like strings, vectors, matrices or cubes. Anything that can be used in combination with the "=" operator and a copy, like memcpy(). A field can be defined like this:

```
field<mat> A;
```

This defines a two dimensional field of matrices, meaning a four dimensional array. The field is called "A". This can be accessed by first two indexes for the field and next two indexes for the matrix. The element A(0,1) for example is the matrix element with indices 0 and 1 in the field. The quantity A(0,1)(2,3) is a double precision number with indices 2,3 in the matrix located in indices 0,1 in the field A.

7.1.2 Matrix Operations

Matrix multiplication is very easy to implement with armadillo. If there are three matrices defined, A, B and C, we can easily call on matrix multiplication by stating:

```
C = A * B
```

Other operations available are additions, subtractions, element wise multiplications and element wise divisions. These are accessed in order like this:

```
C = A + B;
C = A - B;
C = A % B;
C = A / B;
```

Element wise multiplications means that an element in matrix C is calculated as such

$$C(i,j) = A(i,j) \times B(i,j)$$

There is also a useful function called accu(C). This is an accumulation function that adds together all the terms in C.

$$accu(C) = \sum_{ij} C(i,j) \tag{7.1}$$

, where C can be a vector, matrix or a cube. We can combine the accumulation function with element wise multiplication.

```
D = accu(A % B)
```

This produce a double precision value. We can use the combination of element wise multiplication with accumulation in the term

$$D_{ij}^{ab} t_{ij}^{ab} \leftarrow \sum_{cd} I_{ab}^{cd} \tau_{ij}^{cd}. \tag{7.2}$$

If we store I as a field with indexing I(a,b)(c,d) and τ_{ij}^{cd} is stored as a field with indexing τ(i,j)(c,d) then we can use element wise multiplication and accumulation to calculate Eq. (7.2).

$$\sum_{cd} I_{ab}^{cd}\tau_{ij}^{cd} = accu(I(a,b)\%\tau(i,j)). \tag{7.3}$$

Here $I(a,b)$ is a matrix and $\tau(i,j)$ is a matrix of same size. A major positive side of armadillo is that it is possible to link other effective external mathematical libraries to perform the actual matrix operations. We can link BLAS, MKL, OpenBLAS, [52, 51], and many others. Armadillo initiates calls to these libraries automatically and effectively if installed properly. We then get the effectiveness of the best external mathematical libraries, and the simplicity of the armadillo syntax.

7.1.3 Element access

An interesting feature when using armadillo is the way we access elements. In C syntax one usually allocates an array using malloc(). This gives great control over memory access, as we can have even multidimensional arrays sequential in memory.

In armadillo we usually allocate a matrix with the statement $matA$. We can have a field of matrices with $field < mat >$. However each element in the field must be allocated on its own. Also armadillo has a few checks in place to ensure a bug free working code. Based on performance and experience, this is not efficient. The most effective code is one that does not check for bugs at all.

Other types in armadillo such at *mat* or *vector* is quite efficient. When using this library it is important to be aware that not all types in armadillo are as efficient when it comes to memory access. Usually it is best to stick with one- or two-dimensional arrays as much as possible, even make temporary vectors or matrices to avoid accessing a field to much. Trial and error is thus a good tool if we want to use armadillo for high performance computing. This comment applies to armadillo at the time of this thesis. The armadillo developers are continuously working to improve the performance.

Another problem we encountered in our implementation is that OpenMPI does not take armadillo types in its communication functions. Armadillo does have functions that can help modestly in this regard, like the function .memptr(). However, we found it was not an optimal combination.

7.2 Parallel Computing and OpenMPI

The Open Source Message Passing Interface, OpenMPI, will be used in our implementation, Ref. [48]. OpenMPI is a library that makes parallel computing much easier. It removes the need for low level parallel programming.

In this section we will discuss briefly why we need parallel computing and what it is.

7.2.1 The CPU

The CPU, or the Central Processing Unit, is the brain of the computer. This unit processes instructions, many of which require transfer from or to the memory on a computer.

The CPU integrates many components, such as registers, FPUs and caches. The CPU has a "clock" that synchronizes the logic units within the CPU each clock cycle to process instructions. Among other things this clock allows us to accurately measure the time used from one section of the program to another.

Instructions are put in a pipeline for the CPU to execute. While the CPU is processing instructions, it also looks down the pipeline, to see what instructions it will need to perform soon and what values it will need. These values can then be pre-emptively placed in the cache. In the cache they are faster to access when they are needed.

If the CPU makes a wrong guess on for example an if test, the pipeline is filled with instructions that should not be processed. This also means the cache is filled with useless values. We call this a pipeline flush. This slows down performance.

A supercomputer consists of nodes. Each node has a number of CPUs. On the Abel super computing cluster at the University of Oslo, each node has 16 CPUs.

7.2.2 The Compiler

The compiler allows the CPU to understand easy syntax such as C++. The compiler takes the code as input and produce the .o file. In the .o file there are instructions for the CPU to process. The CPU only understands the .o file, as such we must always compile our code. The compiler also creates the pipeline. We want the pipeline to be as optimal as possible, for this reason we need a good compiler.

A normal compiler performs a three step procedure. Step one is to check the code for syntax errors, include problems and other basics.

Step two is to translate the code into an intermediate language. Here optimizations are performed. If we wrote a code segment like this

```
double A, B, C;
A = 50;
B = 20;
C = B * B * B;

// More calculations

B *= A;
```

The compiler will take note that the variable A is defined early on, and used much later. The quantity A is defined, stored into memory, then read from memory and finally it takes part in calculations. The compiler can rearrange our code to optimize this segment

```
double B, C;
B = 20;
C = B * B * B;

// More calculations

double A = 50;
B *= A;
```

Here the variable A is defined and used directly. It is defined were we need it and ready in the pipeline for calculations. If the pipeline for some reason is filled with wrong instructions, we will not take advantage of optimizations such as this.

Step three is to output the .o file. Compilers are extremely complex, we should mention this was a brief and simplified description.

7.2.3 Data

Data is stored in memory as a sequence of 0s and 1s. One 0 or 1 occupies one bit. 8 bits is one byte. The memory is read as bytes. Even a bool variable which can either be true or false is one byte. A bool of value true is stored in memory as 00000001.

Other types of data have different sizes in memory. An int is 4 bytes, a float 4 bytes and a double 8 bytes. Data is usually stored in memory, or rapid access memory, RAM. Here we can access it faster than from disk.

On a supercomputer, each node has a fixed number of memory available. The CPUs on the node can share this memory, or we can distribute it into smaller chunks were each CPU has its own unshared memory.

7.2.4 Bandwidth

The bandwidth is a measure of number of bytes transferred per second. The bandwidth is a feature of the hardware, we will look at it as a constant value.

If we want to send an array of 100 doubles from one computer to another, this will be 800 bytes. If we sent it as floats, it would be 400 bytes. If we assume the bandwidth is same, it would be twice as fast to transfer floats than doubles.

However, we will always use double precision values. But also in situations where we can reduce the size of the array, if it for example is symmetric. If we reduce the size by half, to 50 doubles or 400 bytes, we have saved much time in communication.

The communication inside a node is quite fast on a supercomputer. However when we need to use multiple nodes at once there are challenges. The nodes are not at the same physical distance to each other, this means we cannot achieve the same bandwidth between different nodes. The Abel supercomputer has nodes stacked in a rack. The nodes inside the same rack are closer. The bandwidth is usually higher in communication inside a rack, relative to communication between nodes in different racks.

7.2.5 Designing Parallel Algorithms

When we design a parallel algorithm we look for hotspots. These are computationally intensive areas, and a parallel implementation should be designed to work good around such areas.

However we must be careful, as communication can sometimes overtake computation. This can happen even in computation intensive areas. A measure known as Granularity is known as the ratio of computation versus communication.

7.2.6 Performance

A serial algorithm is evaluated by its runtime. The runtime of a parallel program depends on input size , number of processors and the communication. This is a multidimensional problem, and not so easy to measure.

Sometimes we can use two different algorithms to solve the same problem. One algorithm may be more efficient in serial, while the other is more efficient in parallel.

To measure how good performance our parallel algorithm gives, it must be measured against the best serial algorithm for the given problem. This is true even if the best serial algorithm is in no way close to the same as the parallel algorithm.

One could imagine a serial program that solves a problem in 10 seconds, but is impossible to run in parallel. And we can imagine another algorithm that solves it in 100 seconds, but runs easily in parallel. If we run the second algorithm with 5 CPUs, and say this takes 20 seconds. It would still be better to use the first serial algorithm. The parallel performance can only be described as not good. This is true until you can run the second algorithm in less time than 10 seconds.

A good model of performance we will use is the Speedup, S, defined as

$$S(p) = \frac{T_0}{T_p}. \tag{7.4}$$

With ideal performance $S(p)$ is linear, preferably $S(p) = p$. As we noted in section 7.2.1 values needed for calculation are pre-emptively placed in the catche. If a CPU cannot fit all values needed for calculations in the catche, the CPU must get these values from main memory. This slows down performance considerably.

We consider a large array we want to use in calculations. It is twice as large as the catche. If we introduce two CPUs, we can split the array in two. Each CPU can then place half the array in its catche. We will then avoid the performance loss from memory accessing. This creates the possibility of super linear scaling. This is a situation where we double the number of CPUs, and get more than a doubling in performance. Figure 7.1 is an illustration of different types of scaling.

7.2.7 Overhead

If we try to solve a problem using two processor, it will normally not be twice as fast as it would be on one processor. This is because of overhead. Overhead are things like wasted computations, communication and latency.

Wasted computation would be additional computations required for running the algorithm in parallel. Latency is the time interval required to initiate a communication, and also to tell the processors that the communication is completed.

Runtime in a serial program is often denoted as T_S. The time from the first processor to start, until the last processor exits, is often noted a parallel

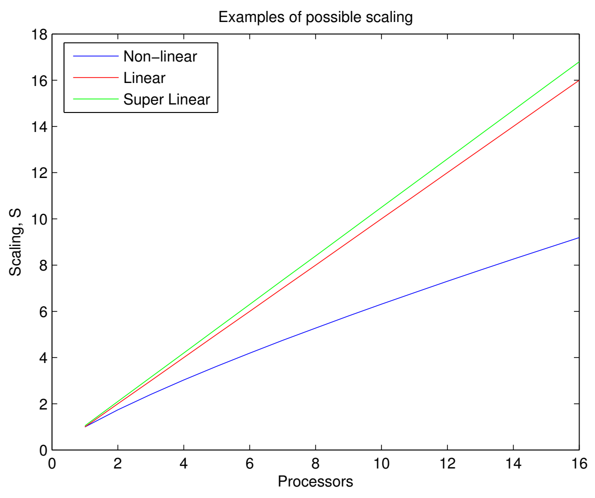


Figure 7.1: Illustration of possible scaling plots. Linear, Super linear and non linear is plotted.

runtime, T_P. The overhead, T_O, can then be described as

$$T_O = pT_P - T_S, \tag{7.5}$$

where p is the number of processors. Overhead is commonly increased as we increase the number of processors.

7.2.8 General Parallel Guidelines

For this implementation we will use a few simple guidelines with MPI. First we want to minimize the number of initiated communications. This is to reduce latency. When a communication is initiated, processors are synchronized. This means all processors enter into an MPI function at the same time. If one CPU is faster than another, this CPU will have to be idle and wait for the others to reach the communication function. Also when exiting a communication, various CPUs do not exit at the same time. This is another reason for minimizing the number of synchronizations.

Second we want to minimize the number of bytes to be communicated, mainly through symmetries. Third we want to use OpenMPI, which has optimized functions for communication implemented. In Appendix A we list many of these functions, with a short description of what they do.

7.2.9 Optimizing Communication

We will not go in detail on how the MPI functions are optimized. A good book on the subject is [49]. We will only entertain a small example.

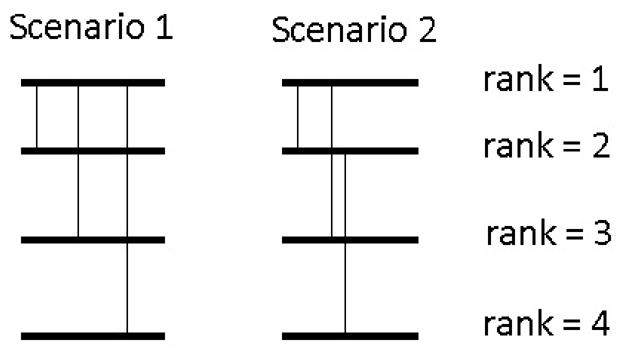


Figure 7.2: Illustration of two scenarios. Scenario one is a naive implementation of a broadcast. Scenario two is one example on how performance can be improved.

This example is illustrated in figure 7.2. Say we have four processors. We want to broadcast a message from processor 1 to all the others. We distinguish the first processor by its rank, it is rank 1. If rank 1 sends its message to rank 2, then rank 3 and then rank 4, there must be three communications performed by rank 1. One communication must wait for the other to finish in this example.

However, if rank 1 sends its message to rank 2. And then rank 1 sends to rank 3 at the same time as rank 2 sends to rank 4, there has only been two individual communication procedures by rank 1. This gives a better performance.

Each vertical line in figure 7.2 represents one send and recieve with MPI. A MPI_Send and MPI_Receive scales as

$$t = t_s + mt_b. \tag{7.6}$$

Here t_s is the startup time, t_b is the bandwidth and m is the number of bytes. In scenario one we would be performing $(P - 1)$ such send/receives, where P is the number of MPI processes

$$T_1 = (P - 1) \times (t_s + mt_b). \tag{7.7}$$

In scenario two we still perform send/recieves, but since they can now be done simontaniously the number of send/recieves scales as $\lceil log_2(P) \rceil$.

$$T_2 = \lceil log_2(P) \rceil \times (t_s + mt_b). \tag{7.8}$$

Ideally we do not want the number of sends/recieves performed by one CPU to increase at all when we increase the number of processors.

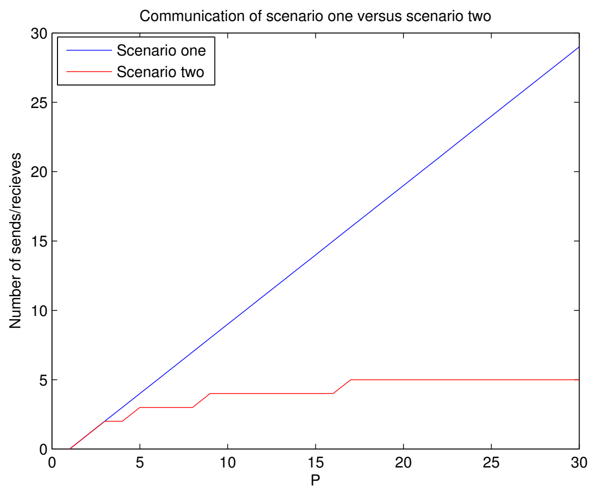


Figure 7.3: Illustration of communication in the two scenarios. Plotted are the number of send/recieves that must be performed.

We also perform performance tests of the actual performance on abel of the OpenMPI broadcast function. Results are presented in figure 7.4. We notice there are indeed optimizations present from the non linear scaling.

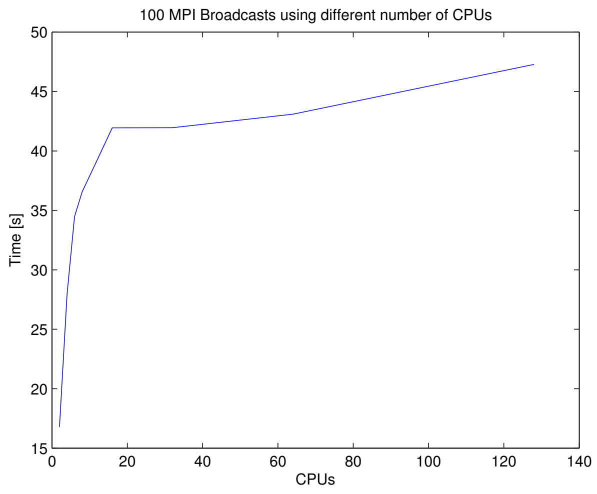


Figure 7.4: Illustration of actual communication with different number of CPUs. Time is measured for 100 Broadcasts with 8×70^4 bytes.

7.2.10 Optimizing Work Distribution

Also in parallel programming it is important that all processors get assigned the same workload. We think of the workload as a series of jobs that can be executed in parallel.

First we must define what is one job and second we must distribute these jobs among processors. Imagine running the following calculation in parallel

$$K = \sum_{ijkl}^{N} X_{ij} Y_{kl} Z_{lk}. \tag{7.9}$$

We first factorize it as

$$Z = \sum_{ij}^{N} X_{ij} \times \sum_{kl}^{N} Y_{kl} Z_{lk}. \tag{7.10}$$

We will look at two possible definitions of one job in this scenario. First, we define a job by its job ID. This job ID can for example be expressed as a function of i and j. For example

$$job\_ID = i + j. \tag{7.11}$$

If we choose this definition we have $N \times N$ jobs to distribute. Alternatively we can define a job ID as a function of i, j, k and l. For example,

$$job\_ID = i + j + k + l. \tag{7.12}$$

We would then have N^4 jobs to distribute. If we have N^4 jobs, we can use N^4 CPUs at max. If we however chose to prior job definition, we could only use N^2 CPUs. In general we want to have as many jobs as possible to distribute, but sometimes this can lead to additional communication.

Another important feature is to optimize the job distribution. If we chose one job_ID to be expressed by i and j, we can visualize the job_ID in a matrix. Each column is a different index i, and each row is an index j. The matrix elements are the job_IDs. If we use $N = 4$ and Eq. (7.11) we would get

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \end{pmatrix}$$

We want all jobs to have a different ID. We redefine the job_ID to be

$$job\_ID = i \times N + j. \tag{7.13}$$

Using this definition our matrix of job_IDs becomes

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

Each matrix element represent a job_ID. Here each job has got its unique ID, and it is easier to distribute. When we distribute work we must use the MPI rank and total number of MPI processes p. For example we can define a condition for each processor that must be true if the processors are to perform the job

```
if (job_ID % p = rank){
    // Perform job
}
```

From the perspective of our CPUs we can use this relation to identify our job distribution. Noted now in the matrix is what processor performs which job. We assume we have $p = 8$.

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

We see the job distribution is optimal, because the amount of work for all CPUs are identical. If we used the job_ID defined in Eq. (7.11) the amount of work for each processor would not be the same. This is a sub-optimal work distribution.

7.2.11 Why Parallel

Constructing a parallel program seems like quite the challenging feature. Every year there are new processors released with improved performance. However, there is a limit on the CPU. The problem with great performance CPUs is that their power consumption is generally very high. A CPU with twice the performance generally needs 3 or 4 times the power, according to a lecture from Intel on parallel programming, [53]. It is therefore much more feasable to have several CPUs with less performance, than one high performance CPU.

So not only does parallel programming enable us to perform calculations faster and on larger systems, it also requires less power. Power consumption is the limiting factor in CPU performance today. Parallel programming is thus very important, and likely to become even more important in the future. On a sidenote, this is a reason why GPUs have become so popular, GPUs are optimized for performance per watt. Christoffer Hirth wrote extensively about this in his thesis, [26]. His principles has not been incorporated in our implementation, but is a likely source of further performance gains.

7.3 OpenMP

OpenMP is another library for parallel programming. It is developed by Intel and can be activated in most compilers. OpenMP use shared memory model. Here the main memory is available on all processors. The key word here is main memory, as each processor has its own cache. OpenMP is very easy to get started with. We will not be using it, but more information is available in Ref. [50].

7.4 External Math Libraries

External Math Libraries are optimized for performance. They have built-in functions to handle matrix-matrix multiplications, vector-matrix multiplications, and similar problems. The best libraries are the likes of OpenBLAS, Ref. [51], and Intel MKL, Ref. [52]. For our implementation we will make use of MKL on the Abel computing cluster. These libraries often give a huge performance gain in matrix operations, relative to a naive for-loop implementation.

We also mention that MKL comes with a parallel version, in where it makes use of OpenMP. Both these libraries are developed by Intel. For our purposes we only made use of the serial version.

Chapter 8

Implementation

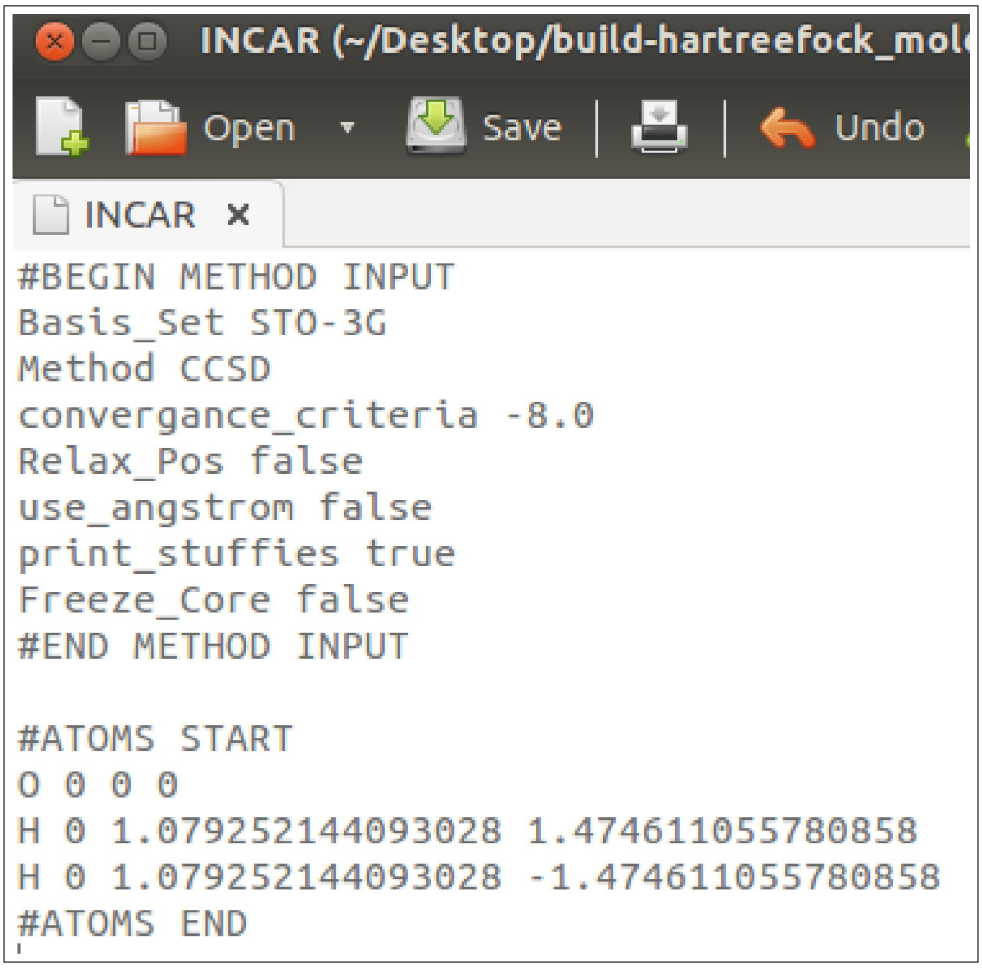
In this chapter we will discuss the implementation of all methods discussed in the previous chapters. We will implement them all in one program. We want to give our program a few positive features. First, we want people who know nothing about programming to be able to make use of our program. A background in chemistry is ideal for making use of quantum chemistry methods. However, a background in chemistry does not always include programming. For this reason, we encourage the chemistry community to look into the complete package presented in this chapter. Second, we want it to be effective, and go in parallel.

This chapter is divided into six sections. First we look at how a program user can give input and run the program. Next we examine the general structure. The following four sections discuss the details of HF, AOtoMO, serial CCSD and parallel implementation of CCSD. All ideas, code and wise remarks in this chapter are written from scratch by the author. The implementation is of course not the only one possible, for this reason we also include some citations to alternative implementations were appropriate.

8.1 Input File

This section will deal with the user friendly part of our program. This means easy input. We must be able to define what method to use, what atoms and where they are placed and other input variables. We want these to be defined in a separate textfile, to ensure the user never needs to recompile or edit any code. The input file must be named "INCAR".

An example of an input file is given in figure 8.1. This is the only file we need in order to change the system or method in use. Our program uses the standard fstream library to read the textfile. We then go through it searching for keywords. The keywords are defined to be the leftmost word



```
#BEGIN METHOD INPUT
Basis_Set STO-3G
Method CCSD
convergance_criteria -8.0
Relax_Pos false
use_angstrom false
print_stuffies true
Freeze_Core false
#END METHOD INPUT

#ATOMS START
O 0 0 0
H 0 1.079252144093028 1.474611055780858
H 0 1.079252144093028 -1.474611055780858
#ATOMS END
```

Figure 8.1: Example of input file for our program

in each line.

Basis_Set is the first keyword. Here we can choose from a variety of basis sets and the program will make use of this. The current options are STO-3G, 3-21G, 4-31G, 6-31G, 6-311ss, 6-311-2d2p and 6-311-3d3p. Most of basis sets are implemented for all atoms for which they are available.

The next keyword is Method. Here the choices are HF, CCSD, CCSDT-1a, CCSDT-1b, CCSDT-2, CCSDT-3, CCSDT-4 and CCSDT. The CCSDT part will be discussed in the next chapter.

The keyword convergence_criteria is defined to be 10^n, where n is given in the textfile. As an example, -8.0 gives a convergence criteria of 10^{-8}.

The same convergence criteria is used for all methods.

The variable Relax_Pos is meant to call a relaxation procedure, but this is not implemented in this version of the program.

The quantity use_angstrom gives the user the option to give atomic coordinates in angstrom, instead of atomic units. The options here are true or false. If it is set to true the coordinates are transformed to atomic units inside the program.

print_stuffies is a variable that gives the user the option if we want extra values printed. If this is set to true there are several interesting numbers printed during calculations. If this is set to false we only print the final energy. This option is added for a situation where we want to perform several hundred smaller calculation. Freeze_Core is an option available for CCSD. This will freeze the core electrons.

The next few lines give the atoms and their positions. The first letter is used to determine the number of electrons and the charge of the nucleus or nuclei. The program does not deal with ions.

The input file stops searching for keywords once they are all found. Hence the user is free to put comments in the input file, as long as they are not placed next to keywords or inside the ATOMS section.

8.2 General Code Overview

In this section we describe the general overview of the code. We will present this as figures, and fill inn the blanks throughout the remaining sections. Each class will be described by its input, output and internal workings.

The first class in use is the initializer. This class takes the input from main and makes sure we use it correctly. If angstroms is used as units, the coordinates are transformed to atomic units. If we want extra print options, this is ensured here. We also define a Hartree Fock object in this class, since all methods in computational chemistry generally start with a HF calculation.

We then make sure the correct method is called, and pass the HF object. For this reason we drew an arrow from HF to initializer only in figure 8.2, since it is now passed as an object to the other methods.

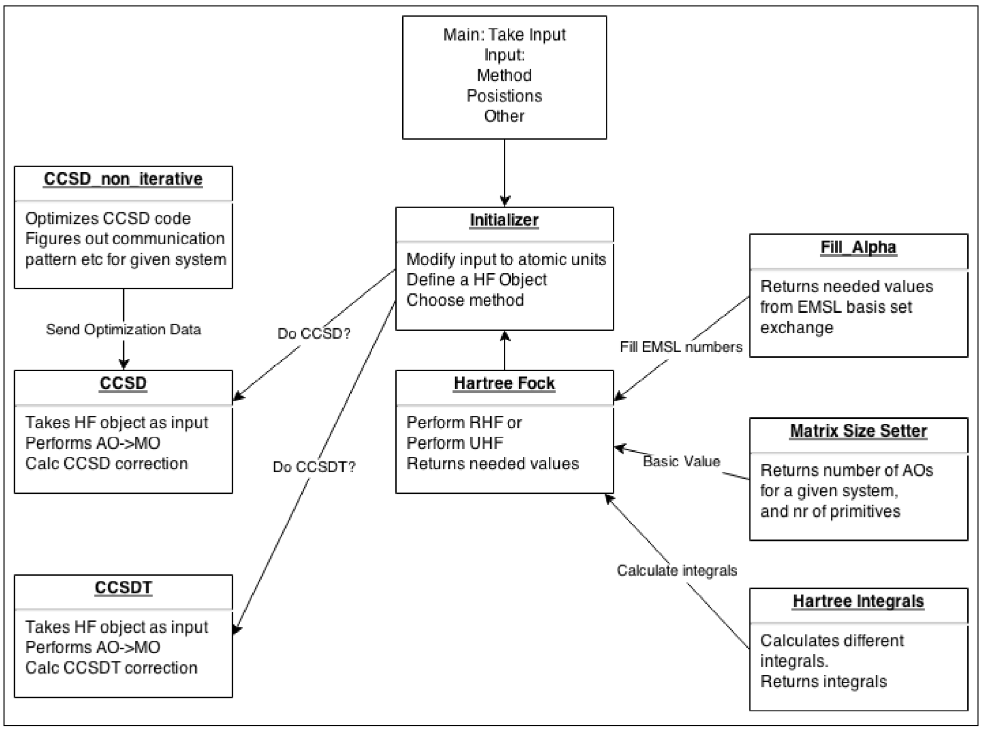


Figure 8.2: Code structure

8.3 Hartree Fock

In this section we discuss the HF implementation in detail. Our Hartree Fock implementation is grounded in the class hartree_fock_solver. The main function is called Get_Energy. In this function we will calculate the HF energy. The main outlay can be seen in figure 8.3.

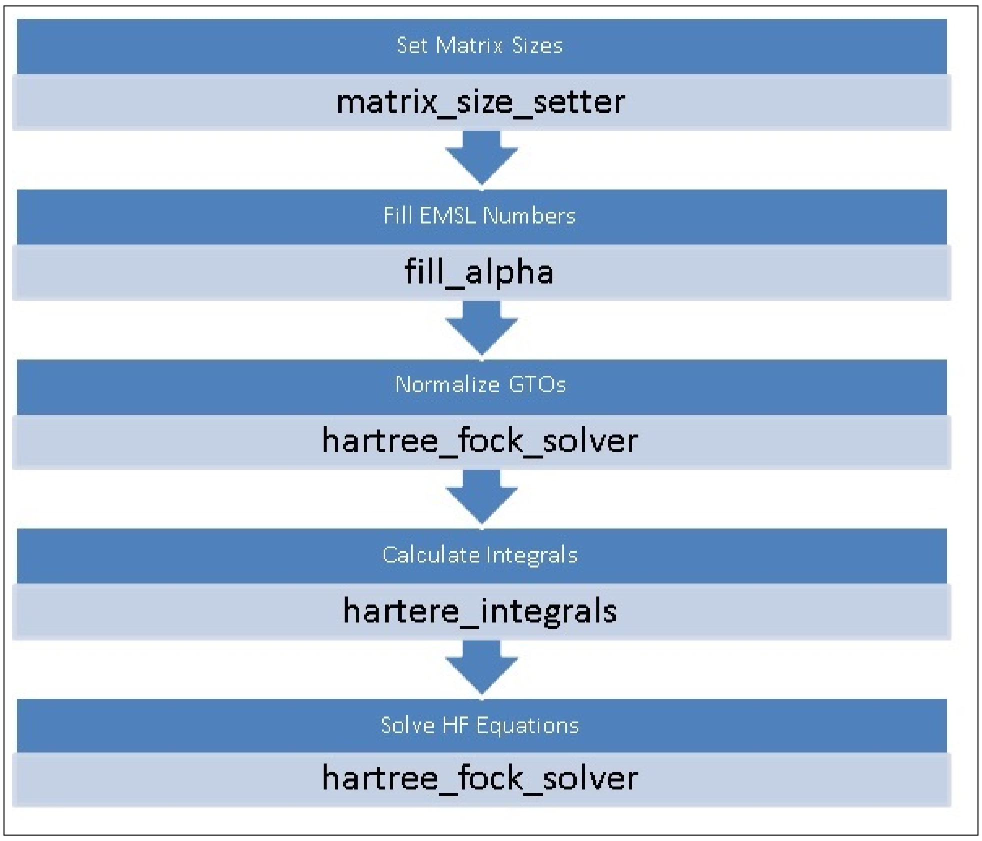


Figure 8.3: Basic Outlay of HF Implementation. First column is what action is done, second column is in what class this action takes place

The code is described in the text. We have also included key lines from the code itself to better illustrate the implementation.

Filling numbers from EMSL

```
    // Set Matrix Sizes
    matrix_size_setter matset(Z, Basis_Set, n_Nuclei);
    Matrix_Size = matset.Set_Matrix_Size();

    // Fill numbers from EMSL
    Fill_Alpha Fyll(n_Nuclei, Z, Basis_Set,
Matrix_Size, matset.Return_Max_Bas_Func());
    alpha = Fyll.Fyll_Opp_Alpha();
    c = Fyll.Fyll_Opp_c();
    n_Basis = Fyll.Fyll_Opp_Nr_Basis_Functions();
    Number_Of_Orbitals = Fyll.Fyll_Opp_Antall_Orbitaler();
    Potenser = Fyll.Fyll_Opp_Potenser();
```

The first procedure performed in this function is to call the matrix_size_setter class. We make an object of this class and send the basis set in use and which atoms are in play. This class then returns how large our arrays must be. We then allocate these arrays.

The next step is going to the fill_alpha class. This class contains data from EMSL, and fills up this data in arrays. The array alpha is filled with values for α_i, the array c is filled with values of c_i and Potenser is filled with the angular momentum. We also make a one dimensional array, Number_Of_Orbitals, which holds information on how many basis functions are in use for a specific atom. The array n_Basis holds how many primitives each of these basis functions consist of.

Normalizing GTOs

```
    // Normalize coefficients from EMSL
    Normalize_small_c();
```

The next step is to multiply in the normalization constant. This is multiplied in with the array c, through the function Normalize_small_c. In this function we have implemented the equations from section 4.5.

Overlap Integrals

The next step is to make an object of the class hartree_integrals. Inside this class we will eventually calculate all the integrals we need. However the first step is to fill up an array of E_t^{ij}. These values are present in all our integrals. For this we use Eqs. (4.41), (4.43) and (4.44). The values for E_t^{ij} will be calculated for all combinations of two primitive GTOS. This enables us to reuse the values in all our integrals, even the electron-electron repulsion.

```
    // Precalculations
    HartInt.Fill_E_ij();

    // Overlap
    O = HartInt.Overlap_Matrix();
```

We then calculate the integrals. The overlap is stored in a matrix S and is calculated using Eq. (4.42).

Kinetic Energy

The two index integrals are stored in a matrix EK. EK consists of our kinetic energy and the nuclei-electron interaction.

```
    // One electron operator
    EK = -0.5*HartInt.Kinetic_Energy()+
    - HartInt.Nuclei_Electron_Interaction();
```

The kinetic energy is calculated using Eq. (4.76).

Hermite Integrals

For the nuclei-electron interaction we need to calculate the Hermite Integrals, $R_t uv^n$. We make a new function to calculate these called Set_R_ijk. This function implements the equations given in Eqs. (4.117), (4.118), (4.119) and (4.120). We include the implementation of Eqs. (4.117) and (4.118). We put the values in a global four dimensional array R_ijk.

```
void Hartree_Integrals::Set_R_ijk(double p, int t, int u, int v,
    rowvec R1, rowvec R2)
{
    int t_max,nn,i,j,k,tt = t,uu = u,vv = v;
    t_max = t+u+v;
    double Boys_arg;
    rowvec Rcp(3);
    Rcp = R1-R2;
    Boys_arg = p*dot(Rcp, Rcp);
    Boys_arg = Boys(Boys_arg, 0);

    // Initialize R^n_0,0,0
    for (nn=0; nn<(t_max+1);nn++){
        R_ijk.at(nn)(0,0,0) = pow(-2*p, nn) *     F_Boys(nn);
    }

    // Fill up R^n_i,0,0
    for (i=0; i<tt; i++){
        for (nn=0; nn<(t_max-i); nn++){
            R_ijk.at(nn)(i+1,0,0) = Rcp(0) * R_ijk.at(nn+1)(i,0,0);
            if (i > 0){
                R_ijk.at(nn)(i+1,0,0) += i * R_ijk.at(nn+1)(i-1,0,0)
                    ;
            }
        }
    }

    // Rest of Set_R_ijk function
```

We here make use of the Boys function.

Boys Function

The function that calculates a value for the Boys function is called Boys. We have two equations we can use, Eq. (4.98) and Eq. (4.100). One works for small x, the other for large x. We define everything less than x = 50 to be small, and everything greater or equal to 50 to be large. We include the implementation of large x.

```
if (x > 50){
    Set_Boys_Start(N);
    F = Boys_Start / pow(2.0, N+1) * sqrt(M_PI/pow(x, 2*N+1));
}
```

For small x we Taylor expand around zero. We choose M = 100 in Eq. (4.98) for the Taylor expansion.

```
else{
    double F=0, sum=0;
    int M;
    for (int j=0; j<100; j++){
        sum = pow(2*x, j);
        M = 2*N+1;
        while (M < (2*N+2+2*j)){
            sum /= M;
            M += 2;
        }
        F += sum;
    }
    F *= exp(-x);
}
```

We then use the recursive relation in Eq. (4.101).

```
F_Boys(N) = F;
for (int i = N; i > 0; i--){
    F_Boys(i-1) = (2*x*F_Boys(i) + exp(-x))/(2*i-1);
}
```

We are left with designing a value to N. N denotes the starting F_n value, from which we will iterate down to the approximate solution. Popular here is putting N as some function of angular momentum, like $N = 6 \times l$. However we just put it to 30. In this value we were able to recreate all the benchmark values in Refs. [38] and [39]. These articles discuss the numerical calculation of the Boys function. Using N = 30 our results were also in agreement with the rest of the Computational Physics Group.

Nuclei-Electron Interaction

With these functions we can implement nuclei-electron interaction as given in Eq. (4.107). We add these into the array EK.

Electron-Electron Interaction

The electron-electron repulsion integrals are stored in a four dimensional field, field_Q. They are calculated through a function called Calc_Integrals_On_The_Fly. This function takes the input of four orbitals, i, j, k, l, and returns its value for $\langle ij|kl\rangle$. The function is an implementation of Eq. (4.113), also using Eq. (4.114).

```
double Calc_Integrals_On_The_Fly(int orb1, int orb2, int orb3,
    int orb4)
{
    int i,j,k,m;

    // Figure out what atom the AO belongs to, need atomic
        position
    i = Calc_Which_Atom_We_Are_Dealing_With(orb1);
    j = Calc_Which_Atom_We_Are_Dealing_With(orb3);
    k = Calc_Which_Atom_We_Are_Dealing_With(orb2);
    m = Calc_Which_Atom_We_Are_Dealing_With(orb4);

    // Here we calculate the two electron integrals
    // We have already stored E_ij^t so we reuse these
    // Symmetry considerations are applied elsewhere.

    int E_counter1, E_counter2; // These ensures we get the
        right E_ij^t
    int n,p,o,q; // Index for primitive GTO
    double temp = 0;
    E_counter1 = E_index(orb1,orb2);
    for (n=0; n<n_Basis(orb1); n++)
    {
        for (p=0; p<n_Basis(orb2); p++)
        {
            E_counter2 = E_index(orb3, orb4);
            for (o=0; o<n_Basis(orb3); o++)
            {
                for (q=0; q<n_Basis(orb4); q++)
                {
                    temp += c(orb1,n)*c(orb2,p)*c(orb3,o)*c(orb4
                        ,q)*
                            HartInt.
                                Electron_Electron_Interaction_Single

                (orb1, orb3, orb2, orb4,
                  i, j, k, m, n, o, p,
                q, E_counter1, E_counter2);

                // E_t^ij is stored for x,y,z direction
            // Hence +3 on the counter
                    E_counter2 += 3;
                }
            }
            E_counter1 += 3;
        }
    }
    return temp; // temp is the value of <ij|kl>
}
```

We also take advantage of the eighfold symmetries, written our in Eqs. (3.19) and (3.20). We constructed the code like this originally to have the option to not store these integrals at all, and instead calculate them as needed. This would be a game changer in terms of what calculations are possible, since memory would now be scaling as n^2 instead of n^4. However we later decided on a memory distribution model was sufficient for our purposes, since we want to use coupled cluster. Coupled Cluster use more memory than HF, so the system size is restricted as is.

Parallel Implementation and Memory Distribution

The hotspot in HF is the two electron integrals. We are not looking to make an optimized HF solver, but we must run this part of the calculation in parallel.

We want the workload of the integrals $\langle ij|kl\rangle$ distributed for a given index i and j. We only calculate one version of each symmetric term.

```
    field_Q.set_size(Matrix_Size, Matrix_Size);
    for (int i = 0; i < Matrix_Size; i++)
    {
        for (int j = 0; j < Matrix_Size; j++)
        {
        // Leave parts of the field un-initialized
        // size = number of MPI procs
        // rank = my MPI rank
            if ((i+j)%size == rank)
            {
                field_Q(i,j) = zeros(Matrix_Size, Matrix_Size);
            }
        }
    }
```

The two electron integrals are a part of the Fock matrix calculation. We want to run this also in parallel, so ensure we can keep the integrals distributed in memory. We remember the Fock matrix was dependant upon

$$\sum_{kl}\langle ij|kl\rangle, \tag{8.1}$$

and

$$\sum_{kl} \langle il|kj \rangle. \tag{8.2}$$

Because of this we define field_Q to store the integrals as such

$$field\_Q(i,k)(j,l) = \langle ij|kl \rangle. \tag{8.3}$$

We place the two indexes to be swapped in the matrix part of our armadillo field. We then store a N^3 sized array of temporary values, F_temp(i,j,k). We then add the terms together in the correct order to produce F_{ij}. Here we can use functions like MPI_Reduce, or make our own implementation of this function to produce the same result.

The important feature is that each processor only calculates terms based on the index i and k. This enables us to leave the indexes not in use in the field undefined, thus distributing the N^4 memory over all our P MPI processors in use (MPI procs). Each processor then only stores $\frac{N^4}{P}$ doubles. The amount of bytes for communication scales as N^3 doubles.

However we earlier calculated the integrals with a work distribution of indexes i and j. This work distribution makes it easier to use symmetries to avoid recalculation of symmetric terms. We therefore also introduce a communication procedure where we reshuffle the terms in field_Q among the MPI procs. The amount of bytes for communication here is $\frac{1}{8}N^4$, and must be done using MPI_Alltoallw or a similar implementation producing an identical result. Our HF implementation is not particularly optimized. Comments on we could optimize this implementation is available in the Future Prospects chapter.

Pre Iterative Steps

The equation to solve in HF is the eigenvalue equation from Eq. (3.47). To do this on a computer we must rewrite it slightly. The equation stands as

$$FC = SC\epsilon. \tag{8.4}$$

We define a matrix V that satisfies

$$V^{\dagger}SV = I, \tag{8.5}$$

where I is the identity matrix. We insert $V^{\dagger}$ to the left on both sides. Also VV^{-1} is inserted into the equations. This leaves

$$V^{\dagger}FVV^{-1}C = V^{\dagger}SVV^{-1}C\epsilon. \tag{8.6}$$

We also define

$$F' = V^{\dagger} F V, \tag{8.7}$$

and

$$C' = V^{-1} C. \tag{8.8}$$

We insert Eq. (8.5), F' and C' into Eq. (8.4).

$$F'C' = C'\epsilon. \tag{8.9}$$

This is a true eigenvalue problem, where ϵ will be the eigenvalues of F' and C' will be the eigenfunctions.

We also define an intermediate P, which will be the electron density.

$$P_{ij} = \sum_{k}^{N} C_i^k C_j^k, \tag{8.10}$$

where N is the number of electrons. We are now ready to begin an iterative procedure. This procedure will be different for RHF and UHF.

RHF Iterative Procedure

For RHF we initially put the density P to be filled with zeroes. In RHF we will have an equal number of electrons with spin up and spin down. This simplifies our density matrix to

$$P_{ij} = \sum_{k}^{N/2} C_i^k C_j^k. \tag{8.11}$$

We use Eq. (3.52) to find the Fock matrix. We first insert P into the equation.

$$F_{ij} = (EK)_{ij} + \sum_{kl} P_{kl}(2\langle ij|kl\rangle - \langle il|kj\rangle). \tag{8.12}$$

We then perform the iterations until we reach self consistency.

```
while $RHF\_continue = true$ do
    Calculate $F$
    $F' = V^\dagger F V$
    Solve $F'C' = C'\epsilon$
    Compute $C = VC'$
    Compute P
    if $RHF = converged$ then
        RHF_continue = false
    end
end
```

Algorithm 1: Psudocode for RHF iterations

After we have reached self consistency we calculate the energy.

```
double Hartree_Fock_Solver::Calc_Energy()
{
    // Optimized RHF energy calculations
    Single_E_Energy = accu(EK % P);
    Two_E_Energy = 0.5*accu(Energy_Fock_Matrix % P) - 0.5*
        Single_E_Energy;
    return Single_E_Energy+Two_E_Energy;
}
```

Using armadillo the energy calculation simplifies to only two lines of code.

UHF Iterative Procedure

For UHF we define two densities, P^α and P^β, which are the densities for spin up and down.

$$P^\alpha_{ij} = \sum_k^{N_\alpha} C^\alpha_{ik} C^\alpha_{jk}. \tag{8.13}$$

$$P^\beta_{ij} = \sum_k^{N_\beta} C^\beta_{ik} C^\beta_{jk}. \tag{8.14}$$

Here N_α is the number of spin up particles, while N_β is the number of spin down particles. These must be defined as input as must be equal to the total number of electrons in the system. We define the starting density to be random uniform numbers. We ensure the two matrices are not equal to each other for the first iteration. We use Eqs. (3.61) and (3.62) to find the Fock matrices.

```
while $UHF\_continue = true$ do
    Calculate $F_\alpha$
    Calculate $F_\beta$
    $F'_\alpha = V^\dagger F_\alpha V$
    $F'_\beta = V^\dagger F_\beta V$
    Solve $F'_\alpha C'_\alpha = C'_\alpha \epsilon_\alpha$
    Solve $F'_\beta C'_\beta = C'_\beta \epsilon_\beta$
    Compute $C_\alpha = VC'_\alpha$
    Compute $C_\beta = VC'_\beta$
    Compute $P_\alpha$
    Compute $P_\beta$
    if $UHF = converged$ then
        UHF_continue = false
    end
end
```

Algorithm 2: Psudocode for UHF iterations

After iterations we again calculate the energy. With armadillo the energy calculation simplifies to just two lines of code.

```
double Hartree_Fock_Solver::Unrestricted_Energy()
{
    // Oprimized energy for UHF
    Single_E_Energy = accu((P_up + P_down) % EK);
    Two_E_Energy = 0.5 * accu(EnF_up % P_up) + 0.5 * accu(
        EnF_down % P_down) - 0.5 * Single_E_Energy;
    return Single_E_Energy + Two_E_Energy;
}
```

Helping Convergence

Sometimes our solution has problems converging. This is a numerical problem and we can introduce a few features to help the convergence along.

Damping is one option. This means updating the density only slightly, by inserting

$$P'_{new} = \gamma P_{old} + (1 - \gamma) P_{new}. \tag{8.15}$$

This reduce the change in density between iterations. We only used this in UHF.

A better alternative is the DIIS method, discussed in section 4.9. We implemented this method for RHF. The first part of our DIIS implementation is calculating the error, Δp.

```
delta_p = F*P*O - O*P*F;
```

We then store the error and Fock matrices for the last M iterations. M is defined to M = 3 in our implementation. After this we construct the matrix B.

```
for (int i = 0; i < number_elements_DIIS; i++){
    for (int j = 0; j < number_elements_DIIS; j++){
        mat1 = Stored_Error.at(i);
        mat2 = Stored_Error.at(j);
        DIIS_B(i,j) = trace(mat1.t() * mat2);
    }
}
```

We then find the coefficients c.

```
DIIS_c = solve(DIIS_B, DIIS_Z);
```

And finally we construct the new Fock matrix, as a linear combination of the previous Fock matrices.

```
F = DIIS_c.at(0) * Stored_F.at(0);
for (int i = 1; i < number_elements_DIIS; i++){
    F += DIIS_c.at(i) * Stored_F.at(i);
}
```

8.4 Atomic Orbital to Molecular Orbital

Atomic Orbital (AO) to Molecular Orbital (MO) is required before we can do any CCSD calculations. In this section we describe how to implement this transformation. We are here looking for a highly optimized implementation. Some background is available in Ref.[35]. However the author found the algorithms in the literature unsatisfactory. For this reason we will present a new algorithm. First, the simplest transformation is:

$$\langle ab|cd\rangle = \sum_{ijkl} C_i^a C_j^b C_k^c C_l^d \langle ij|kl\rangle. \tag{8.16}$$

This scales as n^8 and can be factorized.

$$\langle ab|cd\rangle = \sum_i C_i^a \sum_j C_j^b \sum_k C_k^c \sum_l C_l^d \langle ij|kl\rangle. \tag{8.17}$$

This is usually split into four quarter transformations.

$$\langle aj|kl\rangle = \sum_i C_i^a \langle ij|kl\rangle. \tag{8.18}$$

$$\langle ab|kl\rangle = \sum_j C_j^b \langle aj|kl\rangle. \tag{8.19}$$

$$\langle ab|cl\rangle = \sum_k C_k^c \langle ab|kl\rangle. \tag{8.20}$$

$$\langle ab|cd\rangle = \sum_l C_l^d \langle ab|cl\rangle. \tag{8.21}$$

Each of these quarter transformations scale as n^5. The implementation of this must be done in an effective way in terms of speed and memory. The latter is the most important as the memory here scales as N^4, where N is the number of contracted GTOs, for both $\langle ij|kl\rangle$, $\langle ab|cd\rangle$ and also the intermediates in between each quarter transformation.

```
Data: Psudo Code
Result: Algorithm for parallel AOtoMO transformation
for a=0; a<N do
    for k=0; k<N do
        for l=0; l<N do
            if Grid k and l over threads then
                for j=0; j<N do
                    for i=0; i<N do
                        $QT1(k,l,j)+ = C_i^a \times \langle ij|kl \rangle$
                    end
                end
                for j=0; j<N do
                    for b=0; b<N do
                        $QT2(k,l,b)+ = C_j^b \times QT1(k,l,j)$
                    end
                end
            end
        end
    end
    Communicate $QT2(k,l,b)$
    for b=0; b<N do
        for c=0; c<N do
            if Grid b and c over threads then
                for k=0; k<N do
                    for l=0; l<N do
                        $QT3(b,c,l) = C_k^c \times QT2(k,l,b)$
                    end
                end
                for l=0; l<N do
                    for d=0; d<N do
                        $QT4(b,c,d) = C_l^d \times QT3(b,c,l)$
                    end
                end
            end
        end
    end
    Communicate $QT4(b,c,d)$
    if Store distributed MOs to given thread then
        $\langle ab|cd \rangle = QT4(b,c,d)$
    end
end
```

Algorithm 3: Simple Psudocode for parallel AOtoMO transformation. QT1, QT2, QT3 and QT4 are intermediates

Algorithm 3 is a description of how we optimize this implementation. Further optimizations will come later, but first an illustration of the general idea. We first hold index a constant throughout the transformation. This enables us to use N^3 size intermediates.

Second the grid over k and l is chosen because neither of these are involved as an index in C for the first two quarter transformations. This makes sure that the terms of $QT2(k,l,b)$ calculated by each thread is the fully two quarter transformed term. This avoids the use of MPI_Reduce or similar operations and means only one thread needs to communicate these two quarter transformed terms with specific k and l, minimizing the communication. The total amount of double precision values communicated in the first communication for now is N^3 for each a, making it N^4 in total for all a.

After the first communication each thread has all terms in $QT2(k,l,b)$ available. We then make a new grid over b and c and continue calculations in parallel. The grid could be made over a and b, but the prior makes in general a better work distribution. This is because index a is held fixed. After the fourth quarter transformation each thread has the fully transformed MOs available for certain b and c indexes.

At this point we can distribute the MOs in the same grid as for b and c, and start CCSD calculations. However because we want to have the distribution optimized for CCSD we implement another communication. This communication is N^3 for each a, making it N^4 in total for all a.

After the second communication we simply store the MOs in a memory distributed manner. It is also possible to write to disk.

$QT2$ and $QT4$ must be stored as one dimensional arrays, to minimize the number of communication procedures initiated, hence minimize latency. Also all multiplications are written using external math libraries through armadillo. We should also introduce symmetries to optimize our calculations further. The starting AOs had eight-fold symmetries. So does the resulting MOs. However these symmetries does not hold at all the quarter transformed intermediates. This complicates things slightly.

The second quarter transformed four dimensional array, QT2, will have symmetries in the two untouched indexes, as well as in the two transformed indexes. We were able to make use of this to reduce communication by 75%, since symmetric terms need not be communicated twice. This also holds true at the QT4 level obviously. The algorithm using symmetries and external math libraries is presented in algorithm 4.

```
Data: Psudo Code
Result: Effective Algorithm for parallel AOtoMO transformation
        using external math libraries
for a=0; a<N do
    for k=0; k<N do
        for l=0; l<=k do
            if Calculate on local thread then
                A1(*) = C(a,*) × $\langle kl|**\rangle$
                A2($0 \to a$) = C($0 \to a$, *) × A1
                for b=0; b<=a do
                    QT2(b,k,l) = A2(b)
                end
            end
        end
    end
    MPI_Allgatherv(QT2)
    for b=0; b<=a do
        for c=0; c<N do
            if Calculate on local thread then
                A1(*) = C(c,*) × QT2(b,*,*)
                A2($0 \to c$) = C($0 \to c$, *) × A1
                for d=0; d<=c do
                    QT4(b,c,d) = A2(d)
                end
            end
        end
    end
    MPI_Allgatherv(QT4)
    if Store distributed MOs to given thread then
        $\langle ab|cd\rangle = QT4(b,c,d)$
        or write to disk
    end
end
```
Algorithm 4: Psudocode for parallel AO to MO transformation using armadillo. A1 and A2 are one dimensional intermediates

The communication is somewhat tricky in this algorithm. Since we have inserted symmetries, the size of the message to be transmitted changes dependant upon the index a. This also applies to the displacement. We therefore store both in two dimensional arrays where a is the outer index, the inner is the MPI rank.

We run through the algorithm one time in advance to calculate these variables. We also calculate and store where each processor will start calculations. This is done to remove any pipeline flushes, which can be caused by the CPU wrongly guessing the answer of an if test.

For this reason we define another two dimensional array, this one of size N times the number of MPI procs. In the first two quarter transformation, each rank here stored at what index l will calculations start for a given index k. The next l the same rank will perform calculations on will then be

$$l \to l + p, \tag{8.22}$$

where p is the number of MPI procs. The exact same procedure is repeated for quarter transformation 3 and 4.

We have also in the more advanced algorithm inserted one dimensional arrays A1 and A2. Using these provide more optimize ways of accessing memory. It may at first sight seem like an additional complication to first calculate A2 as a one dimensional array and later store it in QT2, but this is a more efficient way when using armadillo.

The algorithm is implemented in the function

```
void Prepear_AOs(int nr_freeze);
```

The argument is how many core orbitals to freeze. The argument is somewhat wasted, since frozen core approximation is not implemented yet.

8.5 CCSD Serial Implementation

Our CCSD implementation is quite large, actually close to 10 000 lines. However this is small compared to other optimized implementations, which are usually around 40 000 lines of code. Implementation is important in CCSD, since it scales quickly for larger systems. Additional information on on the advancement of CCSD is available in a series of books, Ref.[61]. The most effective implementation to the authors knowledge is the Cyclic Tensor Framework, see Refs.[62], [63] and [63].

In this thesis we will present a simple and effective implementation of CCSD in parallel. First we look at a simple serial implementation. This section discusses the serial implementation. There are two specific goals for this implementation. First getting the energy in the smallest amount of time, second being able to run larger systems.

Even more precise we can state that our goals are:
a) Never get zero in a multiplication
b) All multiplication should be done by external math libraries
c) Do not store anything more than needed

We first present the general structure of the code. Later we will discuss a few details about different optimizations we have implemented. These will be contrasted to what kind of optimizations is commonly implemented in CCSD. Then, there will be a pros and cons list for our implementation. The chapter will be quite technical as there are several considerations behind each optimization, and it all works in combination.

8.5.1 Structure

For our serial program we first define arrays to store all intermediates, MOs and amplitudes. Two arrays are defined for each amplitude, one for the old amplitudes and one for the new amplitudes. We define a convergence criteria, which stops iterations once the difference of energy from one iteration to the next is bellow this criteria.

```
Data: Psudo Code
Result: Structure of CCSD serial program
while CCSD continue = true do
    Set Eold = Enew
    Calc F1
    Calc F2
    Calc F3
    Calc W1
    Calc W2
    Calc W3
    Calc W4
    Calc New t1 amplitudes
    Calc New t2 amplitudes
    Set t1 = t1new
    Set t2 = t2new
    Calc $\tau_{ij}^{ab}$
    Calc New Energy
    if Enew - Eold < Convergence criteria then
        CCSD continue = false
    end
end
```

Algorithm 5: Psudocode for our serial CCSD program

Algorithm 5 illustrates the algorithm as psudocode. Each of the terms behind "Calc" is taken as a separate function to make the code easily readable.

8.5.2 Removing redundant zeroes

We now briefly reconsider the molecular integrals, which were calculated as such

$$\langle pq|rs\rangle = \sum_{\alpha\beta\xi\nu} C_\alpha^p C_\beta^q C_\xi^r C_\nu^s \langle \alpha\beta|\xi\nu\rangle. \tag{8.23}$$

Here $\langle \alpha\beta|\xi\nu\rangle$ are our atomic orbitals (AOs). These come from our RHF calculations. $\langle pq|rs\rangle$ are the molecular orbitals (MOs). MOs here are presented as a linear combination of AOs. The MOs appear in CCSD as a double bar integral. This is defined as such

$$\langle pq||rs\rangle = \langle pq|rs\rangle - \langle pq|sr\rangle. \tag{8.24}$$

Due to spin considerations, if we fill a matrix with $\langle pq||rs\rangle$ it will be filled with mostly zeroes. However when using an RHF based CCSD it is common that all even numbered spin orbitals have the same spin orientation. This means all odd numbered orbitals will also have the same spin orientation. This results in the zeroes forming pattern that we have identified and utilized.

$\langle pq||rs\rangle$ are diagonal in total spin projection. In RHF the total spin is also equal to zero. When we have all odd numbered orbitals with the same spin orientation, and same with even numbered orbitals, this has a practical implication. The implication is that the only terms that will not be equal to zero are those where the sum of the orbital indexes are equal to an even number.

We will now visualize this. We construct a program that performs the AO to MO transformation and print $\langle pq||rs\rangle$ for a fixed $p = 1$ and $r = 1$. In the span of q and s there is formed a matrix, we have noted the terms that will be zero and also the terms that will be non-zero with the indexes (q, s).

$$\begin{pmatrix} (0,0) & 0 & (0,2) & 0 & (0,4) & 0 & \dots \\ 0 & (1,1) & 0 & (1,3) & 0 & (1,5) & \dots \\ (2,0) & 0 & (2,2) & 0 & (2,4) & 0 & \dots \\ 0 & (3,1) & 0 & (3,3) & 0 & (3,5) & \dots \\ (4,0) & 0 & (4,2) & 0 & (4,4) & 0 & \dots \\ 0 & (5,1) & 0 & (5,3) & 0 & (5,5) & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

This array is now stored in our computer as a four dimensional array that we call $I[a][b][c][d]$. We on purpose use a different index in the array than we do for the orbital, even though index a in this example is a referance to orbital p. We can note which orbitals our array-indexes refers to as such
a = p
b = r
c = q
d = s

This will be an array of size $(2N)^4$, where N is the number of contraction Gaussian Type Orbitals (GTOs). We now perform a trick. We want our indexes of I to refer to a different orbital, in practise we want:
a = p
b = r
c = q/2 + (q% 2) N

d = s/2 + (s% 2) N

Where % is the binary operator and we use integer division by 2. The number 2 comes from two spin orbitals per spacial orbital. Now index c is no longer a referance to orbital q, but a referance to orbital [q/2 + (q % 2) N]. If we now visualize the same double bar integral with fixed $p = 1$ and $r = 1$ it looks like this

$$\begin{pmatrix} (0,0) & (0,2) & (0,4) & \dots & 0 & 0 & 0 & \dots \\ (2,0) & (2,2) & (2,4) & \dots & 0 & 0 & 0 & \dots \\ (4,0) & (4,2) & (4,4) & \dots & 0 & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ (1,1) & (1,3) & (1,5) & \dots & 0 & 0 & 0 & \dots \\ (3,1) & (3,3) & (3,5) & \dots & 0 & 0 & 0 & \dots \\ (5,1) & (5,3) & (5,5) & \dots & 0 & 0 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

Performing this trick will always result in a matrix that looks something like this. We can split this matrix into four sub-matrices, one top left, one top right, one bottom left and one bottom right. Regardless of a and b, we will always have either the two left sub-matrices, or the two right sub-matrices always filled with zeroes. These do not need to be stored. If we ensure we *only perform calculations on orbitals with a non-zero contribution* we can change our array-indexing to:
a = p
b = r
c = q/2 + (q% 2) N
d = s/2

This means for two orbital where s = 2 and s = 3 we will have the same d value. However one of these orbitals will always be zero, so if we avoid doing calculations on this there will be no problems. And we also reduce the size of the array to half. Visualizing now the same array it looks like this.

$$\begin{pmatrix} (0,0) & (0,2) & (0,4) & \dots \\ (2,0) & (2,2) & (2,4) & \dots \\ (4,0) & (4,2) & (4,4) & \dots \\ \dots & \dots & \dots & \dots \\ (1,1) & (1,3) & (1,5) & \dots \\ (3,1) & (3,3) & (3,5) & \dots \\ (5,1) & (5,3) & (5,5) & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

And its size will be $\frac{1}{2}(2N)^2$. This kind of indexing can and should be performed on **all** stored integrals, amplitudes and intermediates. This ensures all memory is reduced by at least 50 %. Also if a,b,c,d is referencing orbitals in the same manner in all stored arrays we can still use external math libraries as before. However now we will not be passing any zeroes into these external math libraries, so calculations can be faster. This change in indexing keeps all symmetries and also allow easy row and column access. The row is accessed as usual.

$$\begin{pmatrix} (0,0) & (0,2) & (0,4) & \dots \\ (2,0) & (2,2) & (2,4) & \dots \\ (4,0) & (4,2) & (4,4) & \dots \\ \dots & \dots & \dots & \dots \\ (1,1) & (1,3) & (1,5) & \dots \\ (3,1) & (3,3) & (3,5) & \dots \\ (5,1) & (5,3) & (5,5) & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

A column is slightly different, since we only require either the top half or the bottom half of the matrix.

$$\begin{pmatrix} (0,0) & (0,2) & (0,4) & \dots \\ (2,0) & (2,2) & (2,4) & \dots \\ (4,0) & (4,2) & (4,4) & \dots \\ \dots & \dots & \dots & \dots \\ (1,1) & (1,3) & (1,5) & \dots \\ (3,1) & (3,3) & (3,5) & \dots \\ (5,1) & (5,3) & (5,5) & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

This can be specified using the submatrix command in armadillo. The attributes of the double bar integrals in RHF are such that there will be some remaining zeroes after removing these. This is because total spin must be zero. However there will be another pattern formed where all remaining zeroes are placed in one of two remaining sub-matrices. This can also be accounted for, reducing memory needs by an additional $\frac{1}{8}$. These calculations can also easily be avoided using submatrix calls in armadillo.

Since we are using an RHF basis some matrix elements become independent of spin. This means spin up will have the same value as spin down. This happens mostly for the two dimensional intermediates, and when we store in this manner the practical implications of this becomes identical upper and lower submatrices. In this situation we do not calculate the same term twice. In the future we will refer to this method as compact storage. This method removes all zeroes without defining additional arrays, as is usually done today.

8.5.3 Pre Iterative Calculations

Before calculations can start we perform a few tricks. We do not store our MOs in one gigantic array, instead we split it up into several smaller ones. This is done at the end of the AOtoMO transforamtion. These are variables such as MO3 and MO4, decleared in the header file ccsd_memory_optimized.h.

```
field<mat> MO3, MO4 ...
```

This is a very common procedure in CCSD implementation. It is performed to enable more effectively use of external math libraries. The reason it is more effective is because of memory accessing. If we want to send parts of an array into an external math library, we need first to extract which parts to send. Instead, we can define several smaller arrays like MO3. MO3 is then designed specifically to be sent directly into the external math library.

Ref.[19] also ponders this. In fact, this is such an optimization that even redundant storage of double bar integrals is often used. This means storing a value twice, just to have it easily available for passing to external math library.

Originally, we took advantage of this straight forward optimization. However, in the current implementation we will not be storing redundant values. Instead, we will be storing the single bar integrals, and have functions to map these into two dimensional arrays of double bar integrals ready for external math library use. We have surgically designed each and every for loop such that this mapping is redundant in terms of program efficiency. It does reduce memory requirements drastically.

The splitting of the integrals for us then becomes somewhat redundant in this regard, but as we will see later it is of clinical importance when we implement memory distribution.

Before iterations can start we also allocate memory for our intermediates and amplitudes. We use the principles of section 8.5.2 here. In fact every array to come in contact with an external math library must be stored on this form.

8.5.4 F1, F2 and F3

We now start the iterative procedure. The first three intermediates are two dimensional and very straight forward to calculate. We use Eqs. (6.17), (6.22) and (6.24). Our implementation has some additional complications which will be discussed shortly, but is still equivalent to our initial naive implementation.

```
for (int a = 0; a < unocc_orb; a++){
  for (int m = 0; m < n_Electrons; m++){
     F1(a/2, m/2) = accu(integ2(a,m) % T_1);
  }
}
```

In this initial naive implementation integ2 is one part of the double bar integrals we pulled out. Since we want to store only the single bar integrals, we replace this with a function Fill_integ_2_2D(int a, int m) to fill up a global mat integ2_2D. This is then used in external math libraries.

```
for (int a = 0; a < unocc_orb; a++){
  for (int m = 0; m < n_Electrons; m++){
     Fill_integ_2_2D(a, m);
     F1(a/2, m/2) = accu(integ2_2D % T_1);
  }
}
```

$[F_2]$ and $[F_3]$ are calculated in a similar procedure.

8.5.5 W1, W2, W3 and W4

These intermediates are calculated using Eqs. (6.5), (6.8), (6.15) and (6.26). We include the initial naive implementation of $[W_1]$.

```
for (int i = 0; i < n_Electrons; i++){
    for (int j = i+1; j < n_Electrons; j++){
        Fill_integ8_2D;
        W_1(i,j)(k/2,l/2) = integ8_2D;
    }
}

for (int k = 0; k < n_Electrons; k++) {
    for (int l = 0; l < n_Electrons; l++){
        Fill_integ6_2D(k,l);
        Fill_integ4_2D(k,l);
        for (int i = 0; i < n_Electrons; i++){
            for (int j = i+1; j < n_Electrons; j++){
                W_1(i,j)(k/2,l/2) += accu(integ6_2D.col(j)
                     % T_1.col(i));
                W_1(i,j)(k/2,l/2) -= accu(integ6_2D.col(i)
                     % T_1.col(j));
                W_1(i,j)(k/2,l/2) += 0.5*accu(integ4_2D
                     % tau3.at(i,j));
            }
        }
    }
}
```

Here the mapping into two dimensional double bar integrals is still done as a N^4 procedure, whereas the calculation is now an N^6 procedure. For easy external math library use later we store these variables as W1(i,j)(k,l), W2(i,j)(a,m), W3(i,m)(e,n) and W4(a,i)(c,k). We also use symmetries where they can be applied.

8.5.6 New amplitudes

The T1 amplitudes are calculated using Eq. (6.32). For the T2 amplitudes we use Eq. (6.29). To make this amplitude most optimal for external math libraries we store it as

$$T2(a,i)(b,j). \tag{8.25}$$

8.5.7 τ_{ij}^{ab} and Energy

τ_{ij}^{ab} is calculated in Eq. (6.28). It is stored in a variable tau3(a,b)(i,j) for optimal use in external math libraries. The energy can be calculated using Eq. (5.87). However we can simplify this further by introducing τ_{ij}^{ab}.

$$E_{CCSD} = E_0 + \sum_{ai} f_{ai} t_i^a + \frac{1}{4} \sum_{abij} \langle ij||ab \rangle \tau_{ij}^{ab}. \tag{8.26}$$

Here we have inserted $t_i^a t_j^b = \frac{1}{2}\left(t_i^a t_j^b - t_j^a t_i^b\right)$. Also the term f_{ai} will always be equal to zero when the basis for our CCSD calculations are a diagonalized Fock matrix.

$$E_{CCSD} = E_0 + \frac{1}{4} \sum_{abij} \langle ij||ab \rangle \tau_{ij}^{ab}. \tag{8.27}$$

8.5.8 Dodging Additional Unnecessary Calculations

In section 8.5.2 we discussed how to avoid multiplication where both terms are zero. However, for CCSD we originally had several terms to be multiplied, and we factorized them. This causes another potential optimization, that we wish to introduce with a simplified example. Consider four terms, A, B, C and D, that want to multiply together.

$$F = A \times B \times C \times D. \tag{8.28}$$

Imagine factorizing this would speed up our calculations.

$$F = A \times (B \times (C \times D)). \tag{8.29}$$

Let us define intermediate E.

$$E = B \times (C \times D). \tag{8.30}$$

After we calculated this we are left with

$$F = A \times E. \tag{8.31}$$

At the end of the calculation, it turns out A was equal to zero. This means the entire calculation was wasted, as F would have been zero anyway. Spin consideration causes this situation to occur in CCSD. Luckily because this comes from spin considerations, it is deterministic. We can identify all these situations and avoid calculations.

This is implemented in our code, and we want to present an example from the contributions to t_{ij}^{ab} from $[W4]$.

$$D_{ij}^{ab} t_{ij}^{ab} \leftarrow \sum_{kc} t_{jk}^{bc} \times [W_4]_{ic}^{ak}. \tag{8.32}$$

Index b and j only appear in the T2 amplitudes, while indexes a and i appear in the intermediate. Imagine now that index b is an odd number, while index j is an even number.

In the sum over k and c, k and c can themselves be odd or even. We remember we arranged our MOs so that all odd numbers had same spin orientation. Inside the sum, whenever now c is an odd number we will be exciting two electron into spin up orbitals. If j was an even number we also remove an electron from a spin down orbital.

Regardless of index k this will not result in zero spin in total and the amplitude must be equal to zero with our spin restriction. In section 8.5.2 we noted a situation where one of the two sub-matrices would be zero. This is the situation.

Also, the indexes a, b, i and j must themselves result in zero spin in total, or the amplitude will be zero. This limits the number of possible combinations of t_{jk}^{bc} and $[W_4]_{ic}^{ak}$ to where we can actually avoid calculating some terms of $[W_4]_{ic}^{ak}$ that are not equal to zero.

The easiest and most human-time effective way of implementing this is to simply go through the factorization backwards, to identify which multiplications we did not need. This has been done.

8.6 CCSD Parallel Implementation

In parallel implementation we will make extensive use of memory distribution. In CCSD it is quite normal to read some of the MOs from disk. We will not be doing this, but we will place memory distribution as our number one priority. The code was however originally designed to read from disk, so this option is left easily available.

8.6.1 Memory Distribution

In a serial implementation of CCSD the leading memory consumer is an $\frac{1}{16\times 2}n_v^4$ sized array, $\langle ab||cd\rangle$. The $\frac{1}{16}$ comes from only storing spacial single bar MOs, $\langle ab|cd\rangle$, with n_v being the number of virtual spin orbitals. The factor $\frac{1}{2}$ comes from symmetry. This array is called MO9 in our implementation.

However, the array only appears in the calculation of t_{ij}^{ab}, as this is the only place where the double bar integrals has three or more virtual indexes (which are a, b, c etc). This means we can ensure one processor only requires parts of the array MO9 if we distribute work here correctly. It is also possible to distribute the double bar integrals themselves, [60] presents such an algorithm.

```
for (int a = 0; a < unocc_orb; a++){
    for (int b = a+1; b < unocc_orb; b++){
        // Distribute work with Work_ID variable
        if (Work_ID % size == rank){
            // Perform calculation
        // Only the processor who passes this if test
        // will need <ab||cd> with specific a and b
        }
    }
}
```

All the largest parts of the single bar integrals will be distributed in memory. This leaves the largest un-distributed arrays as our T2 amplitudes and some intermediates. Specifically the old and new t_{ij}^{ab}, $[W_4]$ and τ_{ij}^{ab}.

We will be able to distribute $[W_4]$, through some quite complex operations that actually also provides quite good parallel performance.

Because we want to store the old T2 amplitudes as specified in Eq. (8.25) we are unable to take advantage of symmetries. We are however able take advantage of one symmetry for τ_{ij}^{ab} and the new T2 amplitudes. Also we have the storage of section 8.5.2. This means non distributed memory is scaling as

$$M(n_v, n_o) = \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{16} + \frac{1}{16}\right) n_v^2 n_o^2 \approx n_v^2 n_o^2. \tag{8.33}$$

The factor $\frac{1}{2}$ is the old T2 amplitudes. The $\frac{1}{4}$ is the intermediate τ_{ij}^{ab}. This intermediate improves performance only modestly in our factorization, but is very helpful when optimizing the use of external math libraries. We therefore keep it. The final factors $\frac{1}{16}$ are parts of the MOs we where unable to distribute in memory and also the new T2 amplitudes. The new T2 amplitudes require less memory because we only store one version of each symmetric term. How to distribute $[W_4]$ will be discussed shortly. Contributions from $n_v n_o^3$ is ignored in the non distributed memory scaling.

8.6.2 Three Part Parallel

Our parallel implementation will be quite straight forward. We will split the iterative procedure in three. Part two is the calculation of $[W_4]$. Part three is the amplitudes. Part one is everything else. The split is performed to keep in line with our guiding parallel principles of minimizing communication initiations. In each part we will also discuss what type of performance we can expect with increased number of CPUs in use. This is known as scaling.

Part 3

We first look at the third parallel part, the amplitudes. Each processor allocates memory for the new T2 amplitudes. At first each processor only stores the terms it performs calculations on itself. That is, the T2 amplitudes are distributed in memory.

We want I_{ab}^{cd} distributed in memory. The numbers for this variable is stored in MO9. To make the memory distribution easy, we distribute work for t_{ij}^{ab} based on indexes a and b. These indexes are symmetric in t_{ij}^{ab}. We thus only need calculations on $b > a$. Since we stored the single bar spacial MOs, we must distribute work very delicately if we are to not get to much overhead.

Work is distributed with in a block cyclic manner, with the block always being of size 2. This block size is identical to the number of spin MOs per spacial MO. The optimal work distribution with these limitations has the mathematical formula, with all divisions being integer divisions.

$$Work\_ID = \frac{a}{2} \times \frac{n_v}{2} + \frac{b}{2} - \sum_{n=0}^{a} \frac{n}{2}. \tag{8.34}$$

The number two is the block size, $\frac{n_v}{2}$ is the size of a column in MO9 and the sum ensures we get the optimal distribution for $b > a$. This forumla distributes work over indexes a/2 and b/2 optimally. The work ID is used by the processors to figure out if the calculation is to be performed.

```
// Find new T2 amplitudes, function
for (int a = 0; a < unocc_orb; a++)
{
    sum_a_n += a/2;
    A = a/2;
    AA = A* Speed_Occ - sum_a_n;

    // Potential to read from file here.
    // Read in a 3 dimensional array of single bar integrals for
        a
    // specific index a. Same array is used for a and a+1
    // Can use for example MPI_File_read(...)

    // a is an even number
    for (int b = a+2; b < unocc_orb; b++){ // b is even number
        B = b/2;
        Work_ID = AA+B;
        if (Work_ID % size == rank){
      // Load up 2D arrays for external math libraries
            Fill_integ3_2D(a, b);
            Fill_integ9_2D(a, b);

      // Reindexing of tau for external math library use
            Fill_2D_tau(a, b);

            for (int i = 0; i < n_Electrons; i++){ // i is even
                number
                for (int j = i+2; j < n_Electrons; j++){ // j is
                    even number
                    MY_OWN_MPI[index_counter] =
            (-MOLeftovers(a/2, b/2)(j/2, i/2)
            + MOLeftovers(a/2, b/2)(i/2, j/2)
            + W_5(a,b)(i/2,j/2)
                    - W_5(a,b)(j/2,i/2)
            - accu(W_2(i,j)(a/2, span()) % T_1.row(b/2))
                    + accu(W_2(i,j)(b/2, span()) % T_1.row(a/2))
            + 0.5*accu(W_1(i,j)(span(0, Speed_Elec-1), span()) %
                tau1(span(0, Speed_Elec-1), span())) // Half matrix =
                 0, skip this
            - accu(t2.at(b,i)(span(0, Speed_Occ-1), j/2) % D3.row(a)
                .t())
                    + accu(t2.at(a,i)(span(0, Speed_Occ-1), j/2) %
                        D3.row(b).t())
            + accu(t2.at(a,j)(b/2, span()) % D2.row(i/2))
                    - accu(t2.at(a,i)(b/2, span()) % D2.row(j/2))
            - accu(integ9_2D(span(0, Speed_Occ-1), i/2) % T_1(span
                (0, Speed_Occ-1), j/2))
                    + accu(integ9_2D(span(0, Speed_Occ-1), j/2) %
                        T_1(span(0, Speed_Occ-1), i/2))
            + 0.5*accu(integ3_2D(span(0, Speed_Occ-1), span()) %
                tau3(i,j)(span(0, Speed_Occ-1), span()))) // Half
                matrix = 0, skip this

            / (DEN_AI(a/2,i/2)+DEN_AI(b/2, j/2));

            // This is one new T2 amplitude.
            // Plus one on index counter, and calculate the next
                    index_counter++;
                    j++;
                }
                i++;
            }
        }
        b++;
    }
}
```

The code segment above is a part of the function for the new T2 amplitudes. This is how our actual code looks. It is designed for performance. The outer loop is index a. If we wanted to read from file, we would be reading in the single bar integrals for a specific index a, into an N^3 sized array.

The next loop is index b. Here we figure out if a local processor is to perform these calculations. If the processor shall perform calculations, we fill up the largest arrays needed for external math library use. The smaller arrays used in external math libraries are already filled.

The next loops are i and j. Since spin must be zero, an even number a and b only allows for even number i and j. The other combinations of odd b, even a etc are also implemented in our code but not included here.

Inside the four loops we calculate the amplitude t_{ij}^{ab}. We notice every term is written using external math libraries, with the accumulation function. We also skip calculations using the span() function where appropriate, as noted in the previous section. Finally we divide by the denominator and store the new amplitude in a one dimensional array for easier MPI function use.

Once calculations are completed we must gather the results and update the old T2 amplitudes. The new T2 amplitudes are all stored in a one dimensional array on each local processor, so we only need to initiate one communication procedure. The most effective would be a collective all-to-all communication, where we send in the new amplitudes and gather them/write over the old ones. For this we could use a function like MPI_Allgatherv. However this does not work with armadillo, since we cannot map values into an armadillo field directly with MPI.

This complicates things slightly. We see two solutions to the problem, but neither is as efficient as the before mentioned one.

We can either allocate a new one-dimensional array and perform the prior solution with a mapping of the new amplitudes into the armadillo field afterwards. Or we can perform P one-to-all broadcasts, where P is the number of MPI procs. Then each processors sends its information to others, and this is mapped into the armadillo type array. We chose the prior, but it is slightly less effective. There will be a mapping procedure required. The scaling of the communication will be identical to the scaling of an MPI_Allgatherv.

```
MPI_Allgatherv(MY_OWN_MPI, WORK_EACH_NODE(rank), MPI_DOUBLE,
                             SHARED_INFO_MPI, Work_Each_Node_T2_Parallel,
                                  Displacement_Each_Node_T2_Parallel,
                             MPI_DOUBLE, MPI_COMM_WORLD);
```

Here my own MY_OWN_MPI holds the information calculated on the processor. SHARED_INFO_MPI will contain the full new non distributed T2 amplitudes. The new amplitudes are symmetric, and we only store one version of each symmetric term. SHARED_INFO_MPI is the array we counted as non distributed new T2 amplitudes in section 8.6.1. But our algorithm is really designed to distribute the the new T2 amplitudes. However because we use armadillo with MPI we need this extra variable.

Without the complication arising from armadillo and MPI we could also skip this mapping of symmetries into the old T2 amplitudes.

```
for (int K = 0; K < size; K++){
    sum_a_n = 0;
    for (int a = 0; a < unocc_orb; a++){
        sum_a_n += a/2;
        A = a/2;
        AA = A * Speed_Occ - sum_a_n;

        for (int b = a+2; b < unocc_orb; b++){
            B = b/2;
            INDEX_CHECK = AA+B;
            if (INDEX_CHECK % size == K){
                for (int i = 0; i < n_Electrons; i++){
                    for (int j = i+2; j < n_Electrons; j++){
                        temp = SHARED_INFO_MPI[index_counter];

             // Map out symmetries after communication
                        t2(a,i)(b/2, j/2) = temp;
                        t2(b,i)(a/2, j/2) = -temp;
                        t2(a,j)(b/2, i/2) = -temp;
                        t2(b,j)(a/2, i/2) = temp;

                        index_counter++;
                        j++;
                    }
                    i++;
                }
            }
            b++;
        }
    }
}
```

Part 2

Next is part two of the parallel implementation, which is the $[W_4]$ calculation. Here we also want to distribute this variable in memory. We want to store $[W_4]$ as described previous to make use of external math libraries most effectively.

$$W4(a, i)(c, k). \tag{8.35}$$

Most of the contributions to $[W_4]$ are themselves distributed in memory on the indexes a and k. We therefore perform calculations on a local processor in a cyclic grid over these two indexes. A local processor thus holds

$$W4(a, *)(*, k). \tag{8.36}$$

Here star means all terms in this index. If we temporarily swaps the indexes i and k we can store W4(a,k)(*,*), and calculate these terms.

```
for (int a = 0; a < unocc_orb; a++){
    for (int m = Where_To_Start_Part2(rank,a); m < n_Electrons; m
        +=jump){
        // Fill 2D arrays ready for external math libraries
        // These are distributed in memory
        Fill_integ7_2D(a,m);
        Fill_integ5_2D(a,m);

        for (int e = 0; e < unocc_orb; e++){
            Fill_integ2_2D_even_even(e, m);
            for(int i = 0; i < n_Electrons; i++){
                W4(a,m)(e,i) = -integ7_2D(e/2,i/2)
          - accu(W_3.at(i,m)(e/2,span()) % T_1.row(a/2))
                    + accu(integ5_2D(span(0, Speed_Occ-1),e/2) % T_1
                        (span(0, Speed_Occ-1),i/2))
                    + 0.5*accu(integ2_2D % t2.at(a,i));
                i++;
            }
            e++;
        }
    }
    a++;
}
```

The Where_To_Start_Part2(rank,a) variable will be explained later. However, when this intermediate contributes to the T2 amplitudes we need the full matrix that is stored in $W4(a, i)$.

Therefore we perform a communication. To pick the correct MPI function we also need to know what to do with the array after the communication. Afterwards we want to multiply

$$\sum_{ck} W4(a,i)(c,k) \times t2(b,j)(c,k). \tag{8.37}$$

This multiplication will run in parallel, with work distributed cyclically over a and i. The optimal work distribution forumla is

$$Work\_ID = a \times n_o + i. \tag{8.38}$$

The communication needed to get the correct array needed prior and after is thus an all-to-all personalized communication. We will use MPI_Alltoallw. Each processor here sends its own personalized message to the other MPI procs. The message is reduced to a one dimensional array in MY_OWN_MPI before communication.

```
MPI_Alltoallw(MY_OWN_MPI, Global_Worksize_2[rank],
    Global_Displacement_2[rank], mpi_types_array, SHARED_INFO_MPI
    , Global_Worksize_2_1[rank], Global_Displacement_2_1[rank],
    mpi_types_array, MPI_COMM_WORLD);
```

We then perform the multiplication in Eq. (8.37), with work distributed over a and i. We want this contribution to be added to the new T2 amplitudes, which themselves are work distributed over a and b. This means we add another MPI_Alltoallw communication to get the correct data to the correct processor.

We have introduced a temporary memory distributed variable $W5(a,b)(i,j)$ to store this contribution to t_{ij}^{ab}. The positive features of this algorithm is that we indeed get all the variables distributed in memory, and communication procedures initiated are two All-to-All communications. All-to-All is generally the most effective kind of communication in MPI. We note that the number of initiated communications is independent of number of processors. This is exactly in line with our parallel implementation guidelines states earlier. The scaling of this communication will be identical to the scaling of two MPI_Alltoallw functions. This function is highly optimized. Also the work distribution is optimal.

Part 1

The final part of our parallel implementation consist of everything else. Here we construct $[W_1]$, $[W_2]$, $W_3]$, $[F_1]$, $[F_2]$ and $[F_3]$. The contribution from $[F_1]$ to $[F_2]$ and $[F_3]$ is a n^3 contribution. Thus we do not need to run this part in parallel. The energy and τ_{ij}^{ab} is also calculated in serial in the current implementation. These are n^4 terms, and will be the leading non parallel calculations.

To perform this part of the parallel implementation we make cyclical grids of different kinds. We can reuse the array of new T2 amplitudes, since it is not needed at this step in the calculations. This enables less memory usage. We fill the array with all numbers calculated on the processor. Then perform communication just as in Part 3, and map the correct numbers into the correct armadillo fields.

```
MPI_Allgatherv(MY_OWN_MPI, Work_Each_Node_part1_Parallel[rank],
    MPI_DOUBLE, SHARED_INFO_MPI, Work_Each_Node_part1_Parallel,
    Displacement_Each_Node_part1_Parallel, MPI_DOUBLE,
    MPI_COMM_WORLD);
```

We combine all these variables into one communication to maximise the number of jobs to distribute in accordance to the principles stated in section 7.2.10. Also to minimize the latency by initializing less communication procedures.

However because we combine several different variables we combine jobs that are not of the same size. This causes problems in our job distribution. The job distribution of part one in our parallel implementation is sub-optimal. In larger calculations some CPUs can get twice the workload of other CPUs. We have used a few tricks to lessen this performance problem. These are things like shifting the job distribution.

```
if ((Work_ID + Shift) % size == rank){
    // Perform job
}
```

The results however are not optimal and as such we cannot expect an optimal performance of this part of the CCSD implementation. Even with this concern combining all remaining variables into one MPI communication was still the better solution, compared to performing communications after each intermediate calculation.

8.6.3 Extra Pre Iterative Procedures

Before we start iterating we must map out a few new variables. These are extra calculations not needed in serial and includes variables such as displacement and size of messages in the communications. They are calculated in the class
ccsd_non_iterative_part.

```
if (Work_ID % size == rank)
{
    // Do calculation
}
```

We also map out which Work_ID each processor are to perform calculations on. This is for example the Where_To_Start_Part2(rank,a) variable. This enables us to remove all if tests like the one above from our iterative procedure. If tests inside a for loop can be very time consuming if the value true or false changes often from one index to the next. This is especially true if we have two processors. In this case, the value would change every time an index is changed. This means the number of pipeline flushes could potentially be large, dependant upon the compiler.

For P processors, the value of the if test changes after (P-1) index changes. Not having any if tests helps performance somewhat, in particular for a small number of MPI procs.

Chapter 9

CCSDT implementation guide

In this chapter will study Coupled Cluster theory by including up to three-particle-three-hole correlations. This means that our ground state will include, in addition to the Hartree-Fock reference state (our vacuum reference state), one-particle-one-hole, two-particle-two-hole and three-particle-three-hole correlations to infinite order in the interactions. This approach to the full Schrödinger equation is called Coupled Cluster with singles, doubles and triples, with the acronym CCSDT. CCSDT includes thus so-called triple contributions. We will not derive the equations, the derivation can be looked up in textbooks like Ref. [24]. This chapter is based on this textbook and a series of papers, such as [21] and referances therein, by Jozef Noga and Rodney Bartlett. Also, Refs. [25] and [22], with an erratum in Ref. [23], are useful and practical reads.

This chapter is written as a guide for implementing CCSDT with simplicity and pertinent benchmarks. CCSDT requires much computation, and contains complex equations. There are however approximations made to the CCSDT equations and these have been given their own name, the so-called CCSDT-n methods. Here n = 1a,1b,2,3,4. This chapter starts with the CCSD equations and adds CCSDT-n methods one by one, arriving ultimately at the full CCSDT equations.

The CCSDT equations can be quite difficult to extract from the literature in an implementation ready form. Each section in this chapter will start with a description of which new terms are included and supply them in a factorized form ready for implementation. We will continuously use equations from [24].

Throughout the chapter we provide also benchmarks for each contribution added. We hope this chapter will be useful for anyone who wish to create a working CCSDT code. The additional code needed for systems not based on Hartree-Fock theory will be provided in the final section.

9.1 System for Benchmarks

We first define our system to be used when benchmarking the various approaches to CCSDT-n. We must ensure the input is correct if we wish to recreate the benchmarked energy. The geometries are taken from Ref. [29]. We will study H_2O with coordinates

Atom	x	y	z
O	0	0	-0.009
H	1.515263	0	-1.058898
H	-1.515263	0	-1.058898

Table 9.1: Equilibrium geometry for H_2O with the DZ basis set. Values are Cartesian coordinates for atom in x, y and z direction in atomic units.

Coordinates are given in atomic units in table 9.1. We use a convergence criteria 10^{-7}. The basis set is available on EMSL as DZ (Dunning). A restricted Hartree-Fock (RHF) calculation using this input gives energy in atomic units (results obtained with the codes developed by us):

$$E_{RHF} = -76.0098. \tag{9.1}$$

The CCSD correlation energy to the system is, obtained with our code,

$$E_{CCSD} = -0.146238. \tag{9.2}$$

The benchmarks can be verified in [21]. We will also supply an additional benchmark of the same system outside of equilibrium. The same input is used except the geometry is given in table 9.2.

Atom	x	y	z
O	0	0	0
H	-2.27289	0	1.574847
H	2.27289	0	1.574847

Table 9.2: Geometry for system outside of equilibrium of H_2O with the DZ basis set. Values are Cartesian coordinates for atom in x, y and z direction in atomic units.

All calculations starts with an initial guess of $t_{ijk}^{abc} = 0$.

9.2 Theory

As stated in the introduction to this chapter, CCSDT includes $\mathbf{T}_3$ correlations as well, that is three-particle-three-hole correlations, namely

$$\mathbf{T} = \mathbf{T}_1 + \mathbf{T}_2 + \mathbf{T}_3. \tag{9.3}$$

All the terms from CCSD will also be included in CCSDT. The indices a, b, c, d, e, f are understood to go over virtual orbitals. The indices i, j, k, l, m, n go over orbitals occupied in the RHF basis. The amplitude equations are well defined.

$$\langle \Psi_i^a | \mathbf{H}_N (1 + \mathbf{T}_2 + \mathbf{T}_1\mathbf{T}_2 + \frac{1}{2}\mathbf{T}_1^2 + \frac{1}{6}\mathbf{T}_1^3 + \mathbf{T}_3)|\Psi_0\rangle_C = 0, \tag{9.4}$$

with $\mathbf{T}_3$ as the new contribution added to the CCSD equations.

$$\begin{aligned}\langle \Psi_{ij}^{ab} | \mathbf{H}_N (1 + \mathbf{T}_2 + \frac{1}{2}\mathbf{T}_2^2 + \mathbf{T}_1 + \mathbf{T}_1\mathbf{T}_2 + \frac{1}{2}\mathbf{T}_1^2 + \frac{1}{2}\mathbf{T}_1^2\mathbf{T}_2 \\ + \frac{1}{6}\mathbf{T}_1^3 + \frac{1}{24}\mathbf{T}_1^4 + \mathbf{T}_3 + \mathbf{T}_1\mathbf{T}_3)|\Psi_0\rangle_C = 0,\end{aligned} \tag{9.5}$$

with new contributions from $\mathbf{T}_3$ and $\mathbf{T}_1\mathbf{T}_3$. Finally

$$\begin{aligned}\langle \Psi_{ijk}^{abc} | \mathbf{H}_N (\mathbf{T}_2 + \mathbf{T}_3 + \mathbf{T}_2\mathbf{T}_3 + \frac{1}{2}\mathbf{T}_2^2 + \mathbf{T}_1\mathbf{T}_2 + \mathbf{T}_1\mathbf{T}_3 + \frac{1}{2}\mathbf{T}_1^2\mathbf{T}_2 \\ + \frac{1}{2}\mathbf{T}_1\mathbf{T}_2^2 + \frac{1}{2}\mathbf{T}_1^2\mathbf{T}_3 + \frac{1}{6}\mathbf{T}_1\mathbf{T}_2)|\Psi_0\rangle_c = 0.\end{aligned} \tag{9.6}$$

All of these are new contributions. The energy expression remains unchanged from CCSD. CCSDT-n methods include more and more of these new contributions.

For the interested reader who wishes to verify the equations we will soon present these with those given in [24]. We need to add further functionalities to the permutation operator, with $\mathbf{P}(a/bc)$ defined as

$$\mathbf{P}(a/bc)f(a,b,c) = f(a,b,c) - f(b,a,c) - f(c,b,a), \tag{9.7}$$

to be read as the permutation where a is exchanged by b and a is exchanged by c. Two of these permutation operators, $\mathbf{P}(a/bc)\mathbf{P}(k/ij)$, give in total nine permutations.

Another form of the permutation operator is $\mathbf{P}(abc)$. This is defined in terms of the following six permutations

$$\begin{aligned}\mathbf{P}(abc)f(a,b,c) =& f(a,b,c) - f(b,a,c) - f(a,c,b)\\ &- f(c,b,a) + f(b,c,a) + f(c,a,b).\end{aligned} \tag{9.8}$$

These two permutation operators can be interchanged if one rewrites Eq. (9.8) as

$$\begin{aligned}\mathbf{P}(abc)f(a,b,c) =& f(a,b,c) - f(b,a,c) - f(a,c,b)\\ &- f(c,b,a) + f(b,c,a) + f(c,a,b)\\ &= [f(cab) - f(acb) - f(bac)]\\ &- [f(cba) - f(bca) - f(abc)]\\ &= \mathbf{P}(c/ab)\left[f(cab) - f(cba)\right].\end{aligned} \tag{9.9}$$

9.3 CCSDT-1a

The simplest inclusion of triples is the CCSDT-1a approximation. This method includes the contribution from $\mathbf{T}_3$ in t_i^a, $\mathbf{T}_3$ in t_{ij}^{ab} and $\mathbf{T}_2$ in t_{ijk}^{abc}. This can be expressed as

$$t_{ijk}^{abc}D_{ijk}^{abc} = \mathbf{P}(a/bc)\mathbf{P}(k/ij)\sum_e I_{bc}^{ek}t_{ij}^{ae} - \mathbf{P}(c/ab)\mathbf{P}(i/jk)\sum_m I_{mc}^{jk}t_{im}^{ab}. \tag{9.10}$$

Here the denominator is defined as the Möller-Plesset denominator.

$$D_{ijk}^{abc} = f_{ii} + f_{jj} + f_{kk} - f_{aa} - f_{bb} - f_{cc}. \tag{9.11}$$

To make life simpler in more advanced CCSDT-n algorithms we already insert intermediates. We define two intermediates

$$[X1]_{ab}^{ei} = I_{e,i}^{ab}, \tag{9.12}$$

and

$$[X2]_{ij}^{am} = I_{am}^{ij}. \tag{9.13}$$

This means our equation for t_{ijk}^{abc} is now

$$\begin{aligned}D_{ijk}^{abc}t_{ijk}^{abc} =& \mathbf{P}(a/bc)\mathbf{P}(k/ij)\sum_e [X1]_{bc}^{ek}t_{ij}^{ae} \qquad (9.14)\\ &- \mathbf{P}(c/ab)\mathbf{P}(i/jk)\sum_m [X2]_{jk}^{mc}t_{im}^{ab}.\end{aligned}$$

CCSDT-1a makes no changes in the calculation of the energy, however the $\mathbf{T}_3$ contributions are added to t_i^a and parts of the $\mathbf{T}_3$ contribution are added to t_{ij}^{ab}. To indicate that the terms from the original CCSD method should also be included we use $\leftarrow$ and write

$$D_i^a t_i^a \leftarrow \frac{1}{4}\sum_{bcjk} I_{j,k}^{b,c} t_{ijk}^{abc} m \tag{9.15}$$

and

$$D_{ij}^{ab} t_{ij}^{ab} \leftarrow \frac{1}{2}\sum_{kcd} I_{bk}^{cd} t_{ijk}^{acd} - \frac{1}{2}\sum_{kcd} I_{ak}^{cd} t_{ijk}^{bcd} - \frac{1}{2}\sum_{mkc} I_{mk}^{jc} t_{imk}^{abc} + \frac{1}{2}\sum_{mkc} I_{mk}^{ic} t_{jmk}^{abc}. \tag{9.16}$$

CCSDT-1a actually gives a surprisingly good approximation to the full CCSDT energy, since the terms between the two usually add and subtract a similar amount to the energy.

The equations for t_i^a are the same for CCSDT as for CCSDT-1a. Implementing the new equations should give the following result in atomic units for the system in equilibrium:

$$E_{CCSDT-1a} = -0.147577. \tag{9.17}$$

For the system out of equilibrium we should get the energy value

$$E_{CCSDT-1a} = -0.209537. \tag{9.18}$$

Our code reproduces excellently both results.

9.4 CCSDT-1b

CCSDT-1b adds the remaining contribution to t_{ij}^{ab}, namely

$$D_{ij}^{ab} t_{ij}^{ab} \leftarrow \sum_{klcd} I_{kl}^{cd} t_{ijk}^{abc} t_l^d \frac{1}{2}\sum_{klcd} I_{kl}^{cd}\left(t_{ikj}^{bcd} t_l^a - t_{ikj}^{acd} t_l^b + t_{kli}^{adb} t_j^c - t_{klj}^{adb} t_i^c\right). \tag{9.19}$$

The energy correction at equilibrium is now:

$$E_{CCSDT-1b} = -0.147580. \tag{9.20}$$

The same system out of equilibrium should have a correlation energy of

$$E_{CCSDT-1b} = -0.209517. \tag{9.21}$$

Again, our code passes perfectly this benchmark.

9.5 CCSDT-2

For CCSDT-2 we add all contributions from T_2 that does not include T_1 to t_{ijk}^{abc}. This explicitly includes the terms

$$\begin{aligned}
D_{ijk}^{abc}t_{ijk}^{abc} \leftarrow & \mathbf{P}(i/jk)\mathbf{P}(abc)\sum_{lde} I_{lb}^{de}t_{il}^{ad}t_{jk}^{ec} + \mathbf{P}(ijk)\mathbf{P}(a/bc)\sum_{lmd} I_{lm}^{dj}t_{il}^{ad}t_{mk}^{bc} \\
& - \frac{1}{2}\mathbf{P}(i/jk)\mathbf{P}(c/ab)\sum_{lde} I_{lc}^{de}t_{il}^{ab}t_{jk}^{de} \\
& + \frac{1}{2}\mathbf{P}(k/ij)\mathbf{P}(a/bc)\sum_{lmd} I_{lm}^{dk}t_{ij}^{ad}t_{lm}^{bc}. \qquad (9.22)
\end{aligned}$$

In our implementation this means changing X_1 and X_2, since we introduced these intermediates earlier. It should be noted that since there are currently no $\mathbf{T}_3$ contributions to t_{ijk}^{abc} these amplitudes does not need to be stored for each iteration. This feature applies to CCSDT-n methods up to but not including CCSDT-4. The new intermediates for CCSDT-2 will be:

$$[X1]_{ab}^{ie} = I_{ab}^{ie} + \frac{1}{2}\sum_{mn} t_{mn}^{ab}I_{mn}^{ei}, \qquad (9.23)$$

and

$$[X2]_{ij}^{am} = I_{ma}^{ij} + \frac{1}{2}\sum_{ef} t_{ij}^{ef}I_{ma}^{ef}. \qquad (9.24)$$

We also introduce two new intermediates

$$[X12]_{ab}^{id} = \sum_{ld} I_{lb}^{ed}t_{il}^{ae}, \qquad (9.25)$$

and

$$[X13]_{ij}^{al} = \sum_{md} I_{ml}^{dj}t_{im}^{ad}. \qquad (9.26)$$

The expression for t_{ijk}^{abc} should be changed accordingly, leaving the contribution from CCSDT-1b untouched. This is indicated by the $\leftarrow$ in the next equation

$$t_{ijk}^{abc} \leftarrow \sum_{e}\mathbf{P}(i/jk)\mathbf{P}(abc)[X12]_{ab}^{ie}t_{jk}^{ec} + \sum_{m}\mathbf{P}(ijk)\mathbf{P}(a/bc)[X13]_{ij}^{am}t_{mk}^{bc}. \qquad (9.27)$$

The energy correction should now be, for the equilibrium configuration,

$$E_{CCSDT-2} = -0.147459. \tag{9.28}$$

While outside of equilibrium we get

$$E_{CCSDT-2} = -0.208938. \tag{9.29}$$

Both these results match perfectly with benchmark. The number of iterations can also serve as additional benchmark. The latter calculation was achieved in our program in 30 iterations.

9.6 CCSDT-3

CCSDT-3 adds all remaining contributions to t_{ijk}^{abc} that do not themselves contain T_3 amplitudes. The equations are available in [24]. For our purposes we continue with the implementation ready equations. For CCSDT-3 we must make a tweak in our intermediates, $X12$ and $X13$. The terms from CCSDT-2 remain, but we add some new ones. We also introduce new intermediates in addition to those we already have, ending with

$$[X12]_{ab}^{ie} \leftarrow -\sum_{l} I_{al}^{id} t_l^b - \sum_{le} I_{lb}^{ed} t_i^e t_l^a - \sum_{lme} I_{lm}^{ed} t_m^b t_{il}^{ae}, \tag{9.30}$$

$$[X13]_{ij}^{am} \leftarrow \sum_{md} I_{ml}^{dj} t_i^d t_m^a - \sum_{d} I_{al}^{id} t_j^d - \sum_{mde} I_{ml}^{de} t_{im}^{ad} t_j^e, \tag{9.31}$$

$$\begin{aligned} [X14]_{ab}^{id} = & \sum_{elm} I_{lm}^{ed} \left[+ t_i^e t_l^a t_m^b - t_l^e t_{im}^{ab} + \frac{1}{2} I_{lm}^{ed} t_i^e t_{lm}^{ab} \right] \\ & + \sum_{lm} I_{lm}^{id} t_l^a t_m^b + \sum_{e} I_{ab}^{ed} t_i^e, \end{aligned} \tag{9.32}$$

and

$$[X15]_{ij}^{am} = \sum_{m} I_{ml}^{ij} t_m^a - \sum_{ed} I_{al}^{de} t_i^d t_j^e + \frac{1}{2} \sum_{med} I_{ml}^{de} \tau_{ij}^{de} t_m^a. \tag{9.33}$$

These two intermediates must be included in our t_{ijk}^{abc} equation and we obtain

$$t_{ijk}^{abc} \leftarrow \sum_{e} \mathbf{P}(a/bc)\mathbf{P}(k/ij)[X14]_{ab}^{ie} t_{jk}^{ec} + \sum_{m} \mathbf{P}(c/ab)\mathbf{P}(i/jk)[X15]_{ij}^{am} t_{mk}^{bc}. \tag{9.34}$$

Inserting these equations we should now have the following energy correlation in equilibrium

$$E_{CCSDT-3} = -0.147450. \tag{9.35}$$

and out of equilibrium

$$E_{CCSDT-3} = -0.208876. \tag{9.36}$$

Both of these results match perfectly with benchmark. For the reader who wishes to optimize a CCSDT program, the four intermediates $[X12]$, $[X13]$, $[X14]$ and $[X15]$ can actually be placed inside $[X1]$ and $[X2]$, using the permutation operator tricks defined in Eq. (9.8).

9.7 CCSDT-4

For CCSDT-4 we will add the terms that are linear in $T3$. This corresponds to the terms

$$\begin{aligned} D^{abc}_{ijk} t^{abc}_{ijk} \leftarrow & \sum_{ld} \mathbf{P}(i/jk)\mathbf{P}(a/bc) I^{id}_{al} t^{dbc}_{ljk} + \frac{1}{2}\sum_{mk} \mathbf{P}(k/ij) I^{ij}_{lm} t^{abc}_{lmk} \\ & + \frac{1}{2}\sum_{de} \mathbf{P}(c/ab) I^{de}_{ab} t^{dec}_{ijk}. \end{aligned} \tag{9.37}$$

We will introduce these terms as three new intermediates, because there are more terms in the full CCSDT approach that can use this factorization. These are defined as

$$[X3]^{lm}_{ij} = \frac{1}{2} I^{ij}_{lm}, \tag{9.38}$$

$$[X4]^{de}_{ab} = \frac{1}{2} I^{de}_{ab}, \tag{9.39}$$

and

$$[X6]^{id}_{al} = I^{id}_{al}. \tag{9.40}$$

Inserting this in the amplitude equation gives

$$\begin{aligned} D^{abc}_{ijk} t^{abc}_{ijk} \leftarrow & \sum_{ld} \mathbf{P}(i/jk)\mathbf{P}(a/bc) [X6]^{id}_{al} t^{dbc}_{ljk} + \sum_{mk} \mathbf{P}(k/ij) [X3]^{lm}_{ij} t^{abc}_{lmk} \\ & + \sum_{de} \mathbf{P}(c/ab) [X4]^{de}_{ab} t^{dec}_{ijk}. \end{aligned} \tag{9.41}$$

We again perform energy calculations on the same systems as before. At equilibrium the energy correction is now

$$E_{CCSDT-4} = -0.147613. \tag{9.42}$$

And off-equilibrium we get

$$E_{CCSDT-4} = -0.209668. \tag{9.43}$$

Our code passes both tests again.

9.8 Full CCSDT

For the full CCSDT we introduce all the remaining terms. We will add these into our existing intermediates, and define a few new ones

$$[X1]^{ic}_{ab} \leftarrow \sum_{lme} \frac{1}{2} I^{ce}_{lm} t^{aec}_{lmi}, \tag{9.44}$$

$$[X2]^{am}_{ij} \leftarrow \sum_{lde} \frac{1}{2} I^{de}_{ml} t^{dea}_{ilk}, \tag{9.45}$$

$$[X3]^{lm}_{ij} \leftarrow \sum_{d} \left(I^{dj}_{lm} t^{d}_{i} - I^{di}_{lm} t^{d}_{j} \right) + \sum_{de} \frac{1}{2} I^{de}_{lm} \tau^{de}_{ij}, \tag{9.46}$$

$$[X4]^{de}_{ab} \leftarrow \sum_{l} \left(I^{de}_{lb} t^{a}_{l} - I^{de}_{la} t^{b}_{l} \right) + \sum_{ml} \frac{1}{2} I^{de}_{lm} \tau^{ab}_{lm}, \tag{9.47}$$

and

$$[X6]^{id}_{al} \leftarrow \sum_{e} I^{ed}_{al} t^{e}_{i} - \sum_{m} I^{id}_{ml} t^{a}_{m} + \sum_{em} I^{ed}_{ml} t^{ae}_{im} - \sum_{em} I^{ed}_{ml} t^{e}_{i} t^{a}_{m}. \tag{9.48}$$

We also introduce two new intermediates $[X7]$ and $[X8]$

$$[X7]^{m}_{i} = -\sum_{ld} I^{di}_{lm} t^{d}_{l} - \frac{1}{2} \sum_{lde} I^{de}_{lm} \tau^{de}_{li}, \tag{9.49}$$

and

$$[X8]^{e}_{a} = \sum_{ld} I^{de}_{la} t^{d}_{l} - \frac{1}{2} \sum_{dlm} I^{de}_{lm} \tau^{da}_{lm}. \tag{9.50}$$

These are added to t^{abc}_{ijk} with the following permutation operators in front

$$D^{abc}_{ijk} t^{abc}_{ijk} \leftarrow \sum_{e} \mathbf{P}(a/bc) [X8]^{e}_{a} t^{ebc}_{ijk} + \sum_{m} \mathbf{P}(i/jk) [X7]^{m}_{i} t^{abc}_{mjk}. \tag{9.51}$$

Implementing all of this, we get the correlation energy in equilibrium to be

$$E_{CCSDT} = -0.147594. \tag{9.52}$$

This is identical to the benchmark case mentioned above. Out of equilibrium we get

$$E_{CCSDT} = -0.2095(20). \tag{9.53}$$

Here we marked a parenthesis around (20) due to the fact that Bartlett in his letters gives this energy as

$$E_{Bartlett} = -0.209519. \tag{9.54}$$

Meaning we have a difference of -0.000001 to Bartletts results. We will assume this is caused by round off errors.

9.9 Excluded Terms

Some terms are zero when using a HF basis, because the Fock eigenvalues are diagonalized. If we want to perform CCSDT calculations for anything other than a HF basis, we must add these terms

$$[X1]^{ic}_{ab} \leftarrow -\sum_{ld} \langle l|F|d\rangle t^{ab}_{li}, \tag{9.55}$$

$$[X15]^{al}_{ij} \leftarrow -\sum_{md} \langle m|F|d\rangle t^{ad}_{ij}, \tag{9.56}$$

$$[X8]^{d}_{a} \leftarrow -\sum_{l} \langle l|F|d\rangle t^{a}_{l}, \tag{9.57}$$

and

$$\begin{aligned} t^{abc}_{ijk} \leftarrow & \sum_{d} \mathbf{P}(c/ab)(1-\delta_{cd})\langle c|F|d\rangle t^{abd}_{ijk} - \sum_{l} \mathbf{P}(k/ij)(1-\delta_{kl})\langle k|F|l\rangle t^{abc}_{ijl} \\ & - \sum_{ld} \mathbf{P}(i/jk)\langle l|F|d\rangle t^{d}_{i} t3^{abc}_{ljk}. \end{aligned} \tag{9.58}$$

Chapter 10

Benchmarks

In this chapter we will benchmark our code. We have already performed calculations using CCSDT and verified them with benchmark values. The full CCSDT method took advantage of all our implementations except for the unrestricted Hartree-Fock (UHF) part. In this chapter we will look at the performance of our codes and find out for sure if they work properly for other systems. We wish also to point to the strengths and the weaknesses of our implementations. We will look at all our methods, RHF, UHF, CCSD, and CCSDT. Also we will test our memory distributed AOtoMO transformation algorithm to its limits. In general we will benchmark our code against LSDALTON, Ref.[41], but we will also provide additional benchmarks in each section. The CCSDT approach is neither available in DALTON nor in LSDALTON.

We also mention that no special flags are used to compile. We will supply plenty of performance results. The flags used were

```
CFLAGS = -pipe -O2 -Wall -W
```

10.1 Small systems

We first perform some initial testing on small systems like water and the hydrogen molecule H_2. These will be compared with the LSDALTON package, aswell as Ref.[40] and Ref.[22].

We use coordinates given in table 10.1 for all tests, except with the DZ basis set, where we use the coordinates we used throughout chapter 9.

These are in atomic units. The coordinates are taken from Ref.[40].

Atom	x	y	z
O	0	0	0
H	0	1.079252144093028	1.474611055780858
H	0	1.079252144093028	-1.474611055780858

Table 10.1: Coordinates for water molecule system. We will perform benchmark calculations on this system with different basis sets. These coordinates are in atomic units. They are not used for the DZ basis set.

Basis Set	RHF	CCSD Correction	Benchmark
STO-3G	-74.9627	-0.0501273	[40]
4-31G	-75.9081	-0.13668	LSDALTON
6-31G	-75.9845	-0.13603	LSDALTON
DZ	-76.0098	-0.146238	[22]

Table 10.2: Benchmark calculations for water molecule

Our results are in agreement with the references down to the final decimal. We examine the RHF calculations with the STO-3G basis set more closely, in its components. These results are presented in table 10.3.

One-electron energy	= -122.219
Two-electron energy	= 38.1615
Repulsion energy	= 9.09485

Table 10.3: One particle energy, Two particle energy and Repulsion energy of RHF calculation on H_2O with STO-3G basis set.

This is in perfect agreement with benchmark. With DIIS turned on this was achieved in 13 iterations with RHF. With DIIS turned off we needed 21 iterations.

10.2 Hydrogen molecule

Our next calculations will be on the diatomic hydrogen molecule. These results will be benchmarked against a Full Configuration Interaction (FCI) study from 1968, Ref.[31]. Pople and others pioneered the effective use of Gaussian Type Orbitals throughout the 1970s. The FCI calculation used Slater Type Orbitals. We will plot the energy as a function of R, where R is the distance between the two nuclei in a.u. The results are available in figure 10.1. We will use the 6-311++G(2d,2p) basis set for both HF and CCSD calculations, and a convergence criteria of 10^{-5}. Calculations are performed from R = 0.6 to R = 4.5 with 0.1 a.u. as intervals.

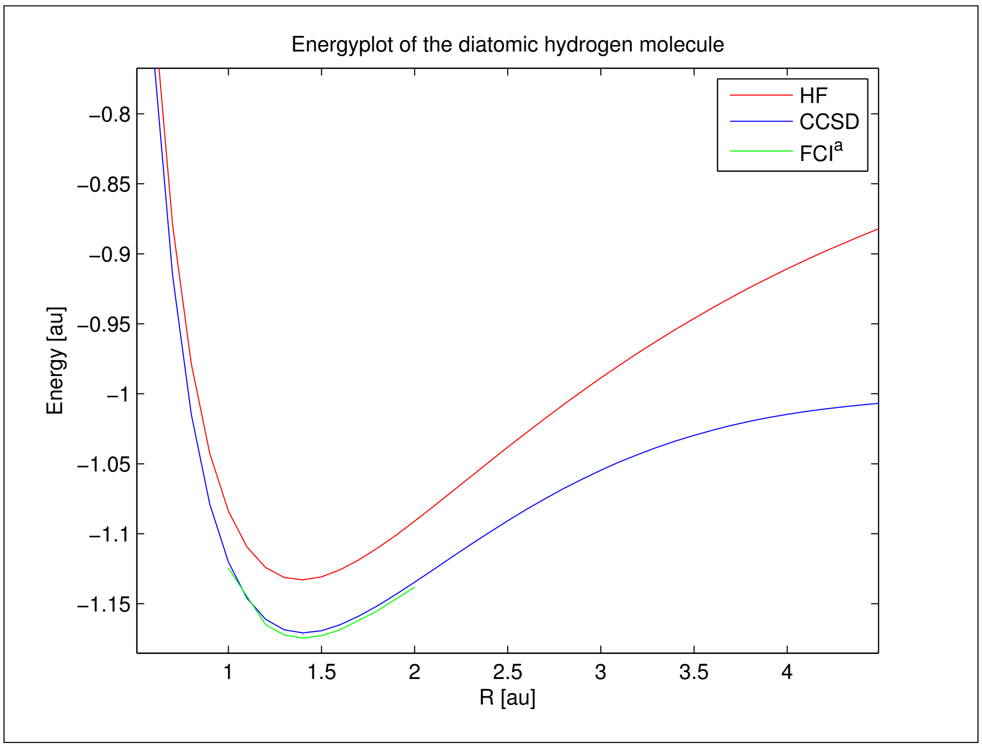


Figure 10.1: Energypotential for H_2 Molecule. [a]FCI results from Ref.[31]

The energy minimum is located at R = 1.4 au with an energy of -1.17086. This was benchmarked against LSDALTON. We could also perform CCSDT calculations on this molecule, but we only have two electrons. This means the answer will be the same as a CCSD calculation, because we do not have three electrons to excite making all the t_{ijk}^{abc} amplitudes 0. This was also confirmed by our program.

Also, this is true for all higher versions of coupled cluster, meaning the only difference between these results and the reference value should be a truncated basis set. We therefore repeat our calculation with the largest Pople basis set the author is aware of, 6-311++G(3df,3pd). This calculation is marked with (a). We use $R = 1.4011$ a.u., which is the equilibrium distance according to our FCI benchmark. We also try the aug-cc-pVQZ basis set. This calculations is marked with (b). Our calculations results in an energy in atomic units

$$E_{CCSD,(a)} = -1.17264. \tag{10.1}$$

and

$$E_{CCSD,(b)} = -1.17394. \tag{10.2}$$

According to the benchmark FCI energy with these nuclei, the equilibrium position is -1.17447 a.u. We notice with increased size basis set we are getting increasingly closer to the FCI calculation with STOs as we increase the size of the basis set.

We did benchmark our results against LSDALTON and the two programs were in agreement for the same basis set. On our energy plot FCI results are plotted for $R \in [1.0, 2.0]$, however, results for R = 1.1 were lacking.

10.3 First row Diatomic molecules

The natural next step is to introduce heavier atoms in our diatomic molecule. In these systems CCSD no longer includes the full correlation, and we will have more sources of error than just the basis set truncation. We use a decent sized basis set, 6-311++G(2d,2p) and a convergence criteria of 10^{-6}. Our results are benchmarked against a paper with DMC calculation and marked with ^{a}, see Ref.[32]. The results are presented in table 10.4.

Molecule	R [au]	E_{HF}	E_{CCSD}	E_0^a	E_R^b
Li_2	5.051	-14.8701	-14.9322	-14.995	0.50
Be_2	4.63	-29.1321	-29.2646	-29.338	0.64
B_2	3.005	-48.8656	-49.1738	-49.415	0.56
C_2	2.3481	-75.3973	-75.7703	-75.923	0.71
N_2	2.068	-108.979	-109.367	-109.542	0.70
O_2	2.282	-149.58	-150.058	-150.326	0.64
F_2	2.68	-198.741	-199.272	-199.529	0.67

Table 10.4: First Row diatomic molecule calculations using RHF and CCSD. We define In this table the molecule is listed to the left. The quantity R is the distance between the nuclei and E_{HF} and E_{CCSD} is the HF and CCSD energies for the system. The energy E_0 is our benchmark value from Ref.[32]

In this table E_0 is our benchmark value, from Ref.[32]. This is the exact, non-relativistic, infinite nuclei mass energy. We define E_R to be the percentage of correlation recovered using CCSD.

$$E_R = \frac{E_{CCSD} - E_{HF}}{E_0 - E_{HF}}. \tag{10.3}$$

We notice the recovered energy is largest for C_2 and N_2.

We also benchmark our results against Henrik Mathias Eidings results with a restricted Hartree-Fock basis (RHF) and Möller-Plesset perturbation theory to second (MP2) and third order in the interaction (MP3) for O_2, see Ref.[4] for more details. His results are marked as ^{a}. Both Eiding and we use the larger 6-311++G(3df,3pd) basis set. Results are presented in table 10.5.

Molecule	HF	MP2^a	MP3^a	CCSD
O_2	-149.588	-150.142	-150.130	-150.138

Table 10.5: Comparison of spin restricted CCSD with MP2 and MP3 for the O_2 molecule at equilibrium. MP2 and MP3 energies from Ref. [4]

We notice the energy is higher for MP3 than for MP2. Our CCSD calculation is lower than MP3, but higher than MP2.

The molecule O_2 is commonly known as an open shell molecule, meaning our spin restriction is likely to cause problems. We therefore repeat this calculation using our unrestricted Hartree Fock implementation.

The first calculation for O_2 is performed with singlet spin orientation, meaning that the total spin is 0. All other input remains the same. We obtain then an energy

$$E_{UHF,0} = -149.647. \tag{10.4}$$

The triplet state, where total spin is 1, gives an energy

$$E_{UHF,1} = -149.674. \tag{10.5}$$

These results are in good agreement with Eiding's calculations. The singlet calculation differ from his calculations with 0.001 in energy. The number of iterations needed was about 80.

10.4 C_{20} Ground State

The molecule C_{20} is a particularly interesting molecule. Calculations using different computational methods seem to provide very different answers to the geometry of the ground state, as noted in Ref. [47]. There are three structures in considerations. They are ring, bowl and cage. All these structures are present in experiments, but the ring structure seems to be the most likely orientation. This is followed by bowl as the second most likely, and cage as the third most likely.

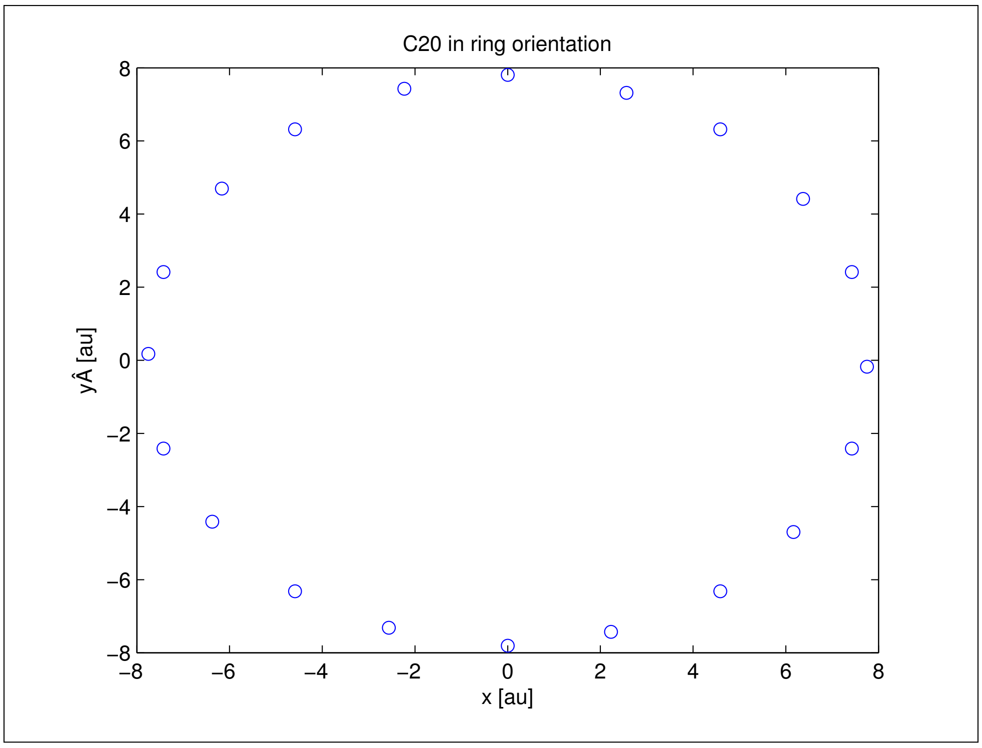


Figure 10.2: Orientation for C_{20} in ring formation.

However, methods considered highly accurate, such as diffusion Monte Carlo (DMC), do not agree with experiment. Different methods do not even agree with each other. All this can be seen in Ref.[45]. One theory for the disagreement is that experiments are not done at 0 Kelvin, while quantum chemistry ground state calculations are. We will perform RHF and CCSD calculations on the system.

We will use the 6-31G basis set, with a convergence criteria of 10^{-4}. If we are to form this molecule we need the energy of the molecule at least to be lower than 20 times the energy of a single carbon. For a reference we calculate the ground state energy of a single carbon atom. The energy for 20 single carbon atoms using RHF is

$$20 \times E_{HF} = -751.6. \tag{10.6}$$

CCSD finds the energy of a 20 single carbon atoms to be

$$20 \times E_{CCSD} = -753.0. \tag{10.7}$$

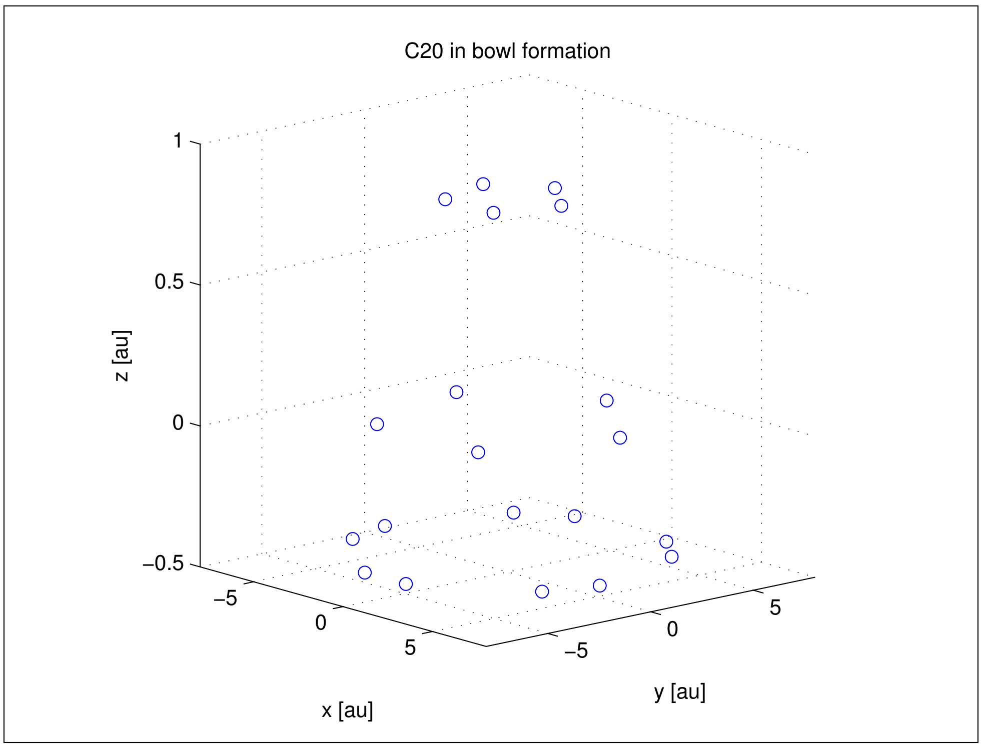


Figure 10.3: Orientation for C_{20} in bowl formation.

The ground state geometry is the one with the lowest energy. We have three orientations from reference, and 20 isolated single carbon atoms is another possible orientation we include.

We would also like to visualize the three different orientations ring, bowl and cage. All coordinates are taken from Refs.[45] and [46]. Both these references use the same coordinates. The geometries for bowl, cage and ring are illustrated in figures 10.3, 10.4 and 10.2.

Orientation	RHF	CCSD
ring	-756.454	-758.261
cage	-756.122	-758.004
bowl	-756.312	-758.195

Table 10.6: C_{20} Energy calculations using different orientations, ring, cage and bowl.

Calculations on ring was performed without DIIS in HF, whereas cage

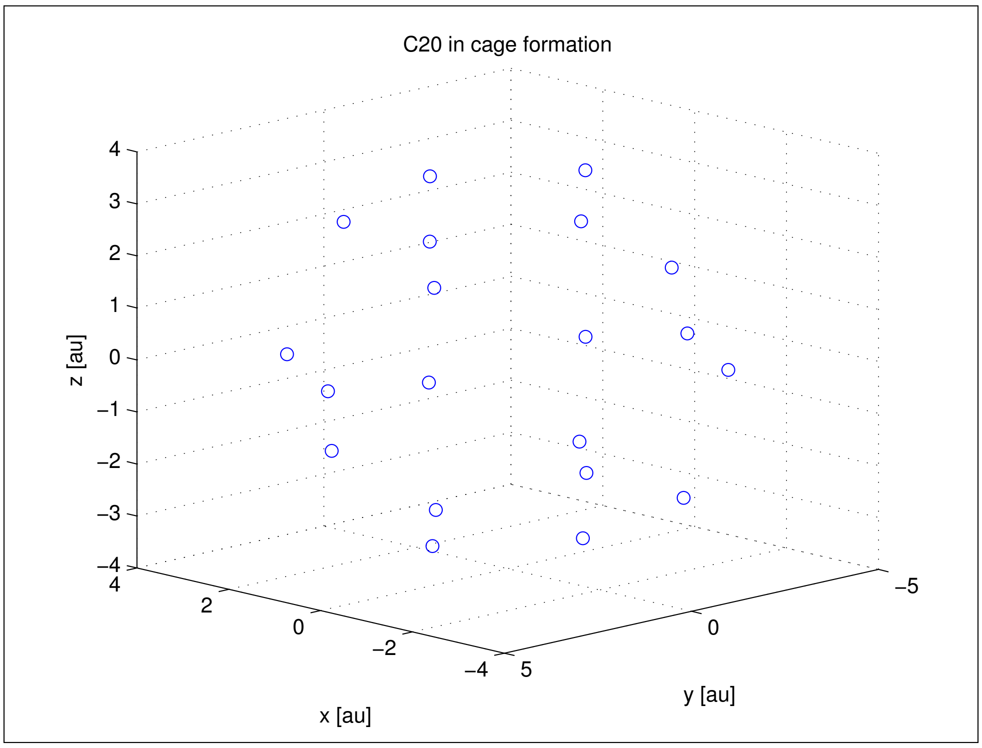


Figure 10.4: Orientation for C_{20} in cage formation.

and bowl had DIIS enabled. This was required to achieve convergence. We see the energy is lower than 20 times the energy of a single carbon for all three orientations bowl, ring and cage and for both methods. Our results are in good agreement with Ref.[46] and experimental results. We have indeed found the ring to be the lowest energy orientation.

To perform this calculation our memory distribution algorithm was required.

10.5 Energy as function of number of AOs

The size of the basis set has a great impact on our calculations. However at some point, the basis set is so large that making it even larger will not provide any noticeable improvement in accuracy. Any increase in basis set size will however always increase the runtime of our program. For this reason there is great interest in knowing what size basis set gives what level of accuracy.

To study this we have performed calculations on a single water molecule,

H_2O. We have performed calculations using the STO-3G, 6-31G, 6-311G**, 6-311++G(2d,2p) and 6-311++G(3df,3pd). These basis sets are listed from smallest to largest. Results are provided in fig. 10.5.

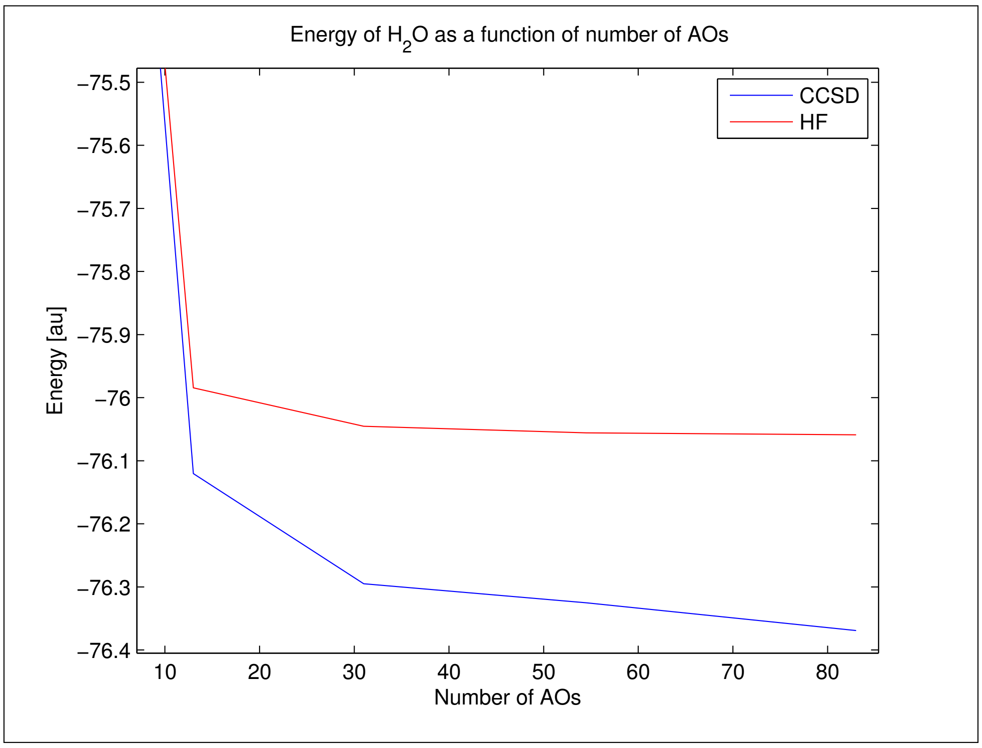


Figure 10.5: Energy of H_2O as a function of number of AOs

We check the convergence with respect to the number of AOs for both HF and CCSD. We see the HF energy has better convergence than CCSD. CCSD is not particularly well converged at all, this was to be expected however based on our H_2 results from earlier.

The largest calculation used approximate 80 AOs. With 80 AOs this would be $n_o = 10$ and $n_v = 150$, meaning $\frac{n_v}{n_o} = 15$. When performing CCSD calculations this fraction is normally expected to be between 5 - 10.

10.6 Hartree Fock Performance Testing

In this section we will test the performance of our Hartree Fock program. We will first look at memory usage, and afterwards discuss the performance of both RHF and UHF for different systems. We have not spent much time on optimizing this part of the program, except for the parallel implementation and memory distribution.

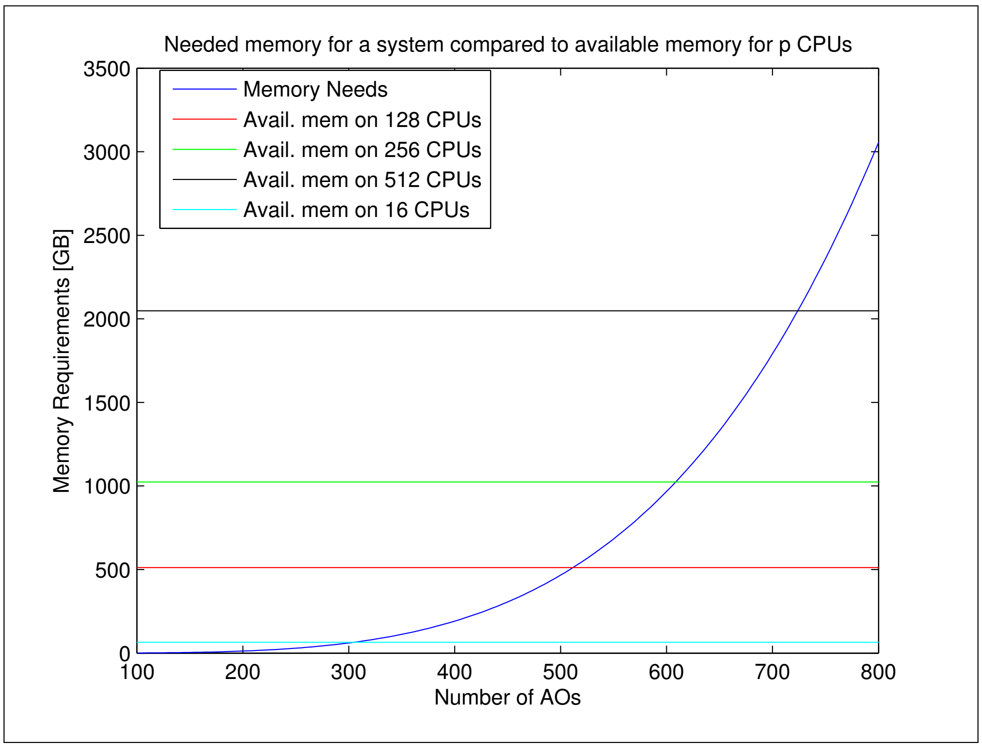


Figure 10.6: Memory Requirements for HF as a function of N

In figure 10.6 we have plotted the memory needs for our HF program as a function N for the serial version. N is the number of AOs. We will perform our calculation on the Abel computing cluster, Ref.[69]. This cluster consist of nodes. Each nodes has 16 CPUs and 64 GB of memory. This means each CPU has 4 GB of memory, if we discount that a small portion of the memory is already occupied by processes already running on the node.

On our figure we have marked the total available memory for a different number of CPUs. We are able to distribute the absolute dominant memory consuming array. This means the crossing point between available memory and needed memory is a good indication of the number of AOs we can run with this number of CPUs. For example 512 CPUs crosses the blue line at $N = 700$. This means we can do approximately 700 AOs with 512 CPUs.

It is possible on abel to ask for more memory for each CPU. There are also high-memory nodes available. In theory we could ask for all 64 GB of memory and only use one CPU on the node. This would however leave the other 15 CPUs unusable for other users, since there is no memory left.

10.6.1 HF performance

To test the performance of our HF implementation we perform calculations on O_2 with the 6-311++G(3df,3pd) basis set. We use convergence criteria of 10^{-8}. Only the four index integrals are run in parallel, so there are some serial calculation involved. The UHF implementation has more communication than RHF. We plot both performances in the same plot for comparison. The raw data is included in table 10.7, and the timings involve all calculations in HF. Results are plotted in figure 10.7.

P	RHF time [s]	UHF time [s]
1	675.78	854.31
2	322.16	411.68
4	182.36	324.26
8	129.94	123.69
16	60.79	73.52
32	36.58	49.87
64	27.55	55.32
128	25.02	44.59
256	34.29	49.1

Table 10.7: Performance of Hartree Fock implementation in parallel for O_2 molecule with 6-311++G(3df,3pd) the basis set.

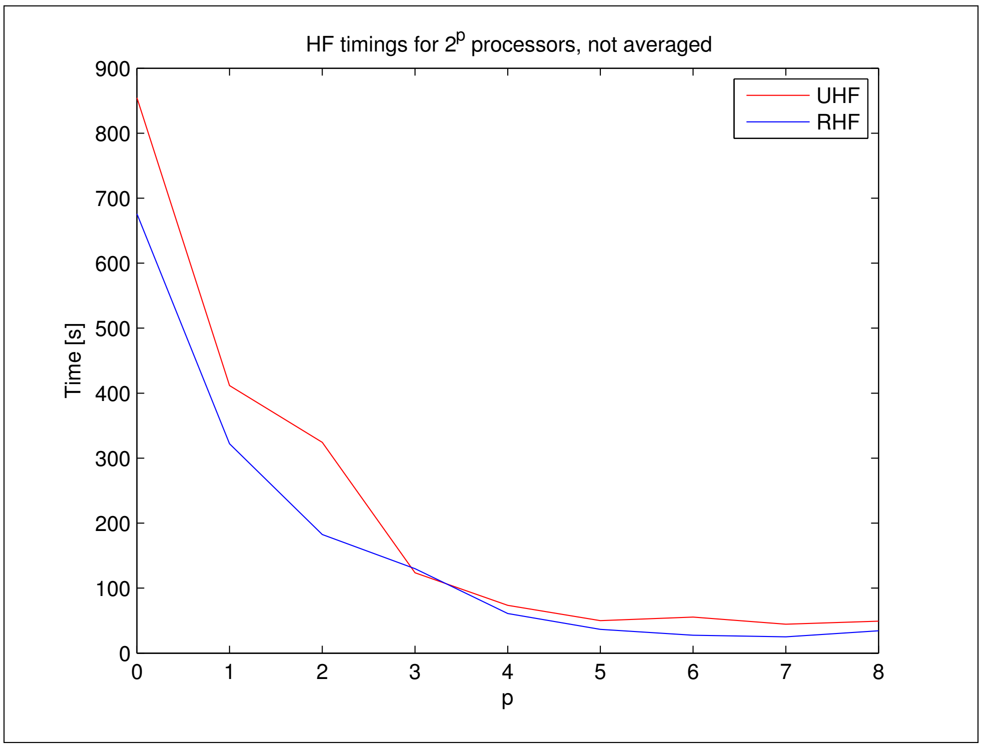


Figure 10.7: Time in seconds for a HF calculation on O_2 with the 6-311++G(3df,3pd) basis set using 2^p processors. Both RHF and UHF calculations included. Results are not averaged over multiple runs.

10.7 AOtoMO Performance Testing

In this section we will test the performance of our AOtoMO transformation algorithm for the four index integrals, as a function of number of CPUs. The raw data is included in table 10.8, and we plot the data in figures 10.8 and 10.9. The quantities of interest are time used in calculation versus time used in communication. Calculations spread over more CPUs can be performed faster. However more CPUs will require more communication. Calculations on 251 and 569 AOs are done on the $C_{12}H_{22}O_{11}$ molecule, sucrose. We use the 6-31G and the 6-311++G** basis sets. The 130 AO calculation is done on an imaginary molecule, simply to measure performance.

We define wall time as the time from the first CPU start until the last CPU finish. We will use this as a measurement of performance of our algorithm. We will also define the point where wall time increase with increased number of CPUs as the time communication overtakes computation.

p	AOs = 130	AOs = 251	AOs = 569
1	123.55	3458	-
2	69.64	1800	-
4	37.73	923	-
8	25.51	618	-
16	17.11	389	21076
32	14.52	297	15556
64	10.35	266	14961
128	10.62	232	11294
256	-	-	9413
512	-	-	9346

Table 10.8: Parallel performance of AOtoMO transformation for 130, 251 and 569 Basis Functions for different number of CPUs

We should first remind ourselves the computational scaling here is N^5, where N is the number of AOs. The communication however scales as N^4. We see from our benchmarks that larger number of AOs scales better with increased number of CPUs.

We notice especially that the wall time increase from 64 to 128 CPUs for 130 AOs. This does not happen for 251 AOs. Indicating a better scaling for higher number of AOs, as expected from the algorithm. We also note that in a less optimized AOtoMO transformation, communication would overtake computation much quicker.

Our memory distributed model makes us able to run larger systems using increased number of CPUs. The memory usage of the algorithm is closely related to that of our Hartree Fock implementation. This is due to the fact that in our HF program we stored all terms of the four index integrals, whereas after HF is completed we will delete the terms corresponding to one symmetry, reducing the memory requirements by $\frac{1}{2}$. The transformed integrals are also stored using this one symmetry, and need $\frac{N^4}{2}$ memory. These two combined equals the memory needed for HF.

For implementation reasons there will be an additional N^3 array needed. This is because we will need one N^3 intermediate for armadillo, and one for the communication in MPI. This changes memory requirements modestly, but becomes a problem for approximate 1000 AOs.

For 569 AOs we are unable to perform the calculation in serial. We start calculations using 16 CPUs. From figure 10.8 we notice the best scaling is

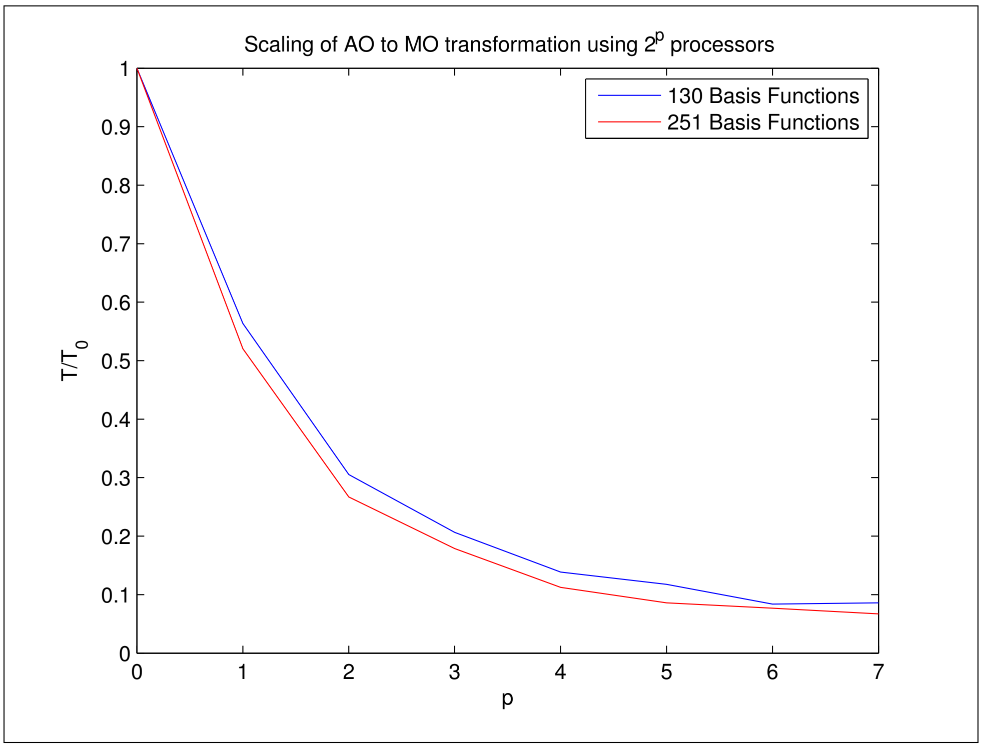


Figure 10.8: AOtoMO transformation scaling for small number of AOs and up to 128 processors, p. Plotted on y - axis is T$/T_0$, where T_0 is the wall time for the serial version. On the x - axis we have 2^p, where p is the number of CPUs

behind us at this number of CPUs. However we can make an educated guess that running this sized system in serial would require several days of computations.

Our results for 569 AOs confirm that the scaling is better with increased number of AOs. Even from 256 to 512 CPUs we improve overall performance.

We also list a few single calculations for different number of AOs and CPUs, to see the performance of the transformation for a variety of calculations. The basis set used for the calculations was 6-311++G(2d,2p). These results are included in table 10.9.

We see up to 600 AOs is very doable calculations. At 735 AOs the transformation itself takes more than one day. However, these timings are very good relative to others, Ref.[36]. We have not found anyone beating our performance. Full AOtoMO transformations are rare in the literature, es-

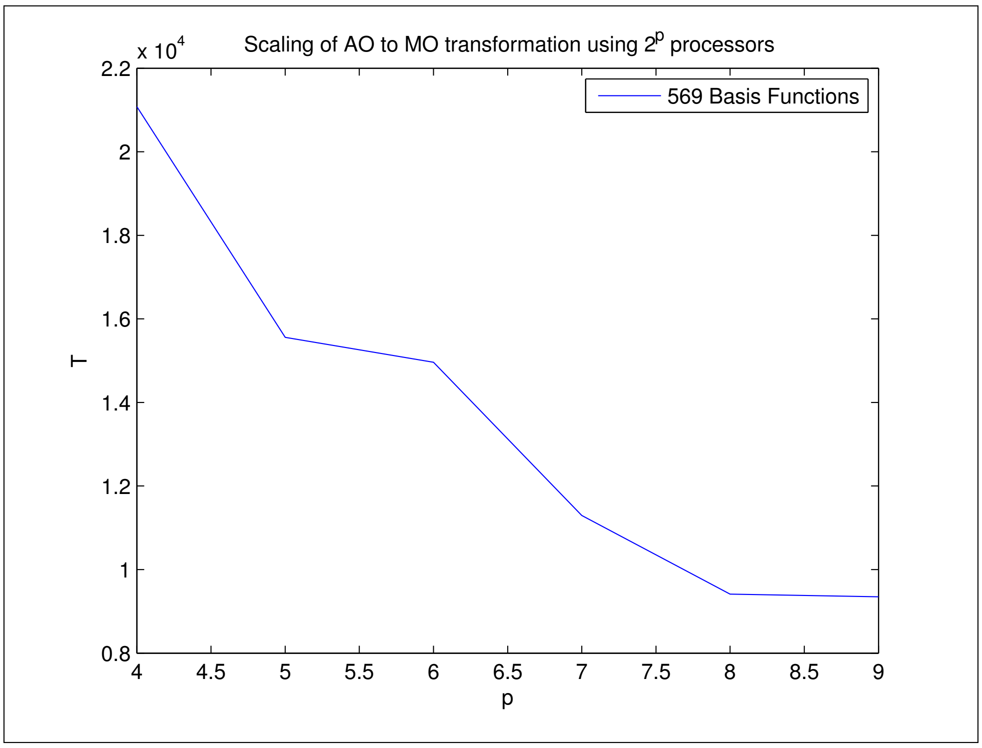


Figure 10.9: AOtoMO transformation wall time scaling for 569 AOs and 2^p CPUs. 16 to 512 CPUs used.

Molecule	CPUs	AOs	AOtoMO
CH_4	16	69	0.98 s
C_2H_6	64	118	9.71 s
C_2OH_6	64	147	19.9 s
$(H_2O)_8$	256	392	23 min
C_{20}	256	580	238 min
$(H_2O)_{15}$	512	735	35.8 hour

Table 10.9: Performance of AOtoMO transformation for different systems in parallel.

pecially for high number of AOs. It is therefore difficult to make comparisons.

Alternatively, on the benchmark page for ACESIII, Ref.[34], they list the wall time of a few CCSD calculations. CCSD regularly use the full AOtoMO transformation, and ACESIII is a very optimized CCSD program. Thus it is interesting to see if CCSD or AOtoMO would dominate wall time in a calculation with an optimized CCSD version.

An Ar_6 calculation with 300 AOs is listed on ACESIII benchmark site with 128 CPUs as 5.9 min per CCSD iteration. Using our AOtoMO transformation, 251 AOs was performed in 3.8 min.

They also list calculations on sucrose, $C_{12}H_{22}O_{11}$, using the 546 AOs, and 23 orbitals frozen. This means 523 unfrozen spin orbitals. They report 29.4 minutes for one iteration of 256 CPUs. Our four index transformation spent 156 minutes on 569 AOs.

These two results indicate the wall time for our transformation for this number of AOs lies approximated between 2-4 CCSD iterations, which is very reasonable. CCSD usually requires 10-20 iterations for convergence in equilibrium. Most of the benchmarks on the ACESIII site for CCSD with up to 512 CPUs is between 300 - 600 AOs, all of which is doable with this algorithm.

It should also be noted that our full AOtoMO transformation only depends upon number of AOs, whereas CCSD depends on the combination of occupied versus virtual orbitals. In a CCSD calculation with $n_v >> n_o$ AOtoMO would be more time consuming relative to a CCSD iteration.

10.8 CCSD Serial Performance

In this section we will test the CCSD performance in serial. We will try a cluster of water molecules, of size $(H_2O)_N$. We will use the 6-311++G(2d,2p) basis set. We also note the serial time of the AOtoMO transformation. Results are presented in table 10.10.

System	AOs	n_o	n_v	CCSD iteration [s]	AOtoMO [s]
(H_2O)	49	10	88	1.17	0.75
$(H_2O)_2$	98	20	176	37	14.41
$(H_2O)_3$	147	30	264	632	297
$(H_2O)_4$	196	40	352	2255	1454

Table 10.10: Performance of serial CCSD program for different number of water molecules using the 6-311++G(2d,2p) basis set.

We test the serial implementation against an unoptimized but factorized CCSD program. This implementation is available through "CCSD1" as method in the input file. The equations are fully factorized, so the theoretical scaling of the equations is still N^6, where N is the number of AOs. However we have not implemented the use of external math libraries, compact storage and the other optimizations discussed in section 8.5. These calculations are not performed on abel. We also use the external math library BLAS. The performance here on one CPU is generally better than on abel supercomputer, but we do not have the high number of CPUs available. Results are presented in table 10.11.

System	n_o	AOs	Unoptimized iter [s]	Optimized iter [s]	Fraction
Ne	10	29	3.5	0.1	35
H_2O	10	49	40	0.78	51
C_2H_4	16	62	670	5.4	124
C_2H_4	16	98		48	

Table 10.11: Comparison between optimized and unoptimized CCSD implementation

10.9 CCSD Parallel Performance

Our CCSD serial implementation is among the fastest. We want to test its parallel performance in detail. First we run a small system of H_2O with the 6-311++G(3df,3pd) basis set for a different number of processors. We use up to 64 processors on this system. The raw data is included in table 10.13. We plot the performance in figure 10.10 and the Speedup, S, in figure 10.11

P	CCSD iteration time [s]
1	4.44
2	2.35
4	1.30
8	0.88
16	0.48
32	0.27
64	0.18

Table 10.12: Parallel performance of CCSD implementation for H_2O with the 6-311++G(3df,3dp) basis set.

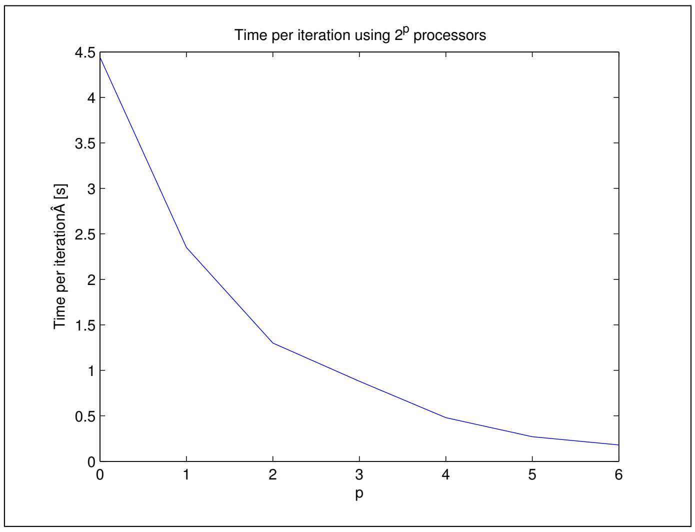


Figure 10.10: Time per iteration of a CCSD run on H_2O with the 6-311++G(3df,3pd) basis set using 2^p processors.

The iteration time for 64 processors oscillated between 0.17 and 0.18. 0.18 was more frequent. This is an excellent scaling for this system, from 4.44 to 0.18 s. We can not expect linear scaling in CCSD, this has never been accomplished. In section 7.2.7 we defined the overhead.

$$T_O = pT_P - T_S \tag{10.8}$$

We can insert the values and calculate the overhead.

P	CCSD iteration time [s]
1	-
2	0.26
4	0.76
8	2.60
16	3.24
32	4.20
64	7.08

Table 10.13: Overhead in parallel CCSD calculation for H_2O.

The overhead says how much computational resources could be saved in total if we waited for the results to be calculated in serial. The overhead is dominated by communication and serial computation. We do not calculate the energy and τ in parallel, these are N^4 calculations. Since these are involved we can never expect the time per iteration to reach exactly zero. S

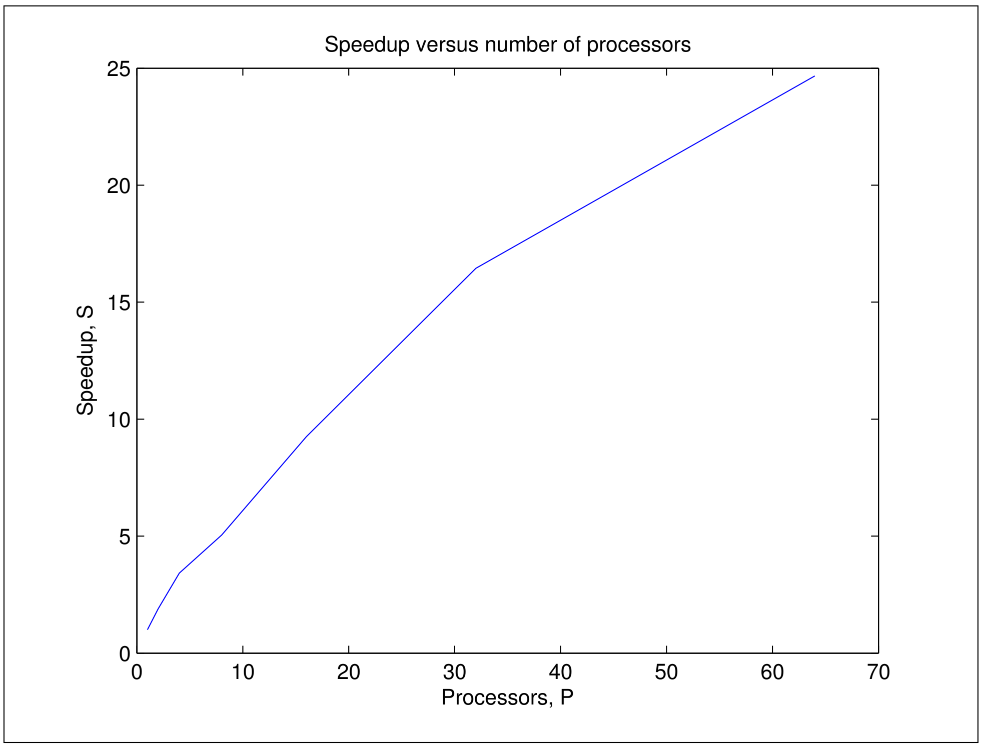


Figure 10.11: The speedup, S, as a function of number of processors for a CCSD iteration on H_2O with the 6-311++G(3df,3pd) basis set.

and T_O are closely related. If $T_O = 0$ we will have linear scaling. However we are happy that we are approaching zero. Communication in CCSD scales as $n_v^2 n_o^2$, while calculation scales as $n_v^4 n_o^2$. As we saw for the AOtoMO transformation, we are likely to get an improved scaling for larger system sizes.

We also test a system of two water molecules. We use the same basis set, to achieve a system of exactly twice the size. We are only interested in the performance of our program, so we can place the two molecules in any location. The raw performance data is included in table 10.14. The performance is plotted in figure 10.12 and S is plotted in figure 10.13.

P	CCSD iteration time [s]
1	300
2	225
4	120
8	59.0
16	30.2
32	15.6
64	8.3
128	4.8

Table 10.14: Measure of parallel performance of CCSD time per iteration with two water molecules with the 6-311++G(3df,3dp) basis set.

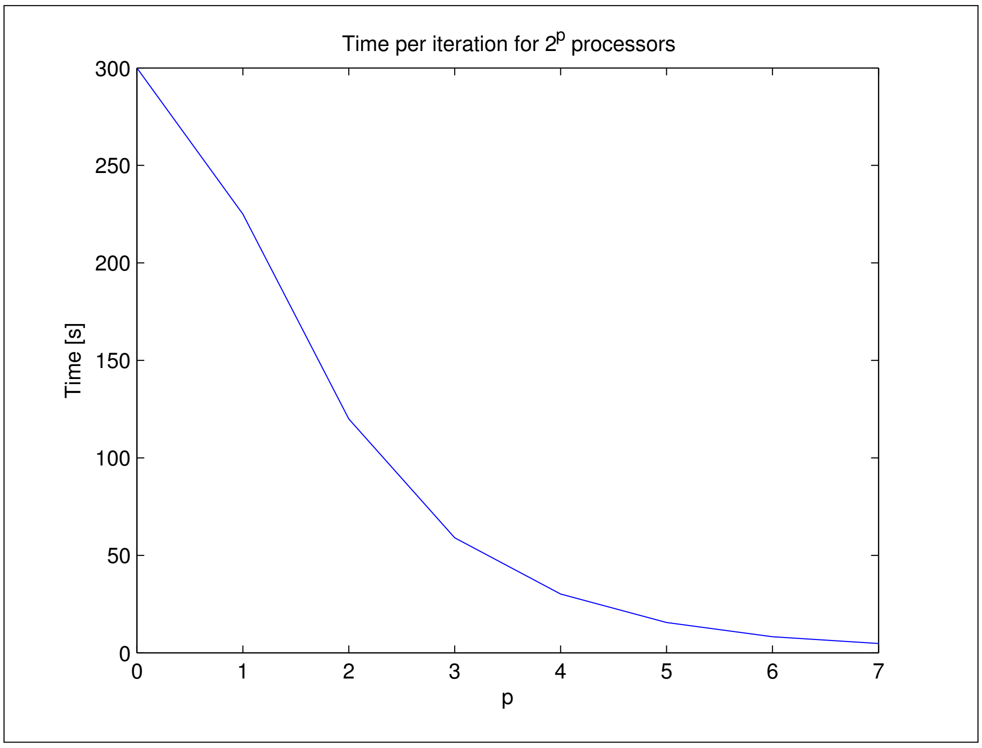


Figure 10.12: Time per iteration of a CCSD run on $(H_2O)_2$ with the 6-311++G(3df,3pd) basis set using 2^p processors.

In figure 10.12 we see some weird behaviour when going from one to two CPUs. This is due to the sub-optimal work distribution noted in section 8.6.2. However from four to eight CPUs, and higher, we do not have this problem. This is because the sub-optimal distribution does not get worse with an increased number of CPUs, it stays at the same sub-optimal level.

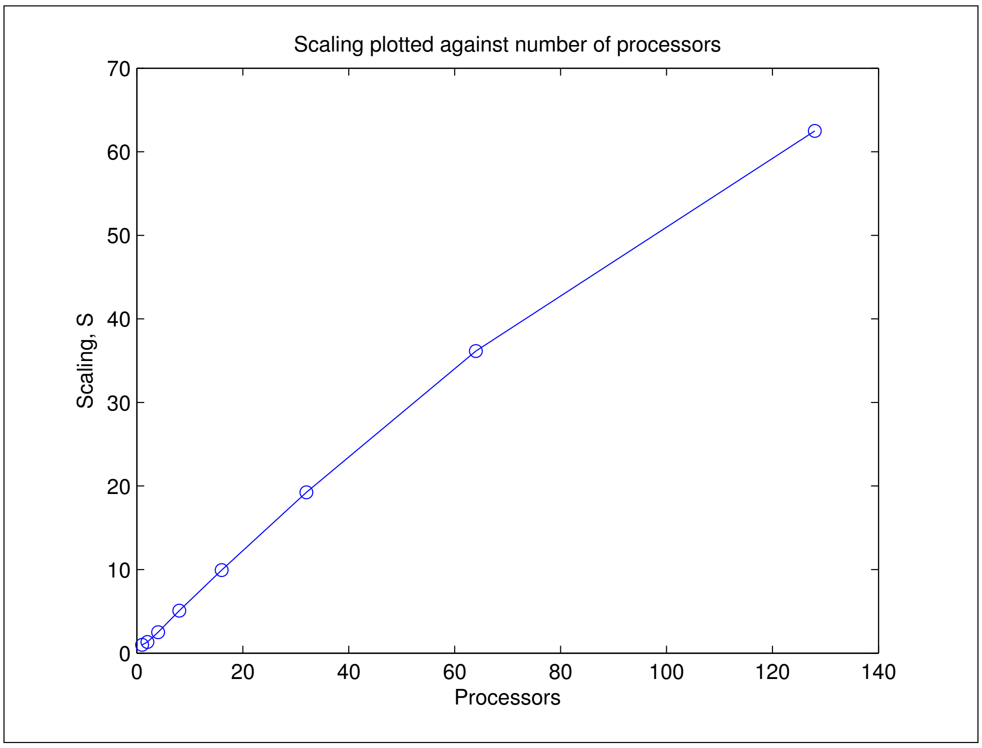


Figure 10.13: The speedup, S, as a function of number of processors for a CCSD iteration on $(H_2O)_2$ with the 6-311++G(3df,3pd) basis set.

We still have a good performance with time per iteration approaching a value close to zero, but with an optimal work distribution in part 1 of the implementation we would approach zero even faster. How sub-optimal the distribution is depends on the system. However distribution of part 2 and 3 is always optimal.

However we do notice that in figure 10.11 the speedup, S, for 64 CPUs is approximately 25. That is

$$S_1(P = 64) = 25, \tag{10.9}$$

whereas S for the larger system we see from figure 10.13 at P = 64 is

$$S_2(P = 64) = 40 \tag{10.10}$$

This is interesting because it shows S improves greatly with larger system size. This means T_O decrease, and we have less wasted computational resources. The improvement from a doubling of system size is much higher here than we saw for AOtoMO, as expected from the larger difference in scaling of computation relative to communication.

We also see this increase in S even with the much more sub optimal work distribution from $P = 1$ to $P = 2$ we see in figure 10.12. This is great news for further optimizations, and will be noted in chapter Future Prospects.

10.10 Potential Energy Plots

Here we present two potential energy plots for the HF and BH molecules. We are in agreement with the benchmarked values in Ref.[70]. Plots are presented in figures 10.14 and 10.15.

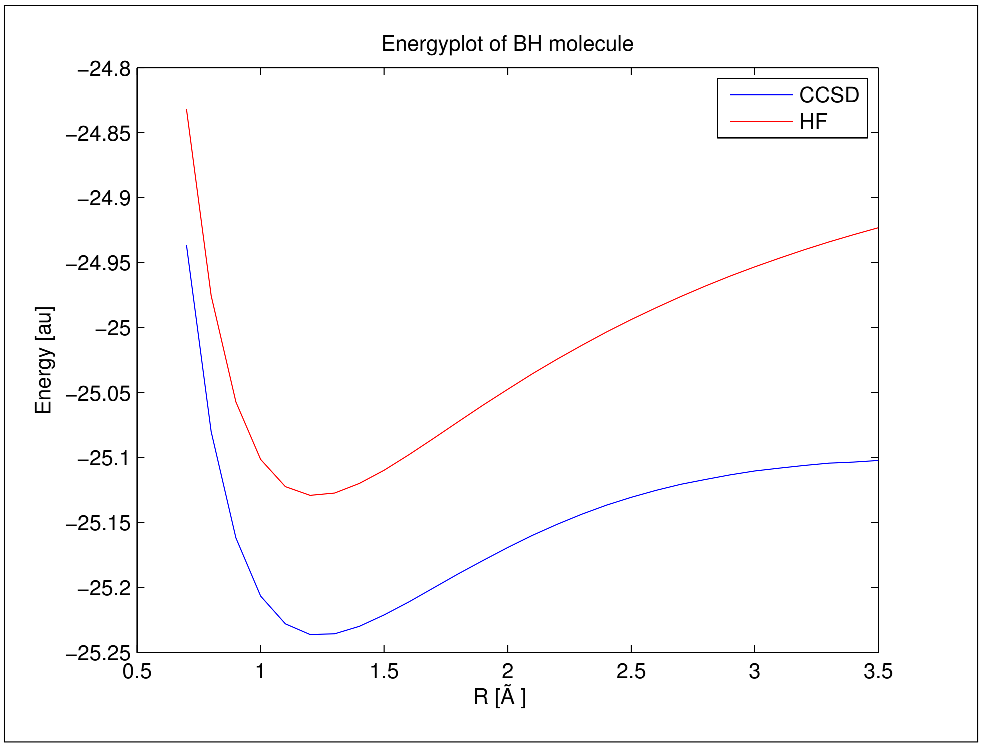


Figure 10.14: Energypotential for BH Molecule, RHF and RHF-CCSD

CCSD generally improve our results. However there are some features with our CCSD calculations we would like to investigate further. We here presents plots of the number of iterations as a function of R for the HF molecule. We also plot the correction to the HF energy, E_{CCSD}, as a function of R.

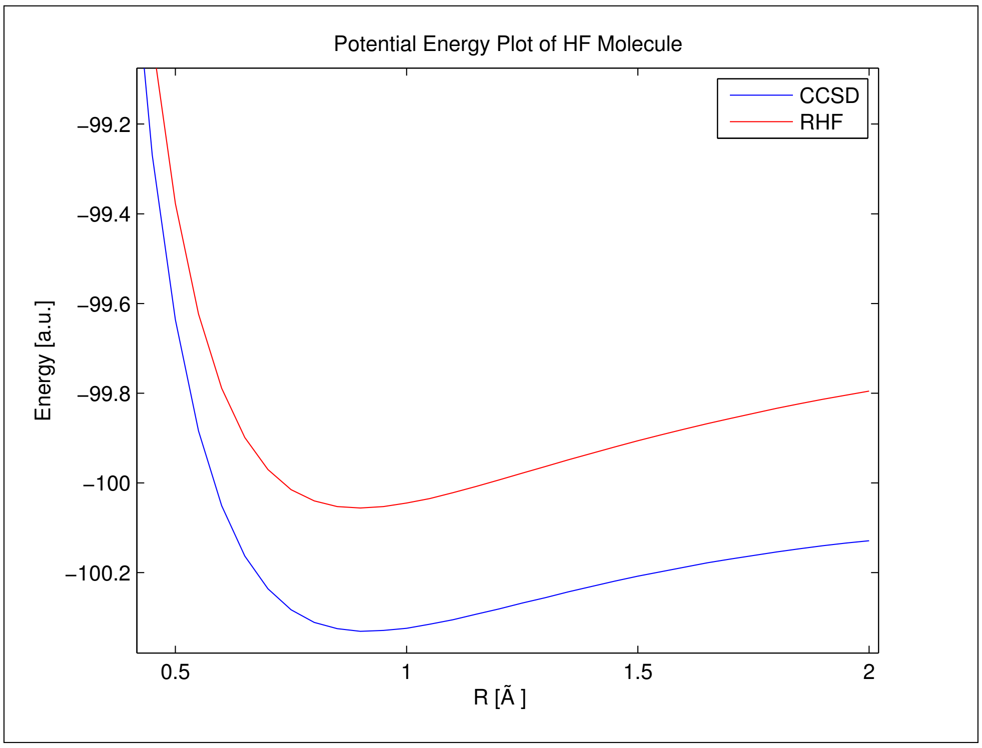


Figure 10.15: Potential energy of HF Molecule. RHF and RHF-CCSD

We notice the number of iterations required for self consistency increase when moving away from the equilibrium geometry. If we move to far away the CCSD correction begin either diverging or oscillating between solutions. We have plotted the number of iterations required for self consistency in figure 10.16. The CCSD correction energy is plotted in figure 10.17. From figure 10.16 we see that CCSD is an equilibrium geometry method. It does not always work outside of equilibrium. If the molecule is to far away from equilibrium we can expect a diverging or oscillating CCSD energy.

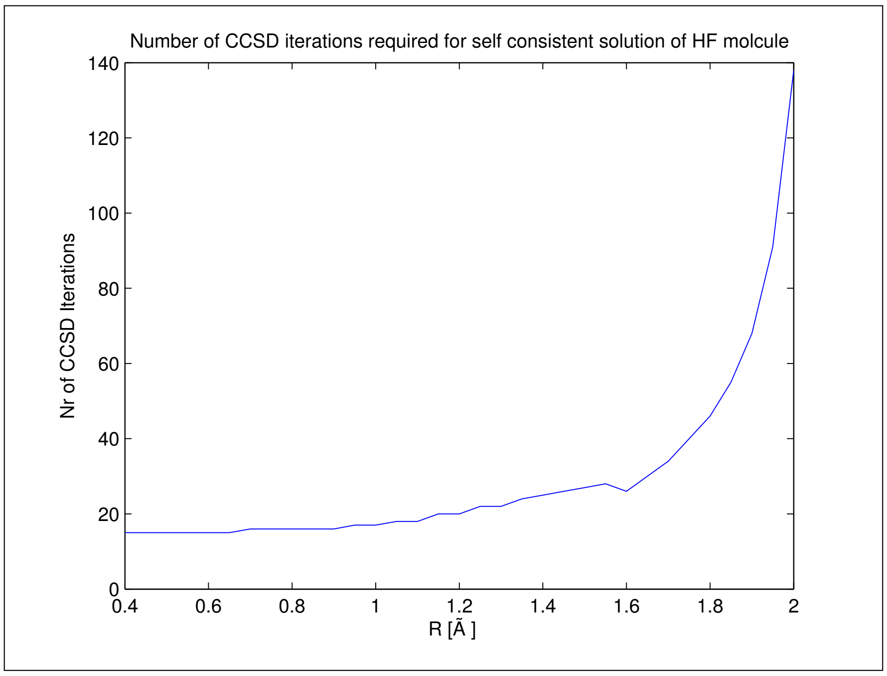


Figure 10.16: HF Molecule

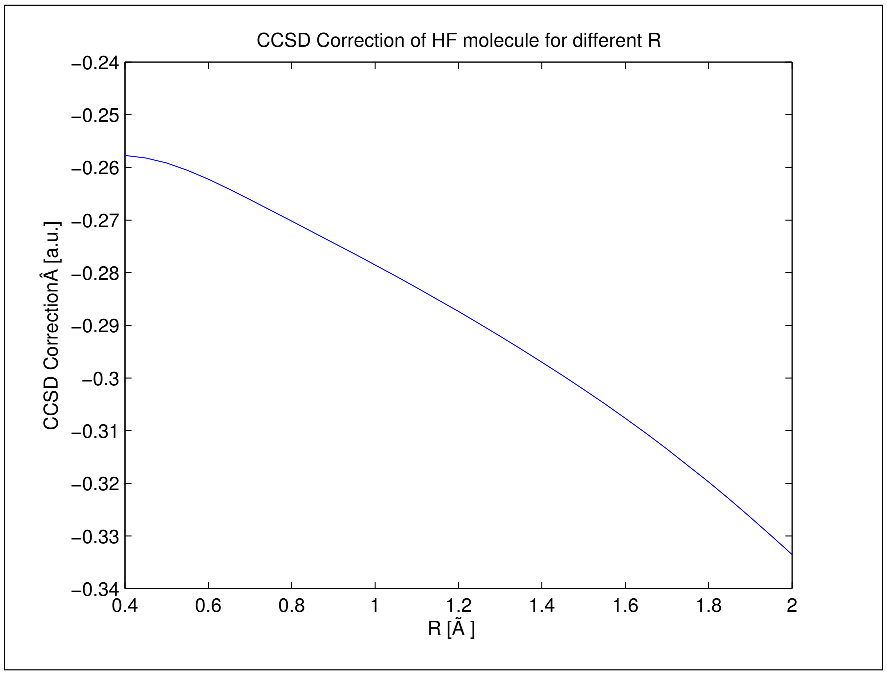


Figure 10.17: HF Molecule

Chapter 11

Results

In this chapter we present our new results. We will experiment with our code, test its performance against existing software and look into features we found interesting during development. We will provide a contribution to the advancement of computational chemistry and specifically the Computational Physics Group at UiO.

11.1 Single Atoms For Future Reference

In this section we do calculations on single atoms. Our hope is that these results may be used as a benchmark for future calculations, using different methods. Since Pople basis sets are not available for all atoms, we can only perform reliable calculations where they are available.

For even number electrons we will use RHF as a referance for CCSD, or CCSDT. For odd number electrons we will use UHF. In UHF we will assume we have one extra electron with spin up, relative to the number of spin down.

We will where available provide a reference energy from Jorgen Hoberget, comparing our results to his DMC calculations, Ref.[88]. Our results are presented in table 11.1.

We see in this table our results are generally on par with Jorgens DMC calculations. Generally speaking, his calculations are more accurate than ours for small systems, whereas for Kr especially we have much better results. For CCSD calculations on any of these atoms we only use one CPU, and do not need much computational resources.

Z	Atom	Basis Set	Method	Energy	DMC
1	H	6-311++G(3df,3pd)	UHF	-0.499818	
2	He	6-311G**	RHF-CCSD	-2.89057	-2.9036(2)
3	Li	6-311++G(2d,2p)	UHF	-7.4321	
4	Be	6-311++G(2d,2p)	RHF-CCSDT	-14.6341	-14.657(2)
5	B	6-311++G(2d,2p)	UHF	-24.5313	
6	C	6-311++G(2d,2p)	RHF-CCSDT	-37.7383	
7	N	6-311++G(2d,2p)	UHF	-54.3402	
8	O	6-311++G(2d,2p)	RHF-CCSD	-74.884	
9	F	6-311++G(2d,2p)	UHF	-99.4014	
10	Ne	6-311++G(2d,2p)	RHF-CCSDT	-128.798	128.765(4)
11	Na	6-311++G(2d,2p)	UHF	-161.847	
12	Mg	6-311++G(2d,2p)	RHF-CCSDT	-199.774	-199.904(8)
13	Al	6-311++G(2d,2p)	UHF	-241.874	
14	Si	6-311++G(2d,2p)	RHF-CCSD	-289.014	
15	P	6-311++G(2d,2p)	UHF	-340.68	
16	S	6-311++G(2d,2p)	RHF-CCSD	-397.692	
17	Cl	6-311++G(2d,2p)	UHF	-459.476	
18	Ar	6-311++G(2d,2p)	RHF-CCSD	-527.056	-527.30(4)
19	K	6-311++G(2d,2p)	UHF	-559.15	
20	Ca	6-311++G(2d,2p)	RHF-CCSD	-677.096	
32	Ge	6-311G**	RHF-CCSD	-2075.66	
33	As	6-311G**	UHF	-2234.12	
34	Se	6-311G**	RHF-CCSD	-2400.17	
35	Br	6-311G**	UHF	-2572.36	
36	Kr	6-311G**	RHF-CCSD	-2752.44	-2749.9(2)

Table 11.1: Single atom energy calculations for future reference, using a variety of methods and basis sets. We define Z as the charge of the atom, Energy as the energy of the given method and DMC as Jorgens DMC calculations from Ref. [88]

11.2 Methods

In our coupled cluster machinery there are two main approximations, a truncated basis set and the limitation of how many excitations are included. In the benchmark chapter we looked into convergence with respect to the basis set. Now we also want to check convergence when we include higher order excitations.

We start with the LiH molecule. We will use the largest Pople type basis set, 6-311++G(3dp,3df). We will use Ref.[85] as a reference. The equilibrium distance given in this paper is R = 3.015 a.u.

The LiH molecule contains four electrons. Here two can be considered core electrons and two valence electrons.

Method	Correction	Energy
HF	-	-7.986 376 7
CCSD	-0.053 444 1	-8.039 820 8
CCSDT-1a	-0.053 536 7	-8.039 913 4
CCSDT-1b	-0.053 536 9	-8.039 913 6
CCSDT-2	-0.053 537 0	-8.039 913 7
CCSDT-3	-0.053 537 4	-8.039 914 1
CCSDT-4	-0.053 557 3	-8.039 934 0
CCSDT	-0.053 555 5	-8.039 932 2
[85], est E_0	-	-8.070 548 0

Table 11.2: Various CCSDT-n method results for the LiH molecule at equilibrium geometry

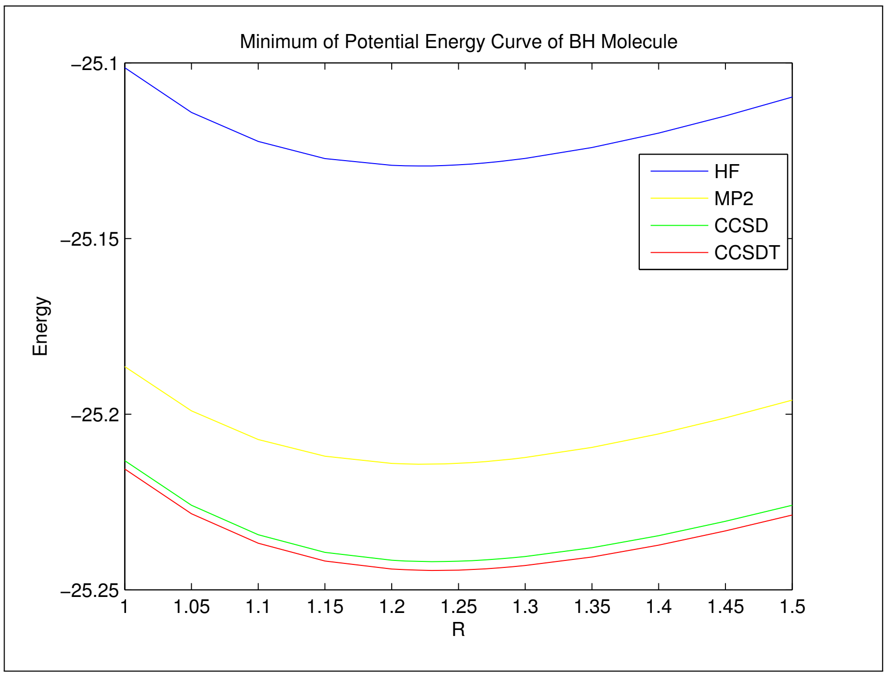


Figure 11.1: BH Potential Energy Minimum plot. Methods in use RHF, MP2, CCSD and CCSDT. Distances in Angstrom.

We notice the energy does not change much. The change is in the fourth decimal. However we suspect this is because we only have two valence electrons.

We also perform calculations on the BH molecule using all our methods. We are interested in a small area around the equilibrium distance found previously in figure 10.14. We will compare RHF, CCSD, CCSDT-1a and CCSDT. We also include MP2 calculations performed with LSDALTON for comparison. Results are available in figures 11.1 and 11.2.

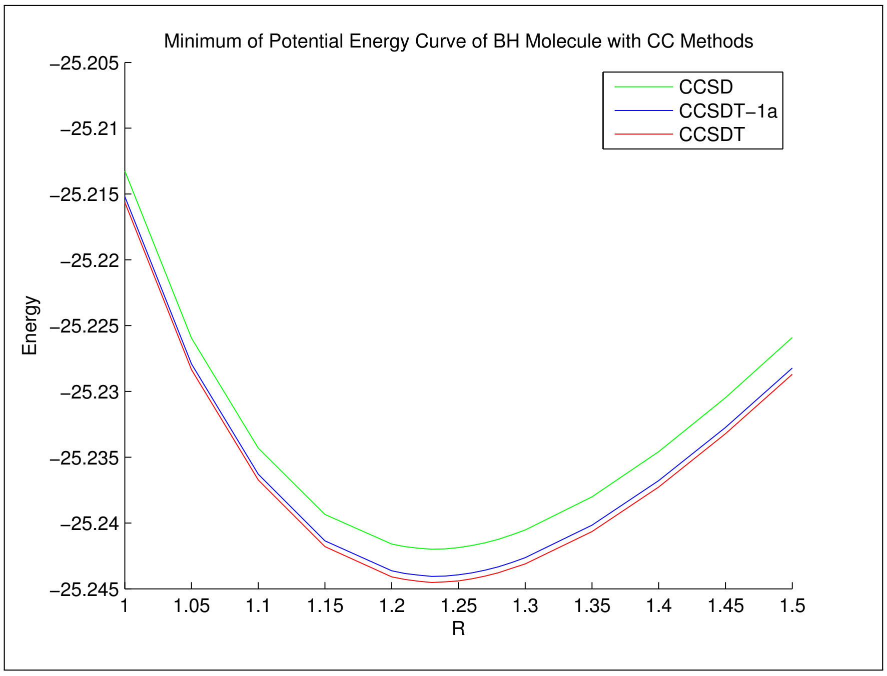


Figure 11.2: BH Potential Energy Minimum plot. Methods in use CCSD, CCSDT-1a and CCSDT. Distances in Angstrom.

We have performed calculations with a resolution of 0.05 Angstrom. Around the minimum the resolution is increased to 0.01 Angstrom. For this system we see different methods provide different results.

The improvement from HF to MP2 is larger than from MP2 to CCSD. Also we see the improvement from MP2 to CCSD is larger than the improvement from CCSD to CCSDT. We see a convergence of the energy with respect to the contributions included. This is illustrated in figure 11.3.

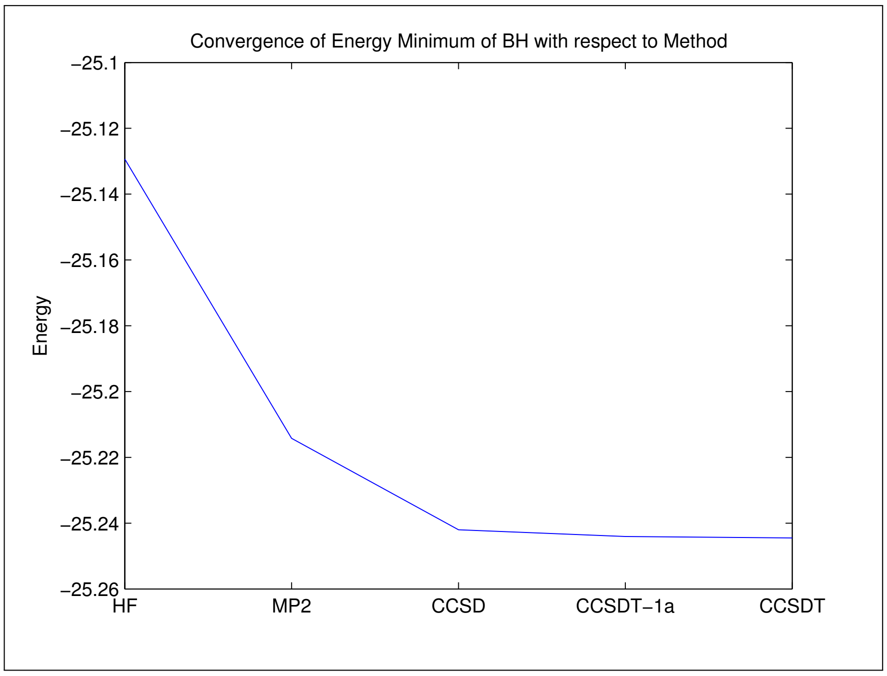


Figure 11.3: BH Energy minimum with respect to Method plot. Methods in use HF, MP2, CCSD, CCSDT-1a and CCSDT. MP2 results from LSDALTON.

We also notice that for HF and MP2 the energy minimum is R = 1.22 Angstrom. For CCSD and CCSDT the minimum is shifted to 1.23 Angstrom.

11.3 CCSD Performance

In this section we will test our CCSD performance against ACESIII. There are a few challenges when making this test, especially that we do not have implemented frozen core in our optimized AOtoMO transformation. We cannot perform the AOtoMO calculation with the unoptimized algorithm for the systems available on the ACESIII benchmark site. They are simply to large. We need the optimized AOtoMO algorithm, and as such we cannot perform frozen core calculations. The benchmarked values on ACESIII benchmark site are with frozen core.

However since CCSD calculations depend on n_o and n_v, we will create a system of the same size and compare performance. We will benchmark against a C_{20} calculation with ACESIII from 2009, [34]. We are interested in program performance. Their calculations are performed with the system specifics described in table 11.3.

System 1	C_{20}
Electrons	120
Basis Set	cc-pVDZ
Contracted GTOs	280
Frozen Orbitals	20 core
n_o for CCSD	80
n_v for CCSD	440

Table 11.3: System input for use in ACESIII

The hardware in use here is the Kraken supercomputer, [74]. In 2009 Kraken was the most powerful supercomputer in the world managed by an academia. Each node on Kraken has the hardware specifications described in table 11.4.

Two 2.6 GHz six-core AMD Opteron processors (Istanbul)
12 cores
16 GB of memory
Connection via Cray SeaStar2+ router

Table 11.4: Hardware in use for ACESIII calculation

We will use a slightly different system, but we have ensured n_v and n_o is approximately the same as for the prior system. Our system is described in details in table 11.5.

System 2	$C_{13}H_2$
Electrons	80
Basis Set	6-311G**
Contracted GTOs	259
Frozen Orbitals	0
n_o for CCSD	80
n_v for CCSD	438

Table 11.5: System input for use in our program

We perform our calculations on the abel computing cluster, [69]. Both ACESIII and our calculations are performed with 200 CPUs. We note our system has two virtual orbitals less.

Code	Time per Iteration [min]
ACESIII	1.6
Our Results	4.17 (250 s)

Table 11.6: Performance results of large parallel CCSD calculation

The AOtoMO transformation took 140 seconds. This is less than one iteration in our program, and close to one iteration in ACESIII.

Compiler flags can optimized our code further. Compiler flags allows the compiler to perform further optimizations, and can sometimes affect the resulting energy. We want the performance to be a realistic measure of how fast our program can produce real results. For this reason we did not use any flags in the compiler until now, but we will try this calculation once more using the flags recommended by abel.

```
CFLAGS = -pipe -O2 -xAVX -mavx -fomit-frame-pointer -fno-alias -
    Wall -W
```

These compiler settings produce a runtime per iteration of $T_P = 230$ s (3.8 m).

Chapter 12

Conclusion

We have developed a complicated many-particle code which can perform *ab initio* calculations of atoms and molecules using the most sophisticated basis functions available. The code includes several methods, with a Hartree-Fock solver used to generate an optimal single-particle basis. This basis serves then as an input to Coupled Cluster theory. Here we have developed an extremely efficient and parallel implementation of both Coupled Cluster with singles and doubles and excitations as well as a serial program of the more complicated triples correlations. Our code produces results in excellent agreement with the existing literature and can also be used to study systems that heretofore have not been investigated using coupled cluster theory. We have performed some calculations, like the convergence of energy with respect to method, we have not seen elsewhere. Our code opens up for many interesting applications. The aim of this chapter is to review our results and benchmarks. We will review each part of our code and draw conclusions of positive and negative features of our implementation, and also review other results.

12.1 Performance

12.1.1 HF

Our Hartree-Fock (HF) code was not particularly optimized. But since we wish to run a Coupled Cluster calculation afterwards, we do not need an extremely fast HF code.

The parallel performance plot, see figure 10.7, exhibits some strange behaviour. We double the number of processors and more than double performance. This seems like super linear scaling, but it is not the case. In HF we distributed the jobs of calculating $\langle ij|kl\rangle$. These are calculated in a sequence, and if we have more than one CPU each CPU gets its own sequence of integrals to calculate. However the amount of memory needed for each integral remains constant. Our implementation excludes super linear scaling, as each individual integral should be calculated in the same amount of time.

We did not average our results over multiple runs, and the same CPU does not always produce the same performance even for the same exact code. This likely explains the situation.

Even though we did not spend as much time optimizing the performance of our restricted Hartree-Fock (RHF) part, it is currently good enough for the largest CCSD calculations we have available. We did not make much use of unrestricted Hartree-Fock part (UHF).

DIIS was implemented and used, but mostly to achieve convergence when needed. If we were to calculate two electron integrals on the fly, DIIS would be much more important. In this situation if we reduced the number of iterations required to a half, we would also reduce the runtime of our program to a half.

12.1.2 AOtoMO

Our AOtoMO performance is good. We have been unable to find anyone beating this performance. The scientific literature from the authors point of view lacks high performance full AOtoMO transformation algorithms and performance results.

Reference [36] lists some results. They do calculations on ethylene (C_2H_4) with cc-pVTZ basis set. This is 114 orbitals. We will compare their results to our results for 130 AOs, listed in section 10.7. We take the best results from Ref.[36]. We also list how much faster our algorithm runs, relative to theirs.

CPUs	[36], 114 AOs [s]	Our, 130 AOs [s]	Fraction
1	699.30	123.55	5.7
2	382.10	69.64	5.5
4	210.90	37.73	5.6
8	132.50	25.51	5.2
16	99.4	17.11	5.8
32	-	14.52	-

Table 12.1: AOtoMO performance measured against existing results.

The AOtoMO transformation comes to dominate calculations in Ref.[36].

With this new algorithm this can be avoided. Ref.[36] is a paper about further developments of the Tensor Contraction Engine, discussed previously. These are leading scientists in computational chemistry. It is very likely their transformation performs quite well relative to others.

The article also mentions the use of point group symmetry, which will be discussed later. This makes coupled cluster calculations faster, and can potentially speed up the AOtoMO transformation also. No mention was made if they applied it in the transformation. The difference in number of AOs is also significant. We did perform calculations for 118 AOs in section 10.7. With 64 CPUs the runtime was half that of 130 AOs with 64 CPUs. With this in mind the outperformance is likely much higher than the fractions listed here.

The key feature of our algorithm is memory distribution. The non distributed memory scales as N^3. However when the number of AOs become so large that N^3 does not fit in memory, we can introduce an additional intermediate after two quarter transformatinos. This would be of size N^4 and distributed in memory. Then we would have non distributed memory scaling as N^2, and would only need a large amount of CPUs. The amount of CPUs in use for the largest CCSD calculations make our algorithm for the full AOtoMO transformation feasible, in most cases.

However not all post HF methods require the full AOtoMO transformation. Much work has been done to avoid the transformation partially or entirely. If performing for example MP2 calculations only a partial AOtoMO can be performed with good results. See Refs.[71], [72] and [73]. In general, CCSD without approximations is one of the faster post-HF method that do require the full AOtoMO transformation. For these reasons we conclude that our AOtoMO transformation is extremely effective. For most calculations the transformation become insignificant in terms of runtime.

12.1.3 CCSD

The CCSD implementation is quite large, and performs very well. Our results are approximate 2.5 times slower than ACESIII for the benchmarked system. However CCSD scales as N^6, and 2.5 times slower for this sized system is not bad. We also have several obvious further optimizations available. This will be discussed in detail in the chapter Future Prospects. ACES has been optimized over more than 20 years. This is the first implementation of CCSD in quantum chemistry on the computational physics group. We do conclude that it is very possible to beat ACESIII, [34], and other high performance software in performance.

The parallel performance is sub optimal. Especially when going from 1 to 2 CPUs, as seen in figure 10.12. The scaling from 4 to 8 CPUs and so on is much better. This is because the work distribution stays at the same sub-optimal level once a given number of CPUs are in use. The sub-optimal distribution is discussed in section 8.6.2, and a solution is proposed in chapter Future Prospects.

The serial algorithm is extremely fast. Serial performance for CCSD is not as interesting because whatever the performance is, we will need multiple CPUs for large systems when using our memory distributed model. We wanted to make comparisons of performance with LSDALTON, but LSDALTON applies approximations to enable faster calculations on larger systems. For this reason we could only benchmark smaller systems down to the final decimal. Also we were unable to print time per iteration in LSDALTON. .

In section 10.8 we also noticed a better performance on the office computer relative to abel. This may be caused by armadillo having problems working with Intel MKL. This can be a problem because armadillo does not return error if the linking is not correct, it only returns suboptimal performance. Also we did not use any special compiler settings in the makefile to optimize performance.

However a single CPU on abel is not expected to outperform significantly. The great thing with supercomputers is the massive number of CPUs available. As such we cannot expect any huge performance gain if this is a problem. This situation remains slightly unclear.

In section 10.8 we also see how our optimized implementation compares to the unoptimized, but factorized, implementation. We see the scaling of performance is improved dramatically. Small optimizations for small systems turn into huge performance gains when the system size increase.

12.2 Basis sets and Convergence

We notice the CCSD energy was not converged in figure 10.5. Similar results are noted in Refs.[81] and [4]. We expected HF to converge faster than CCSD based on the discussion in section 4.8. The CCSD energy for H_2O decreased with approximately 0.1 a.u. from the second largest to the largest Pople type basis set. The HF energy is much better converged.

We also note that the CCSD calculation scales with the number of virtual and occupied orbitals. The hotspot in HF is the two electron integrals, which are based on the number of primitive GTOs. This is a fundamental difference. Pople type basis sets use Cartesian GTOs. For $l = 2$ we then have 6 orbitals. Spherical GTOs only have 5 orbitals for $l = 2$. Since the values for α_i are shared for these orbitals with $l = 2$, it would not make much of a difference in an optimized HF code. It does however make a huge difference in CCSD, as more orbitals results in more time consuming calculations.

For these reasons we conclude that Spherical GTOs are better suited for CCSD. The basis sets that should be used are of the type cc-pVDZ, cc-pVTZ, aug-cc-pVDZ etc. The implementation of spherical GTOs is a bit more complicated according to the literature.

12.3 CCSDT

Our implementation of CCSDT is not optimized and as such we can only run smaller systems.

For LiH we noticed the energy did not change much from CCSD to any of the CCSDT-n methods. This system only has two valence electrons. This means any triple excitation involves excitation of a core electron. For this system the error from the limited basis set is dominating.

For the BH molecule we noticed a larger contribution from triple excitations. Since our CCSDT implementation is not optimized we are limited to smaller system sizes. CCSDT calculations for the BH molecule potential energy curve took one day on abel.

In figure 10.5 we see the energy as a function of basis set size. In figure 11.3 we see the energy as a function of method in use. These are the sources of errors in coupled cluster theory. Both of them are converging towards some value. The Coupled Cluster method is in theory exact if we do not truncate and have an infinite large basis set. Our results are in agreement. From the two figures we conclude the largest errors for the systems we have studied come from the limited basis set, since this plot shows a worse convergence.

We also noticed the equilibrium distance between the atoms changed with different methods. We conclude that not only the energy is converging, but also other physical properties.

Also interesting is how good the CCSDT-1a model recovers the triple excitation contribution for the BH molecule. This method is much faster with lower scaling. Also the new T3 amplitudes in CCSDT-1a does not depend on the old T3 amplitudes. This means we do not need to store them, since they are not reused.

Chapter 13

Future Prospects

In this chapter we discuss future prospect for all parts of our implementation. We will give our answer on how to enable larger and faster calculations. Our main focus will be on the CCSD part of our implementation, but most or all of the discussion will apply also to CCSDT.

13.1 Hartree Fock Performance

Our implementation of HF is not optimized. There are several obvious optimizations available to enable larger molecules in calculations. The most obvious is making use of α_i values.

Several orbitals share α_i values. As such the two electron integrals over primitives can be reused. In our implementation we have included a few comments on which values are changed and which are constant when the only change from one orbital to another is the value of c_i and the composition of angular momentum, m, n and o. The comments are included throughout the code, but mainly in the function Electron_Electron_Interaction_Single in the Hartree_Integrals class.

The way our implementation make use of symmetries and especially our parallel implementation made it difficult to make use of this optimization for now.

Also a huge performance gain is available through integral screening. Integral screening is described in detail in Refs. [8] and [9]. These references also describe other HF optimizations.

13.2 CCSD Performance

The CCSD implementation is quite large and is constructed from scratch. We had no prior experience with coupled cluster. For this reason we are left with several potential optimizations. These will be discussed here.

13.2.1 Work Distribution

The first is the work distribution of part 1. This is suboptimal, but the only reason it is was time constraints. The work distribution should be optimized further, by using the shifts already introduced.

In our program we first distribute the jobs of calculating terms in $[W_1]$. When this is complete, the CPU that got the last job with this array is noted. When we start distributing jobs for $[W_2]$, the first job will go to the next CPU after the one that got the last job in $[W_1]$. This should be continued, so that any irregularities in job distribution is removed as much as possible.

We implemented this shift on one of the arrays in part 1 of parallel CCSD implementation. From figure 10.12 we see the terrible scaling from 1 to 2 CPUs. This is normally were we expect the best scaling. Close to 100% speed up is not uncommon in CCSD from 1 to 2 CPUs. If this work distribution is fixed we will likely see a fantastic increase in performance.

13.2.2 Memory Access

The next optimization we did not have time to implement is the problem with field in armadillo. Armadillo field is inefficient. When we removed fields and implemented one dimensional vectors as intermediates in our AOtoMO transformation we experienced a significant performance gain. Most resent armadillo patches contains some optimizations on fields. The code will likely improve in performance if armadillo is simply patched.

However even matrix and vector are not as efficient as normal C malloc() syntax. If we want the fastest CCSD program in the world armadillo should be removed and replaced with normal arrays of doubles. This would also remove the additional mapping and reduce the number of lines in our implementation. This would not only make it faster, but also more readable.

13.2.3 All Out Memory Distribution

Also the memory distribution can be improved. If we were to activate OpenMP in our implementation, and hold the largest arrays as shared in memory over 16 CPUs, we would have 64 GB of memory available. This is enough to perform almost all the benchmark calculations from ACESIII and NWChem, [34], [33]. However we did not experience any performance gain using OpenMP. Also a shared memory model is limited to the number of CPUs on a node. We want a memory distribution algorithm where the systems that fit in memory are arbitrarily large for an infinite number of CPUs.

This can only be achieved if we either read everything from disk or distribute both arrays in all matrix multiplications. We will illustrate with an example on the term

$$t_{ij}^{ab} \leftarrow \sum_{cd} I_{ab}^{cd} \tau_{ij}^{cd}. \tag{13.1}$$

In our current implementation we hold τ_{ij}^{cd} in memory on all CPUs, and distribute I_{ab}^{cd} among P CPUs. The memory of these two terms scales as

$$M = \frac{n_v^2}{P} \times n_v^2 + n_v^2 n_o^2. \tag{13.2}$$

We want to define communication groups. We will split the P CPUs in U communication groups, where each group consists of O CPUs.

$$P = U \times O. \tag{13.3}$$

We want O CPUs to distribute τ_{ij}^{cd} among themselves, and we want to distribute I_{ab}^{cd} among the U communication groups. The memory in this situation would scale as

$$M = \frac{n_v^2}{U} \times n_v^2 + \frac{n_o^2}{O} \times n_v^2. \tag{13.4}$$

If memory scales as Eq. (13.2) $n_v^2 n_o^2$ is the limiting factor, and we still have highly limited available system size. However if memory scales as Eq. (13.4) we can optimize U and O to fit any system in memory, if $P \to \infty$. The limitation will be n_v^2.

This turns into an optimization problem with two variational parameters U and O. We also have the constraint that $P = U \times O$. We want to optimize the distribution of O and U based on performance, and ensure that the system fits in memory.

Using communication groups we cannot use the same communication model as is implemented now. We will still need all new amplitudes t_{ij}^{ab} in all communication groups. To achieve this there must be a second communication group implemented. This group will hold all processors that will store the same value of the new T2 amplitudes. This will be U number of CPUs. We can then use a all-to-all personalized communication, like

MPI_Alltoallw. In this communication one CPU from each of the second communication groups will take part. Then each of the second communication group must perform a one-to-all broadcast of this information.

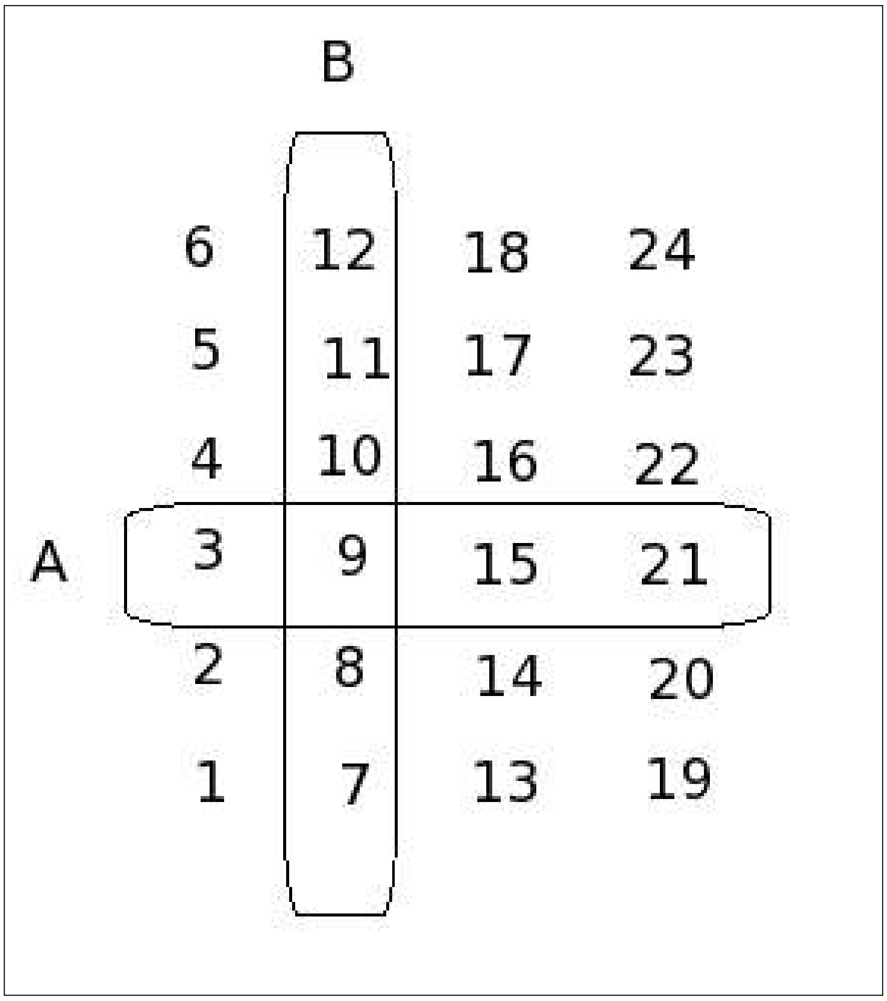


Figure 13.1: Illustration of parallel model. CPUs in row A share common communication rank. CPUs in column B share common calculation rank. One CPU has a unique set of the two.

After the one-to-all broadcast, each of the first defined communication groups will have complete access to the new T2 amplitudes. As such they will be ready for the new iteration. This complication is necessary because we assumed we could not store the full new T2 amplitudes on one CPU. For this reason we cannot use the same communication group as defined in the calculations, because this would require first an all-to-all broadcast of the full new amplitudes followed by a personalized one-to-all scatter. This would exceed memory limits.

This algorithm sounds like a good idea, but it is complicated. Also we initialize more communications. However, we would like to propose a simple and straight forward way to implement this. First we should define two new variables on each processor. These will be dependant upon its rank, and will be unique to each CPU. This is illustrated in figure 13.1.

```
Calculation_Rank = (int) rank / P;
Communication_Rank = rank % (P + 1);
```

These integer division ensure multiple CPUs will get the same value for Calculation_Rank. The modulus with (P+1) ensures all CPUs with the same Calculation_Rank get unique values for Communication_Rank. We want all CPUs with the same Communication_Rank to store the same values for t_{ij}^{ab}. This can be achieved by giving the same displacement and number of bytes to recieve in MPI_Alltoallw, which is an all-to-all scatter function used in our implementation earlier. The displacement should then depend upon the new variable Communication_Rank, and not the normal MPI rank.

Also the additional work distribution can be achieved from adding additional if tests.

```
// Psudocode for optimal work distribution
// in new proposed implementation
for (int a = 0; a < unocc_orb; a++){
    for (int b = 0; b < unocc_orb; b++){
         Work_ID1 = F(a,b); // Some function of a,b
         if (Work_ID1 % P == Calculation_Rank){
            // All CPUs with common
            // calculation rank enter here

            for (int i = 0; i < n_Electrons; i++){
                for (int j = 0; j < n_Electrons; j++){
                    Work_ID2 = F(i,j); // Some function of i,j
                    if (Work_ID2 % U == Calculation_Rank){
                        // Perform calculation. Only one CPU will have
                        // unique set of
                        // calculation and communication rank.
                    }
                }
            }
        }
    }
}
```

In other word the implementation does not really need communication groups to work. The proposed algorithm here will also likely improve performance, in accordance to the principles stated in section 7.2.10. If we distribute work based on four indexes we have a greater number of smaller jobs to distribute. More jobs is easier to distribute evenly, as such the scaling will likely improve. We also avoid broadcasting information, and instead scatter.

Since a scattering of information can be achieved ineffectively by a broadcast, it is safe to assume a scatter is more effectively implemented in MPI.

13.2.4 Removing mapping of two electron integrals

Also the additional mapping introduced to only store the single bar integrals can be removed. This was implemented early on. At this point the author did not believe a full parallel distributed CCSD algorithm was achievable in one master thesis. This will increase performance slightly. However if the points noted in section 13.2.3 is implemented, only storing single bar integrals is preferred from a memory perspective.

13.2.5 Removing Four Dimensional Arrays

In CCSD it is quite normal to contract two and two indexes. The term t_{ij}^{ab} is often not stored as a four dimensional array. It is stored as two dimensional. Often indexes a and b are contracted into an index e, where e can be defined as

$$e = a \times n_v + b. \tag{13.5}$$

Also i and j can be contracted into an index k, where k can be defined as

$$k = i \times n_o + j. \tag{13.6}$$

Using these definitions t_{ij}^{ab} can be calculated directly as a matrix-matrix multiplication. This is way more efficient in terms of memory accessing. We have not jet taken advantage of this optimization, but we have not excluded it.

Implementing this will be slightly more complex when we use the memory distributed model. If we go back to the prior example

$$t_{ij}^{ab} \leftarrow I_{ab}^{cd} \tau_{ij}^{cd}. \tag{13.7}$$

The standard implementation is to define this as a matrix-matrix multiplication where the matrices are of size $n_v^2 \times n_v^2$ and $n_o^2 \times n_v^2$. This will result in a matrix of size $n_o^2 \times n_v^2$. We distribute work based on a and b. This will mean if we contracted the indexes we would distribute work based on the index e. Each processor would perform a matrix-matrix multiplication where the matrices are of size $\frac{n_v^2}{P} \times n_v^2$ and $n_v^2 \times n_o^2$. This would result in a matrix of size $\frac{n_v^2}{P} \times n_o^2$.

If the communication groups in section 13.2.3 is used the matrices for matrix-matrix multiplication would be of size $\frac{n_v^2}{U} \times n_v^2$ and $\frac{n_o^2}{O} \times n_v^2$. The resulting matrix would be of size $\frac{n_v^2}{U} \times \frac{n_o^2}{O}$. Using matrix-matrix multiplications directly will likely cause a fantastic performance gain, and also enable better use of OpenMP. OpenMP was ineffective because the arrays are to small. In a real matrix-matrix multiplication the arrays will be larger.

Also in our current implementation the number of calls to external math libraries scales as N^4. If we implemented two dimensional arrays the number of calls to external math libraries would be constant. This would likely improve the scaling of performance with respect to system size.

13.2.6 Read From File

Another solution to memory concerns is to read the largest array from file. We go back to the previous example

$$t_{ij}^{ab} \leftarrow I_{ab}^{cd} \tau_{ij}^{cd}. \tag{13.8}$$

The variable to read from file is I_{ab}^{cd}. This was previously distributed in memory and with a large number of CPUs it did not take up much space. However, if we read it from file and have it available on all CPUs we can distribute τ_{ij}^{cd} in memory. This enables much larger calculations.

13.2.7 Summary

Several further optimizations has been proposed for our already highly optimized CCSD code. Most of the optimizations require much work. With the exception of section 13.2.6 and maybe section 13.2.3, all proposed optimizations will likely increase performance further. The author believes it is highly likely that a better performance than ACESIII is achievable with only these proposed optimizations.

To the authors knowledge this would be the first time a highly optimized CCSD program is constructed using only a few collective OpenMPI function calls. It is much more popular to use algorithms such as Cannons Algorithm, Refs.[75], [49], or 2.5D communication which has become increasingly popular in recent years, Refs.[63] and [89].

Another popular optimization in recent years is chunksize. Please see Ref.[36] and references therein.

13.3 Further Method Development

We have developed most of our methods with a spin restriction. Removing this spin restriction is of course possible, and required for certain situations. For CCSDT a good article is Ref.[76]. CCSD unrestricted spin can be found in references therein. The CCSDTQ method is described in detail in Ref.[58].

Also possible is merging the code presented here with Henriks, Ref.[4]. This would create a more complete package of quantum chemistry methods. We would have available RHF, UHF, RMP2, UMP2, RMP3, UMP3, RCCSD, RCCSDT and RCCSDT-n methods. R and U signifies restricted and unrestricted spin. In this section we will focus mainly on new approximations that can enable much larger CCSD calculations. We have not examined these approximations in detail, but we will list a few approximations we found interesting.

13.3.1 Natural Orbitals

The largest CCSD calculations performed today take advantage of natural orbitals, Ref.[77]. Simply put, natural orbitals involve dividing orbitals into three segments. Occupied, active and unoccupied. The occupied orbitals are the core orbitals. We can assume these are always occupied. This limits the size of the occupied space, or in other words reduce n_o, since we do not need to include them in CCSD calculations.

Unoccupied orbitals in this context are orbitals so highly excited they are rarely occupied. This is approximated to never occupied, and the orbitals are ignored when performing CCSD. This limits the virtual space, or in other words reduce n_v.

We are left with the active orbitals. These are the valence orbitals and a few orbitals close to the same energy level. However natural orbitals are not as straight forward. Please see Ref.[78]. The performance gain here is high because we limit both n_v and n_o. CCSD scales as $n_o^2 n_v^4$, so we will have much better performance.

13.3.2 Frozen Core

The Frozen Core approximation in quantum chemistry is not the exact same as one would assume. To freeze the core means to ignore the lowest energetic molecular orbitals. If we want to freeze six core orbitals, we ignore the first six molecular orbitals. The hope is that these will correspond to the core orbitals, and in most cases they do.

13.3.3 Local Coupled Cluster

Natural Orbitals can be combined with Local Coupled Cluster, to make Local-Natural-Orbitals Coupled Cluster. Local Coupled Cluster takes advantage of the density as an approximation to achieve performance gains, Ref.[79]. The performance gains available here is in magnitudes of n.

13.3.4 Point Group Symmetry

Points Group Symmetry is especially impressive. These are specified as C1, D1, D2 etc. They reference a certain symmetry. If this symmetry exist in the molecule we can give it as input to our program and take advantage of it, Ref.[80].

Dependant upon the extent of the symmetry, this can provide good performance gains, Refs.[18], [19]. The performance gain available here is not in a magnitude of n, but as a fraction.

13.3.5 Divide and Conquer

Also divide and conquer is a popular approximation. This is relevant to matrix-matrix multiplications. A large matrix-matrix multiplication can be divided into smaller matrices. Each of the smaller matrices can be calculated in parallel, Ref.[82].

Combining all four approximations would give huge performance gains, but they are approximations. Limiting n_v, n_o, reducing the scaling and also adding a fraction in front of wall time, makes large calculations possible for CCSD. The largest the author is aware of is approximately 2000 AOs.

13.3.6 Summary

Most of these approximations can be implemented without changing the CCSD implementation itself. Natural orbitals is an optimization on the basis from HF. Optimizing the basis from HF is practically reducing n_v and n_o. It is not uncommon to see n_v reduced by 50-80%. This reduction is made after a HF calculation and makes the much larger CCSD calculations possible. If we reduce 2000 AOs by 80% we only have $(n_o + n_v) = 800$ in CCSD. 2000 AOs with a decent sized basis set can be a molecule of 60-70 atoms.

Natural Orbitals is a way of improving the basis. Preliminary estimates are made on how likely a virtual orbital is to be occupied, and a cut off is defined where all orbitals beneath this value is ignored. This means 200 Natural Orbitals will likely give much better results than 200 Atomic Orbitals.

We can combine these approximations with the already fast CCSD implementation presented in this thesis.

13.4 Other Methods

Also other methods can be of interest.

13.4.1 DFT

Density Functional Theory (DFT) is the most commonly used method in quantum chemistry. The cost is similar to that of HF and the results are generally more accurate. DFT performs particularly well for metals. We had no Pople type basis sets available for metals. Ref.[83].

13.4.2 Monte Carlo

Monte Carlo (MC) has been applied with a HF and even CCSD probability distribution. Ref.[84].

13.4.3 Perturbation

Möller Plesset calculations provide a slightly less accurate energy correction than CCSD. However it can potentially be calculated much faster. These methods are named MP2, MP3 and so on, dependant on what terms are included. Ref.[4] describes the details.

13.4.4 CCn

The center for computational and theoretical chemistry (CCTC) group at the university of Oslo has developed its own coupled cluster methods. One method, CC2, is closely related to CCSD. CC3 is closely related to CCSDT and so on. Ref.[86]. These are iterative methods.

13.4.5 Approximate Contributions

CCSD(t) is considered the gold standard in quantum chemistry, Ref.[87]. CCSD(t) contains CCSD plus an approximation to higher order amplitudes. Also CCSD[T], formerly known as T(CCSD), is an approximation of triples excitation. Any excitation can be approximated. This is methods like CCSDT(Q), CCSD(TQ) and so on.

13.5 Getting Closer to E_0

We concluded that the largest source of error is now the limited basis set. This was due to the fact that larger Pople basis sets does not exist on the EMSL website. Although it is possible to perform calculations using Cartesian GTOs with basis sets designed for spherical GTOs, it is not optimal. Future work should be implementing spherical GTOs, rather than including higher order excitations.

Since the largest error currently is the basis set we need a larger basis set to get more accurate results. A larger basis set does require more calculations. However a more accurate method generally require higher scaling. Both of these options are evils in terms of computational resources, but a larger basis set is the lesser of the two. This means the outlook for quickly achieving more accurate results is very bright.

Chapter 14

Appendix A

Listed here are some useful MPI functions that deals with communication. Their input can be found on google. This list can be used to figure out what functions exist, and a few basics of what they do.

14.1 MPI Functions

MPI_Bcast()
One to all operation. Broadcasts information from one node to all nodes.

MPI_Reduce()
All to one operation. Gathers information from all nodes onto one node.

MPI_Scatter()
One to all operation. Sends unique information of same size to each node, including self.

MPI_Gather()
All to one operation. Gathers unique information of same size from each node, including self, exact opposite of scatter.

MPI_Scatterv()
One to all operation. Sends unique information of optionally different size to each node, including self.

MPI_Gatherv()
All to one operation. Gathers unique information of optionally different size from all nodes onto one node.

MPI_Recv()
One to one operation. Receives information. Use with MPI_Send.

MPI_Send()
One to one operation. Sends information. Use with MPI_Recv.

MPI_Isend()
One to one operation. Same as MPI_Send but this stores information in a buffer and continues with operations. Then sends the buffer once link with MPI_Irecv has been established.

MPI_Irecv()
One to one operation. Receive information, use with MPI_Isend.

MPI_Get_Count()
Returns precise count of data items received.

MPI_Sendrecv()
One to one operation. Sends and receives information between two nodes. This is faster than initiating two sends and two receives.

MPI_Sendrecv_replace()
One to one operations. Sends and receives information between two nodes, and replaces information. This is faster than initiating two sends and two receives.

MPI_Barrier()
Synchronizes all nodes.

MPI_Allgather()
All to all operation. Gathers information from all nodes like MPI_Gather, but then spreads information to all nodes. Faster than MPI_Gather followed by MPI_Bcast, but same results.

MPI_Alltoall()
All to all operation, broadcasts information from all nodes to all nodes. Faster than MPI_Bcast from all nodes individually, but same result.

MPI_Alltoallv()
All to all operation, like MPI_Alltoall but more freedom.

MPI_Alltoallw()
All to all operation. Each node can have different size of buffer for broadcast, very general function.

MPI_Comm_Split()

Creates a new communication group (communicator). The all in "All to all operation" refer to all nodes in a selected communication group, like MPI_COMM_WORLD. Using this function, "all" can be limited to certain nodes, reducing the number of nodes that take part and increasing speed.

MPI_Comm_size()
Returns how many nodes are a part of a communicator.

14.2 MPI Datatypes

Here are a few basic types in MPI, that is used in the communication and mostly not listed in the function descriptions.

MPI_CHAR , signed char

MPI_SHORT , signed short int

MPI_INT , signed int

MPI_LONG , signed long int

MPI_UNSIGNED_CHAR , unsigned char

MPI_UNSIGNED_CHAR , unsigned short int

MPI_UNSIGNED , unsigned int

MPI_UNSIGNED_LONG , unsigned long int

MPI_FLOAT , float

MPI_DOUBLE , double

MPI_LONG_DOUBLE , long double

MPI_PACKED

MPI_BYTE

Bibliography

[1] Lecture series as part of the Georgia Tech's Summer Lectures Series in Theoretical Chemistry. Available for free online at this url:
`http://vergil.chemistry.gatech.edu/opp/summer-lectures.html`

[2] *Modern Quantum Chemistry.*
Introduction to Advanced Electronic Structure Theory
Attila Szabo, Neil. S. Ostlund

[3] MA Thesis University of Oslo 2014.
Bridging Quantum Mechanics and Molecular Dynamics with Artificial Neural Networks.
S. A. Dragly.

[4] MA Thesis University of Oslo 2014.
Ab Initio Studies of Molecules
H. M. Eiding.

[5] MA Thesis University of Oslo 2014.
From Quantum to Molecular, A Review of Gaussian Basis Sets in Ab Initio Molecular Dynamics.
M. H. Mobarhan.

[6] Koopmans Theorem.
`http://www.youtube.com/watch?v=5T_HyShi9W0`

[7] *Molecular electronic-structure theory. Wiley, 2013*
T. Helgaer, P. Jorgensen and J. Olsen.

[8] *Quantum Chemistry and Molecular Properties: Molecular Integral Evaluation. 2006.*
T. Helgaker.
`http://folk.uio.no/helgaker/talks/SostrupIntegrals_06.pdf`

[9] *Molecular Integral Evaluation. 2010.*
T. Helgaker.
`http://folk.uio.no/helgaker/talks/SostrupIntegrals_10.pdf`

[10] Larry E. McMurchie and Ernest R. Davidson. Journal of Computational Physics. Volume 26, Issue 2, February 1978, Pages 218-231.

[11] John A. Pople and Warren J. Hehre.
Journal of Computational Physics. Volume 27, Issue 2, May 1978, Pages 161-168.

[12] The Hermite Polynomials from wikipedia.
`http://en.wikipedia.org/wiki/Hermite_polynomials`

[13] Feller, D., J. Comp. Chem., 17(13), 1571-1586, 1996.

[14] Schuchardt, K.L., Didier, B.T., Elsethagen, T., Sun, L., Gurumoorthi, V., Chase, J., Li, J., and Windus, T.L.
J. Chem. Inf. Model., 47(3), 1045-1052, 2007

[15] The EMSL Basis Set Exchange website:
`https://bse.pnl.gov/bse/portal`

[16] *An Introduction to Coupled Cluster Theory for Computational Chemists.* T. Daniel Crawford and Henry S. Schaeffer III.

[17] Edgar Solomonik, Devin Matthews, Jeff Hammond, James Demmel.
DOI 10.1109/IPDPS.2013.112
URL:
`http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6569864`

[18] P. Carsky, L. J. Schaad, B. A. Hess, M. Urban, and J. Noga. J. Chem. Phys. 87,411 (1987).

[19] John F. Stanton, Jürgen Gauss, John D. Watts, and Rodney J. Bartlett. Chem. Phys. 94 (6), 15 March 1991.

[20] Henrik Koch, Ove Christiansen, Rika Kobayashi, Poul Jorgensen , Trygve Helgaker. Chemical Physics Letters 228 (1994) 233-238

[21] Jozef Noga and Rodney J. Barlett. Chemical Physics Letters. Volume 134, Issue 2, 20 February 1987, Pages 126-132.

[22] Jozef Noga, Rodney J. Bartlett The Journal of Chemical Physics 86, 7041 (1987).

[23] Erratium: J. Noga and R. J. Bartlett. The Journal of Chemical Physics 86, 7041 (1987).

[24] *Meny Body Methods in Chemistry and Physics: MBPT and Coupled Cluster Theory (Cambridge Molecular Science)*
Isaiah Shavitt and Rodney J. Bartlett

[25] Gustavo E. Scuseria and Timothy J. Lee. The Journal of Chemical Physics 93, 5851 (1990)

[26] MA Thesis University of Oslo 2012.
Studies of quantum dots: Ab initio coupled-cluster analysis using opencl and gpu programming.
Christoffer Hirth.

[27] C. C. J. Roothaan. Rev. Mod. Phys., 23:69, 1951.

[28] J. A. Pople and R. K. Nesbet. J. Chem. Phys., 22:571, 1954

[29] Paul Saxe, Henry F Shaefer III. Chemical Physics Letters Volume 79 Issue 2, 15 April 1981 Pages 202-204

[30] Armadillo source:
`http://arma.sourceforge.net/`

[31] W. Kolos and L. Wolniewicz. The Journal of Chemical Physics 49, 404 (1968)

[32] Claudia Filippi and C. J. Umrigar, J. Chem. Phys, 105, 213, 1996.

[33] NWChem Benchmarks Website:
`http://www.nwchem-sw.org/index.php/Benchmarks`

[34] ACESIII Benchmarks Website:
`http://www.qtp.ufl.edu/PCCworkshop/PCCbenchmarks.html`

[35] Kazuto Nakata, Tadashi Murase, Toshihiro Sakuma and Toshikazu Takada. Journal of Computational and Applied Mathematics 149 (2002) 351 - 357

[36] Automatic Code Generator for Many-Body Electronic Structure Methods: The Tensor Contraction Engine.
Alexander A. Auer, Gerald Baumgartner, David E. Bernholdt, Alina Bibireata, Venkatesh Choppella, Daniel Cociorv, Xiao yang Gao, Robert Harrison, Sriram Krishnamoorth, Sandh ya Krishnan, Chi-Chung Lam, Qingda Lu, Marcel Nooijen, Russell Pitzer, J. Ramanujam, P. Sadayappan and Alexander Sibiryak.
`http://www.csc.lsu.edu/~gb/TCE/Publications/Bartlett-MolPhys06.pdf`
Submitted to Molecular Physics, R. J. Bartlett Festschrift Special Issue

[37] Jonathan L. Bentz, RyanM.Olson, Mark S. Gordon, Michael W. Schmidt,Ricky A. Kendall. Computer Physics Communications 176 (2007) 589-600

[38] I. I. Guseinov, B. A. Mamedov. Journal of Mathematical Chemistry, Vol. 40, No. 2, August 2006.

[39] B. A. Mamedov. Journal of Mathematical Chemistry Vol. 36, No 3, July 2004.

[40] `http://institute.loni.org/NWChem2012/documents/tce-session.pdf`

[41] LSDALTON Program.
K. Aidas, C. Angeli, K. L. Bak, V. Bakken, R. Bast, L. Boman, O. Christiansen, R. Cimiraglia, S. Coriani, P. Dahle, e. K. Dalskov, U. Ekström, T. Enevoldsen, J. J. Eriksen, P. Ettenhuber, B. Fernàndez, L. Ferrighi, H. Fliegl, L. Frediani, K. Hald, A. Halkier, C. Hättig, H. Heiberg, T. Helgaker, A. C. Hennum, H. Hettema, E. Hjertnes, S. Host, I. M. Hoyvik, M. F. Iozzi, B. Jansik, H. J. Aa. Jensen, D. Jonsson, P. Jorgensen, J. Kauczor, S. Kirpekar, T. Kjergaard, W. Klopper, S. Knecht, R. Kobayashi, H. Koch, J. Kongsted, A. Krapp, K. Kristensen, A. Ligabue, O. B. Lutnes, J. I. Melo, K. V. Mikkelsen, R. H. Myhre, C. Neiss, C. B. Nielsen, P. Norman, J. Olsen, J. M. H. Olsen, A. Osted, M. J. Packer, F. Pawlowski, T. B. Pedersen, P. F. Provasi, S. Reine, Z. Rinkevicius, T. A. Ruden, K. Ruud, V. Rybkin, P. Salek, C. C. M. Samson, A. Sanchez de Meras, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. Sneskov, A. H. Steindal, K. O. Sylvester-Hvid, P. R. Taylor, A. M. Teale, E. I. Tellgren, D. P. Tew, A. J. Thorvaldsen, L. Thogersen, O. Vahtras, M. A. Watson, D. J. D. Wilson, M. Ziolkowski and H. Aagren.
The Dalton quantum chemistry program system.
WIREs Comput. Mol. Sci. (doi: 10.1002/wcms.1172).
Dalton, a molecular electronic structure program, Release DALTON2013 (2013), see http://daltonprogram.org.
lsDalton, a linear scaling molecular electronic structure program, Release DALTON2013 (2013), see http://daltonprogram.org.
URL:
`http://daltonprogram.org`

[42] Simplewick, Contractions for Latex:
`http://www.fzu.cz/~kolorenc/tex/simplewick/`

[43] Algorithm2e, Algorithms in Latex:
`http://mlg.ulb.ac.be/files/algorithm2e.pdf`

[44] Latex Package: tikz, Matrix Representation in Latex.

[45] Krishnan Raghavachari, D. L. Strout, G. K. Odom, G. E. Scuseria, J. A. Pople, B. G. Johnson and P. M. W. Gill. Chemical Physics Letters, Volume 214, Issues 3-4, 5 November 1993, Pages 357-361.

[46] http://altair.physics.ncsu.edu/projects/c20/c20.html

[47] Wei An, Yi Gao, Satya Bulusu and Xiao Cheng Zeng.
Article from the Published Research - Derpartment of Chemistry at DigitalCommons@University of Nabraska.
http://digitalcommons.unl.edu/chemzeng/20/

[48] OpenMPI Website:
http://www.open-mpi.org/

[49] *Introduction to Parallel Computing, Second Edition.*
Ananth Grama, Anshul Gupta, George Karypis and Vipin Kumar.

[50] OpenMP Website. http://openmp.org/wp/

[51] OpenBLAS Website. http://www.openblas.net/

[52] Intel MKL Website. https://software.intel.com/en-us/intel-mkl

[53] Youtube Video Lectures: http://www.youtube.com/watch?v=cMWGeJyrc9w

[54] G. E. Scuseria, T. J. Lee, H. F. Schaefer III. Chemical Physics Letters, Volume 130, Number 3, 3 October 1986.

[55] Antara Dutta and C. David Sherrill. Journal of Chemical Physics, Volume 118, Number 4, 22 January 2003.

[56] M.L. Boas. Mathematical Methods in the Physical Sciences. Wiley, 2005

[57] So Hirata. J. Phys. Chem. A. 2003, 107, 9887 - 9897.

[58] A multi-reference coupled-cluster method using a single-reference formalism.
PhD Thesis University of Arizona (1991).
Nevin Oliphant.

[59] Boys S F 1950 Proc. R. Soc. A 200 54

[60] Jonathan L. Bentz, Ryan M. Olson, Mark S. Gordon, Michael W. Schmidt, Ricky A. Kendall.Computer Physics Communications 176 (2007) 589-600.

[61] *Recent Progress in Coupled Cluster Methods volume 11.*
Theory and Applications.
J. Leszczynski.
P. Carsky, J. Paldus, J. Pittner.

[62] 2.5D Communication in CCSD, 2012 seminar free available on youtube. Edgar Solomonik.
https://www.youtube.com/watch?v=MCl5fGvVaLU

[63] 2.5D Communication in CCSD, 2013 slides.
Edgar Solomonik.
http://www.eecs.berkeley.edu/~solomon/talks/matrix-seminar-2013.pdf

[63] 2.5D Communication in CCSD, 2014 slides.
Edgar Solomonik.
http://www.eecs.berkeley.edu/~solomon/talks/ctf-ExMatEx-mar-2014.pdf

[64] Gustavo E. Scuseria, Curtis L. Janssen, Henry F. Schaefer III. Journal of Chemical Physics 89, 7382 (1988).

[65] Gustavo E. Scuseria, Andrew C. Scheiner, Timothy J. Lee, Julia E. Rice and Henry F. Schaefer III. Journal of Chemical Physics 86, 2881 (1987).

[66] P. Pulay. Journal of Computational Chemistry, Vol 3. No 4, 556-560 (1982).

[67] P. Pulay. Chemical Physics Letters. Volume 73, number 2. 15 July 1980.

[68] The Mathematics of DIIS.
http://vergil.chemistry.gatech.edu/notes/diis/node2.html

[69] We want to acknowledge the help received from the Department for Research Computing at USIT, the University of Oslo IT-department.

[70] Arteum D. Bochevarov and C. David Sherrill. The Journal of Chemical Physics, 122, 234110, (2005).

[71] Mauro Del Ben, Jürg Hutter, and Joost VandeVondele. Journal of Chemical Theory and Computation, 8(11):4177-4188,

[72] Joost VandeVondele, Matthias Krack, Fawzi Mohamed, Michele Parrinello, Thomas Chassaing, Jürg Hutter. Computer Physics Communications 167 (2005) 103-128

[73] Martin Schutz and Frederick R. Manby. Phys. Chem. Chem. Phys., 2003, 5, 3349-3358

[74] Kraken Cray Supercomputer.
http://www.nics.tennessee.edu/computing-resources/kraken

[75] Cannon Algorithm Explained, youtube lecture.
http://www.youtube.com/watch?v=sB-Dh4DsOy0

[76] The Journal of Chemical Physics 93, 6104 (1990)

[77] Josep M. Bofill and Peter Pulay. J. Chem. Phys. 90, 3637 (1989).

[78] Christoph Riplinger and Frank Neese. The Journal of Chemical Physics 138, 034106 (2013)

[79] Journal of Chemical Physics; Sep2013, Vol. 139 Issue 9, p094105.

[80] Point Group Symmetry Tutorial.
`http://symmetry.otterbein.edu/tutorial/pointgroups.html`

[81] Igor Ying Zhang, Yi Luo and Xin Xu. The Journal of Chemical Physics, 133, 104105, (2010).

[82] Divide and Conquer PDF From Berkeley University.
Lester Mackey, Ameet Talwalkar, Michael I. Jordan.
`http://www.cs.berkeley.edu/~ameet/dfc.pdf`

[83] `http://www.uio.no/studier/emner/matnat/fys/FYS-MENA4111/`

[84] Alessandro Roggero, Abhishek Muherjee and Francesco Pederiva. Phys Rev B 88, 115138 (2013).

[85] Wei-Cheng Tung1, Michele Pavanello and Ludwik Adamowicz. J. Chem. Phys 134, 06117 (2011)

[86] Henrik Koch et al. J. Chem. Phys. 106 (5), 1 February 1997

[87] Krishnan Raghavachari, Gary W. Trucks, John A. Pople and Martin Head-Gordon Chem. Phys. Lett. 157, 479. 1989.

[88] Quantum Monte-Carlo Studies of Generalized Many-Body Systems.
MA Thesis.
Jorgen Hoberget.
June 2013.

[89] Electrical Engineering and Computer sciences.
University of California at Berkeley.
August 2, 2014.
Technical Report No. UCB/EECS-2014-143.
`http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-143.pdf`

[90] `http://vergil.chemistry.gatech.edu/notes/hf-intro/node4.html`

[91] `http://www.phys.sinica.edu.tw/TIGP-NANO/Course/2011_Spring/classnotes/CMS_20110511.pdf`

[92] Code developed during master thesis:
https://github.com/otnorli/CCSD_PARALLEL

[93] F. Coester. Nucl. Phys. 7, 421 (1958).

[94] F. Coester and H. Kümmel, Nucl. Phys. 17, 477 (1960).

[95] Paldus, J. (2005). "The beginnings of coupled-cluster theory: an eyewitness account". In Dykstra, C. Theory and Applications of Computational Chemistry: The First Forty Years. Elsivier B.V. p. 115.

[96] http://www.mathworks.se/programs/nrd/matlab-trial-request.html?ref=ggl&s_eid=ppc_6415