FYS4411
Variational Monte Carlo simulations of atoms and molecules

Stian Roti Svendby

June 14, 2015

Contents

1 Abstract

We have numerically estimated the ground state energy of helium, beryllium and neon, as well as the simple diatomic molecules H_2 and Be_2, using different wave functions, and studied the structures of the systems by plotting the onebody density. This has been done with variational Monte Carlo methods using both brute force Metropolis algorithm and Metropolis-Hasting algorithm. To optimize the speed of the code, we have implemented efficient general functions to compute the Slater determinant and the Jastrow factor for different systems, with a lot of speed gain compared to a brute force numerical solver. We have also parallelized the code to gain a speedup in the computations.

A correlation factor (Jastrow factor) in our trial wave function was necessary to get an estimate close to the real ground state energy. Including importance sampling (quantum force) also reduced the variance in the computations, and we got a relative error of the ground state in the range 0.5 % to 2 %. This is an error due to lack of precision in the variational parameters α and β and not fully optimal trial wave functions. We used *Steepest decent method* to search for optimal variational parameters, but the algorithm was unstable, and there was some need for tuning of the program when switching between systems.

2 Introduction

Atoms and molecules are the building blocks in everything, and scientists have studies these small guys for more than two hundred years [19]. Earlier they had to do varius experiments at the lab to investigate and discover the structure of different atoms and molecules. Over the last decade, the CPU speed has done an enormous jump, making us able to perform heavy computer simulations on complex systems, even with cheap computers. With a little quantum mechanics at hand, we should be able to exchange the labratory experiments with computer simulations to investigate atoms and molecules. Of course, we should always attack problems from several angels to compare and support results, so numerical experiments is not a replacement for labratory experiments. We will actually compare our numerical results with values obtained at labratory experiments to verify results. The advange of using a computer is that we can easily play around with different quantities when our program is capable of reproduce known results, making it possible to investigate properties that can be hard to work with at the lab.

Usually, in classical mechanics, we analyze forces acting on and between objects in our system, before we develop and solve systems of differential equations to estimate positions at a later time. This is doable for an easy system like the hydrogen atom, but when the system consist of more particles in three dimensions, the system of equations grows fastly, and the CPU expences become too large to solve the system at all with classical mechanics. The solution is quantum mechanics and a statistical approach.

Variatinal Monte Carlo (VMC or just MC) methods are based on stochastic processes, where we have probabilities of rejection or acception of a transition. For small systems at atomic level, MC methods are well suited for numerical experiments. We will perform MC simulations using *Metropolis algorithm* with wave function ratios to calculate transition probabilities. In our case a transition will be a movement of a single electron, and we opt for the ground state of the system, meaning that we are searching for the system configuration that minimizes the energy in the system. In this project we will investigate the atoms *helium* (He), *beryllium* (Be) and *neon* (Ne), but also the simple diatomic molecules H_2 and Be_2. It's important to mention that all systems have filled up orbitals, making life a little easier in this project. We will look at the ground state energy, mean electron-electron distance and study charge and onebody density, but also look at different wave functions and *hamilton operators* from quantum mechanics.

In this report there is a lot of theory in the method section who explain concepts from quantum mechanics, how algorithms have been implemented in our program, but also a lot of ways to optimize the code to earn a great speed up. After the method section we will present results in tables and figures. After the conclusion with main findings and thought you will find a lot of code examples on how different algoritms is implemented in our code.

3 Theory and methods

3.1 Quantum mechanic eigenvalue equation

In order to investigate systems at atomic level, we have to use quantum mechanics. Since we in this project mainly are interested in finding the ground state of different atoms and molecules, we use the *time−independent Schrödingere quation* (SE) to compute the expectation value of the energy, given as [1]:

$$\hat{\boldsymbol{H}}\psi = E\psi$$

Here $\hat{\boldsymbol{H}}$ is the *hamiltonian* operator for the energy of the system, E is the energy we solve the equantion for, and ψ (or trial function ψ_T) is a choosen time-independent wave function. This equation states that the expectation value of E is given as [1]:

$$\langle E\rangle = \langle\hat{\boldsymbol{H}}\rangle = \frac{\int \psi_T^*\hat{\boldsymbol{H}}\psi_T d\boldsymbol{r}}{\int \psi_T^*\psi_T d\boldsymbol{r}}$$

We have that the ground state energy E_0 is a lower bound:

$$E_0 \leq \langle\hat{\boldsymbol{H}}\rangle$$

When searching for the ground state, we have to minimize the energy to get as close as possible to E_0. In order to do this, we need to find trial wave functions that describes the system well. In the next subsections we will look at different wave functions and hamiltonian operators for different atoms and molecules.

3.2 Hydrogenic wave functions

In order to solve SE given above, we have to find wave functions that are able to decribe the system we want to investigate. The hydrogen atom is the only system where we have an exact analytical solution [3]. That means that the calculation of $\langle E\rangle$ will give the exact energy with zero variance [1]. The hydrogenic wave function is given by:

$$\psi_T(\boldsymbol{r}_1, \alpha) = e^{-\alpha r_1}$$

Here $r_1 = \sqrt{x_1^2 + y_1^2 + z_1^2}$ is the radial distance between the nucleus and the surrounding electron, and α is a *variational parameter*. With $\alpha = 1$, the wave function reproduce the exact solution. With this in mind, we construct a (trial) wave function for helium (He) with two sorrounding electrons as the product of two hydrogenic wave functions:

$$\psi_T(\boldsymbol{r}_1, \boldsymbol{r}_2, \alpha) = \psi_1(\boldsymbol{r}_1, \alpha)\psi_2(\boldsymbol{r}_2, \alpha) = e^{-\alpha(r_1+r_2)}$$

Here er have two electron distances to the nucleus, and α has to determine in a way that minimizes the energy. We see that our trial wave function in construced with two single-particle wave functions $\psi_i(\boldsymbol{r}_i, \alpha)$. Unlike the hydrogen atom, He will also have interaction between electrons. This should deviate from the pure hydrogenic wave function, and may reduce the precision in our calculations. To deal with this problem, we can include another factor to our trial wave function known as *Jastrow factor*. The Jastrow factor is also known as a *correlation factor*. This factor includes the electron-electron distance, $r_{12} = |\boldsymbol{r}_1 + \boldsymbol{r}_2|$, making us able include the effect of electron repulsation. This, together with another variational parameter β (that we also have to determine), hopefully gives us a better trial wave function given as:

$$\psi_T(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_{12}, \alpha, \beta) = e^{-\alpha(r_1+r_2)}e^{\frac{r_{12}}{2(1+\beta r)}}$$

If we now go back to wave functions without Jastrow factor, we have to extend the theory a bit. Since electrons are fermions, they cannot be in the same state with equal quantum numbers at the same time due to *Paulis exclusion priciple* [3], which state that the total wave function has to be anti-symmetric. For He we have that the total wave function has to fulfill:

$$\psi(r_1, r_2) = \psi_a(\boldsymbol{r}_1)\psi_b(\boldsymbol{r}_2) - \psi_a(\boldsymbol{r}_2)\psi_b(\boldsymbol{r}_1)$$

We assume that one particle is in state ψ_a and that the other particle is in state ψ_b. This implies that if $\psi_a = \psi_b$ the total wave function will vanish, and hence they cannot be in the same state with equal spin. For He with two electrons we will in the grond state have one electron with spin up and the other electron with spin down. This gives different quantum numbers, so both electrons can stay in the first orbital given by the wave function above. When moving over to bigger atoms, there will not be space for more eletrons in the first orbital, so we will need more of them. The general formula for hydrogen orbitals can be found in Griffiths on page 152 [3], but we dont bother writing it down here (since it looks ugly), but rather state which orbitals we obtain with quantum numbers

n (principal quantum number), l (azimuthal quantum number) and m_l (magnetic quantum number). The general formula can in short form be written as:

$$\psi_{nlm_l}(r,\theta,\phi) = R_{nl}(r)Y_l^{m_l}(\theta,\phi)$$

For orbital $1s$ and $2s$ we have $Y_0^0(\theta,\phi) = 1$, and hence obtain:

$$\psi_{1s} = \psi_{100}(r,\phi,\theta) = e^{-\alpha r}$$

$$\psi_{2s} = \psi_{200}(r,\phi,\theta) = \big(1 - \frac{\alpha}{2}r\big)e^{-\frac{\alpha r}{2}}$$

When moving up to orbital $2p$ we also have to pay attention to angels, so $Y_l^m(\theta,\phi) \neq 1$. The problem with spherical harmonics is that they are complex. To get around this problem we introduce *real solid harmonics*, which is a mapping from spherical coordinates to cartesian coordinates only giving real values. The real solid harmonics are given in the tabel below [1]:

$m_l \backslash l$	0	1	2	3
+3				$\frac{1}{2}\sqrt{\frac{5}{2}}(x^2-3y^2)x$
+2			$\frac{1}{2}\sqrt{3}(x^2-y^2)$	$\frac{1}{2}\sqrt{15}(x^2-y^2)z$
+1		x	$\sqrt{3}xy$	$\frac{1}{2}\sqrt{\frac{3}{2}}(5z^2-r^2)x$
0		y	$\frac{1}{2}(3z^2-r^2)$	$\frac{1}{2}(5z^2-3r^2)x$
-1		z	$\sqrt{3}yz$	$\frac{1}{2}\sqrt{\frac{3}{2}}(5z^2-r^2)y$
-2			$\sqrt{3}xz$	$\sqrt{15}xyz$
-3				$\frac{1}{2}\sqrt{\frac{5}{2}}(3x^2-y^2)y$

Table 1: Real solid harmonics used in mapping from spherical coordinates to cartesian coordinates in $Y_l^{m_l}(\theta,\phi)$ [1].

Using Table 1 we can now construct the next orbitals as:

$$\psi_{2px} = \psi_{211}(r,\phi,\theta) = xe^{-\frac{\alpha r}{2}}$$

$$\psi_{2py} = \psi_{210}(r,\phi,\theta) = ye^{-\frac{\alpha r}{2}}$$

$$\psi_{2pz} = \psi_{21-1}(r,\phi,\theta) = ze^{-\frac{\alpha r}{2}}$$

Here we recognize $Y_l^{m_l}(\theta,\phi)$ as x, y and z after the mapping with solid harmonics found in Table 1. The mapping will work (not alter the energies) as long an external magnetic field not is applied to the system [1], and we are not going to apply any kind of magnetic field on our system. Then we have five different orbitals that can hold two fermion each, meaning that we have enought orbitals for doing simulations on the neon atom, which is the biggest atom we are going to look at. For heavier atoms, we have to add more orbitals.

With this in hand, we are later going to use trial wave functions both with and without Jastrow factor to compare them up to benchmarks for both He, Be and Ne, but also on molecules H_2 and Be_2.

3.3 The helium atom and general atom hamiltonian

The dimensionless hamiltonian for He in three dimensions is given by:

$$\hat{\boldsymbol{H}} = -\frac{\nabla_1^2}{2} - \frac{\nabla_2^2}{2} - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}$$

This can be used in SE to compute the expectation value of the energy, and contain all information about the system (the helium atom). The two first terms $(-\frac{\nabla_1^2}{2} - \frac{\nabla_2^2}{2})$ computes the kinetic energy from the two electrons moving around the nucleus, the two next terms $(-\frac{2}{r_1} - \frac{2}{r_2})$ the potential energy due to the electron attraction from the nucleus, and the last term $(\frac{1}{r_{12}})$ the electron-electron potential due to repulsion between them. $\nabla^2 = \Delta$ is the

double derivative in three spatial dimensions, also known as *Laplacian operator*. We are going to use the same kind of hamiltonian for Be and Ne, given on a general form as:

$$\hat{\boldsymbol{H}} = \sum_{i=1}^{Z}(-\frac{\nabla_i^2}{2} - \frac{Zq}{r_i}) + \sum_{i=1}^{Z}\sum_{i<j}\frac{q^2}{r_{ij}}$$

We notice that we have dimensionless *Coulomb potential* [18] given as $V_c = \frac{q_1 q_2}{r}$, so with electron charge $q = 1$ and $Z = nucleous\, charge$ we get:

$$\hat{\boldsymbol{H}} = \sum_{i=1}^{Z}(-\frac{\nabla_i^2}{2} - \frac{Z}{r_i}) + \sum_{i=1}^{Z}\sum_{i<j}\frac{1}{r_{ij}}$$

This is the general hamiltonian we use for atoms in this project.

3.4 Slater determinants

If we study larger atoms such as beryllium, we have to introduce a *Slater determinant*, since we cannot have more than two eletrons in the lowest orbital. In our project we use wave functions for Be on the form:

$$\psi_T(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{r_3}, \boldsymbol{r_4}, \alpha, \beta) = Det(\psi_1(\boldsymbol{r}_1, \alpha)\psi_2(\boldsymbol{r}_2, \alpha)\psi_3(\boldsymbol{r}_3\alpha)\psi_4(\boldsymbol{r}_4, \alpha))\prod_{i<j}^{4} e^{\frac{a_{ij}r_{ij}}{1+\beta r_{ij}}}$$

where the first part is the Slater determinant, while the last part is the Jastrow factor. We will first look at the Slater determinant part, and to make it cleaner below, we set $\psi_i(\boldsymbol{r}_i, \alpha) = \psi_i(\boldsymbol{r}_i)$. The Slater determinant is given by:

$$Det(\psi_1(\boldsymbol{r}_1)\psi_2(\boldsymbol{r}_2)\psi_3(\boldsymbol{r}_3)\psi_4(\boldsymbol{r}_4)) = \frac{1}{\sqrt{4}}\begin{vmatrix} \psi_1(\boldsymbol{r}_1) & \psi_2(\boldsymbol{r}_1) & \psi_3(\boldsymbol{r}_1) & \psi_4(\boldsymbol{r}_1) \\ \psi_1(\boldsymbol{r}_2) & \psi_2(\boldsymbol{r}_2) & \psi_3(\boldsymbol{r}_2) & \psi_4(\boldsymbol{r}_2) \\ \psi_1(\boldsymbol{r}_3) & \psi_2(\boldsymbol{r}_3) & \psi_3(\boldsymbol{r}_2) & \psi_4(\boldsymbol{r}_3) \\ \psi_1(\boldsymbol{r}_4) & \psi_2(\boldsymbol{r}_4) & \psi_3(\boldsymbol{r}_4) & \psi_4(\boldsymbol{r}_4) \end{vmatrix}$$

For ψ_i we use the hydrogen orbitals:

$$\psi_{1s}(\boldsymbol{r}_i) = e^{-\alpha r_i}$$

$$\psi_{2s}(\boldsymbol{r}_i) = (1 - \frac{\alpha}{2}r_i)e^{-\frac{\alpha}{2}r_i}$$

This determinant is actually zero ([1], page 52) since the spin up and spin down states are equal. But we can rewite the determinant. If we define

$$det\uparrow(1,2) = \frac{1}{\sqrt{2}}\begin{vmatrix} \psi_1(\boldsymbol{r}_1) & \psi_2(\boldsymbol{r}_1) \\ \psi_1(\boldsymbol{r}_2) & \psi_2(\boldsymbol{r}_2) \end{vmatrix}$$

$$det\downarrow(3,4) = \frac{1}{\sqrt{2}}\begin{vmatrix} \psi_3(\boldsymbol{r}_2) & \psi_4(\boldsymbol{r}_3) \\ \psi_3(\boldsymbol{r}_4) & \psi_4(\boldsymbol{r}_4) \end{vmatrix}$$

we can rewrite $Det(\psi_1(\boldsymbol{r}_1)\psi_2(\boldsymbol{r}_2)\psi_3(\boldsymbol{r}_3)\psi_4(\boldsymbol{r}_4))$ (using elementary linear algebra) as:

$$\begin{aligned} Det(\psi_1(\boldsymbol{r}_1)\psi_2(\boldsymbol{r}_2)\psi_3(\boldsymbol{r}_3)\psi_4(\boldsymbol{r}_4)) = {} & det\uparrow(1,2)det\downarrow(3,4) - det\uparrow(1,3)det\downarrow(2,4) \\ & -det\uparrow(1,4)det\downarrow(3,2) - det\uparrow(2,3)det\downarrow(1,4) \\ & -det\uparrow(2,4)det\downarrow(1,3) + det\uparrow(3,4)det\downarrow(1,2) \end{aligned}$$

The determinant is still zero! From Moskowitz and Kalos [13] we have that the slater determinant can be approximated by:

$$Det(\psi_1(\boldsymbol{r}_1)\psi_2(\boldsymbol{r}_2)\psi_3(\boldsymbol{r}_3)\psi_4(\boldsymbol{r}_4)) \propto \frac{1}{\sqrt{4}}\begin{vmatrix} \psi_{1s}(\boldsymbol{r}_1) & \psi_{1s}(\boldsymbol{r}_2) \\ \psi_{2s}(\boldsymbol{r}_1) & \psi_{2s}(\boldsymbol{r}_2) \end{vmatrix}\begin{vmatrix} \psi_{1s}(\boldsymbol{r}_3) & \psi_{1s}(\boldsymbol{r}_4) \\ \psi_{2s}(\boldsymbol{r}_3) & \psi_{2s}(\boldsymbol{r}_4) \end{vmatrix}$$

This gives a nonzero value. The first part is a matrix with spin up, and the last part is a matrix with spin down, and $\frac{1}{\sqrt{4}}$ is just a normalization factor for beryllium. This can be implemented with the code below, with *argument* as a vector with the four instantaneous radial distances between the nucleus and the electrons:

```
double VMCSolver::SlaterDeterminant(){
    vec argument = zeros(nParticles);
  argument = r_centre;
    double wf = (psi1s(argument(0))*psi2s(argument(1))-psi1s(argument(1))*psi2s(argument(0)))*(
        psi1s(argument(2))*psi2s(argument(3))-psi1s(argument(3))*psi2s(argument(2)));
    return wf;
}
```

We note that we have skipped the factor $\frac{1}{\sqrt{4}}$ since it vanish in Metropolis ratio in our algorithm. Generally we can split up the slater determinant as:

$$Det(\psi_1(\boldsymbol{r}_1), \psi_2(\boldsymbol{r}_2), ..., \psi_N(\boldsymbol{r}_N)) \propto det\uparrow det\downarrow$$

We should notice that this ansatz is not anti-symmetric under the exchange of electrons with oppsite spin, but it still gives the same expectation value as the full Slater determinant, as long our hamiltonian is spin independent (which it is) [1].

3.5 The Jastrow factor

We have earlier mentioned the Jastrow factor we introduced to include the electron-electron repulsation. The Jastrow factor can generally be computed by:

$$\psi_C = e^{\sum_{i<j} \frac{a_{ij} r_{ij}}{1+\beta r_{ij}}}$$

Since two fermions cannot be in the same state unless they have different spin, and we want to minimize the energy to obtain the ground state, we choose to give the first half of electrons spin *up* and the remaining half spin *down*. a_{ij} is a matrix containing information about spins of the electrons. We have that $a_{ij} = \frac{1}{4}$ if equal spin and $a_{ij} = \frac{1}{2}$ if opposite spin [3]. In our program a_{ij} is implemented as a matrix holding all combinations of eletron spins. If we assume a_{ij} is computed and the distance between the electrons is known, code to compute the Jastrow factor is then given by:

```
double VMCSolver::JastrowFactor(){
    double Psi = 1.0;
    for(int j=0; j < nParticles; j++){
        for(int i=0; i < j; i++){
            Psi *= exp((a_matrix(i,j)*r_distance(i,j))/(1.0 + beta*r_distance(i,j)));
        }
    }
    return Psi;
}
```

3.6 Local energy and implementation

We have now introduced trial wave functions and the hamiltonian we are going to use to calculate the energy of the atom configurations. The choosen wave function is also our probability distribution:

$$P(\boldsymbol{R}, \alpha, \beta) = \frac{|\psi_T(\boldsymbol{R}, \alpha, \beta)|^2}{\int |\psi_T(\boldsymbol{R}, \alpha, \beta)|^2 d\boldsymbol{R}}$$

with $\boldsymbol{R} = (r_1, r_2, ..., r_Z, r_{1,2}, ...r_{z-1,z})$. From SE we introduce *local energy*, defined as:

$$E_L(\boldsymbol{R}_i, \alpha, \beta) = \frac{1}{\psi_T(\boldsymbol{R}, \alpha, \beta)} \hat{\boldsymbol{H}} \psi_T(\boldsymbol{R}, \alpha, \beta)$$

$$= -\frac{1}{2\psi_T(\boldsymbol{R}, \alpha, \beta)} \nabla_1^2 \psi_T(\boldsymbol{R}, \alpha, \beta) - \frac{1}{2\psi_T(\boldsymbol{R}, \alpha, \beta)} \nabla_2^2 \psi_T(\boldsymbol{R}, \alpha, \beta) - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}$$

With this in hand, we are able to estimate the expectation value of the energy for a given atom by:

$$\langle E \rangle = \int P(\boldsymbol{R}, \alpha, \beta) E_L(\boldsymbol{R}, \alpha, \beta) dR \approx \frac{1}{N} \sum_{i=1}^{N} P(\boldsymbol{R}_i, \alpha, \beta) E_L(\boldsymbol{R}_i, \alpha, \beta)$$

The estimate $\langle E \rangle$ is actually just a MC simulation to solve the integral. We try a random move in space for an electron, use Metropolis algorithm to accept or reject the move, before we have to calculate the new local energy (using the wave function) every time we try a new configuration of the atom. The new local energy is added to a sum for each MC cycle. To get our estimate for the ground state, we just divide the sum of local energies by the number of MC cycles N, or MC cycle times electrons depending on how the sum of local energies is constructed in the code. How this is done will be explained in detail later.

Since we have to compute the local energy every time we move a particle, this is also the most CPU critical part of the computations. We see that we have to carry out two double derivatives for He with two electrons. The laplacian can be solved bruce force numerically, but that is an inefficient way of solving it. If possible, as for He, we can instead carry out the analytical solution of the local energy, making us able to save a lot of CPU time for the specific atom. This is also a nice way to verify that the numerical machinery is working as expected. The estimated energy using a numerical solver should be very close the solution using an analytical solver.

The derivatives are easiest carried out by polar-coordinates by hand or by using *symbolic python* (sympy). The local energies of He for the to different wave functions can then be written:

$$E_{L1} = (\alpha - Z)\left(\frac{1}{r_1} + \frac{1}{r_2}\right) + \frac{1}{r_{12}} - \alpha^2$$

$$E_{L2} = E_{L1} + \frac{1}{2(1+\beta r_{12})^2}\left(\frac{\alpha(r_1+r_2)}{r_{12}}(1 - \frac{\boldsymbol{r}_1\boldsymbol{r}_2}{r_1 r_2}) - \frac{1}{2(1+\beta r_{12})^2} - \frac{2}{r_{12}} + \frac{2\beta}{1+\beta r_{12}}\right)$$

A simple code to find E_{L1} with sympy can be found in "Code implementation 6.1". For bigger atoms like Ne it will be impossible to carry out analytical expressions by hand, and the expressions produced by sympy code will be extremely long. We will later introduce optimized calculations to take care of this latter.

3.7 Cusp conditions

We need our wave function to satisfie the *cusp condition* when $r_1 \to 0$ or $r_2 \to 0$ or $r_{12} \to 0$. Since we have $\frac{1}{r_1}$, $\frac{1}{r_2}$ and $\frac{1}{r_{12}}$ from the potential energy part, we see that the energy tends to infinity if any of the denominators goes to zero, known as *singularities*. Therefore our trial wave function has to be constructed in such a way that singularities cancel, which means that the kinetic energy also have to tend to infinity in this limit. For the local energy we get:

$$E_L(\boldsymbol{R}) = \frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\hat{\boldsymbol{H}}\psi_T(\boldsymbol{R},\alpha,\beta) = \frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\left(-\frac{\nabla^2}{2} - \frac{Z}{r}\right)\psi_T(\boldsymbol{R},\alpha,\beta) + finite\,terms$$

We then let $r \to 0$, and this gives:

$$E_L(R) = \frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\left(-\frac{1}{2}\frac{d^2}{dr^2} - \frac{1}{r}\frac{d}{dr} - \frac{Z}{r}\right)\psi_T(\boldsymbol{R},\alpha,\beta) + finite\,terms$$

For small values of r, the dominating terms is

$$\lim_{r_1 \to 0} E_L(R) = \frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\left(-\frac{1}{r}\frac{d}{dr} - \frac{Z}{r}\right)\psi_T(\boldsymbol{R},\alpha,\beta)$$

since the second derivative not diverge at the origin. Therefore we get:

$$0 = \frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\left(-\frac{1}{r}\frac{d}{dr} - \frac{Z}{r}\right)\psi_T(\boldsymbol{R},\alpha,\beta)$$

$$\frac{1}{\psi_T(\boldsymbol{R},\alpha,\beta)}\frac{d\psi_T(\boldsymbol{R},\alpha,\beta)}{dr} = -Z$$

This imples that a trial wave function can be on the form

$$\psi_T(\boldsymbol{R},\alpha,\beta) \propto e^{-Zr}$$

, which our hydrogen orbitals is. In the case with $r_{12} \to 0$ we will get

$$\psi_T(\boldsymbol{R}, \alpha, \beta) \propto e^{r_{ij}}$$

, which the Jastrow factor fulfills.

3.8 Monte Carlo simulatons and The Metropolis algorithm

As mentioned we use Metropolis algorithm to accept or reject random suggested moves of electrons. A suggested move is given by [1]:

$$y = x + r\delta$$

, where x is the old position, r is a random number between -0.5 and 0.5, and δ is a choosen step length. In our program suggested moves is done by the code:

```
rNew(i,j) = rOld(i,j) + stepLength*(ran2(&idum) - 0.5);
```

, where *steplength* is a predefined value choosen to give an acceptance ratio about 0.5 in Metropolis algorithm. $ran2(\&idum)$ generates random numbers between 0 and 1. With the new configuration of the atom we use Metropolis algorithm [1]. The Metropolis algorithm is given by:

$$A(y, x) = min(1, q(y, x))$$

$$q(y, x) = \frac{|\psi_T(y)|^2}{|\psi_T(x)|^2}$$

The code to implement Metropolis algorithm is given by the following code:

```
if(ran2(&idum)<=(waveFunctionNew*waveFunctionNew)/(waveFunctionOld*waveFunctionOld)){
    for(int j = 0; j < nDimensions; j++){
        rOld(i,j) = rNew(i,j);
        waveFunctionOld = waveFunctionNew;
    }
}
else{
    for(int j = 0; j < nDimensions; j++){
        rNew(i,j) = rOld(i,j);
}
```

We see that if the new configuration gives lower energy, there will be a probability of 0.5 (with the correct step length) for accepted move. If the energy is higher in the new configuration than in the previous one, we reject the move.

We can also introduce *importance sampling*, or in this particular case apply a *quantum force*. That means that we apply a force that make the electrons tend to its minimum, so a new suggested move will almost always give a lower energy. A new step is suggested as:

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t}$$

, where $D = 0.5$ is a factor from the kinetic energy, $F(x)$ is the quantum force, $\Delta t \in [0.001, 0.01]$ (for stable values) is the timestep, and ξ is a is a gaussian probability distribution. The quantum force is given by [1]:

$$F = 2\frac{1}{\psi_T}\nabla\psi_T$$

and pushes the electrons towards regions where the wave function is large, and hence giving accepted moves. Suggested moves are in this case done by the code:

```
rNew(i,j) = rOld(i,j) + GaussianDeviate(&idum)*sqrt(timestep)+QForceOld(i,j)*timestep*D;
```

GaussianDeviate is the gaussian probability distribution, *timestep* controls the step length and the acceptance ratio of suggested steps. A low value of *timestep* $= \Delta t$ will give high acceptance ratio, but the steps will also become shorter, so we must have a balance (values in [0.001,0.01]) for stable result. $QForceOld$ is computed by F above.

Instead of the brute force Metropolis algorithm, we switch to $Metropolis-Hasting\,algorithm$ to take advantage of the quantum force, and the algorithm is given by:

$$A(y,x) = min(1, q(y,x))$$

$$q(y,x) = \frac{G(x,y,\Delta t)|\psi_T(y)|^2}{G(y,x,\Delta t)|\psi_T(x)|^2}$$

We see that we multiply with the $Greensfunction$ (G) on both sides of the fraction compared to the standard Metropolis algorithm. The Greenfunction is implemented in our program with the following code:

```
GreensFunction = 0.0;
for(int j=0; j < nDimensions; j++){
    GreensFunction += 0.5*(QForceOld(i,j)+QForceNew(i,j))*                    (D*timestep
        *0.5*(QForceOld(i,j)-QForceNew(i,j))-rNew(i,j)+rOld(i,j));          }
GreensFunction = exp(GreensFunction);
```

From the code we see that the Greenfunction make us able to include the quantum force in $q(y,x)$. After computing the Greenfunction, we can implement our new acceptance-rejection algorithm (Metropolis-Hasting algorithm):

```
if(ran2(&idum)<=GreensFunction*(waveFunctionNew*waveFunctionNew)/(waveFunctionOld*waveFunctionOld)
    ){
    for(int j = 0; j < nDimensions; j++){
        rOld(i,j) = rNew(i,j);
        QForceOld(i,j) = QForceNew(i,j);
        waveFunctionOld = waveFunctionNew;
    }
}
else{
    for(int j = 0; j < nDimensions; j++){
        rNew(i,j) = rOld(i,j);
        QForceNew(i,j) = QForceOld(i,j);
    }
}
```

We now have all the theory we need to compute the ground state for He, Be and Ne with and without Jastrow factor, and with and without importance sampling.

3.9 Energy, gradients and Laplacians for atoms

Previous we have seen that we have to compute both gradients and laplacians of the trial wave function. We saw that we needed the gradient in the quantum force:

$$F = 2\frac{1}{\psi_T}\nabla\psi_T$$

and the laplacian in the calculation og the expectation value of the energy:

$$\langle E\rangle = \sum_{i=1}^{Z}(-\frac{\nabla_i^2\psi_T}{2} - \frac{Z}{r_i}) + \sum_{i=1}^{Z}\sum_{j=1,j\neq i}^{Z}\frac{1}{r_{ij}}$$

In the first case $\nabla\psi_T$ leads to vectors in three space dimensions. In the second case with $\nabla^2\psi_T$ leads to a value. The gradients and laplacians for the five orbitals we have looked at earlier are given in Table 2:

	ψ	$\frac{\partial\psi}{\partial x}$	$\frac{\partial\psi}{\partial y}$	$\frac{\partial\psi}{\partial z}$	$\nabla^2\psi$
ψ_{1s}	$e^{-\alpha r}$	$-\frac{\alpha}{r}xe^{-\alpha r}$	$-\frac{\alpha}{r}ye^{-\alpha r}$	$-\frac{\alpha}{r}ze^{-\alpha r}$	$\frac{\alpha}{r}(\alpha r-2)e^{-\alpha r}$
ψ_{2s}	$\left(1-\frac{\alpha}{2}r\right)e^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}x(\alpha r-4)e^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}y(\alpha r-4)e^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}z(\alpha r-4)e^{-\frac{\alpha r}{2}}$	$-\frac{1}{8}\frac{\alpha}{r}((\alpha r)^2-10\alpha r+16)e^{-\frac{\alpha r}{2}}$
ψ_{2px}	$xe^{-\frac{\alpha r}{2}}$	$-\frac{1}{r}(\frac{1}{2}\alpha x^2-r)e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}xye^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}xze^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}x(\alpha r-8)e^{-\frac{\alpha r}{2}}$
ψ_{2py}	$ye^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}xye^{-\frac{\alpha r}{2}}$	$-\frac{1}{r}(\frac{1}{2}\alpha y^2-r)e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}yze^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}y(\alpha r-8)e^{-\frac{\alpha r}{2}}$
ψ_{2pz}	$ze^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}xze^{-\frac{\alpha r}{2}}$	$-\frac{1}{2}\frac{\alpha}{r}yze^{-\frac{\alpha r}{2}}$	$-\frac{1}{r}(\frac{1}{2}\alpha z^2-r)e^{-\frac{\alpha r}{2}}$	$\frac{1}{4}\frac{\alpha}{r}z(\alpha r-8)e^{-\frac{\alpha r}{2}}$

Table 2: Gradients and laplacians for the five first hydrogenic orbitals.

3.10 Optimization of the machinery

In our calculations of the local energy (kinetic part), we have to calculate the Laplacian of the wave function every time we move an electron. If quantum force is applied, we also have to compute the gradient of the wave function. This means that we have to differentiate the Slater determinant and the Jastrow factor with respect to all spatial coordinates and particles for every move of an electron, which usually will be the most time-consuming part of our code. To compute the determinant of a $N \times N$ matrix we would brute force use Gaussian elimination of order $O(N^3)$. In our case we have $N = number\,of\,particles \times d = dimensions$ independent coordinates, which means that we have to evaluate the entire Slater determinant $N \times d$ times to calulate the local energy and $N \times d$ times to calulate the quantum force. This leads to an order of $O(d \times N^4)$, that is very much computation for larger atoms like Neon. What we would like to do, is to find an efficient way of computing the gradient and Laplacian of the Slater determinant, as well as the ratio used in metropolis algorithm. This would speed up our program a lot for Be and Ne.

We start out with naming our Slater determinant matrix "D". A matrix element in D is defined as

$$d_{ij} = \psi_j(\boldsymbol{r}_i)$$

, with $\psi_j(\boldsymbol{r}_i)$ as a single particle wave function. The important thing we have to realize, is that when differentiating the Slater determinant with respect to a given electron (moving only one electron at the time), only one of the the rows in the corresponding Slater determinant is changed, making it unnecessary to recompute the whole determinant. The solutions turns out to be using the inverse of the Slater matrix with elements defined as [2]:

$$d_{i,j}^{-1} = \frac{C_{ji}}{|D|}$$

with C_{ji} as the cofactor matrix. This matrix will vanish in our calculations, and will not cause any troubles.

First we want an efficient expression to evaluate the ratio in the metropolis algorithm. If we run the program without the Jastrow factor in the trial wave function, the ratio is given by [2]:

$$R_{SD} = \frac{|D(\boldsymbol{r}^{new})|}{|D(\boldsymbol{r}^{old})|} = \frac{\sum_{j=1}^{N} d_{ij}(\boldsymbol{r}^{new})C_{ij}(\boldsymbol{r}^{new})}{\sum_{j=1}^{N} d_{ij}(\boldsymbol{r}^{old})C_{ij}(\boldsymbol{r}^{old})}$$

We have that $C_{ij}(\boldsymbol{r}^{new}) = C_{ij}(\boldsymbol{r}^{old})$, $D_{i,j}D_{ji}^{-1} = \delta_{ij} = 1$, and using the expression we have for $d_{i,j}^{-1}$, we end up with

$$R_{SD} = \frac{\sum_{j=1}^{N} d_{ij}(\boldsymbol{r}^{new})d_{ji}^{-1}(\boldsymbol{r}^{old})}{\sum_{j=1}^{N} d_{ij}(\boldsymbol{r}^{old})d_{ji}^{-1}(\boldsymbol{r}^{old})} = \sum_{j=1}^{N} d_{ij}(\boldsymbol{r}^{new})d_{ji}^{-1}(\boldsymbol{r}^{old})$$

or

$$\boxed{R_{SD} = \sum_{j=1}^{N} \phi_j(\boldsymbol{r}_i^{new})d_{ji}^{-1}(\boldsymbol{r}^{old})}$$

This is the ratio we want to use in Metropolis algorithm with order $O(N)$. Implementation can be found in "Code implementation 6.2".

Then we want efficient expressions for

$$\frac{\nabla\psi_T}{\psi_T}$$

$$\frac{\nabla^2\psi_T}{\psi_T}$$

used in the calculation of quantum force and local energy. We have when moving only one electron at a time that [2]:

$$\frac{\nabla_i |D(\boldsymbol{r})|}{|D(\boldsymbol{r})|} = \sum_{j=1}^{N} \nabla_i d_{ij}(\boldsymbol{r})d_{ji}^{-1}(\boldsymbol{r}) = \sum_{j=1}^{N} \nabla_i \phi_j(\boldsymbol{r}_i)d_{ji}^{-1}(\boldsymbol{r})$$

$$\frac{\nabla_i^2 |D(\boldsymbol{r})|}{|D(\boldsymbol{r})|} = \sum_{j=1}^{N} \nabla_i^2 d_{ij}(\boldsymbol{r})d_{ji}^{-1}(\boldsymbol{r}) = \sum_{j=1}^{N} \nabla_i^2 \phi_j(\boldsymbol{r}_i)d_{ji}^{-1}(\boldsymbol{r})$$

Implementation can be found in "Code implementation 6.3".

The update of the inverse matrix are done by

$$d_{kj}^{-1}(\boldsymbol{r}^{new}) = \begin{cases} d_{kj}^{-1}(\boldsymbol{r}^{old}) - \frac{d_{ki}^{-1}(\boldsymbol{r}^{old})}{R_{SD}} \sum_{l=1}^{N} d_{il}(\boldsymbol{r}^{new}) d_{lj}^{-1}(\boldsymbol{r}^{old}) & if\, j \neq i \\ \frac{d_{ki}^{-1}(\boldsymbol{r}^{old})}{R_{SD}} \sum_{l=1}^{N} d_{il}(\boldsymbol{r}^{old}) d_{lj}^{-1}(\boldsymbol{r}^{old}) & if\, j = i \end{cases}$$

With the property $D_{i,j}D_{ji}^{-1} = 1$, this can simpler be written:

$$d_{kj}^{-1}(\boldsymbol{r}^{new}) = \begin{cases} d_{kj}^{-1}(\boldsymbol{r}^{old}) - \frac{d_{ki}^{-1}(\boldsymbol{r}^{old})}{R_{SD}} \sum_{l=1}^{N} d_{il}(\boldsymbol{r}^{new}) d_{lj}^{-1}(\boldsymbol{r}^{old}) & if\, j \neq i \\ \frac{d_{ki}^{-1}(\boldsymbol{r}^{old})}{R_{SD}} & if\, j = i \end{cases}$$

Implementation can be found in "Code implementation 6.4"

With the updating algorithm we only need to invert the Slater matrix once, using LU decomposition ($O(N^3)$), or just using the $inv()$ function in *Armadillo* [24]. A single derivative is of order $O(N)$ and we have to take $d \times N$ derivatives, which leads to an order of $O(d \times N^2)$. We also have $O(N^2)$ for updating the inverse matrix. This means that we get a far better scaling than the brute force solver of order $O(d \times N^4)$.

We also want to split up the Slater determinant and write it as a product of a spin up part and a spin down part, as mentioned earlier. In our case we give the first half of electrons spin up, and the second half spin down. The important thing is that when moving one electron at a time, we only need to update one of the matrizes at each move. As a reminder the Slater determinant can then be written as

$$Det(\psi_1(\boldsymbol{r}_1), \psi_2(\boldsymbol{r}_2), ..., \psi_N(\boldsymbol{r}_N)) \propto det \uparrow det \downarrow$$

, and is correct as long as the hamiltonian is independent of spin.

For the correlation part, or the Jastrow factor we have (as mentioned above):

$$\psi_C = e^{\sum_{i<j} \frac{a_{ij} r_{ij}}{1+\beta r_{ij}}}$$

When the Jastrow factor is included in our trial wave function we get $R = R_{SD}R_C$ as the ratio used in metropolis algorithm. Therefore we also need an effcient way of calculation R_C. This is given by [2]:

$$R_C = \frac{\psi_C^{new}}{\psi_C^{new}} = \frac{e^{U_{new}}}{e^{U_{old}}} = e^{\Delta U}$$

, with

$$\Delta U = \sum_{i=1}^{k-1} (f_{ik}^{new} - f_{ik}^{old}) + \sum_{i=k+1}^{N} (f_{ki}^{new} - f_{ki}^{old})$$

, with

$$f_{ki} = \frac{a_{ki} r_{ki}}{1 + \beta r_{ki}}$$

Implementation of R_C can be found in "Code implementation 6.5". When R_{SD} and R_C is computed after a move, we have $R = R_{SD}R_C$, and the rule for acceptance of a move is given by the expression:

$$if(rand\,num <= GreensFunction * R^2)$$

When the correlation functions only depends on the relative distance between particles, as in our case, we can use linear $Padé - Jastrow$ form [1] to compute the gradient needed for the quantum force, and also local energy. The gradient becomes:

$$\frac{1}{\psi_{PJ}} \frac{\partial \psi_{PJ}}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\boldsymbol{r}_{ik}}{r_{ik}} \frac{\partial f_{ik}}{\partial r_{ik}} - \sum_{i=k+1}^{N} \frac{\boldsymbol{r}_{ki}}{r_{ki}} \frac{\partial f_{ki}}{\partial r_{ki}}$$

The Laplacian needed for the local energy is given by

$$\frac{\nabla_k^2 \psi_{PJ}}{\psi_{PJ}} = \left(\frac{\nabla_k \psi_{PJ}}{\psi_{PJ}} \right)^2 + \sum_{i=1}^{k-1} \left[\left(\frac{d-1}{r_{ik}} \right) \frac{\partial f_{ik}}{\partial r_{ik}} + \frac{\partial^2 f_{ik}}{\partial^2 r_{ik}} \right] - \sum_{i=k+1}^{N} \left[\left(\frac{d-1}{r_{ki}} \right) \frac{\partial f_{ki}}{\partial r_{ki}} + \frac{\partial^2 f_{ki}}{\partial^2 r_{ki}} \right]$$

, with

$$\frac{\partial f_{ik}}{\partial r_{ik}} = \frac{a_{ik}}{(1 + \beta r_{ik})^2}$$

$$\frac{\partial^2 f_{ik}}{\partial^2 r_{ik}} = -\frac{2a_{ik}\beta}{(1+\beta r_{ik})^3}$$

, where we have to sum over all particles when computing the local energy. The first term (we name it *GradientSquared*) can be computed by summing up squares of $\frac{1}{\Psi_{PJ}}\frac{\partial \Psi_{PJ}}{\partial x_k}$ and can effectively be computed in the same function as quantum force. How this is done is shown in "Code implementation 6.6"

In "Code implementation 6.6", we notice the calculation of *JastrowGradient*. This gradient has to be multiplied with the *SlaterGradient* as an energy cross-term. All calculations for the energy related to the including of Jastrow factor is found in "Code implementation 6.7".

The total kinetic energy with Jastrow activated is now given by:

$$-\frac{1}{2}\frac{\nabla^2\psi_T}{\psi_T} = -\frac{1}{2}\left(\frac{\nabla^2|D|_\uparrow}{|D|_\uparrow} + \frac{\nabla^2|D|_\downarrow}{|D|_\downarrow} + \frac{\nabla^2\psi_{PJ}}{\psi_{PJ}} + 2\left[\frac{\nabla|D|_\uparrow}{|D|_\uparrow} + \frac{\nabla|D|_\downarrow}{|D|_\downarrow}\right]\cdot\frac{\nabla\psi_{PJ}}{\psi_{PJ}}\right)$$

The total energy is given by this expression for the kinetic energy, pluss the potential energy due to interaction electron-electron and electron-nucleus interaction.

3.11 Variational parameters

We have mentioned the variational parameters α and β, and told that we have to make an optimal choise to minimize the energy as much as possible. But how is it done? The brute force way is to run over a lot of different values for variational parameters, and see what gives the lowest energy, or alternatively the lowest variance [1]. In the case with two parameters, we then get a mesh of values where we have to pick the point that gives the lowest energy. If we run over a mesh for He, we get the result given by Figure 1:

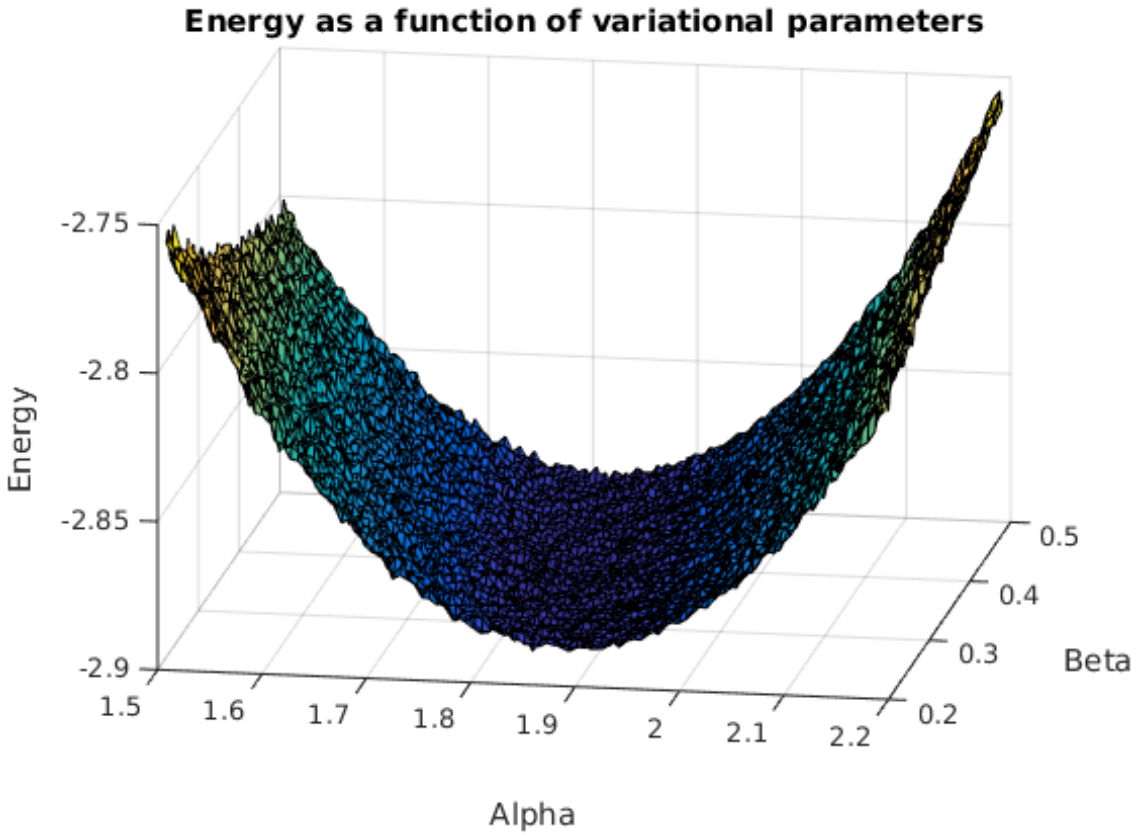


Figure 1: We can see which pair of α and β that minimizes the energy for the Helium atom with Jatrow factor.

As we can see from the plot, the variation in α have a lot more influence on the energy than β. But we have a problem. Since the etimated ground state energy change for each MC simulation even for the same parameters, it is hard to make a proper choise of parameters. We also have to chose between discrete point on the mesh, and making the mesh finer cost very much CPU. In our case it works fair enought to make plot like Figure 1 to select

variational parameters, but we will consider a better solution to this pick out the optimal parameters with in the next subsection.

3.12 Derivatives of the energy and Steepest descent

We want an algorithm that can find the variational parameter that minimize the energy. In our case we consider the variational parameter β in the Jastrow part of our trial wave function. Before we present the simple algorithm, we need something to minimize. In this case we want to minimize the beta-derivative of the computed energy. The formula for this is [8]:

$$\frac{\partial E}{\partial \beta} = 2\left[\left\langle E_L \frac{\partial ln\psi_T}{\partial \beta}\right\rangle - \langle E_L\rangle\left\langle \frac{\partial ln\psi_T}{\partial \beta}\right\rangle\right]$$

We have $ln\psi_T = ln\psi_{SD\uparrow} + ln\psi_{SD\downarrow} + ln\psi_C$, which implies $\frac{\partial ln\psi_T}{\partial \beta} = \frac{\partial ln\psi_{SD\uparrow}}{\partial \beta} + \frac{\partial ln\psi_{SD\downarrow}}{\partial \beta} + \frac{\partial ln\psi_C}{\partial \beta}$. Since the SD part of our trial wave function is independent of β, the two last terms vanish, and we are left with:

$$\frac{\partial ln\psi_T}{\partial \beta} = \frac{\partial ln\psi_C}{\partial \beta} = \frac{\partial}{\partial \beta}\left(\sum_{j=1}^{N}\sum_{j<i}\frac{a_{ij}r_{ij}}{1+\beta r_{ij}}\right)$$

Further we get:

$$\frac{\partial ln\psi_T}{\partial \beta} = \sum_{j=1}^{N}\sum_{i<j} -\frac{a_{ij}r_{ij}^2}{(1+\beta r_{ij})^2}$$

This is computed with the following code:

```
// compute energy of beta derivative of Jastrow factor
double VMCSolver::beta_derivative_Jastrow(){
    double derivative_sum = 0.0;
    for(int j=0; j<nParticles; j++){
        for(int i=0; i<j; i++){
            double divisor = (1.0 + beta*r_distance(i,j));
            derivative_sum += (-a_matrix(i,j)*r_distance(i,j)*r_distance(i,j))/(divisor*divisor);
        }
    }
    return derivative_sum;
}
```

To calculate $\left\langle E_L \frac{\partial ln\psi_T}{\partial \beta}\right\rangle$, we have to compute E_L (as usual) and $\frac{\partial ln\psi_T}{\partial \beta}$ in each MC cycle, multiply them, add them up in a sum, and divide by $Cycles \times electrons$ to get the expectation value. $\langle E\rangle$ (the ground state energy estimate) and $\left\langle \frac{\partial ln\psi_T}{\partial \beta}\right\rangle$ is found in the same way, but without the multiplying part. Then we have all we need to find $\frac{\partial E}{\partial \beta}$.

The algorithm we will use to find the optimal β is *Steepest descent* [20], which is an optimization algorithm of first order. The idea is to take a step in the negative direction of the gradient from the current position. In our case we have to pick an initial value of β, compute $\frac{\partial E}{\partial \beta}$, and take a step in the opposite direction of this. This in much better explained by some piece of code:

```
// steepest decent optimization algorithm to find optimal beta
void VMCSolver::findOptimalBeta(int my_rank, int world_size){
    fstream outfile;
    int nCycles = 100000;
    int max_iter = 500;
    double step = 0.01;
    const double precision = 1.0e-4;
    beta = 0.25;
    double beta_new = beta;
    double beta_old = 0.0;
    int counter = 0;
    while(abs(beta_new - beta_old) > precision) {
    MonteCarloIntegration(nCycles, outfile, my_rank, world_size);
        beta_old = beta_new;
        beta_new = beta_old - step*E_betaDerivative;
        beta = beta_new;
```

```
17          counter++;
18          if(counter > max_iter){
19              cout << "Max iter reached when searching for optimal beta!" << endl;
20              cout << "beta: " << beta_new << endl;
21              break;
22          }
23      }
24  }
```

We see that if we call this function, we perform series of short MC simulations that returns $EbetaDerivative$, used to compute the new beta value, until $abs(beta_{new} - beta_{old}) > precision$. Its also wise to set a maximum number of iters, so we not have to wait forever if the algorithm fails. In this code we use the same step lenght for all steps, but a more sophisticated algorithm should take shorter and shorter steps. This simple method also force us to use pretty many MC cycles (here we use 100 000) to obtain stable values.

But we also have a variational parameter α, what can we do with that? We could use a similar approach as with β, but a better solution is to use optimized GTO basis sets to get rid of α. This is explained in the next section.

3.13 GTO orbitals

"GTO" stands for $Gaussian\,Type\,Orbitals$. On the webpage https://bse.pnl.gov/bse/portal [7], we find a lot of different basis sets for different atoms in the periodic table. This is values found experimentally by chemists. Constructing gaussian shaped orbitals taking advantage of this data, should make us able to get even closer to the real value of the ground state in our MC simulations. We are going to use a basis called "3-21G" for He, Be and Ne. The basis sets contains optimized values for α and corresponding coefficients c. We will now explain how the gaussian orbitals are constructed.

We define [5]:

$$G^{\alpha}_{i,j,k} = N_{i,j,k} x^i y^j z^k e^{-\alpha r^2}$$

Here i, j, k is the quantum numbers n, l, m_l that we look at earlier when we discussed hydrogen-like orbitals. α is a value taken from the GTO basis, and r is the electron-nucleus distance as usual. $N_{i,j,k}$ is a normalization factor given by [6]:

$$N_{i,j,k} = \left(\frac{2\alpha}{\pi}\right)^{\frac{3}{4}} \left(\frac{(8\alpha)^{i+j+k} i!j!k!}{(2i)!(2j)!!(2k)!}\right)^{\frac{1}{2}}$$

Further we define:

$$\Phi_v = \sum_{\mu} c_{\mu v} G^{\alpha_\mu}_{i,j,k}$$

Here v and μ is indices to carry out the correct coefficients from the GTO basis table we choose to use. The orbitals are then given as:

$$\psi = \sum_{\sigma} K_\rho \Phi_v$$

K_ρ are coeffcient from $Hartee - Fock\,calculations$, and σ is an index telling how many terms we have to add up for each K_ρ. This explanation may be unclear, so we show what ψ looks like for He and Be with GTO. We set up the values from the GTO basis in a matrix as given in Tabel 3:

α_i	c_i	c_i
α_1	c_1	
α_2	c_2	
α_3	c_3	
α_4	c_4	c_7
α_5	c_5	c_8
α_6	c_6	c_9

(a) Content in a GTO table for Be, coefficients found experimentally by chemists.

$K_\rho\, 1.\, basis\, function$	$K_\rho\, 2.\, basis\, function$
K_1^1	K_1^2
K_2^1	K_2^2
K_3^1	K_3^2
K_4^1	K_4^2
K_5^1	K_5^2
K_6^1	K_6^2
K_7^1	K_7^2
K_8^1	K_8^2
K_9^1	K_9^2

(b) Coefficients from $Hartee - Fock\, calculations$ for two basis functions needed for Be.

Table 3: Coefficients used in the construction of GTO for Be.

We note that $\alpha_7 = \alpha_4$, $\alpha_8 = \alpha_5$ and $\alpha_9 = \alpha_6$ from the table above. For He we get the basis function (we only need one basis function since we have splitted our Slater matrix in spin up and spin down parts):

$$\psi_{He} = K_1\Big(c_1 N_{0,0,0} e^{-\alpha_1 r^2}\Big) + K_2\Big(c_2 N_{0,0,0} e^{-\alpha_2 r^2} + c_3 N_{0,0,0} e^{-\alpha_3 r^2}\Big)$$

For Be with four electrons we need two basis functions of the form:

$$\begin{aligned}
\psi_{Be}^b = {} & K_1^b\Big(c_1 N_{0,0,0} e^{-\alpha_1 r^2} + c_2 N_{0,0,0} e^{-\alpha_2 r^2} + c_3 N_{0,0,0} e^{-\alpha_3 r^2}\Big) \\
& + K_2^b\Big(c_4 N_{0,0,0} e^{-\alpha_4 r^2} + c_5 N_{0,0,0} e^{-\alpha_5 r^2}\Big) + K_3^b\Big(c_6 N_{0,0,0} e^{-\alpha_6 r^2}\Big) \\
& + K_4^b\Big(c_7 N_{1,0,0} e^{-\alpha_7 r^2} + c_8 N_{1,0,0} e^{-\alpha_8 r^2}\Big) + K_5^b\Big(c_9 N_{1,0,0} e^{-\alpha_9 r^2}\Big) \\
& + K_6^b\Big(c_7 N_{0,1,0} e^{-\alpha_7 r^2} + c_8 N_{0,1,0} e^{-\alpha_8 r^2}\Big) + K_7^b\Big(c_9 N_{0,1,0} e^{-\alpha_9 r^2}\Big) \\
& + K_8^b\Big(c_7 N_{0,0,1} e^{-\alpha_7 r^2} + c_8 N_{0,0,1} e^{-\alpha_8 r^2}\Big) + K_9^b\Big(c_9 N_{0,0,1} e^{-\alpha_9 r^2}\Big)
\end{aligned}$$

, where we use different K_ρ^b-values for different basis functions. For Ne, with new α and c coefficients, we get the exact same form of the basis functions as for Be, but with different K_ρ^b-values and five different basis functions ($b = 1, .., 5$).

We also have to find the gradients and Laplacians of orbitals as usual, which in this case means that we have to find the derivatives of $G_{i,j,k}^\alpha$ with respect to r. That is actually a simple task, and the result can be found in our code, so we don't bother to write it down here. All we have to do now is to implement the basis functions and the corresponding gradients and Laplacians in the exact same way as with hydrogenic wave functions. How GTO in implemented can be found in "Code implementation 6.8".

3.14 Implementation of molecules and modification of our program

In the previous sections we have looked at atoms. Now we will look at simple the diatomic molecules H_2 and Be_2. The machinery is the same for molecules as for atoms, but we have to do some modifications on the trial wave function. We also have to keep track of two different electron-nucleus distances, since we have two separate nucleons. The extra nucleus also give rise to a new potential energy between the nucleons, as well as additional potential energy between electrons and the extra nucleus. The new hamiltonian we obtain for diatomic molecules (with homogeneous atoms) is:

$$\hat{\boldsymbol{H}} = \sum_{i=1}^{2Z}\Big(-\frac{\nabla_i^2}{2} - Z\Big(\frac{1}{r_{1i}} + \frac{1}{r_{2i}}\Big) + \frac{Z^2}{|R_\rho|}\Big) + \sum_{i=1}^{Z}\sum_{i<j}\frac{1}{r_{ij}}$$

Here we recognize r_{1i} and r_{2i} as the distance from electron i to the two nucleons, and $R_\rho = \boldsymbol{r}_1 - \boldsymbol{r}_2$, with $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ as the position vectors for the nucleons. We have for H_2 $R_\rho = 1.40$, [13] known as the *Bohr radii*, and $R_\rho = 4.63$ for Be_2 [23].

With only one nucleon, it was obvious to place the nucleus in origo. Now when we got two, we have to place them out. If we name the distance between the nucleons (inter-proton distance) R_ρ, it's natural to plaze both in the same xy-plane, but with coordinates $z = -\frac{R_\rho}{2}$ and $z = +\frac{R_\rho}{2}$ along the z-axis. Then our position vectors become:

$$\boldsymbol{r}_{1i} = \boldsymbol{r}_i + \frac{R_\rho}{2}$$

$$\boldsymbol{r}_{2i} = \boldsymbol{r}_i - \frac{R_\rho}{2}$$

, where $\boldsymbol{r}_i$ is the position vector for electron i relative to origo.

Our trial wave function for H_2 is on the form:

$$\psi_T(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_{12}, \alpha, \beta) = \psi(\boldsymbol{r}_1, \boldsymbol{R})\psi(\boldsymbol{r}_2, \boldsymbol{R})e^{\frac{r_{12}}{2(1+\beta r)}}$$

, with:

$$\psi_1(\boldsymbol{r}_i, \boldsymbol{R}) = e^{-\alpha r_{1i}} + e^{-\alpha r_{2i}}$$

For Be_2 we need four orbitals, and the next three orbitals are given by:

$$\psi_2(\boldsymbol{r}_i, \boldsymbol{R}) = e^{-\alpha r_{1i}} - e^{-\alpha r_{2i}}$$

$$\psi_3(\boldsymbol{r}_i, \boldsymbol{R}) = \big(1 - \frac{\alpha}{2} r_{1i}\big)e^{-\frac{\alpha r_{1i}}{2}} + \big(1 - \frac{\alpha}{2} r_{2i}\big)e^{-\frac{\alpha r_{2i}}{2}}$$

$$\psi_4(\boldsymbol{r}_i, \boldsymbol{R}) = \big(1 - \frac{\alpha}{2} r_{1i}\big)e^{-\frac{\alpha r_{1i}}{2}} - \big(1 - \frac{\alpha}{2} r_{2i}\big)e^{-\frac{\alpha r_{2i}}{2}}$$

We note that we only have to use hydrogenic orbital 1s and 2s to construct orbitals for Be_2, because of opposite signs between the wave functions. In the implementation of gradients and the Laplacians, we do it in the exact same way as for atoms, but we have to be careful with the derivation with respect to z since the nucleons is not placed in origo anymore.

It can be shown that in the limit where all distances approach zero that:

$$\alpha = 1 + e^{(\frac{-R_\rho}{\alpha})}$$

, which make us able to plot the two sides of the equation to find α as the intersection between them, leaving β as the only variational parameter. We have earlier presented the steepest decent method to pick out the value of β that minimizes the energy.

For H_2 we obtain:

$$\alpha = 1 + e^{(\frac{-1.40}{\alpha})} \implies \alpha = 1.356$$

3.15 Blocking

Now we got all the theory to make etimates of the ground state energy for both atoms and simple diatomic molecules, but we have to find a proper way to present the estimate. This is done by include the standard deviation or σ and present the estimated energy as:

$$E_{VMC} \pm \sigma$$

If our samples are uncorrelated, the standard deviation of our etimate of the mean would have been:

$$\sigma = \sqrt{\frac{1}{N}\big(\langle E^2_{mean}\rangle - \langle E_{mean}\rangle^2\big)}$$

But since our samples actually are correlated, the computed σ will be too low, and the correct standard deviation is given by [1]:

$$\sigma = \sqrt{\frac{1 + 2\tau/\Delta t}{N}\big(\langle E^2_{mean}\rangle - \langle E_{mean}\rangle^2\big)}$$

, where τ is *correlation time* (time between uncorrelated samples) and Δt is time between samples. Unfortunately τ is unknown and cost a lot of CPU, fortunately this can be solved with *blocking*. Blocking means that we run a MC simulation, store every computed energy, divide the stored energy into different block sizes, and compute and plot the standard deviation (SD) as a function of the blocksize. The SD for each blocksize is found by the formula:

$$\sigma_{trial} = \sqrt{\frac{1}{N_{block}} \sum_{N_{block}} \langle E_{block}^2 \rangle - \left(\frac{1}{N_{block}} \sum_{N_{block}} \langle E_{block} \rangle \right)^2}$$

A matlab code for doing this is found in "Code implementation 6.9".

After plotting the SD as a function of block size, we should see a "plateu" where the SD is independent of the block size, and the SD on this plateu we name σ_{block}, with the corresponding $N_{blocksize}$ where the plateu starts. Then the actual σ we want to find is given by:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}}$$

, where N is the size of the dataset we have done blocking on. Now we are able to get result and present them properly.

3.16 Verifications

On a big project like this with many lines of code, we have to do some verifications in the progress. In section 3.6 about local energy we wrote the closed form expressions for the local energy for He. This made us able to run the program both with an analytical expression for the local energy, as well as a numerical solver. We could also turn on and off the Jastrow factor to see if it was working as expected as well. If we run with the same random seed in MC simulations, we get the results given in Table 4:

$N = 10^6$	*Numerical*	*Analytical*
without Jastrow factor	-2.81272	-2.81269
with Jastrow factor	-2.89373	-2.89334

Table 4: Estimate of ground state energy of Helium.

We see that the numerical and the analytical solver gives approximately the same results, indicating that the numerical solver works as expected.

We would also like to test that the optimized Slater and Jastrow machinery is working as expected. In section 3.2 about hydrogenic wave functions, we mentioned that the expectation value of the energy become exact if $\alpha = 1$. This gives rise to a very useful test to check if the Slater machinery is working. If we set $\alpha = Z$ and remove the interaction term $\frac{1}{r_{ij}}$ in the hamiltonian, we should reproduce benchmarks down to numerical precision. We have from quantum mechanics that [3]:

$$\langle \psi | \hat{\boldsymbol{H}} | \psi \rangle = \sum_{i=1}^{Z} -\frac{Z^2}{2m_i}$$

This leads to $He = -4$, $Be = -20$ and $Ne = -200$. We are able to reproduce these values down to numerical precision, so it looks like the Slater machinery is working as expected.

To test other parts, such as the energy contribution from the Jastrow factor, we removed the Slater part of the wave function, and compared the computed energy to the analytical expression.

3.17 Parallelization of the Variational Monte Carlo solver

To parallelize the code we use MPI. This is simply done by running multiple MC simulations simultaneously, collect the estimated energies, and divide by the number of processes:

```
1 int my_rank, world_size;
2 double energy, sum_energy;
3
4 // Initialize the parallel environment
5 MPI_Init (&nargs, &args);
6 MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
```

```
7  MPI_Comm_size (MPI_COMM_WORLD, &world_size);
8
9  //VMCSolver *solver = new VMCSolver();
10 //energy = solver->runMonteCarloIntegration(nCycles, my_rank, world_size);
11
12 // Collect energy estimates
13 MPI_Reduce(&energy, &sum_energy, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
14
15 // Final energy estimate with multiple processors
16 if(my_rank==0){
17     cout << "Totalt cycles: " << nCycles << endl;
18     cout << endl << "Ground state estimate: " << sum_energy/world_size << endl;
19 }
20
21 // Clean up and close the MPI environment
22 MPI_Finalize();
```

The nature of Monte Carlo makes it very appropriate to parallelize the code. How how much speedup we obtain will be studied later.

3.18 Monte Carlo solver

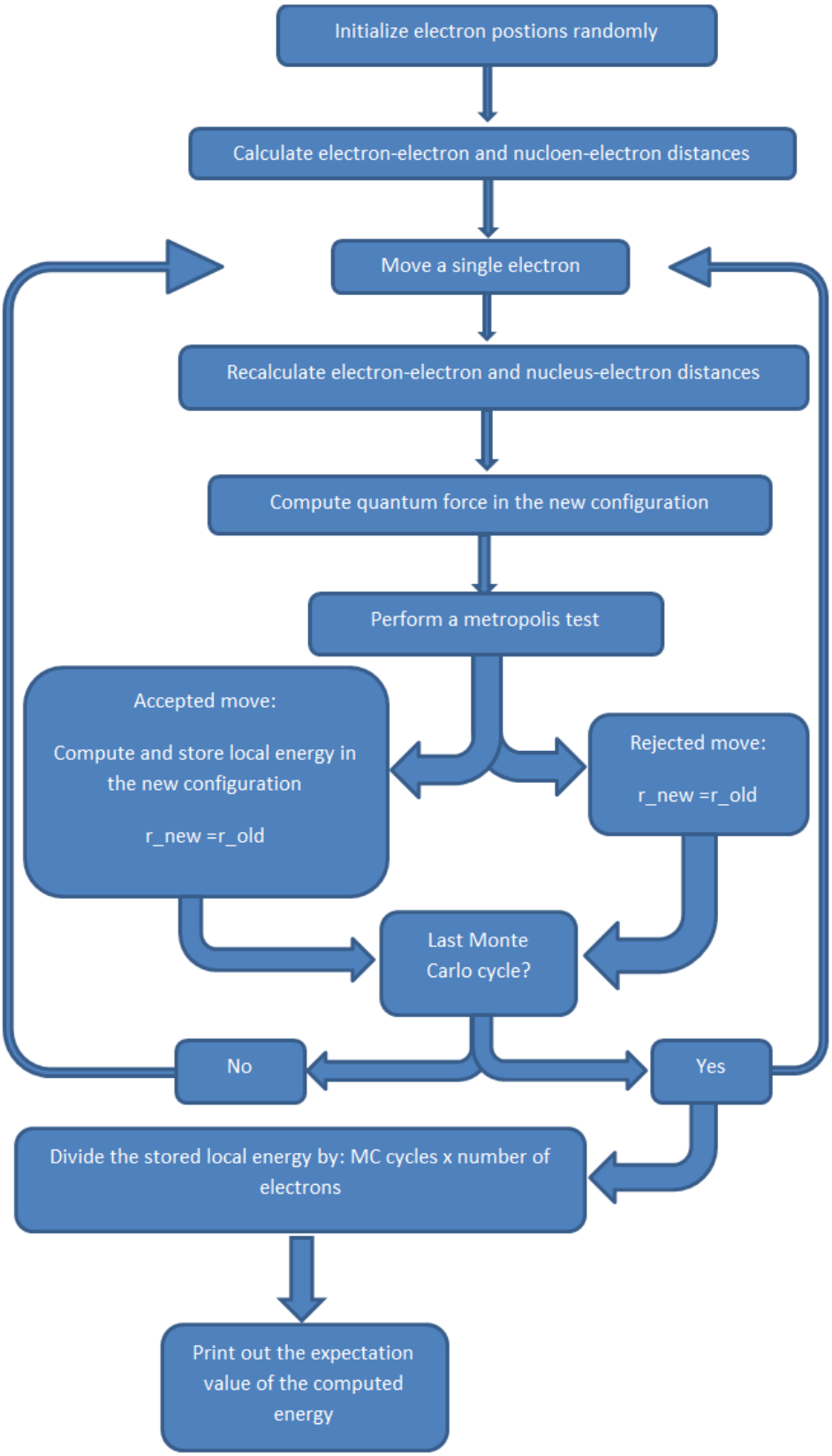


Figure 2: Flow chart for our Monte Carlo solver.

4 Results and discussion

4.1 Variational parameters and ground state energy for atoms

In the method section we explained how to find optimal values for variational parameters. In the case with only one variational parameter α for He, we could graphically obtain the optimal α from Figure 3:

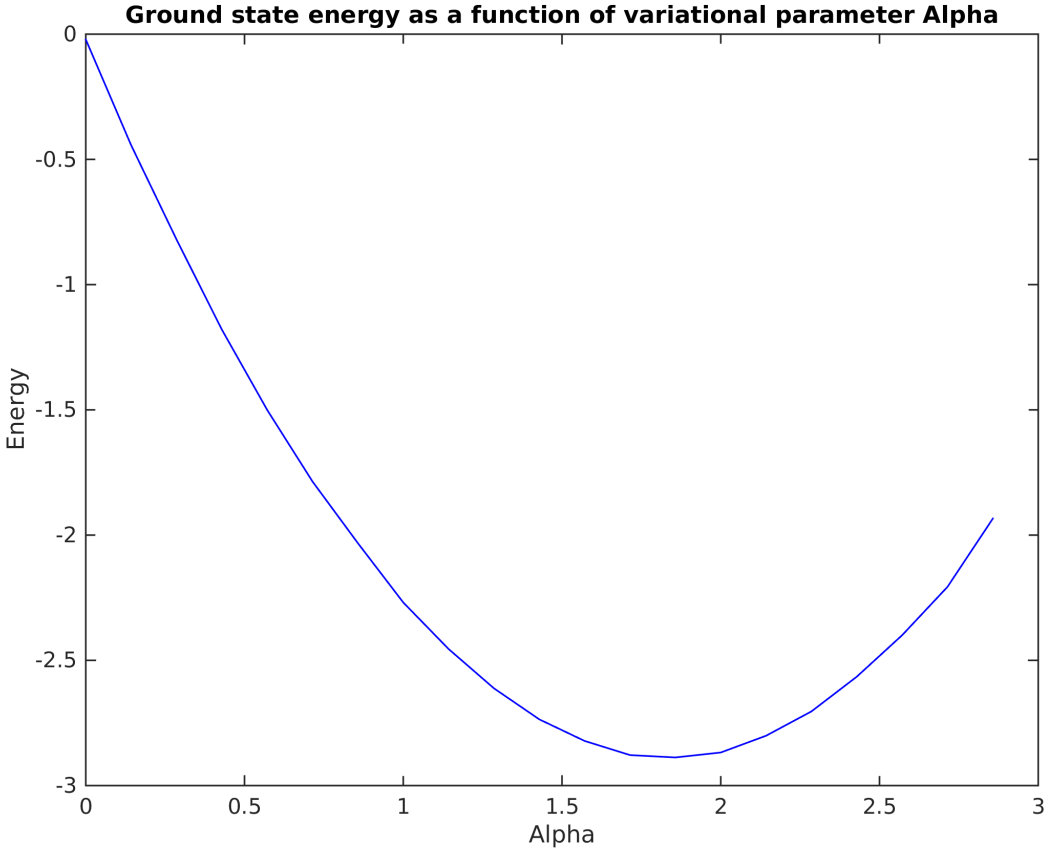


Figure 3: We see how the estimated ground state for He changes with differetn values of alpha. We reach a minimum about $\alpha = 1.85$.

As we remember from the method section, we can use the same procedure when we have two variational parameters, but then we have to run over a mesh. For He we found graphically the optimal parameters: $\alpha = 1.85$ and $\beta = 0.35$. In the same way we found $\alpha = 3.9$ and $\beta = 0.1$ for Be. Since this method turned out to be very time consuming, and the manually pick of optimal values from graphical evaluation of the mesh was pretty unaccurate, we was looking for alternative solutions. With introducing GTO basis, we got rid of the need of finding optimal α, and we could use steepest decent method to find the optimal β. With steepest decent we found $\beta = 0.259$ for He and $\beta = 0.206$ for Be. We see that the graphical results are fairly close those we obtain with steepes decent, but it's more convenient to find values automatic and save a lot of CPU.

As an example to prove that the values from steepest decent gives us reliable results, we can compare the energy to a reference value. For Be (with the same random generated numbers) we get with $\beta = 0.1$ (from graphical evaluation) that $E_0 = -14.510\, a.u.$ ($a.u. = atomic\, units$), compared to $\beta = 0.206$ (from steepest decent method) which gives $E_0 = -14.516\, a.u.$. The true value we try to find is $E_0 = -14.667\, a.u.$ [14]. We see that $\beta = 0.206$ brings us a little bit closer to the true minimum. If we use α values from graphical evaluation, the results obtained are summarized in the table below (with Jastrow factor included in trial wave function):

$Atom$	α	$Manual\ \beta$	$E_0\ (manual\ \beta)$	$Steepest\,decent\ \beta$	$E_0(steepest\,decent\ \beta)$	$Reference\ E_0$
He	1.85	0.35	-2.893	0.259	-2.901	-2.9037
Be	3.9	0.1	-14.510	0.206	-14.516	-14.667
Ne	10.2	0.1	-128.08	0.098	-128.04	-128.94

Table 5: Estimated energies with β obtained from graphical evaluation and the steepes decent method. Here we have used $N = 10^6$ MC cycles.

We see from the table above that the energy estimate for Ne didn't get better with the β from steepes decent, but this can be due to lack of accuracy in α and to random effects in the MC solver.

It's also interesting to look at the values of α. We have mentioned earlier how the solution is exact for hydrogen if $\alpha = 1$. But why doesn't α become 2 for He when our trial wave function is construted of two hydrogen-orbitals? The reason for this is the interaction between the electrons, where the repulsation between them destroys a perfectly symmetric electric field around the nucleus.

4.2 Variance in Monte Carlo simulations

When doing MC simualtions, we should gain precision with increased number of MC cycles. To test that this is the case, we can plot the variance as a function of MC cycles, and expect the variance to decrease for increasing number of MC cycles. Doing a MC simulation with Jastrow factor (without importance samling) with He, gives the figure below:

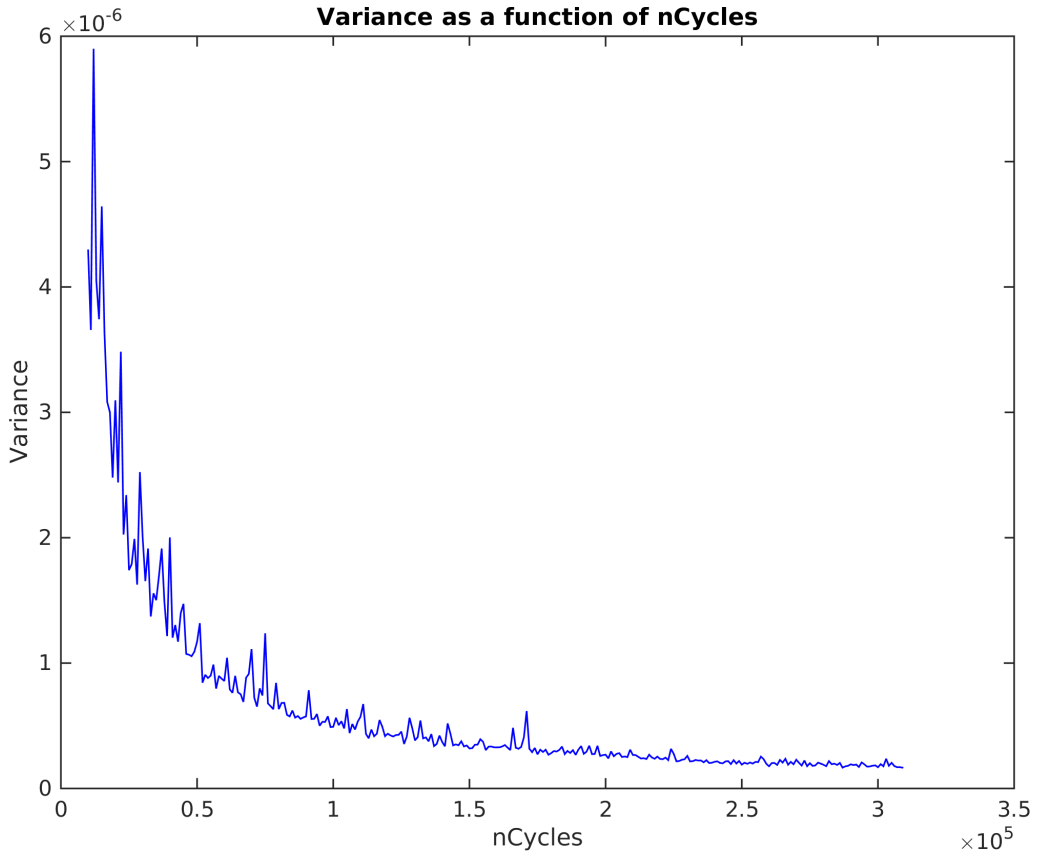


Figure 4: We see how the variance become smaller when MC cycles are increased in simulation of the Helium atom.

As expected, the variance of the energy is decreasing for increasing MC cycles. We always want to use as many cycles as possible within the range of CPU-time to get the best possible approximation. We should note that the variance computed here is *not* the real variance of the estimated ground state energy because of correlation. To find the true variance we need to perform blocking.

4.3 Speed against different solvers

We have talked about the advantage of finding analytical expression if possible, but usually the expression fastly becomes ugly. Therefore we implemented an optimized analytical machinery making us uble to run our program pretty fast for big atoms such as Ne. It can be interesting to see how much speedup we get with the analytical expression or the optimized solver, compared to a brute force numerical solver. The result for He is given in the table below:

$N = 10^8$	*numerical*	*optimized*	*analytical*	*rel. speedup optimized*	*rel. speedup analytical*
without Jastrow factor	295.21 s	143.44 s	134.20 s	2.06	2.20
with Jastrow factor	617.11 s	167.28 s	161.25 s	3.69	3.83

Table 6: Simulation to estimate ground state energy for helium. We see that we get a great relative speedup in the MC simulation with analytical expressions or the optimized solver compared to the numerical solver, both with and without Jastrow factor included in the trial wave function.

It should be clear that implementing the optimized solver really is worth the bill. The more complex our trial wave function become, the more is the pay from the optimized solver compared to the numerical solver.

4.4 Optimization with Jastrow factor and importance sampling

We want to see how the estimated E_0 and the mean electron-electron distance is affected by the Jastrow factor and importance sampling. In the two tables below we have performed simulations on He with different combinations:

$N = 10^8$	*importance sampling off*	*importance sampling on*
without Jastrow factor	-2.8214	-2.8214
with Jastrow factor	-2.8902	-2.8905

Table 7: Estimate of ground state energy of Helium.

$N = 10^8$	*importance sampling off*	*importance sampling on*
without Jastrow factor	1.1824	1.1841
with Jastrow factor	1.3564	1.3569

Table 8: Estimate of the mean distance between the electrons in the Helium atom.

From the tables above we observe that the importance sampling (with $\Delta t = 0.002$) has no significant effect on the ground state estimate or the electron-electron distance. On the other hand, the Jastrow factor is of big importance. We see that we get much closer to the reference E_0 from section 4.1 when Jastrow is included in our trial wave function. We also observe how the repulsation between the electrons increase the distance between them when Jastrow is included.

If we now look at importance sampling, we want to check how the energy and mean electron-electron distance depends on the choosen value of Δt (*timestep*), mentioned in section 3.8. For the timestep used in Table 7 and Table 8, we could not observe any effect caused by importance sampling, but we want to check if that is the case for all values of Δt. The result from a simulation with $N = 10^7$ are given in Figure 3:

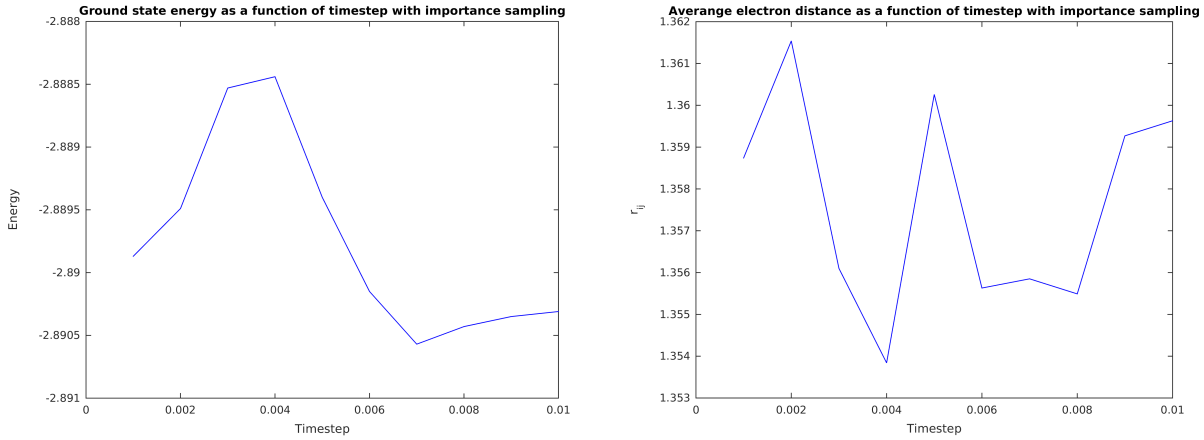


Figure 5: Energy and mean distance between electrons as a function of Δt.

We note the small scale on the y-axis for both figures in Figure 5. We can't see any clear dependence between timestep and energy from Figure 5, so the value of Δt is not important within the range of Δt values ([0.001,0.01]). The same conclusion can also be drawn for averange electron distance r_{ij}. So a proper value for Δt could be 0.002, but we have to make sure that we have enought MC cycles to move electrons to the ground state configuration.

It's important to mention that when we not used importance sampling, we had to construct a very simple algorithm to find the step length that gave us an acceptance ratio about 0.5 in the Metropolis algorithm. In the case with importance sampling, we will almost always accept the moves as long as the Δt is small. Since the quantum force pushes the electrons in the right direction (where the wave function is large) we will accept moves, and it's reasonable to think that we need fewer MC cycles to reach an energy close to the ground state energy. A plot show what it looks like:

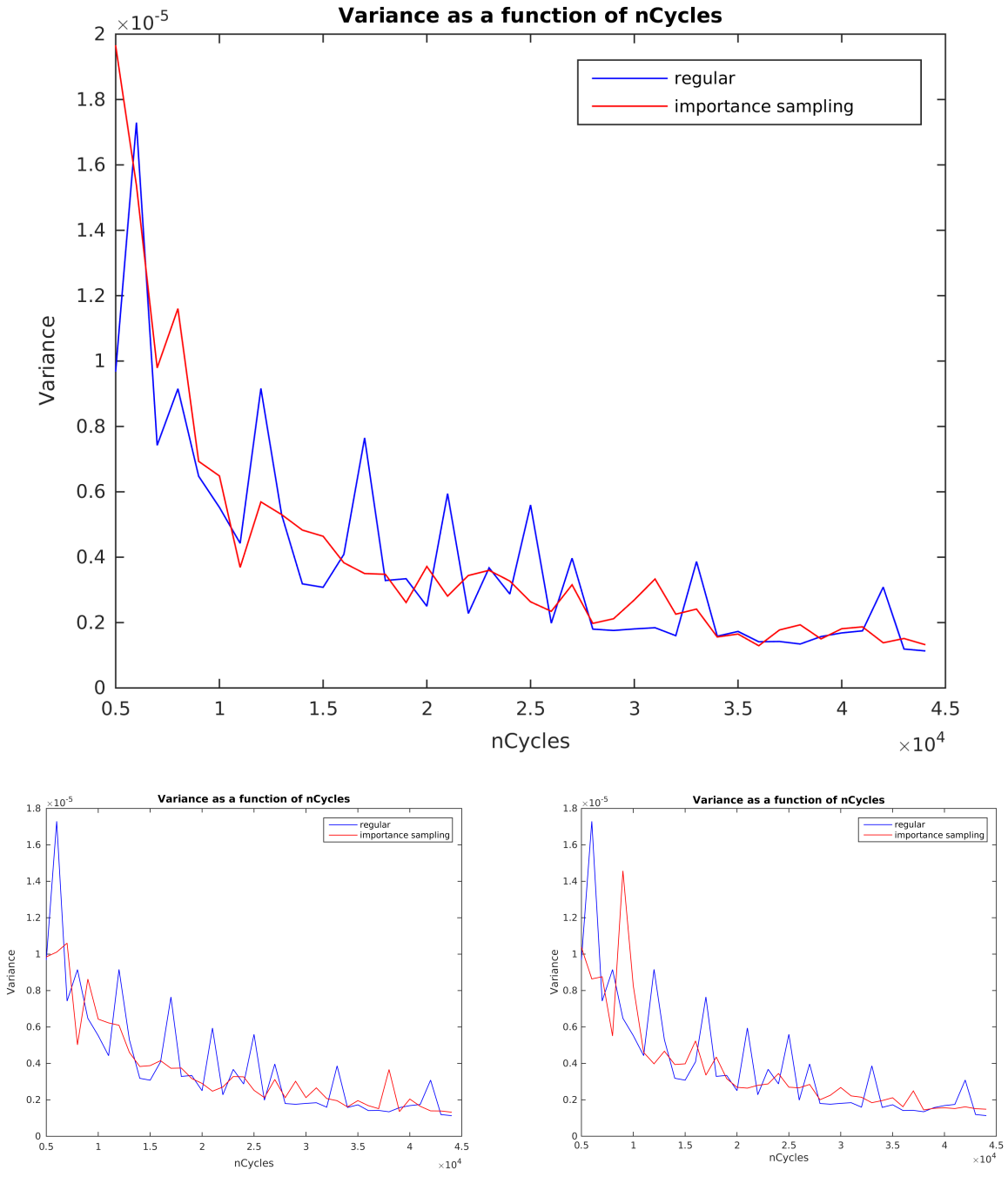


Figure 6: We see that the variance actually decreases equally with and without importance sampling, but with importance sampling there are less variation in the variance, so there should be less variation in the computed energy for few cycles. $\Delta t = 0.002$ *in the first figure,* $\Delta t = 0.005$ *in the figure down left, and* $\Delta t = 0.01$ *in the figure down right.*

Surprisingly we cannot see any clear difference in how fast the estimated energy tends to the ground state energy when quantum force is applied. However there is less variance in the calculations with importance sampling, so we will for fewer cycles get a result with less risk of "bad luck" due tue variance.

We have seen how importance sampling leads to less variance in simulations of E_0, and that almost every suggested electron-move is accepted, but what about the CPU expences? Let's look at Figure 7 (with Jastrow factor):

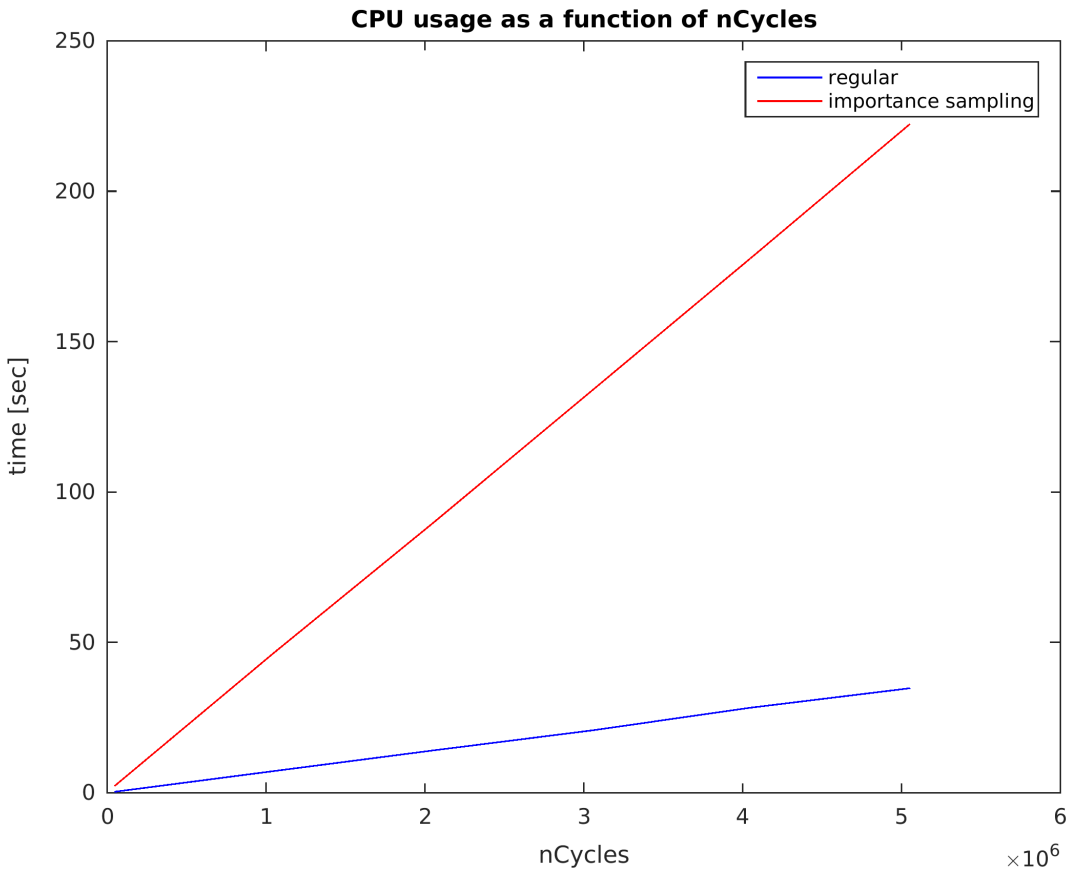
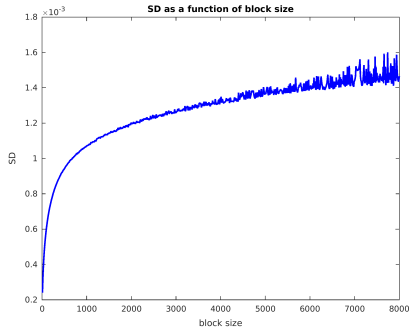


Figure 7: CPU-time as a function of MC cycles for He with and without importance sampling.

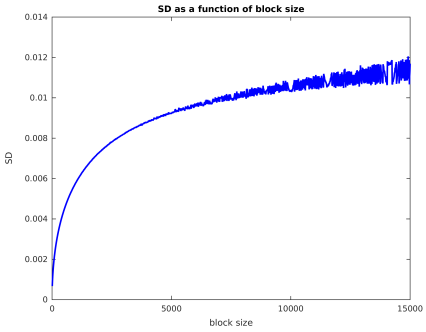
From Figure 7 we see that including importance sampling increase the CPU-time a lot, so we have to pay a lot to reduce the variance.

4.5 Finding the real variance with Blocking

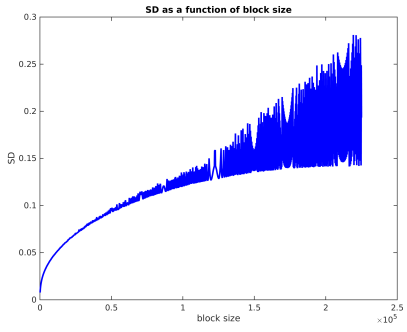
As mentioned in section 3.15, we can't calculate the standard deviation with a simple formula because of correlation between samples. Blocking was metioned as the solution to this problem. Performing blocking on He, Be and Ne gives:



(a) Blocking performed on He with $N = 10^7$. We see a "plateu" where $\sigma_{block} \approx 1.4 \times 10^{-3}$ and $N_{blocksize} \approx 5000$.



(b) Blocking performed on Be with $N = 10^7$. We see a "plateu" where $\sigma_{block} \approx 0.011$ and $N_{blocksize} \approx 10000$.



(c) Blocking performed on Ne with $N = 10^7$. We see a "plateu" where $\sigma_{block} \approx 0.14$ and $N_{blocksize} \approx 200000$.

Figure 8: Blocking performed on He, Be and Ne.

The corresponding standard deviations are found by:

He:

Using value $\sigma_{block} \approx 1.4 \times 10^{-3}$, $N_{blocksize} \approx 5000$ and $N = 10^7$, we find the standard deviation:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}} = \frac{1.4 \times 10^{-3}}{\sqrt{10^7/5000}} = 3.130 \times 10^{-5}$$

Be:

Using value $\sigma_{block} \approx 0.011$, $N_{blocksize} \approx 10000$ and $N = 10^7$, we find the standard deviation:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}} = \frac{0.011}{\sqrt{10^7/10000}} = 3.48 \times 10^{-4}$$

Ne:

Using value $\sigma_{block} \approx 0.14$, $N_{blocksize} \approx 200000$ and $N = 10^7$, we find the standard deviation:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}} = \frac{0.14}{\sqrt{10^7/200000}} = 1.98 \times 10^{-2}$$

4.6 Onebody density and charge density

Before we look at the results and start the discussion around onebody density, it's very helpful to take a look at what the hydrogenic orbitals look like. This is show in the figure below:

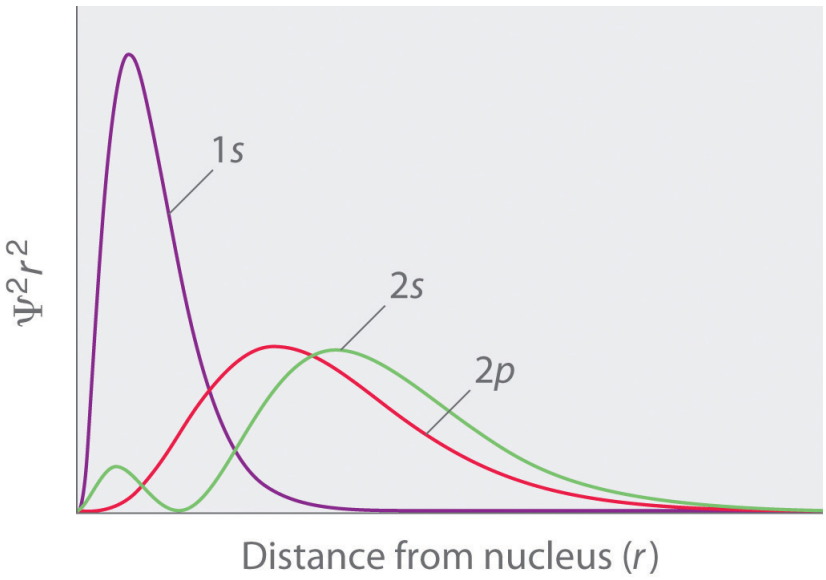


Figure 9: Electron probability density for different hydrogenic orbitals (given in the figure) [17].

The figure above is a nice tool to discuss our results, which is given by the figures below:

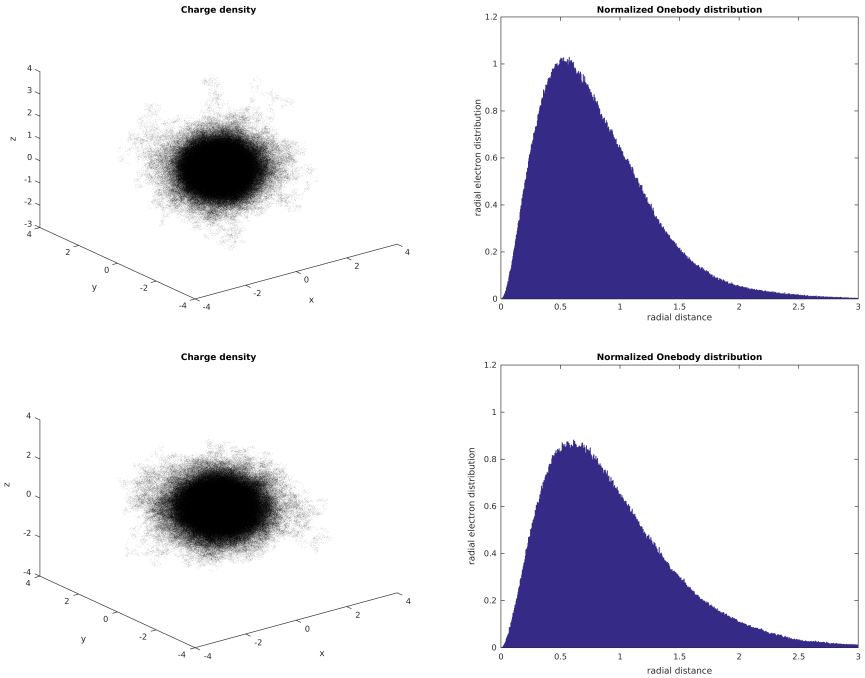


Figure 10: Onebody density helium. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

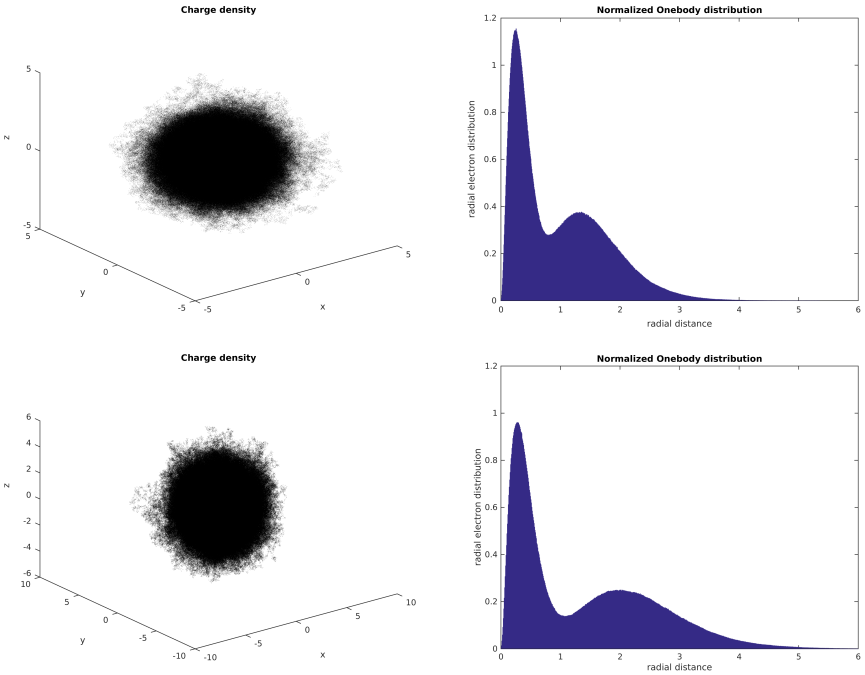


Figure 11: Onebody density beryllium. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

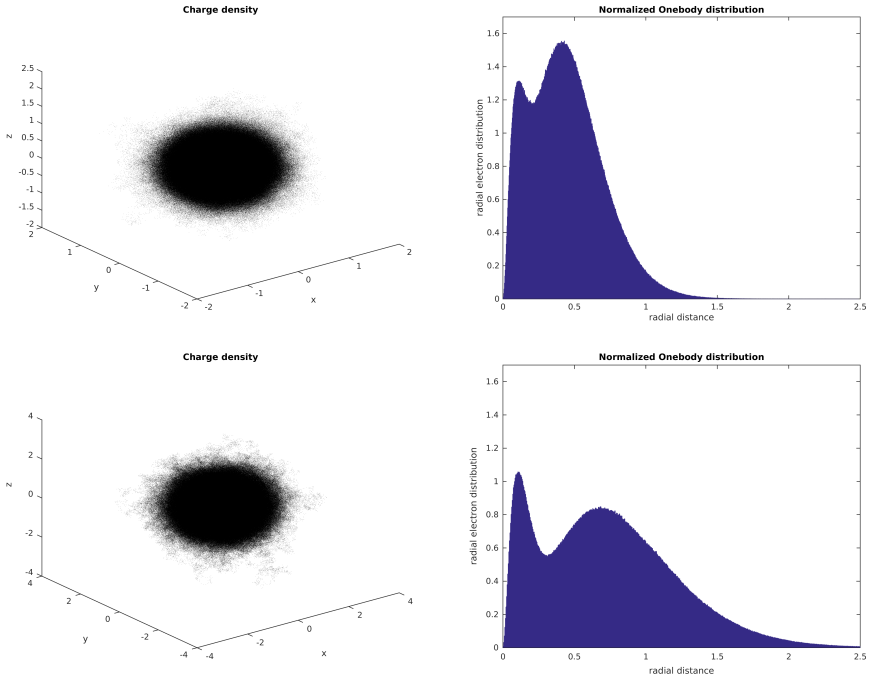


Figure 12: Onebody density neon. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

In the three figures above, we have plotted results for He, Be and Ne. It's used pure hydrogenic wave function in the upper part of each figure, and with Jastrow factor in the lower part. In the left coloumn we have just plotted the position of one electron at all times. In the right coloumn we have plotted a histogram of radial distance from the nucleus.

The left column gives us an idea of what the electron density looks like, but it's a lot of interesting information we cannot see from this angle, and we hardly see any difference when including Jastrow factor for any of the atoms.

In the right coloum we get far more interesting results. For He we spot the 1s orbital for the electrons. It's not much difference between pure hydrogenic wave function and when including an interaction factor, but we can observe some wider spread when Jastrow is included, due to electron repulsion.

Since we have filled up two orbitals for Be, we would from Figure 9 expect to see a distribution with two peaks, orbital 1s and 2s. We can clearly spot two peaks in both figures, but the plot with Jastrow has some wider spread.

For Ne with three full orbitals, we would from Figure 9 expect to see orbital 1s and a combination 2s and 2p. Here we clearly see two peaks in the distribution in both figures, and also here we observe more spread with Jastrow. We are also able to weakly distinguish orbital 2s and 2p in the right peak in the figure with Jastrow. Since orbital 2s and 2p interfere a lot, it's hard to tell them apart.

We see that the Jastrow factor is of great importance in our experiments, especially for beryllium and neon. Jastrow doesn't make a huge difference in the plots, but the small difference makes us able to easier spot different orbitals in the atoms, and drive our results much closer to the "correct" results/shapes in Figure 9.

4.7 Molecules

We are now moving over from atoms to diatomic molecules, using Jastrow factor and importance sampling in the calculations. The new property we had to take care of with molecules, is the two nucleons instead of one. We want to study how the ground state energy depends on the inter-proton distance R_ρ. If we run over different R_ρ and compute the ground state energy, we have to use steepest decent method to find the optimal β for each R_ρ. For H_2 we use $\alpha = 1.356$ from section 3.14, and get the result given in the figure below:

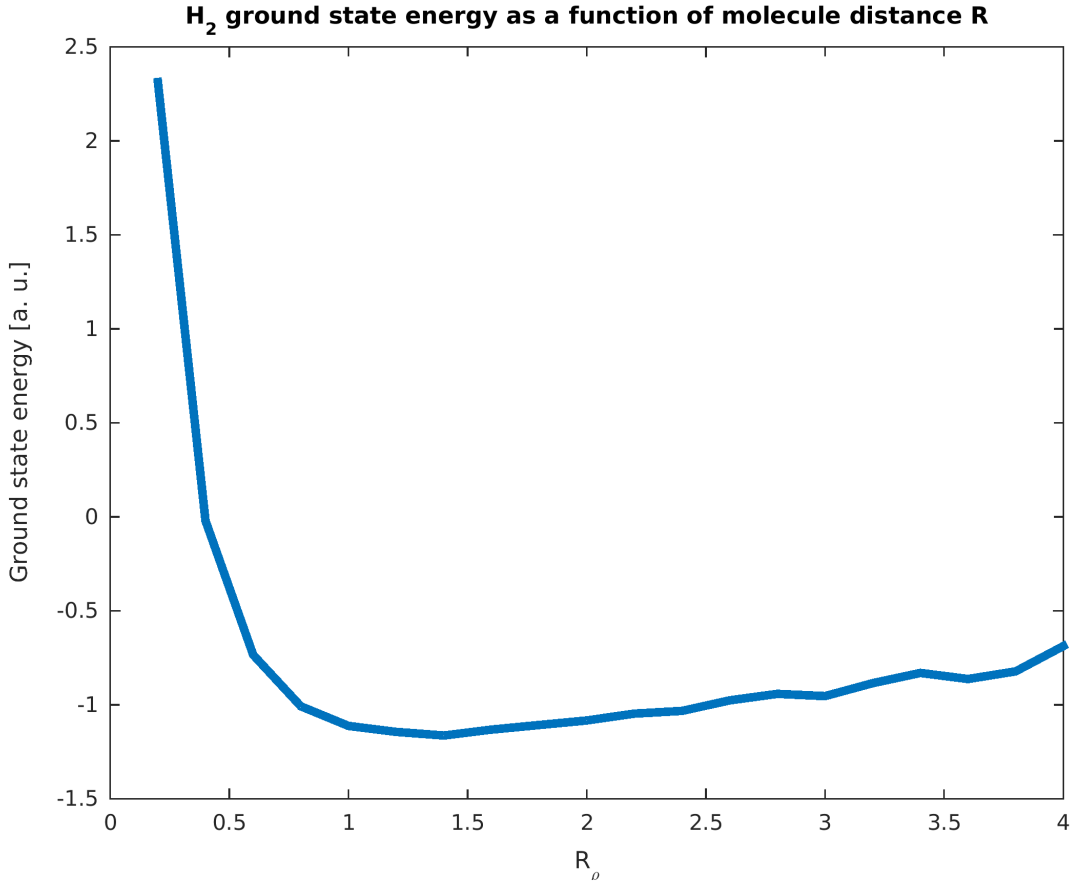


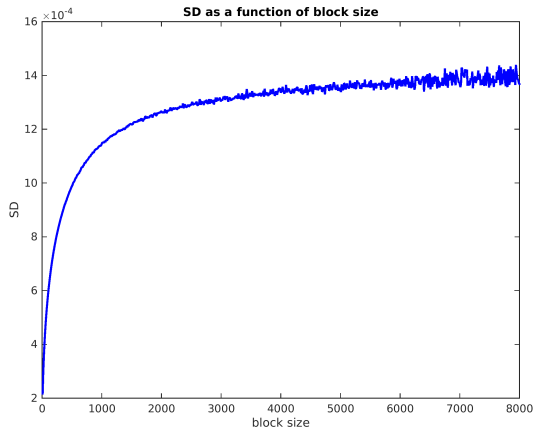
Figure 13: Ground state energy of H_2 as a function of inter-proton distance, $\alpha = 1.356$.

From the figure above, we se that we obtain the minimum energy at the bohr radii 1.40 for H_2, as expected. The corresponding optimal variational parameter if found to be $\beta = 0.276$. If we run a simulation with the obtained values $\alpha = 1.356$, $\beta = 0.276$ and $R_\rho = 1.40$, we find the averange electron-electron distance to be ≈ 2.18, with $E_0 = -1.1575$. We se that the averange distance has increased from 1.3912 in He to 2.18 in H_2, with the same number of electrons (two). We observe that this increase in mean distance is pretty close to a half bohr radii = 1.4. We get:

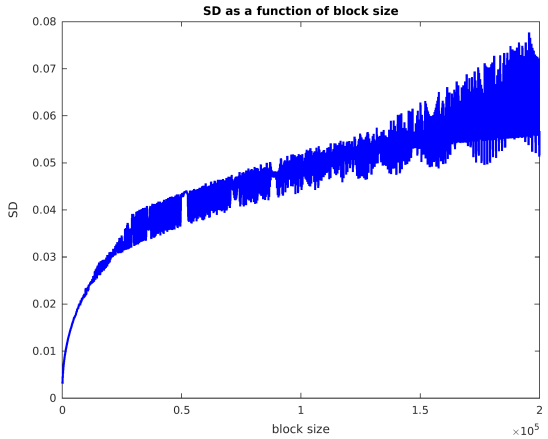
$$1.3912 + \frac{1.4}{2} = 2.09$$

This is logical and support that the results obtained are trustworthy.

For completeness we also estimate σ in the energy for H_2 and Be_2, and blocking gives:



(a) Blocking performed on H_2 with $N = 10^7$. We see a "plateu" where $\sigma_{block} \approx 14 \times 10^{-4}$ and $N_{blocksize} \approx 5000$.



(b) Blocking performed on Be_2 with $N = 10^7$. We see a "plateu" where $\sigma_{block} \approx 0.05$ and $N_{blocksize} \approx 150000$.

Figure 14: Blocking performed on H_2 and Be_2.

H_2:

Using value $\sigma_{block} \approx 14 \times 10^{-4}$, $N_{blocksize} \approx 5000$ and $N = 10^7$, we find the standard deviation:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}} = \frac{14 \times 10^{-4}}{\sqrt{10^7/5000}} = 9.89 \times 10^{-6}$$

Be_2:

Using value $\sigma_{block} \approx 0.05$, $N_{blocksize} \approx 150000$ and $N = 10^7$, we find the standard deviation:

$$\sigma = \frac{\sigma_{block}}{\sqrt{N/N_{blocksize}}} = \frac{0.05}{\sqrt{10^7/150000}} = 1.94 \times 10^{-3}$$

Above we used wave functions for H_2 on the form:

$$\psi(\boldsymbol{r}_i, \boldsymbol{R}) = e^{-\alpha r_{1i}} + e^{-\alpha r_{2i}}$$

, but what happens if we change the sign from $+$ to $-$? Again we run over different R_ρ-values with the new trial wave function $\psi(\boldsymbol{r}_i, \boldsymbol{R}) = e^{-\alpha r_{1i}} - e^{-\alpha r_{2i}}$. The result is given by the figure below:

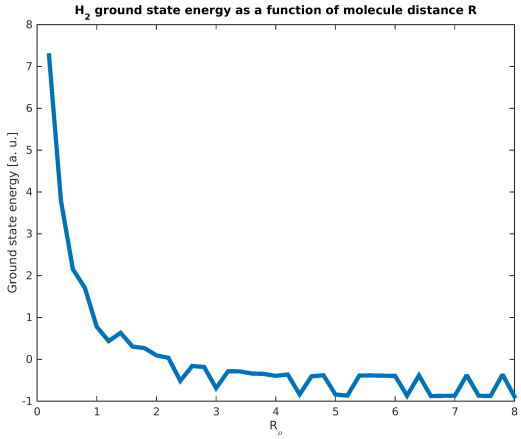
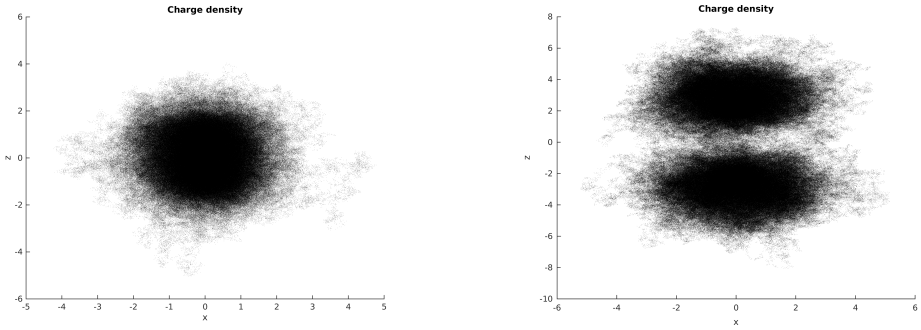


Figure 15: Ground state energy of H_2 (with minus in the trial wave function) as a function of inter-proton distance, $\alpha = 1.356$.

In this plot we doesn't get a nice minimum as we did with a plus sign in our trial wave function. In addition we are far away from the reference energy (see table 9), and we get oscillations due to too few MC cycles and unstable

β values, which we didn't get with the other wave function. It's not easy to draw any conclusions from this, and it's possible that the wave function with minus sign actually make no physical sense for H_2. It's also possible that the optimal $\alpha = 1.356$ is incorrect in the new trial wave function.

We should also look at charge desities for molecules. Results (with Jastrow factor) are given i the figure below:



(a) Onebody density H_2.

(b) Onebody density Be_2.

Figure 16: Visualization of H_2 and Be_2.

In the figure above we are not able to distinguish the two nucleons in H_2 since they are too close, but for Be_2 with much bigger inter-molecular distance we clearly see two nucleons. Radial electron density plot are of less interest for molecules, so it's not included in this report.

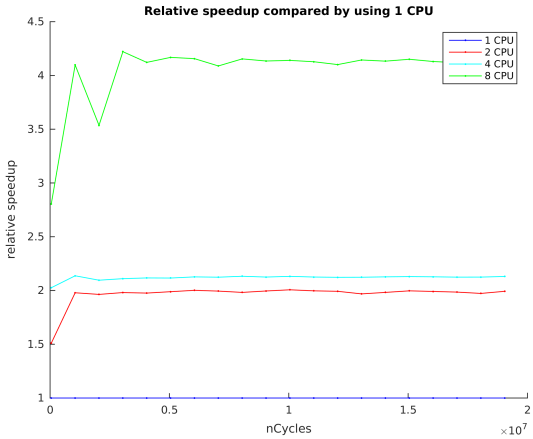
4.8 A note on GTO in this project

In section 3.13 we explained how to use GTO basis sets. In theory, we should get even better ground state estimates with GTO compared to our hydrogenic-like orbitals. This is not the case in our experiments. We get for He (with $N = 10^7$ and Jastrow factor) an energy ≈ -2.86, which is a poorer estimate than we got with hydrogen like orbitals. For Be we got an energy ≈ -13.97, and for Ne an energy ≈ -119.67. All simualtions leads to poorer estimates of the ground state energy that with hydrogenic trial wave functions with Jastrow. Why this is the case, is hard to explain. It could be something wrong in our code, but it could also be due to less accurate orbitals, too few MC cycles ($N = 10^7$), or non-optimal variational parameters.

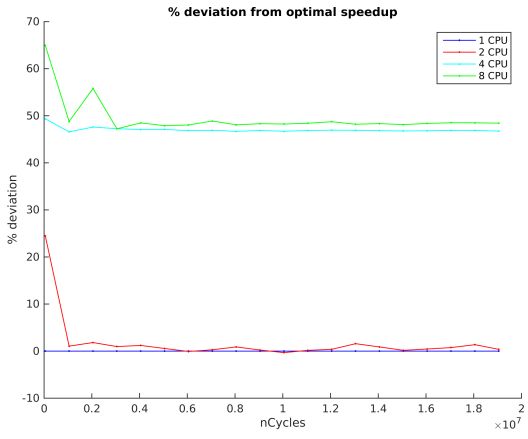
4.9 Speedup due to parallelization

If we now go over to the parallelization of the code, we can investigate the speedup using more processors. The computer used in this project have a quad Core (4 CPUs) with support of hyperthreading (additional 4 virtual CPUs). If we try with 1, 2, 4 and 8 processors, we can se how long time the program use to run a given number of MC cycles, and compare the time to the time spend when using only one processor. If we divide the time with one CPU by the time used with more CPUs, we get how many times faster the program runs.

Theoretically, the optimal speedup scales with number of cores used in the simulation. But there are many factors that can slow down the speedup. In our case, we are lucky, because there are very little overhead due to communication between processors. There are still limitations due to memory speed and memory bandwidth [23], and when using 8 CPUs, we are actually using 4 CPUs with multithreading. This could slow down our program a bit. The results are given below:



(a) Relative speedup between equal simulations with different number of processors.



(b) Deviation in % from theoretical optimal speedup using different number of processors.

Figure 17: The two figures above show the speedup using different number of GPSs.

We see an almost optimal speedup when using 2 CPUs. When using 4 CPUs, we actually get a very poor speedup compared to using 2 CPUs, and almost 50% from optimal theoretical speedup. This can be due tue how the operating system choose processors (in this case Ubuntu 14.04). It is possible that we are using 2 physical cores and 2 virtual cores with hyperthreading, with a limitation of shared memory. We have the same effect with 8 CPUs, with a speedup almost 50% from optimal theoretical speedup. Therefore, the hyperthreading seems to give poor improvement on the computational efficiency, but the overhead due to communication between prosessors have very little cost. We can only observe slow down for a very few MC cycles, as shown in the two plots above. MC simulations are very easy to parallelize with high gain in performance.

4.10 Summary of observations

A summary of runs for different systems (with hydrogenic wave functions and Jastrow factor) is given in the table below. *Relative error* (rel. error) is given as: $\frac{|E_0 - E_{VMC}|}{|E_{VMC}|}$ [16].

System	α	β	σ	E_{VMC}	$ref.\,E_0$	$rel.\,error$	$mean\,r_{ij}$	N
He	1.85	0.259	3.1×10^{-5}	-2.8901	-2.9037	0.47 %	1.3912	10^7
Be	3.9	0.206	3.5×10^{-4}	-14.4967	-14.667	1.17 %	2.1209	10^7
Ne	10.2	0.098	2.0×10^{-2}	-127.985	-128.94	0.75 %	1.1620	10^7
H_2	1.356	0.276	9.9×10^{-6}	-1.1575	-1.1746	1.48 %	2.1792	10^7
Be_2	3.725*	0.246*	1.9×10^{-3}	-28.765	-29.339	2.00 %	4.1245	10^7

Table 9: A summary of quantities obtained for different systems from MC simulations.

* These values are borrowed from Morten Ledum and Håvard Tveit Ihle's report in 2013 [23], since there was little time to search for the optimal value. In fact we shouldn't need to find the optimal α since we planned to use GTO basis sets on molecules as well. The steepst decent method was also unstable for Be_2 making it hard to find a good value for β, but experiments indicates that the value of β not have a huge effect on the etimate within a certain range (e. g. 0.001 to 20). In addition we also had to find the optimal R_ρ for Be_2, and also this value was borrowed in the estimation of the energy, with $R_\rho = 4.63$.

5 Conclusions and perspectives

Our main goal in this project was to develop a program that was able to estimate the ground state of some atoms and simple diatomic molecules. We have been able to reproduce the ground state energy within a relative error between 0.5 % and 2 % for all systems compared to tabulated reference values. Including a correlation factor in our trial wave function made a big improvement to our energy estimates, leading to a wider spread in the electron distribution plots. For all atoms and molecules in this project, we were able to reproduce the expected system configuration with plots of onebody desity and radial electron distribution. We saw how importance sampling

reduced the variance in the computations leading to less variance in the ground state energy. It also turned out that you have to put a lot of effort in finding good variational parameters to optimize the estimate of the ground state. We were also able to speed up the program a lot by using optimized expression to calculate gradients and Laplacians for the Slater gradient and the correlation factor.

The most critical part of this project was the steepest decent method used to find the optimal value for β. For Ne and Be_2 it was unstable, and we had to try out different step lengths and number of MC cycles. Still because of randomization in intial positions, the algorithm could fail finding an optimal value. Increasing MC cycles could make it more stable, but then we will use a lot of CPU time, and the advantage of a minimization algorithm is falling apart.

From the results we observe very little variation in our energy estimates, even with pretty few MC cycles. This indicates that even if we had performed, let's say 10^{10} MC cycles, we still would't get much closer to the reference value. This is due to lack of exact variational parameters. In addition our trial wave functions are not perfect, but a good GTO basis sets would propably lead us closer to the reference value. Unfortunately we didn't get better estimates with 3-21G basis set, and are unsure what's the reason for this is.

The most interesting part of the project was the plots of onebody density and radial electron distribution. It was facinating that our program was able to reproduce known patterns in the atom and molecule structures. We could cleary distinguish different orbitals in our atoms, and for Be_2 we could see how two nucleons were surrounded by electrons. By looking at such physical plots, we get a far better understanding of what we actually are simulating, than we get by just looking at wave functions and hamiltonians.

Developing code in this project have been challenging due tue a lot of indexing and need for updating in the MC procedure. Especially the implementation of the optimized machinery for computing gradients and Laplacian of the wave function was very hard to get through without making any errors. It was also challenging to make the steepst decent method to work ok, but still it's not optimal to work with.

Further improvements are many. First of all we could run simulations with many more MC cycles than 10^7, but then the blocking using Matlab tok really long time before we eventually ran out of RAM on the computer. We should also work more on GTOs and try out different basis sets to improve the energy estimates. Here we hopefully get better trial wave functions and at the same time get rid of the tedious search after an optimal variational parameter. We should also really consider using another minimization algorithm than steepst decent, since it behaves unstable for bigger systems. *Conjugate gradient method* [21] is an option, but we should also consider looking for a method like *Newtons method* [22] with adaptive step length. In addtion we could write our program in a way that makes us able to simulate other atoms and molecules, not only special cases with filled up orbitals.

6 Code implementations

6.1 Calculate derivatives with sympy (symbolic python)

```
import sympy as sp

def LocalEnergy(phi):
    return  ((-sp.diff(phi, x1, 2) - sp.diff(phi, y1, 2) - sp.diff(phi, z1,2) +          (- sp.
        diff(phi, x2, 2) - sp.diff(phi, y2, 2) - sp.diff(phi, z2, 2)))/(2*phi) -        Z/r1 - Z
        /r2 + 1 /r12)

# make variables symbolic
x1, x2, y1, y2, z1, z2, alpha, Z = sp.symbols('x1, x2, y1, y2, z1, z2, alpha, Z')

# creates a symbolic equivalent of r1 and r2
R1, R2, R12 = sp.symbols('r1 r2 r12')
r1 = sp.sqrt(x1*x1 + y1*y1 + z1*z1) r2 = sp.sqrt(x2*x2 + y2*y2 + z2*z2) r12 = sp.sqrt((x1 - x2)**2
     + (y1 - y2)**2 + (z1 - z2)**2)

# our wavefunction
phi1 = sp.exp(-alpha*(r1 + r2))

# compute the local energy
local_energy = LocalEnergy(phi1)

#Simplify the expression local_energy = local_energy.subs(r12,R12).factor().subs(r1, R1).subs(r1
    **2,R1**2).subs(r2,R2).subs(r2**2,R2**2).simplify().collect(Z).collect(alpha)

# Print out analytical expression for the local energy as latex and c code
print sp.printing.latex(local_energy)
print sp.printing.ccode(local_energy)
```

6.2 Implement efficient Slater determinant ratio

```
// compute the R_sd ratio
void VMCSolver::compute_R_sd(int i){
    R_sd = 0.0;
    if(i < nParticles/2){
        for(int j=0; j<nParticles/2; j++){
            R_sd += SlaterPsi(rNew, i, j)*D_up_old(j, i);
        }
    }
    else{
        for(int j=0; j<nParticles/2; j++){
            R_sd += SlaterPsi(rNew, i, j)*D_down_old(j, i-nParticles/2);
        }
    }
}
```

6.3 Implement efficient gradient and laplacian of trial wave function

```
// compute slater first derivative
void VMCSolver::SlaterGradient(int i){
    compute_R_sd(i);
    if(i < nParticles/2){
        for(int k=0; k<nDimensions; k++){
            double derivative_up = 0.0;
            for(int j=0; j<nParticles/2; j++){
                derivative_up += Psi_first_derivative(i, j, k)*D_up_old(j, i);
            }
            SlaterGradientNew(i, k) = (1.0/R_sd)*derivative_up;
        }
    }
    else{
        for(int k=0; k<nDimensions; k++){
            double derivative_down = 0.0;
                for(int j=0; j<nParticles/2; j++){
                    derivative_down += Psi_first_derivative(i, j, k)*D_down_old(j, i-
                        nParticles/2);
                }
                SlaterGradientNew(i, k) = (1.0/R_sd)*derivative_down;
            }
        }
    }

// compute slater second derivative
double VMCSolver::SlaterLaplacian(){
    double derivative_up = 0.0;
    double derivative_down = 0.0;
    for(int i=0; i<nParticles/2; i++){
        for(int j=0; j<nParticles/2; j++){
            derivative_up += Psi_second_derivative(i, j)*D_up_new(j, i);
                        derivative_down += Psi_second_derivative(i+nParticles/2, j)*
                    D_down_new(j, i);
        }
    }
    double derivative_sum = derivative_up + derivative_down;
    return derivative_sum;
}
```

6.4 Implement efficient way of updating the Slater matrix

```
// update Slater matrix
```

```
void VMCSolver::update_D(mat& D_new, const mat& D_old, int i, int selector){
    i = i - nParticles/2*selector;
    for(int k=0; k<nParticles/2; k++){
        for(int j=0; j<nParticles/2; j++){
            if(j!=i){
                double sum = 0;
                for(int l=0; l<nParticles/2; l++){
                    sum += SlaterPsi(rNew, i+nParticles/2*selector, l)*D_old(l,j);
                }
                D_new(k,j) = D_old(k,j) - D_old(k, i)*sum/R_sd;
            }
            else{
                D_new(k,j) = D_old(k, i)/R_sd;
            }
        }
    }
}
```

6.5 Implement efficient Jastrow factor ratio

```
// compute C matrix
void VMCSolver::fillJastrowMatrix(mat &CorrelationMatrix){
    for(int k=0; k < nParticles; k++){
        for(int i=k+1; i < nParticles; i++){
            CorrelationMatrix(k,i) = a_matrix(k,i)*r_distance(k,i)/(1.0 + beta*
                r_distance(k,i));
        }
    }
}

// compute R_c ratio
void VMCSolver::compute_R_c(){
    double deltaU = 0.0;
    for(int k=0; k < nParticles; k++){
        for(int i=0; i < k; i++){
            deltaU += C_new(i, k) - C_old(i, k);
        }
        for(int i=k+1; i < nParticles; i++){
            deltaU += C_new(k, i) - C_old(k, i);
        }
    }
    R_c = exp(deltaU);
}
```

6.6 Implement Quantum force

```
// compute the quantum force used in importance sampling
void VMCSolver::QuantumForce(const mat &r, mat &F){
    GradientSquared = 0.0;
    for(int k=0; k<nParticles; k++){
        for(int j=0; j<nDimensions; j++){
            if(activate_JastrowFactor){
                double GradientSum = 0.0;
                for(int i=0; i<k; i++){
                    GradientSum += (r(k,j)-r(i,j))/r_distance(i,k)*JastrowDerivative(i,k
                        );
                }
                for(int i=k+1; i<nParticles; i++){
                    GradientSum -= (r(i,j)-r(k,j))/r_distance(k,i)*JastrowDerivative(k,i
                        );
                }
```

```
                F(k,j) =  2.0*(SlaterGradientNew(k,j) + GradientSum);
                GradientSquared += GradientSum*GradientSum;
                JastrowGradient(k, j) = GradientSum;
            }
            else{
                F(k,j) =  2.0*SlaterGradientNew(k,j);
            }
        }
    }
}
```

6.7 Implement efficient gradient and laplacian of Jastrow factor

```
// compute derivatives used in gradient
void VMCSolver::computeJastrowDerivative(int k){
    for(int i=0; i<k; i++){
        double divisor = 1.0 + beta*r_distance(i, k);
        JastrowDerivative(i, k) = a_matrix(i, k)/(divisor*divisor);
    }
    for(int i=k+1; i<nParticles; i++){
        double divisor = 1.0 + beta*r_distance(k, i);
        JastrowDerivative(k, i) = a_matrix(k, i)/(divisor*divisor);
    }
}

// compute double derivatives used in Jastrow laplacian
void VMCSolver::computeJastrowLaplacian(int k){
    for(int i=0; i<k; i++){
        double divisor = 1.0 + beta*r_distance(i, k);
        JastrowLaplacianNew(i, k) = -2.0*a_matrix(i, k)*beta/(divisor*divisor*divisor);
    }
    for(int i=k+1; i<nParticles; i++){
        double divisor = 1.0 + beta*r_distance(k, i);
        JastrowLaplacianNew(k, i) = -2.0*a_matrix(k, i)*beta/(divisor*divisor*divisor);
    }
}

// compute Jasstrow energy and energy crossterm
double VMCSolver::computeJastrowEnergy(){
    double sum = 0.0;
    energytermSlaterJastrow = 0.0;
    sum += GradientSquared;
    for(int k=0; k<nParticles; k++){
        for(int i=0; i<k; i++) {
            sum += (nDimensions - 1)/r_distance(i, k)*JastrowDerivative(i, k) +
                    JastrowLaplacianNew(i, k);
        }
        for(int i=k+1; i<nParticles; i++) {
            sum += (nDimensions - 1)/r_distance(k, i)*JastrowDerivative(k, i) +
                JastrowLaplacianNew(k, i);
        }
    }
    for(int k=0; k<nDimensions; k++){
        energytermSlaterJastrow += dot(SlaterGradientNew.col(k), JastrowGradient.col(k))
            ;
    }
    return sum;
}
```

6.8 Brute force implementation of GTO

```
// Called for setting up Slater determinant with GTO double VMCSolver::SlaterPsi(int particle, int orb){
    double psi = 0.0;
    if(AtomType=="He"){
            double alpha1 = GTO_values(0, 0);
            double alpha2 = GTO_values(1, 0);
            double alpha3 = GTO_values(2, 0);

            double c1 = GTO_values(0, 1);
            double c2 = GTO_values(1, 1);
            double c3 = GTO_values(2, 1);

            double K1 = GTO_coef_values(0, 0);
            double K2 = GTO_coef_values(1, 0);

            psi = K1*(c1*G_func(alpha1, particle, 0, 0, 0))
                + K2*(c2*G_func(alpha2, particle, 0, 0, 0) + c3*G_func(alpha3, particle, 0, 0, 0))
                    ;
        }
    else{
        int Kp_set;
        Kp_set = orb;

        double alpha1 = GTO_values(0, 0);
        double alpha2 = GTO_values(1, 0);
        double alpha3 = GTO_values(2, 0);
        double alpha4 = GTO_values(3, 0);
        double alpha5 = GTO_values(4, 0);
        double alpha6 = GTO_values(5, 0);
        double alpha7 = GTO_values(6, 0);
        double alpha8 = GTO_values(7, 0);
        double alpha9 = GTO_values(8, 0);

        double c1 = GTO_values(0, 1);
        double c2 = GTO_values(1, 1);
        double c3 = GTO_values(2, 1);
        double c4 = GTO_values(3, 1);
        double c5 = GTO_values(4, 1);
        double c6 = GTO_values(5, 1);
        double c7 = GTO_values(6, 1);
        double c8 = GTO_values(7, 1);
        double c9 = GTO_values(8, 1);

        double K1 = GTO_coef_values(0, Kp_set);
        double K2 = GTO_coef_values(1, Kp_set);
        double K3 = GTO_coef_values(2, Kp_set);
        double K4 = GTO_coef_values(3, Kp_set);
        double K5 = GTO_coef_values(4, Kp_set);
        double K6 = GTO_coef_values(5, Kp_set);
        double K7 = GTO_coef_values(6, Kp_set);
        double K8 = GTO_coef_values(7, Kp_set);
        double K9 = GTO_coef_values(8, Kp_set);

        psi = K1*(c1*G_func(alpha1, particle, 0, 0, 0) + c2*G_func(alpha2, particle, 0, 0, 0) + c3
            *G_func(alpha3, particle, 0, 0, 0))
        + K2*(c4*G_func(alpha4, particle, 0, 0, 0) + c5*G_func(alpha5, particle, 0, 0, 0))
        + K3*(c6*G_func(alpha6, particle, 0, 0, 0))
        + K4*(c7*G_func(alpha7, particle, 1, 0, 0) + c8*G_func(alpha8, particle, 1, 0, 0))
        + K5*(c9*G_func(alpha9, particle, 1, 0, 0))
        + K6*(c7*G_func(alpha7, particle, 0, 1, 0) + c8*G_func(alpha8, particle, 0, 1, 0))
        + K7*(c9*G_func(alpha9, particle, 0, 1, 0))
        + K8*(c7*G_func(alpha7, particle, 0, 0, 1) + c8*G_func(alpha8, particle, 0, 0, 1))
        + K9*(c9*G_func(alpha9, particle, 0, 0, 1));
    }
    return psi;
}

// Compute G in GTO orbitals for given parameters
double VMCSolver::G_func(double GTO_alpha, int particle, int i, int j, int k){
    double x, y, z, r, N;
    x = rNew(particle, 0);
    y = rNew(particle, 1);
    z = rNew(particle, 2);
```

```
    N = Normalization_factor(GTO_alpha, i, j, k);
    double G = N*pow(x, i)*pow(y, j)*pow(z, k)*exp(-GTO_alpha*(x*x + y*y + z*z));
    return G;
}

// General expression for computing the normalization factor in GTO orbitals
double VMCSolver::Normalization_factor(double GTO_alpha, int i, int j, int k){
    int fac_i = factorial_func(i);
    int fac_j = factorial_func(j);
    int fac_k = factorial_func(k);
    int fac_2i = factorial_func(2*i);
    int fac_2j = factorial_func(2*j);
    int fac_2k = factorial_func(2*k);
    double frac_over = pow(8.0*GTO_alpha, i+j+k) * (fac_i*fac_j*fac_k);
    double frac_under = fac_2i*fac_2j*fac_2k;
    double N = pow(((2.0*GTO_alpha)/pi), (3.0/4.0)) * sqrt(frac_over/frac_under);
    return N;
}

// Computes factorial of a given number
double VMCSolver::factorial_func(int number){  int factorial = 1;
    for(int i=2; i<=number; i++){
        factorial *= i;
    }
    return factorial;
}
```

6.9 Matlab code to perform blocking on data set

```
for i=1:block_trials
    blocks = data_size/(i);
    energy_trial_sum = 0;
    energySquared_trial_sum = 0;
    for j=1:blocks
        block_size = data_size/blocks;
        index_start = floor(1 + (j-1)*block_size);
        index_stop = floor(j*block_size);
        n = (index_stop - index_start) + 1;

        energy_mean_block = sum(energy(index_start:index_stop))/n;
        energy_trial_sum = energy_trial_sum + energy_mean_block;
        energySquared_trial_sum = energySquared_trial_sum+ energy_mean_block*energy_mean_block;
    end

    energy_mean_blockSize = energy_trial_sum/blocks;
    energySquared_mean_blockSize = energySquared_trial_sum/blocks;
    variance_mean_blockSize = (energySquared_mean_blockSize - (energy_mean_blockSize*
        energy_mean_blockSize))/blocks;
    block_size_values(i) = floor(block_size);
    block_size_variance(i) = variance_mean_blockSize;
end

SD = sqrt(block_size_variance);
plot(block_size_values, SD)
```

7 Bibliography

References

[1] M. Hjorth-Jensen, *Variational Monte Carlo methods*, lecture notes in FYS4411, University of Oslo, (2015).

[2] M. Hjort-Jensen, *Computational Physics*, lectures notes in FYS3150, University of Oslo, (2014).

[3] D. J. Griffiths, *Introduction to Quantum Mechanics*, Second Edition, (2005).

[4] H. T. Ihle and M. Ledum, Variational Monte Carlo method project FYS4411, University of Oslo, (2013).

[5] O. T. Norli, *Coupled Cluster Studies in Computational Chemistry*, Master Thesis University of Oslo, (2014), Chapter 4.

[6] S. A. Dragly, *Bridging Quantum Mechanics and Molecular Dynamics with Artificial Neural Networks*, Master Thesis University of Oslo, (2014), Section 3.5-3.6.

[7] Gaussian Type Orbitals, 3-21G Basis Sets, https://bse.pnl.gov/bse/portal, (2015).

[8] M. Hjorth-Jensen, *Conjugate gradient methods and other optimization methods*, lecture notes in FYS4411, University of Oslo, (2015).

[9] B. L. Hammond, W. A. Lester and P. J. Reynolds, *Monte Carlo methods in Ab Inition Quantum Chemistry*, World Scientific, Singapore, 1994, chapters 2-5 and appendix B.

[10] B. H. Bransden and C. J. Joachain, *Physics of Atoms and molecules*, Longman, 1986. Chapters 6, 7 and 9.

[11] S. A. Alexander and R. L. Coldwell, Int. Journal of Quantum Chemistry, **63** (1997) 1001.

[12] C. J. Umrigar, K. G. Wilson and J. W. Wilkins, Phys. Rev. Lett. **60** (1988) 1719.

[13] Moskowith and Kalos, Int. Journal of Quantum Chemistry, 1107 (1981). Results for He and H_2.

[14] E. Buendia et al., J. Chem. Phys. 131, (2009). Results for Be and Ne.

[15] Filippi Singh and Umringar, J. Chemical Physics **105**,123 (1996). Results on Be_2.

[16] Relative error: [http://mathworld.wolfram.com/RelativeError.html||http://mathworld.wolfram.com/RelativeError.html]

[17] Orbital illustrations, http://catalog.flatworldknowledge.com/bookhub/4309?e=averill_1.0-ch06_s05#, (June 10, 2015).

[18] Coulomb potential, http://cps-www.bu.edu/Wasser/robert/work/node9.html, (June 10, 2015).

[19] History of the discovery of atoms, http://www.nobeliefs.com/atom.htm, (June 10, 2015).

[20] Steepes decent algorithm, http://en.wikipedia.org/wiki/Gradient_descent, (June 10, 2015).

[21] M. Hjorth-Jensen, *Conjugate gradient methods and other optimization methods*, lecture notes in FYS4411, University of Oslo, (2015).

[22] Newtons Method algorithm, http://en.wikipedia.org/wiki/Newton%27s_method, (June 10, 2015).

[23] Memory Bandwidth: http://en.wikipedia.org/wiki/Memory_bandwidth, (June 10, 2015).

[24] Armadillo, C++ linear algebra library: http://arma.sourceforge.net/docs.html