

# FYS4411

## Variational Monte Carlo simulations of atoms and molecules

Stian Roti Svendby

June 12, 2015

### Contents

<b>1 Abstract</b>	<b>2</b>
<b>2 Introduction</b>	<b>2</b>
<b>3 Theory and methods</b>	<b>3</b>
3.1 Quantum mechanic eigenvalue equation . . . . .	3
3.2 Hydrogenic wave functions . . . . .	3
3.3 The helium atom and general atom hamiltonian . . . . .	4
3.4 Slater determinants . . . . .	5
3.5 The Jastrow factor . . . . .	6
3.6 Local energy and implementation . . . . .	6
3.7 Cusp conditions . . . . .	7
3.8 Monte Carlo simulatons and The Metropolis algorithm . . . . .	8
3.9 Energy, gradients and Laplacians for atoms . . . . .	9
3.10 Optimization of the machinery . . . . .	10
3.11 Variational parameters . . . . .	12
3.12 Derivatives of the energy and Steepest descent . . . . .	13
3.13 GTO orbitals . . . . .	14
3.14 Implementation of molecules and modification of our program . . . . .	15
3.15 Blocking . . . . .	16
3.16 Verifications . . . . .	17
3.17 Parallelization of the Variational Monte Carlo solver . . . . .	17
3.18 Monte Carlo solver . . . . .	19
<b>4 Results and discussion</b>	<b>20</b>
4.1 Variational parameters and ground state energy for atoms . . . . .	20
4.2 Variance in Monte Carlo simulations . . . . .	21
4.3 Speed against different solvers . . . . .	22
4.4 Optimization with Jastrow factor and importance sampling . . . . .	22
4.5 Finding the real variance with Blocking . . . . .	25
4.6 Onebody density and charge density . . . . .	26
4.7 Molecules . . . . .	28
4.8 A note on GTO in this project . . . . .	31
4.9 Speedup due to parallelization . . . . .	31
4.10 Summary of observations . . . . .	32
<b>5 Conclusions and perspectives</b>	<b>32</b>
<b>6 Code implementations</b>	<b>33</b>
6.1 Calculate derivatives with sympy (symbolic python) . . . . .	33
6.2 Implement efficient Slater determinant ratio . . . . .	33
6.3 Implement efficient gradient and laplacian of trial wave function . . . . .	34
6.4 Implement efficient way of updating the Slater matrix . . . . .	34
6.5 Implement efficient Jastrow factor ratio . . . . .	35
6.6 Implement Quantum force . . . . .	35

6.7	Implement efficient gradient and laplacian of Jastrow factor . . . . .	36
6.8	Brute force implementation of GTO . . . . .	36
6.9	Matlab code to perform blocking on data set . . . . .	38
<b>7</b>	<b>Bibliography</b>	<b>38</b>

## 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 (applied 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  $\alpha$  and  $\beta$  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 [17]. 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 laboratory 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 laboratory experiments. We will actually compare our numerical results with values obtained at laboratory 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.

*Variational Monte Carlo* (VMC or just MC) methods are based on stochastic processes, where we have probabilities of rejection or acceptance of a transition. For small systems at atomic level, . We will perform MC simulations using *Metropolis algorithm* with wave functions 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 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 can find a lot of code examples on how different algoritms is implemented.

### 3 Theory and methods

#### 3.1 Quantum mechanic eigenvalue equation

In order to investigate systems on 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 Schrödinger equation* (SE) to compute the expectation value of the energy, given as [1]:

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

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

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

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

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

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 wavefunctions 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 describe 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:

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

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  $\alpha$  is a *variational parameter*. With  $\alpha = 1$ , the wave function reproduces the exact solution. With this in mind, we construct a (trial) wave function for helium (He) with two surrounding electrons as the product of two hydrogenic wave functions:

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

Here we have two electron distances to the nucleus, and  $\alpha$  has to determine in a way that minimizes the energy. We see that our trial wave functions are constructed with two single-particle wave functions  $\psi_i(\mathbf{r}_i, \alpha)$ . 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  $r_{12} = |\mathbf{r}_1 - \mathbf{r}_2|$  which is the electron-electron distance, making us able to include the effect of electron repulsion. This together with another variational parameter  $\beta$  (that we also have to determine), hopefully gives us a better trial wave function given as:

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

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 *Pauli's exclusion principle* [3], which states that the total wave function has to be anti-symmetric. For He we have that the total wave function has to fulfill:

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

We assume that one particle is in state  $\psi_a$  and that the other particle is in state  $\psi_b$ . 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 ground 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 electrons in the first orbital, so we need more of them. The general formula for hydrogen orbitals can be found in Griffiths on page 152 [3], but we don't bother

writing it down here since it look 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:

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

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

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]:

$m_l \setminus l$	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, 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 enough 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:

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

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 nucleous, 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:

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

We notice that we have dimensionless *Coulomb potential* [16] given as  $V_c = \frac{q_1 q_2}{r}$ , so with electron charge  $q = 1$  and  $Z = \text{nucleous charge}$  we get:

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

This is the general hamiltonian we use for atoms.

### 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 electrons in the lowest orbital. In our project we use wave functions for Be on the form:

$$\psi_T(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \alpha, \beta) = \text{Det}(\psi_1(\mathbf{r}_1, \alpha)\psi_2(\mathbf{r}_2, \alpha)\psi_3(\mathbf{r}_3, \alpha)\psi_4(\mathbf{r}_4, \alpha)) \prod_{i < j}^4 e^{\frac{\alpha_{ij} r_{ij}}{1+\beta r_{ij}}}$$

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  $\psi_i(\mathbf{r}_i, \alpha) = \psi_i(\mathbf{r}_i)$ . The Slater determinant is given by:

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

For  $\psi_i$  we use the hydrogen orbitals:

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

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

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

$$\begin{aligned} \text{det } \uparrow (1, 2) &= \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) \end{vmatrix} \\ \text{det } \downarrow (3, 4) &= \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_3(\mathbf{r}_2) & \psi_4(\mathbf{r}_3) \\ \psi_3(\mathbf{r}_4) & \psi_4(\mathbf{r}_4) \end{vmatrix} \end{aligned}$$

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

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

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

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

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 following code, with *argument* as a vector with the four instantaneous radial distances between the nucleus and the electrons:

```

1 double VMCsolver::SlaterDeterminant(){
2     vec argument = zeros(nParticles);
3     argument = r_centre;
4     double wf = (psi1s(argument(0))*psi2s(argument(1))-psi1s(argument(1))*psi2s(argument(0)))*(
5         psi1s(argument(2))*psi2s(argument(3))-psi1s(argument(3))*psi2s(argument(2)));
6     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:

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

We should notice that this ansatz is not antisymmetric 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 repulsion. 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 spinn and  $a_{ij} = \frac{1}{2}$  if opposite spin [REF]. 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:

```

1 double VMCsolver::JastrowFactor(){
2     double Psi = 1.0;
3     for(int j=0; j < nParticles; j++){
4         for(int i=0; i < j; i++){
5             Psi *= exp((a_matrix(i, j)*r_distance(i, j))/(1.0 + beta*r_distance(i, j)));
6         }
7     }
8     return Psi;
}

```

### 3.6 Local energy and implementation

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

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

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

$$\begin{aligned} E_L(\mathbf{R}_i, \alpha, \beta) &= \frac{1}{\psi_T(\mathbf{R}, \alpha, \beta)} \hat{H} \psi_T(\mathbf{R}, \alpha, \beta) \\ &= -\frac{1}{2\psi_T(\mathbf{R}, \alpha, \beta)} \nabla_1^2 \psi_T(\mathbf{R}, \alpha, \beta) - \frac{1}{2\psi_T(\mathbf{R}, \alpha, \beta)} \nabla_2^2 \psi_T(\mathbf{R}, \alpha, \beta) - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}} \end{aligned}$$

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

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

The estimate is actually a MC simulation to solve the above 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:

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

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

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. To handle this problem we will later introduce optimized calculations to take care this latter.

### 3.7 Cusp conditions

We need our wave function to satisfy the *cusp condition* when  $r_1 \rightarrow 0$  or  $r_2 \rightarrow 0$  or  $r_{12} \rightarrow 0$ . 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:

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

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

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

For small values of  $r$ , the dominating terms is

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

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

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

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

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

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

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

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

, which the Jastrow factor fulfills.

### 3.8 Monte Carlo simulations 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  $\delta$  is a chosen steplength. In our program suggested moves are done by the code:

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

, where *steplength* is a predefined value chosen 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. 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:

```
1 if(ran2(&idum) <= (waveFunctionNew * waveFunctionNew) / (waveFunctionOld * waveFunctionOld)) {
2     for(int j = 0; j < nDimensions; j++) {
3         rOld(i, j) = rNew(i, j);
4         waveFunctionOld = waveFunctionNew;
5     }
6 }
7 else {
8     for(int j = 0; j < nDimensions; j++) {
9         rNew(i, j) = rOld(i, j);
10    }
```

We see that if the new configuration gives lower energy, there will be a probability of 0.5 (for the correct steplength) 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  $\xi$  is a gaussian probability distribution. The quantum force is given by:

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

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:

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

*GaussianDeviate* is the gaussian probability distribution, *timestep* control the steplength 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 without *quantum force* applied, we switch to Metropolis-Hastings algorithm 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:

```

1 GreensFunction = 0.0;
2 for(int j=0; j < nDimensions; j++){
3     GreensFunction += 0.5*(QForceOld(i, j)+QForceNew(i, j))*           (D*timestep
4         *0.5*(QForceOld(i, j)-QForceNew(i, j))-rNew(i, j)+rOld(i, j));
5     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-Hastings algorithm):

```

1 if(ran2(&idum)<=GreensFunction*(waveFunctionNew*waveFunctionNew)/(waveFunctionOld*waveFunctionOld)){
2     for(int j = 0; j < nDimensions; j++){
3         rOld(i, j) = rNew(i, j);
4         QForceOld(i, j) = QForceNew(i, j);
5         waveFunctionOld = waveFunctionNew;
6     }
7 } else{
8     for(int j = 0; j < nDimensions; j++){
9         rNew(i, j) = rOld(i, j);
10        QForceNew(i, j) = QForceOld(i, j);
11    }
12 }
}

```

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:

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

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:

	$\psi$	$\frac{\partial \psi}{\partial x}$	$\frac{\partial \psi}{\partial y}$	$\frac{\partial \psi}{\partial z}$	$\nabla^2 \psi$
$\psi_{1s}$	$e^{-\alpha r}$	$-\frac{\alpha}{r} x e^{-\alpha r}$	$-\frac{\alpha}{r} y e^{-\alpha r}$	$-\frac{\alpha}{r} z e^{-\alpha r}$	$\frac{\alpha}{r} (\alpha r - 2) e^{-\alpha r}$
$\psi_{2s}$	$(1 - \frac{\alpha}{2} r) e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} x (\alpha r - 4) e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} y (\alpha r - 4) e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} z (\alpha r - 4) e^{-\frac{\alpha r}{2}}$	$-\frac{1}{8} \frac{\alpha}{r} ((\alpha r)^2 - 10\alpha r + 16) e^{-\frac{\alpha r}{2}}$
$\psi_{2px}$	$x e^{-\frac{\alpha r}{2}}$	$-\frac{1}{r} (\frac{1}{2} \alpha x^2 - r) e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} x y e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} x z e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} x (\alpha r - 8) e^{-\frac{\alpha r}{2}}$
$\psi_{2py}$	$y e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} x y e^{-\frac{\alpha r}{2}}$	$-\frac{1}{r} (\frac{1}{2} \alpha y^2 - r) e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} y z e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} y (\alpha r - 8) e^{-\frac{\alpha r}{2}}$
$\psi_{2pz}$	$z e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} x z e^{-\frac{\alpha r}{2}}$	$-\frac{1}{2} \frac{\alpha}{r} y z e^{-\frac{\alpha r}{2}}$	$-\frac{1}{r} (\frac{1}{2} \alpha z^2 - r) e^{-\frac{\alpha r}{2}}$	$\frac{1}{4} \frac{\alpha}{r} z (\alpha r - 8) e^{-\frac{\alpha r}{2}}$

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  $N = \text{number of particles} \times d = \text{dimensions}$  independent coordinates, which means that we have to evaluate the entire Slater determinant  $N \times d$  times to calculate the local energy and  $N \times d$  times to calculate 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

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

, with  $\psi_j(\mathbf{r}_i)$  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 rows in the corresponding Slater determinant is changed, making it unnecessary to recompute the whole determinant. The solution 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:

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

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

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

or

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

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

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

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

Implementation can be found in “Code implementation 6.3”.

The update of the inverse matrix are done by

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

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

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

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. 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

$$\text{Det}(\psi_1(\mathbf{r}_1), \psi_2(\mathbf{r}_2), \dots, \psi_N(\mathbf{r}_N)) \propto \det \uparrow \det \downarrow$$

, 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:

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

, 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:

$$\text{if}(\text{rand num} \leq \text{GreensFunction} * R^2)$$

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:

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

The Laplacian needed for the local energy is given by

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

, 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 crossterm. 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, plus the potential energy due to interaction electron-electron and electron-nucleous interaction.

### 3.11 Variational parameters

We have mentioned the variational parameters  $\alpha$  and  $\beta$ , and told that we have to make an optimal choice 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:

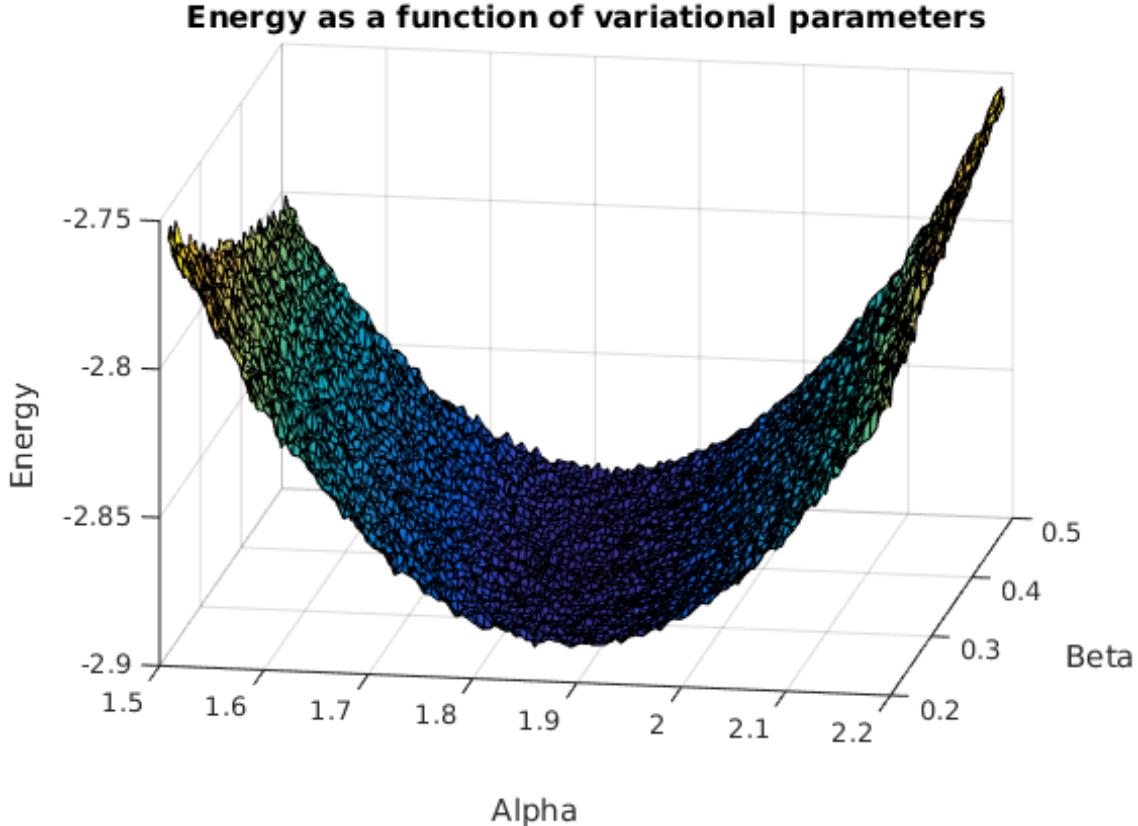


Figure 1: We can see which pair of  $\alpha$  and  $\beta$  that minimizes the energy for the Helium atom with Jastrow factor.

As we can see from the plot, the variation in  $\alpha$  have a lot more influence on the energy than  $\beta$ . But we have a problem. Since the estimated ground state energy change for each MC simulation even for the same parameters, it is hard to make a proper choice of parameters. We also have to choose between discrete points on the mesh, and making the mesh finer cost very much CPU. In our case it works fair enough to make a 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  $\beta$  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  $\beta$ , 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:

```

1 // compute energy of beta derivative of Jastrow factor
2 double VMCsolver::beta_derivative_Jastrow(){
3     double derivative_sum = 0.0;
4     for(int j=0; j<nParticles; j++){
5         for(int i=0; i<j; i++){
6             double divisor = (1.0 + beta*r_distance(i,j));
7             derivative_sum += (-a_matrix(i,j)*r_distance(i,j)*r_distance(i,j))/(divisor*divisor);
8         }
9     }
10    return derivative_sum;
11 }
```

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  $\beta$  is *Steepest descent* [18], 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  $\beta$ , compute  $\frac{\partial E}{\partial \beta}$ , and take a step in the opposite direction of this. This is much better explained by some piece of code:

```

1 // steepest decent optimization algorithm to find optimal beta
2 void VMCsolver::findOptimalBeta(int my_rank, int world_size){
3     fstream outfile;
4     int nCycles = 100000;
5     int max_iter = 500;
6     double step = 0.01;
7     const double precision = 1.0e-4;
8     beta = 0.25;
9     double beta_new = beta;
10    double beta_old = 0.0;
11    int counter = 0;
12    while(abs(beta_new - beta_old) > precision) {
13        MonteCarloIntegration(nCycles, outfile, my_rank, world_size);
14        beta_old = beta_new;
15        beta_new = beta_old - step*E_betaDerivative;
16        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  $\alpha$ , what can we do with that? We could use a similar approach as with  $\beta$ , but a better solution is to use optimized GTO basis sets to get rid of  $\alpha$ . 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  $\alpha$  and corresponding coefficients  $c$ . We will now explain how the gaussian orbitals are constructed.

We define [5]:

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

Here  $i, j, k$  is the quantum numbers  $n, l, m_l$  that we look at earlier when we discussed hydrogen-like orbitals.  $\alpha$  is a value taken from the GTO basis, and  $r$  is the electron-nucleous 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:

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

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

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

$K_{\rho}$  are coefficient from *Hartee – Fock calculations*, and  $\sigma$  is an index telling how many terms we have to add up for each  $K_{\rho}$ . This explanation may be unclear, so we show what  $\psi$  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:

$\alpha_i$	$c_i$	$c_i$
$\alpha_1$	$c_1$	
$\alpha_2$	$c_2$	
$\alpha_3$	$c_3$	
$\alpha_4$	$c_4$	$c_7$
$\alpha_5$	$c_5$	$c_8$
$\alpha_6$	$c_6$	$c_9$

(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):

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

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

$$\begin{aligned} \psi_{Be}^b &= K_1^b \left( c_1 N_{0,0,0} e^{-\alpha_1 r^2} + c_2 N_{0,0,0} e^{-\alpha_2 r^2} + c_3 N_{0,0,0} e^{-\alpha_3 r^2} \right) \\ &+ K_2^b \left( c_4 N_{0,0,0} e^{-\alpha_4 r^2} + c_5 N_{0,0,0} e^{-\alpha_5 r^2} \right) + K_3^b \left( c_6 N_{0,0,0} e^{-\alpha_6 r^2} \right) \\ &+ K_4^b \left( c_7 N_{1,0,0} e^{-\alpha_7 r^2} + c_8 N_{1,0,0} e^{-\alpha_8 r^2} \right) + K_5^b \left( c_9 N_{1,0,0} e^{-\alpha_9 r^2} \right) \\ &+ K_6^b \left( c_7 N_{0,1,0} e^{-\alpha_{17} r^2} + c_8 N_{0,1,0} e^{-\alpha_{18} r^2} \right) + K_7^b \left( c_9 N_{0,1,0} e^{-\alpha_{19} r^2} \right) \\ &+ K_8^b \left( c_7 N_{0,0,1} e^{-\alpha_{27} r^2} + c_8 N_{0,0,1} e^{-\alpha_{28} r^2} \right) + K_9^b \left( c_9 N_{0,0,1} e^{-\alpha_{29} r^2} \right) \end{aligned}$$

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

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 like atoms) is:

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

Here we recognize  $r_{1i}$  and  $r_{2i}$  as the distance from electron  $i$  to the two nucleons, and  $R_\rho = \mathbf{r}_1 - \mathbf{r}_2$ , with  $\mathbf{r}_1$  and  $\mathbf{r}_2$  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  $Be_2$  [21].

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 call the distance between the nucleons (inter-proton distance)  $R_\rho$ , it's natural to place 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:

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

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

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

Our trial wave function for  $H_2$  is on the form:

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

, with:

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

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

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

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

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

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 we therefore have to set up new orbitals for molecules, but they look exactly like they did with hydrogenic orbitals only that we have to add up two functions with different electron-nucleon distance. The same yields for the gradient and the Laplacian, but we have to be careful with the derivation with respect to  $z$  since the nucleons is not placed in origo.

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  $\alpha$  as the intersection between them, leaving  $\beta$  as the only variational parameter. We have earlier presented the steepest decent method to pick out the value of  $\beta$  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 estimates 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  $\sigma$  and present the estimated energy as:

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

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

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

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

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

, where  $\tau$  is *correlation time* (time between uncorrelated samples) and  $\Delta t$  is time between samples. Unfortunately  $\tau$  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 “plateau” where the SD is independent of the block size, and the SD on this plateau we name  $\sigma_{block}$ , with the corresponding  $N_{blocksize}$  where the plateau starts. Then the actual  $\sigma$  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]:

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

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, can be done by removing the Slater part of the wave function, and compare 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_ncycles:" << 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

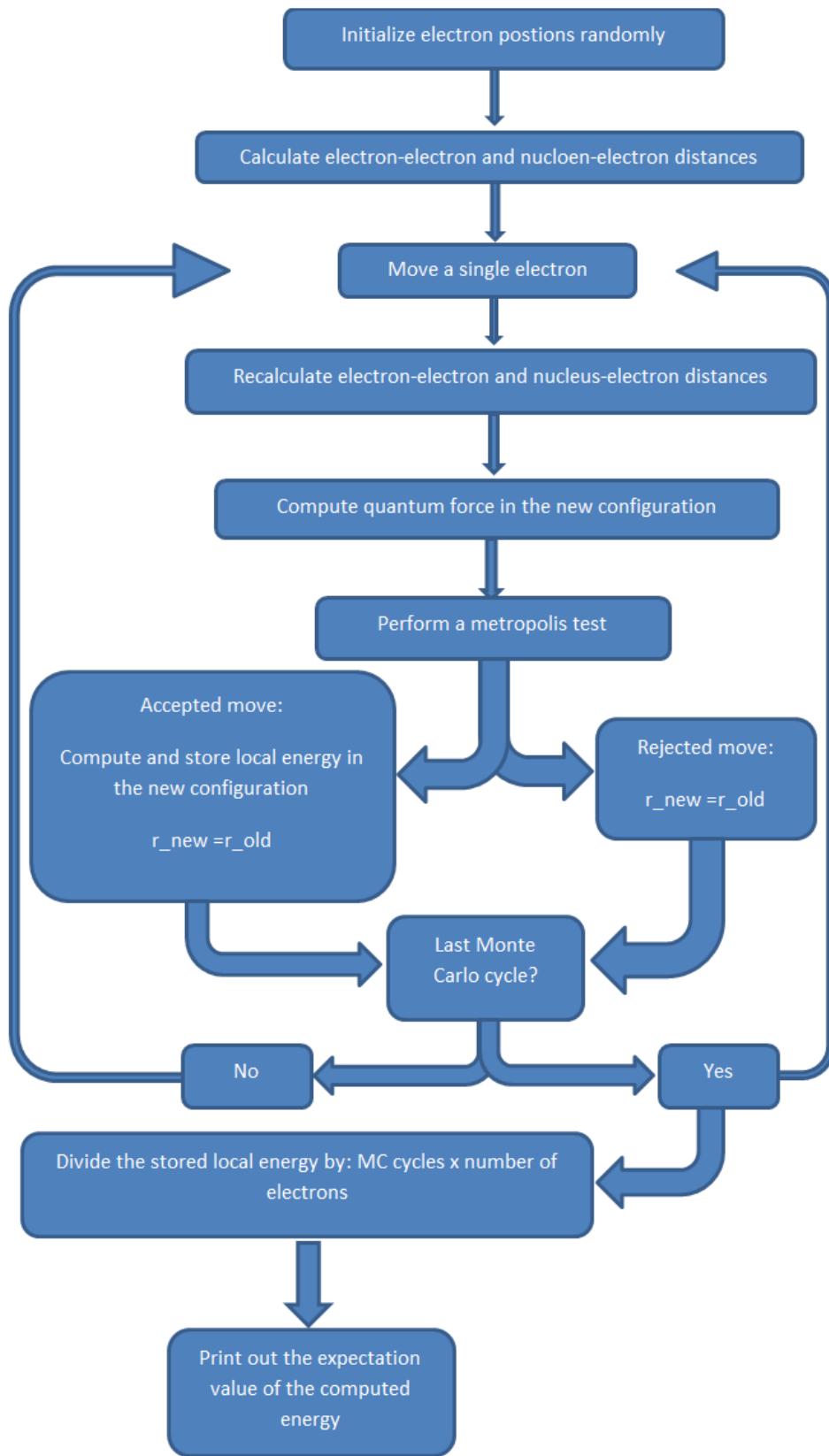


Figure 2: Flow chart for our Monte Carlo solver.

## 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  $\alpha$  for He, we could graphically obtain the optimal  $\alpha$  from Figure 3:

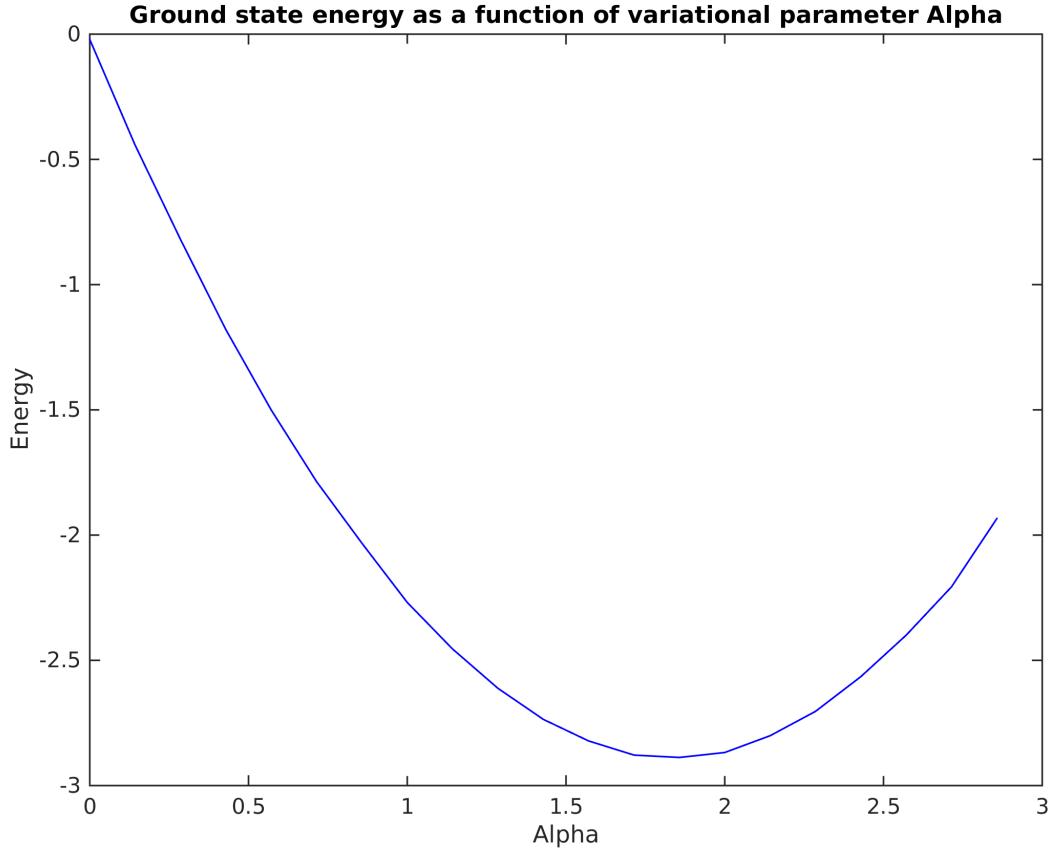


Figure 3: We see how the estimated ground state for He changes with different values of alpha. We reach a minimum about  $\alpha = 1.85$ .

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 were looking for alternative solutions. With introducing GTO basis, we got rid of the need of finding optimal  $\alpha$ , and we could use steepest descent method to find the optimal  $\beta$ . With steepest descent 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 steepest descent, 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 descent 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  $E_0 = -14.510 \text{ a.u.}$  ( $\text{a.u.} = \text{atomic units}$ ), compared to  $\beta = 0.206$  (from steepest descent method) which gives  $E_0 = -14.516 \text{ a.u.}$ . The true value we try to find is  $E_0 = -14.667 \text{ a.u.}$  [REF]. We see that  $\beta = 0.206$  brings us a little bit closer to the true minimum. If we use  $\alpha$  values from graphical evaluation, the results obtained are summarized in the table below (with Jastrow factor included in trial wave function):

Atom	$\alpha$	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  $\beta$  obtained from graphical evaluation and the steepest descent 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  $\beta$  from steepest descent, but this can be due to lack of accuracy in  $\alpha$  and to random effects in the MC solver.

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

## 4.2 Variance in Monte Carlo simulations

When doing MC simulations, we should gain better 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 sampling) with He, gives the figure below:

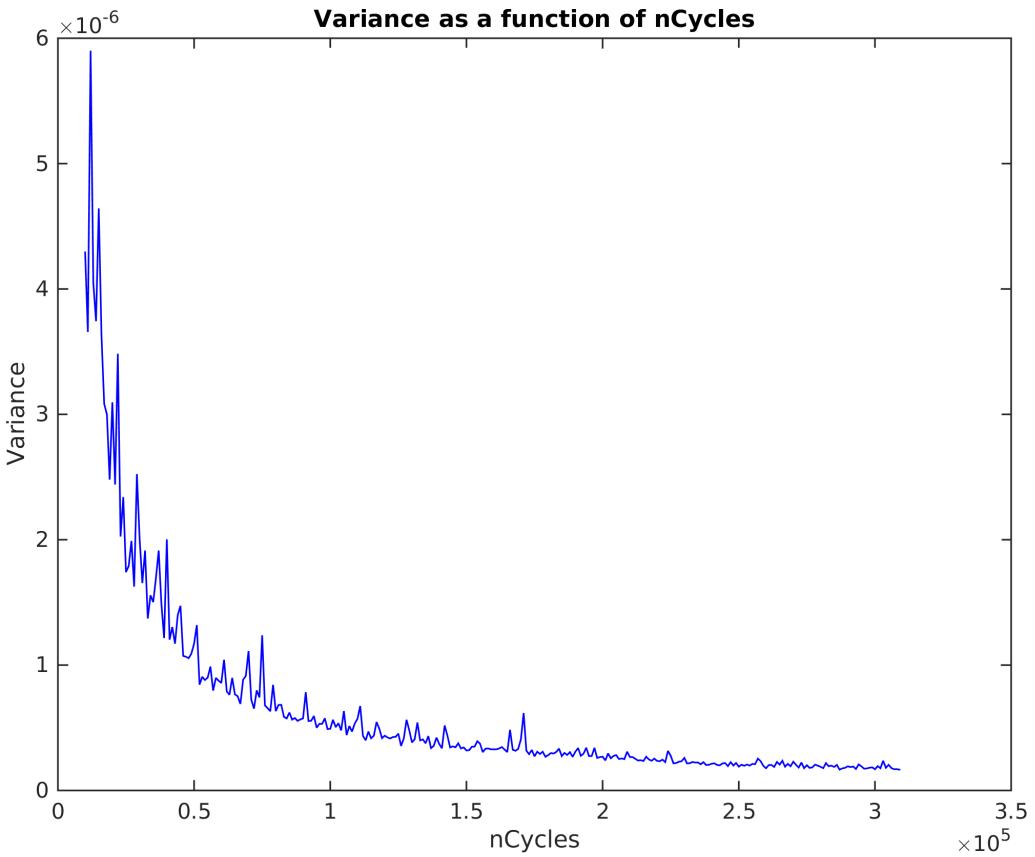


Figure 4: We see how the variance becomes smaller when MC cycles are increased in simulation of the Helium atom.

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 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 fast become ugly. Therefore we implemented an optimized analytical machinery making us able to run our program pretty fast for big atoms such as Neon. 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:

$N = 10^8$	<i>numerical</i>	<i>optimized</i>	<i>analytical</i>	<i>rel. speedup optimized</i>	<i>rel. speedup analytical</i>
<i>without Jastrow factor</i>	295.21 s	143.44 s	134.20 s	2.06	2.20
<i>with Jastrow factor</i>	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 numerical calculations, 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 with He with different combinations:

$N = 10^8$	<i>importance sampling off</i>	<i>importance sampling on</i>
<i>without Jastrow factor</i>	-2.8214	-2.8214
<i>with Jastrow factor</i>	-2.8902	-2.8905

*Table 7: Estimate of ground state energy of Helium.*

$N = 10^8$	<i>importance sampling off</i>	<i>importance sampling on</i>
<i>without Jastrow factor</i>	1.1824	1.1841
<i>with Jastrow factor</i>	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 see how the repulsion 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 chosen value of  $\Delta t$  (*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  $\Delta t$ . The result from a simulation with  $N = 10^7$  are given in Figure 3:

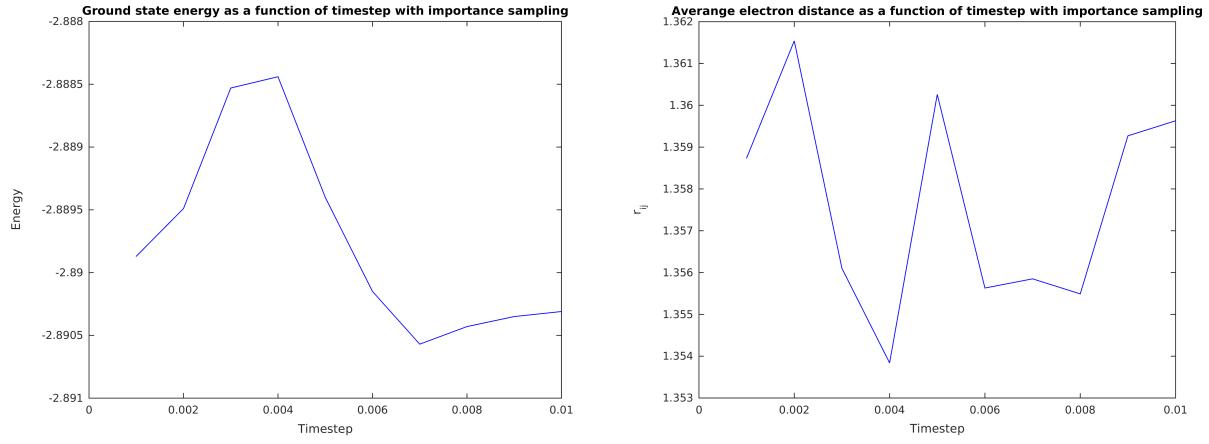


Figure 5: Energy and mean distance between electrons as a function of  $\Delta t$ .

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  $\Delta t$  is not important within the range of  $\Delta t$  values ( $[0.001, 0.01]$ ). The same conclusion can also be drawn for average electron distance  $r_{ij}$ . So a proper value for  $\Delta t$  could be 0.002, but we have to make sure we have enough MC cycles to move electrons to the ground 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  $\Delta t$  is small. Since the quantum force pushes the electrons in the right direction (where the wavefunction 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:

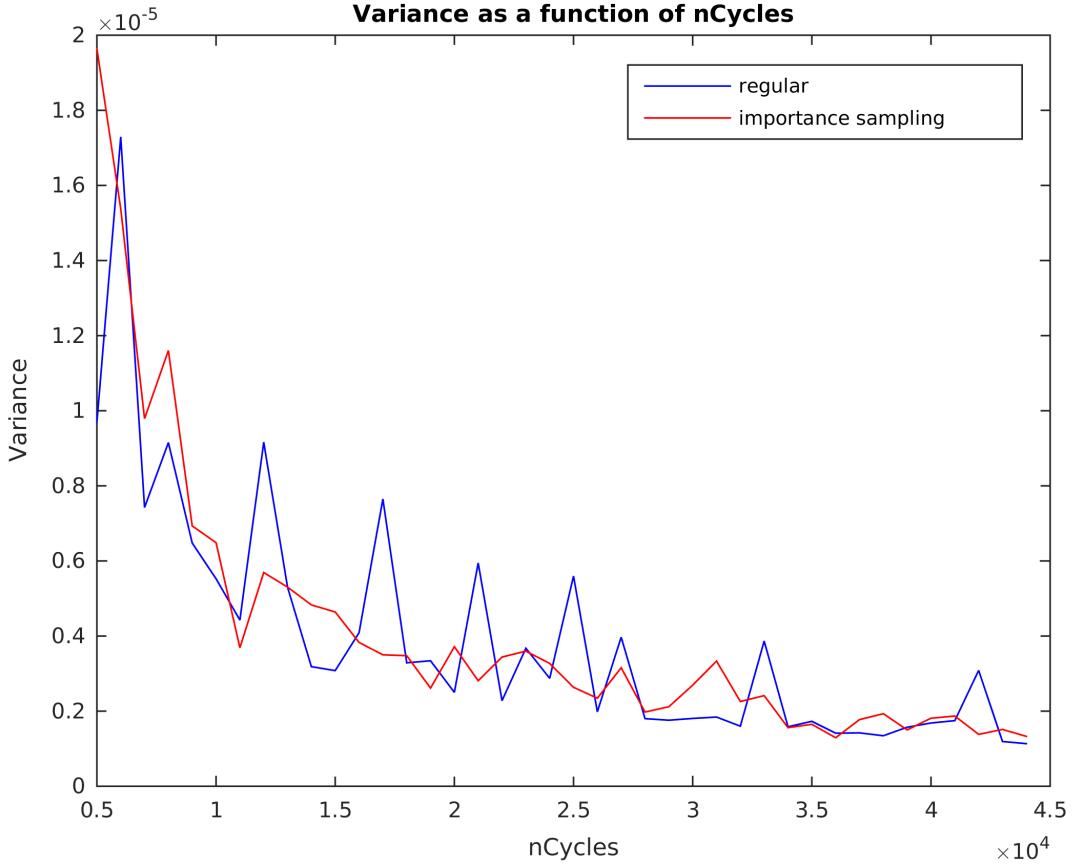


Figure 6: We see that the variance actually decreases equally with and without importance sampling, but with importance sampling there are less variation in the variance, so there should be less variation in the computed energy for few cycles.  $\Delta t = 0.002$  in the first figure,  $\Delta t = 0.005$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

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

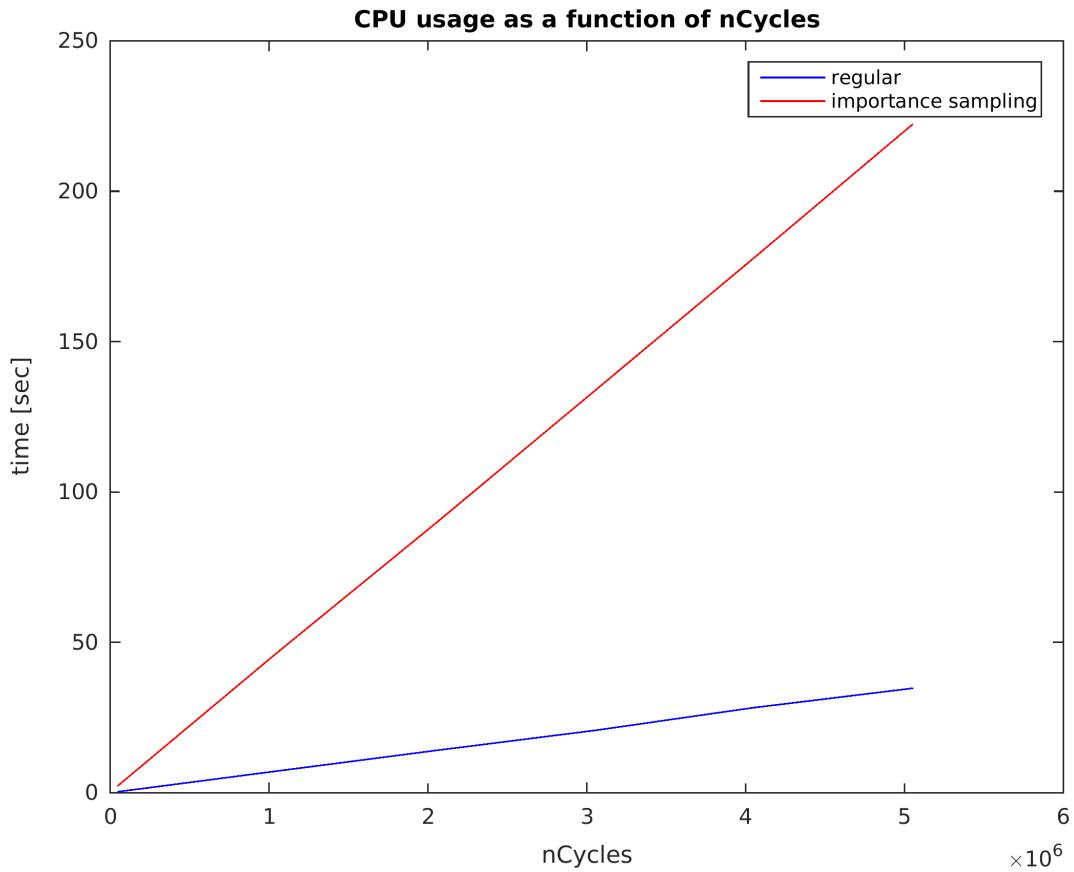
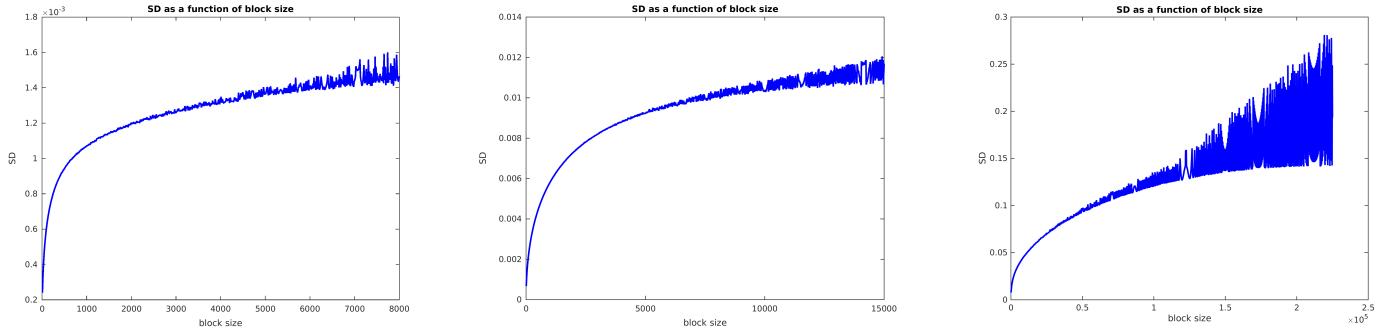


Figure 7: CPU-time as a function of MC cycles for helium with and without importance sampling.

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 mentioned 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 “plateau” 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 “plateau” where  $\sigma_{block} \approx 0.011$  and  $N_{blocksize} \approx 10000$ .

(c) Blocking performed on Ne with  $N = 10^7$ . We see a “plateau” 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, it's very helpful to take a look at what the hydrogenic orbitals look like. This is show in the figure below:

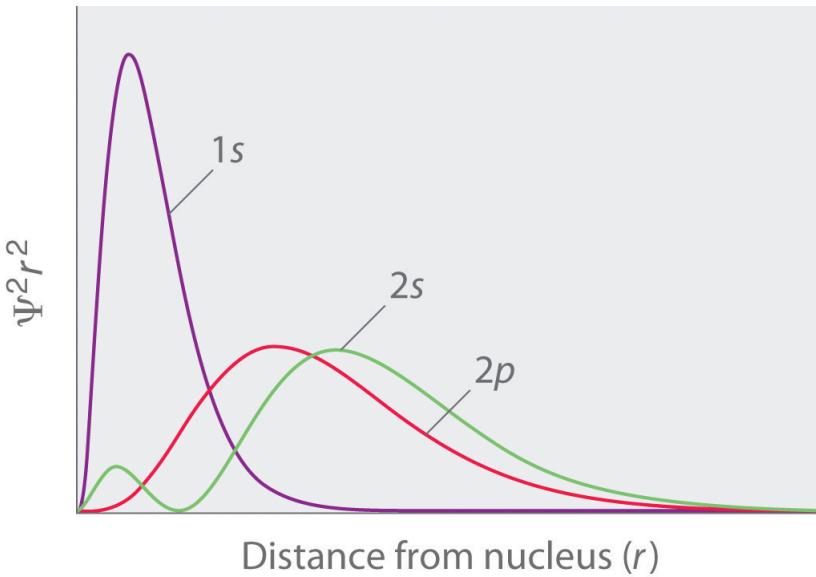


Figure 9: Electron probability density for different hydrogenic orbitals (given in the figure) [15].

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

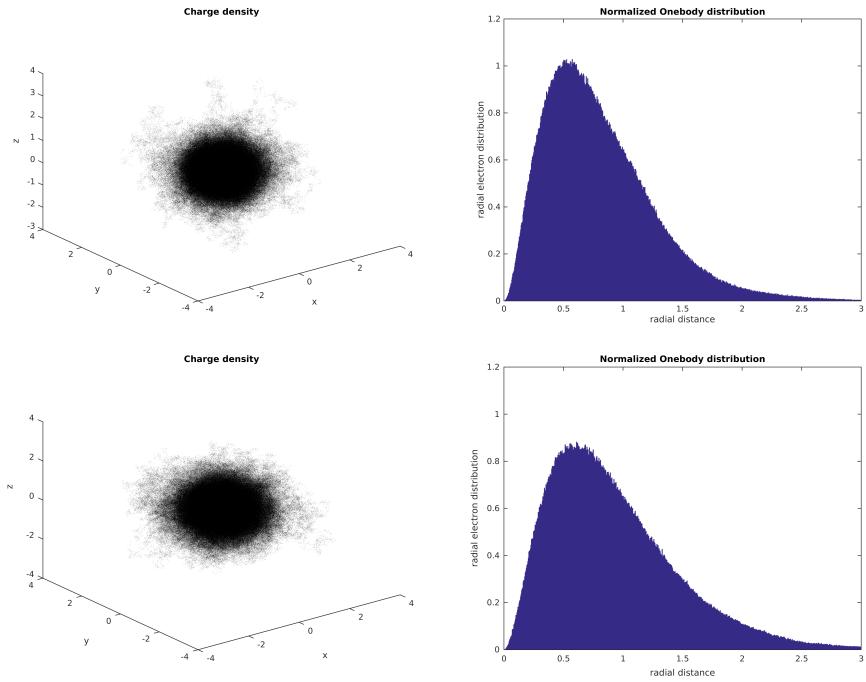


Figure 10: Onebody density helium. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

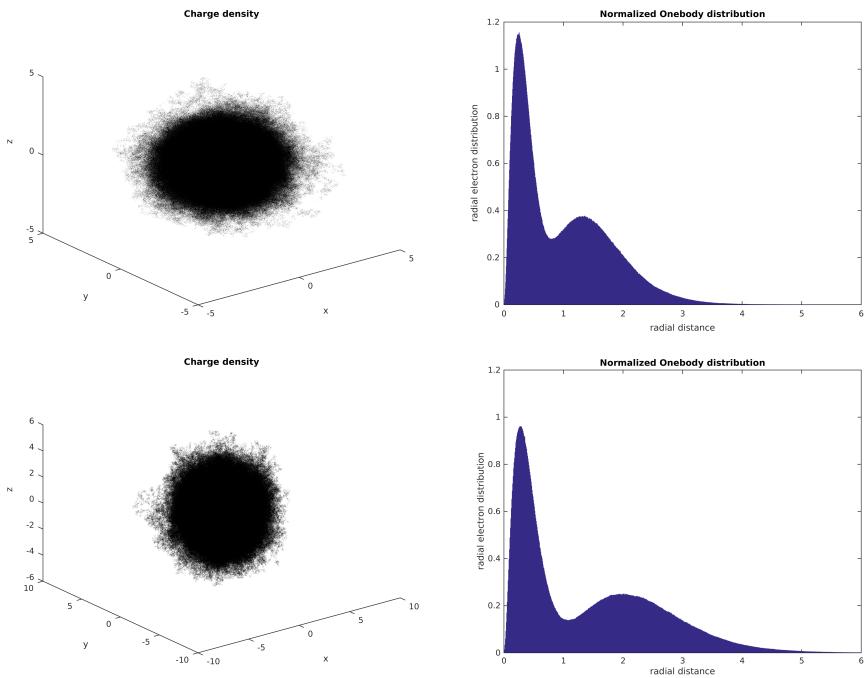


Figure 11: Onebody density beryllium. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

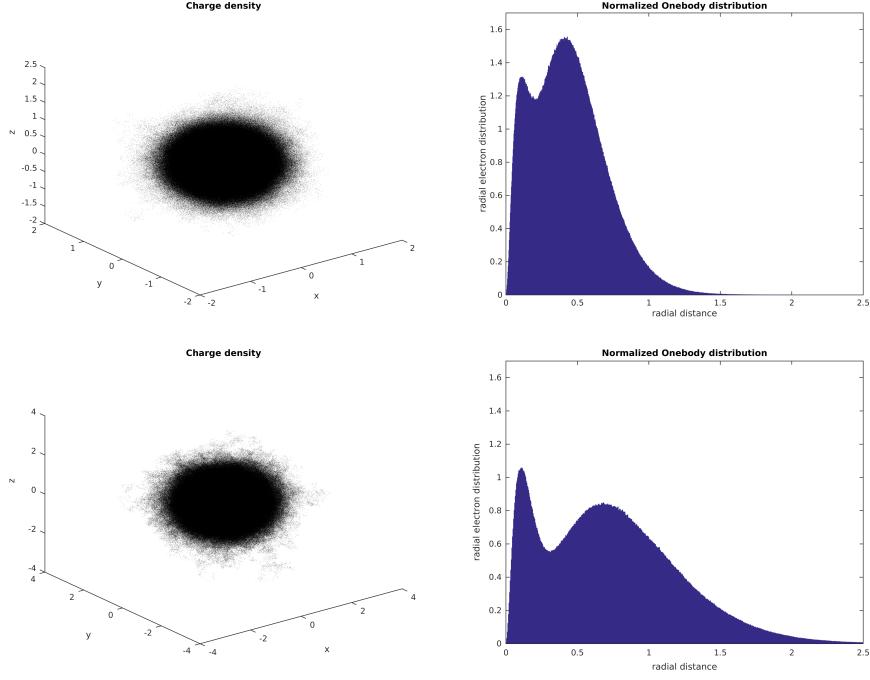


Figure 12: Onebody density neon. Pure hydrogenic wave function in the upper part, with Jastrow factor in the lower part.

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 helium 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 beryllium, 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 neon 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 some 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 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  $R_\rho$ . If we run over different  $R_\rho$  and compute the ground state energy, we have to use steepest gradient method to find the optimal  $\beta$  for each  $R_\rho$ . For  $H_2$  we use  $\alpha = 1.356$  from section 3.14, and get the result given in the figure below:

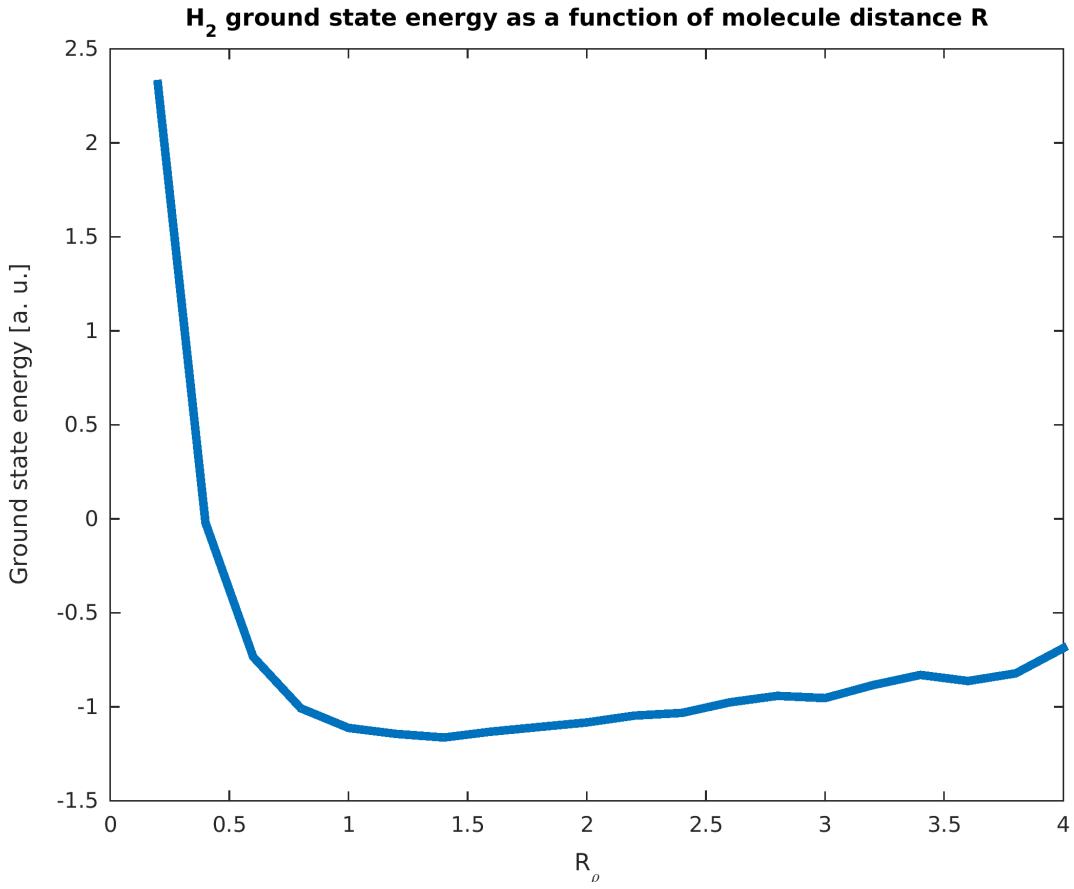


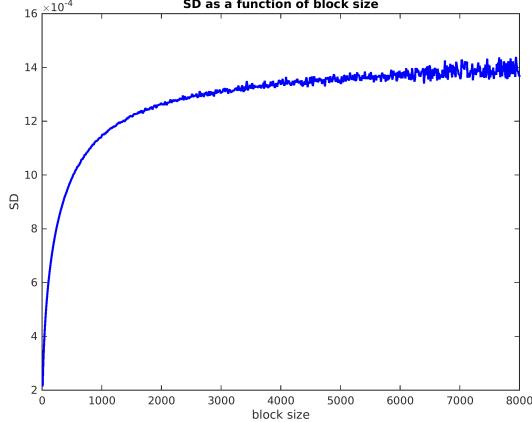
Figure 13: Ground state energy of  $H_2$  as a function of inter-proton distance,  $\alpha = 1.356$ .

From the figure above, we see that we obtain the minimum energy at the bohr radii 1.40 for  $H_2$ , as expected. The corresponding optimal variational parameter is 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 average electron-electron distance to be  $\approx 2.18$ , with  $E_0 = -1.1575$ . We see that the average 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:

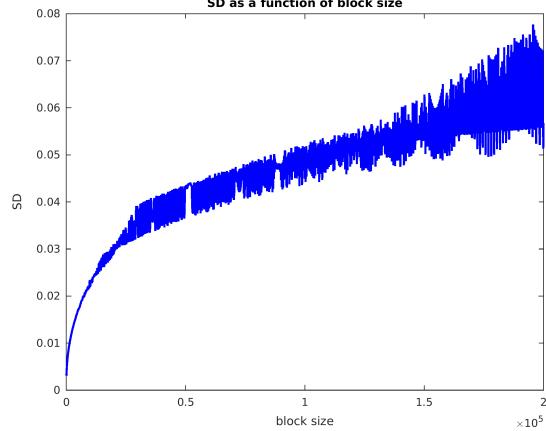
$$1.3912 + \frac{1.4}{2} = 2.09$$

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

For completeness we also estimate  $\sigma$  for  $H_2$  and  $Be_2$ , and blocking gives:



(a) Blocking performed on  $H_2$  with  $N = 10^7$ . We see a “plateau” 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 “plateau” 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:

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

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

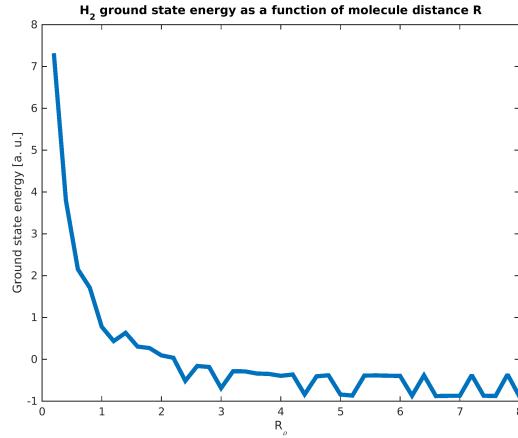
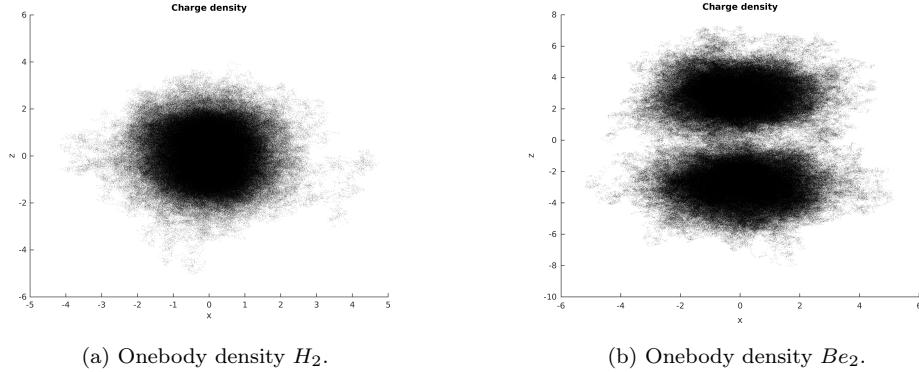


Figure 15: Ground state energy of  $H_2$  (with minus in the trial wave function) as a function of inter-proton distance,  $\alpha = 1.356$ .

In this plot we doesn't get a nice minimum as we did with a pluss sign in our trial wave function. In addition we are far away from the reference energy, and we get oscillations due to too few MC cycles and unstable  $\beta$  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. It's also possible that the optimal  $\alpha = 1.356$  is incorrect in the new trial wave function.

We should also look at charge densities and radial electron distribution for molecules. Results (with Jastrow factor) are given in the figure below:



*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 plots are of little interest for molecules, so it's not included in this report.

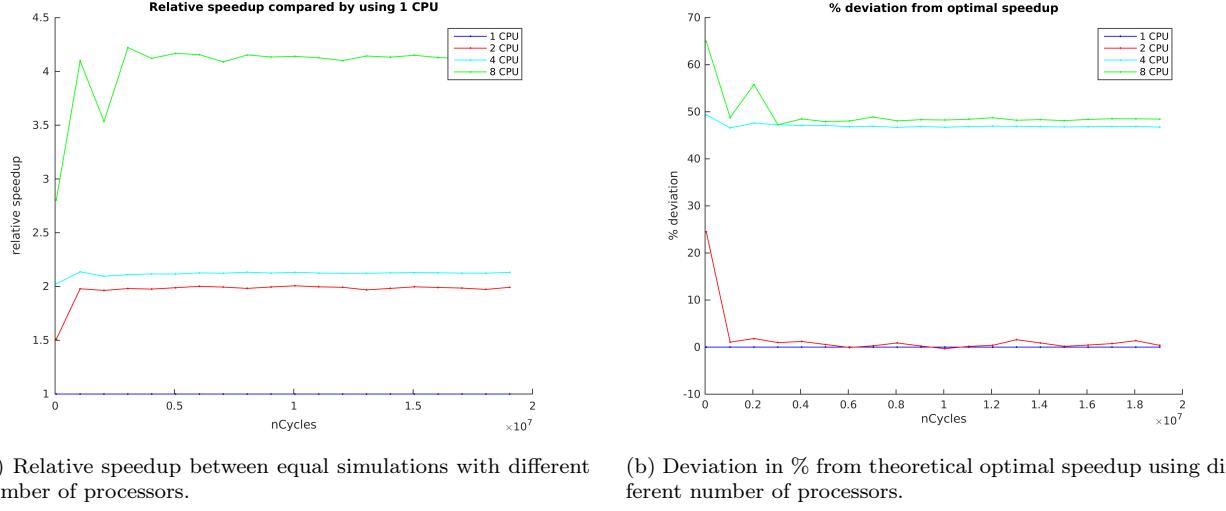
## 4.8 A note on GTO in this project

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  $\approx -2.86$ , which is a poorer estimate than we got with hydrogen like orbitals. For Be we got an energy  $\approx -13.97$ , and for Ne an energy  $\approx -119.67$ . All simulations leads to poorer estimates of the ground state energy than with hydrogenic trial wave functions. 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 see how long time the program uses to run a given number of MC cycles, and compare the time to the time spent 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 [21], and when using 8 CPUs, we are actually using 4 CPUs with multithreading. This should 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 to 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 processors 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 is given in the table below. *Relative error* (rel. error) is given as:  $\frac{|E_0 - E_{VMC}|}{|E_{VMC}|}$  (<http://mathworld.wolfram.com/RelativeError.html>).

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

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

\* These are borrowed from Morten Ledum and Håvard Tveit Ihle's report in 2013 [21], since there was little time to search for the optimal value. In fact we shouldn't need to find the optimal  $\alpha$  since we planned to use GTO basis sets on molecules as well. The steepest descent method was also unstable for  $Be_2$  making it hard to find a good value for  $\beta$ , but experiments indicates that the value of  $\beta$  not have a huge effect on the estimate within a certain range (e. g. 0.001 to 20). In addition we also had to find the optimal  $R_\rho$  for  $Be_2$ , and also this value was also 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 density 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 with 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  $\beta$ . 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 GTO basis sets would propably lead us closer to the reference value. Unfortunately we didn't get better estimates, and are unsure what's the reason for this is.

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 took 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 steepest decent, since it behaves unstable for bigger systems. *Conjugate gradient method* [19] is an option, but we should also consider looking for a method like *Newton's method* [20] 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)

```

1 import sympy as sp
2
3 def LocalEnergy(phi):
4     return ((-sp.diff(phi, x1, 2) - sp.diff(phi, y1, 2) - sp.diff(phi, z1, 2) +
5             diff(phi, x2, 2) - sp.diff(phi, y2, 2) - sp.diff(phi, z2, 2))/(2*phi) -
6             (- sp.
7                 Z/r1 - Z
8                 /r2 + 1 /r12)
9
10 # make variables symbolic
11 x1, x2, y1, y2, z1, z2, alpha, Z = sp.symbols('x1, x2, y1, y2, z1, z2, alpha, Z')
12
13 # creates a symbolic equivalent of r1 and r2
14 R1, R2, R12 = sp.symbols('r1 r2 r12')
15 r1 = sp.sqrt(x1*x1 + y1*y1 + z1*z1) r2 = sp.sqrt(x2*x2 + y2*y2 + z2*z2) r12 = sp.sqrt((x1 - x2)**2
16     + (y1 - y2)**2 + (z1 - z2)**2)
17
18 # our wavefunction
19 phil = sp.exp(-alpha*(r1 + r2))
20
21 # compute the local energy
22 local_energy = LocalEnergy(phil)
23
24 #Simplify the expression local_energy = local_energy.subs(r12,R12).factor().subs(r1, R1).subs(r1
25 **2,R1**2).subs(r2,R2).subs(r2**2,R2**2).simplify().collect(Z).collect(alpha)
26
27 # Print out analytical expression for the local energy as latex and c code
28 print sp.printing.latex(local_energy)
29 print sp.printing.ccode(local_energy)

```

### 6.2 Implement efficient Slater determinant ratio

```

1 // compute the R_sd ratio
2 void VMCSolver::compute_R_sd(int i){

```

```

3     R_sd = 0.0;
4     if(i < nParticles/2){
5         for(int j=0; j<nParticles/2; j++){
6             R_sd += SlaterPsi(rNew, i, j)*D_up_old(j, i);
7         }
8     }
9     else{
10        for(int j=0; j<nParticles/2; j++){
11            R_sd += SlaterPsi(rNew, i, j)*D_down_old(j, i-nParticles/2);
12        }
13    }
14 }
```

### 6.3 Implement efficient gradient and laplacian of trial wave function

```

1 // compute slater first derivative
2 void VMCsolver::SlaterGradient(int i){
3     compute_R_sd(i);
4     if(i < nParticles/2){
5         for(int k=0; k<nDimensions; k++){
6             double derivative_up = 0.0;
7             for(int j=0; j<nParticles/2; j++){
8                 derivative_up += Psi_first_derivative(i, j, k)*D_up_old(j, i);
9             }
10            SlaterGradientNew(i, k) = (1.0/R_sd)*derivative_up;
11        }
12    }
13    else{
14        for(int k=0; k<nDimensions; k++){
15            double derivative_down = 0.0;
16            for(int j=0; j<nParticles/2; j++){
17                derivative_down += Psi_first_derivative(i, j, k)*D_down_old(j, i-
18                    nParticles/2);
19            }
20            SlaterGradientNew(i, k) = (1.0/R_sd)*derivative_down;
21        }
22    }
23
24 // compute slater second derivative
25 double VMCsolver::SlaterLaplacian(){
26     double derivative_up = 0.0;
27     double derivative_down = 0.0;
28     for(int i=0; i<nParticles/2; i++){
29         for(int j=0; j<nParticles/2; j++){
30             derivative_up += Psi_second_derivative(i, j)*D_up_new(j, i);
31             derivative_down += Psi_second_derivative(i+nParticles/2, j)*
32                 D_down_new(j, i);
33         }
34     }
35     double derivative_sum = derivative_up + derivative_down;
36     return derivative_sum;
37 }
```

### 6.4 Implement efficient way of updating the Slater matrix

```

1 // update Slater matrix
2 void VMCsolver::update_D(mat& D_new, const mat& D_old, int i, int selector){
3     i = i - nParticles/2*selector;
4     for(int k=0; k<nParticles/2; k++){
5         for(int j=0; j<nParticles/2; j++){


```

```

6   if(j!=i){
7     double sum = 0;
8     for(int l=0; l<nParticles/2; l++){
9       sum += SlaterPsi(rNew, i+nParticles/2*selector, l)*D_old(l,j);
10    }
11    D_new(k,j) = D_old(k,j) - D_old(k, i)*sum/R_sd;
12  }
13  else{
14    D_new(k,j) = D_old(k, i)/R_sd;
15  }
16}
17}
18}

```

## 6.5 Implement efficient Jastrow factor ratio

```

1 // compute C matrix
2 void VMCSolver::fillJastrowMatrix(mat &CorrelationMatrix){
3   for(int k=0; k < nParticles; k++){
4     for(int i=k+1; i < nParticles; i++){
5       CorrelationMatrix(k,i) = a_matrix(k,i)*r_distance(k,i)/(1.0 + beta*
6         r_distance(k,i));
7     }
8   }
9
10 // compute R_c ratio
11 void VMCSolver::compute_R_c(){
12   double deltaU = 0.0;
13   for(int k=0; k < nParticles; k++){
14     for(int i=0; i < k; i++){
15       deltaU += C_new(i, k) - C_old(i, k);
16     }
17     for(int i=k+1; i < nParticles; i++){
18       deltaU += C_new(k, i) - C_old(k, i);
19     }
20   }
21   R_c = exp(deltaU);
22 }

```

## 6.6 Implement Quantum force

```

1 // compute the quantum force used in importance sampling
2 void VMCSolver::QuantumForce(const mat &r, mat &F){
3   GradientSquared = 0.0;
4   for(int k=0; k<nParticles; k++){
5     for(int j=0; j<nDimensions; j++){
6       if(activate_JastrowFactor){
7         double GradientSum = 0.0;
8         for(int i=0; i<k; i++){
9           GradientSum += (r(k,j)-r(i,j))/r_distance(i,k)*JastrowDerivative(i,k
10             );
11         }
12         for(int i=k+1; i<nParticles; i++){
13           GradientSum -= (r(i,j)-r(k,j))/r_distance(k,i)*JastrowDerivative(k,i
14             );
15         }
16       }
17       F(k,j) = 2.0*(SlaterGradientNew(k,j) + GradientSum);
18       GradientSquared += GradientSum*GradientSum;
19       JastrowGradient(k, j) = GradientSum;
20     }
21   }
22 }

```

```

18         else{
19             F(k,j) = 2.0*SlaterGradientNew(k,j);
20         }
21     }
22 }

```

## 6.7 Implement efficient gradient and laplacian of Jastrow factor

```

1 // compute derivatives used in gradient
2 void VMCSolver::computeJastrowDerivative(int k){
3     for(int i=0; i<k; i++){
4         double divisor = 1.0 + beta*r_distance(i, k);
5         JastrowDerivative(i, k) = a_matrix(i, k)/(divisor*divisor);
6     }
7     for(int i=k+1; i<nParticles; i++){
8         double divisor = 1.0 + beta*r_distance(k, i);
9         JastrowDerivative(k, i) = a_matrix(k, i)/(divisor*divisor);
10    }
11 }
12
13 // compute double derivatives used in Jastrow laplacian
14 void VMCSolver::computeJastrowLaplacian(int k){
15     for(int i=0; i<k; i++){
16         double divisor = 1.0 + beta*r_distance(i, k);
17         JastrowLaplacianNew(i, k) = -2.0*a_matrix(i, k)*beta/(divisor*divisor*divisor);
18     }
19     for(int i=k+1; i<nParticles; i++){
20         double divisor = 1.0 + beta*r_distance(k, i);
21         JastrowLaplacianNew(k, i) = -2.0*a_matrix(k, i)*beta/(divisor*divisor*divisor);
22     }
23 }
24
25 // compute Jasstrow energy and energy crossterm
26 double VMCSolver::computeJastrowEnergy(){
27     double sum = 0.0;
28     energytermSlaterJastrow = 0.0;
29     sum += GradientSquared;
30     for(int k=0; k<nParticles; k++){
31         for(int i=0; i<k; i++) {
32             sum += (nDimensions - 1)/r_distance(i, k)*JastrowDerivative(i, k) +
33                     JastrowLaplacianNew(i, k);
34         }
35         for(int i=k+1; i<nParticles; i++) {
36             sum += (nDimensions - 1)/r_distance(k, i)*JastrowDerivative(k, i) +
37                     JastrowLaplacianNew(k, i);
38         }
39     }
40     for(int k=0; k<nDimensions; k++){
41         energytermSlaterJastrow += dot(SlaterGradientNew.col(k), JastrowGradient.col(k))
42         ;
43     }
44     return sum;
45 }

```

## 6.8 Brute force implementation of GTO

```

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

```

```

5     double alpha2 = GTO_values(1, 0);
6     double alpha3 = GTO_values(2, 0);
7
8     double c1 = GTO_values(0, 1);
9     double c2 = GTO_values(1, 1);
10    double c3 = GTO_values(2, 1);
11
12    double K1 = GTO_coef_values(0, 0);
13    double K2 = GTO_coef_values(1, 0);
14
15    psi = K1*(c1*G_func(alpha1, particle, 0, 0, 0))
16      + K2*(c2*G_func(alpha2, particle, 0, 0, 0) + c3*G_func(alpha3, particle, 0, 0, 0))
17      ;
18  }
19 else{
20   int Kp_set;
21   Kp_set = orb;
22
23   double alpha1 = GTO_values(0, 0);
24   double alpha2 = GTO_values(1, 0);
25   double alpha3 = GTO_values(2, 0);
26   double alpha4 = GTO_values(3, 0);
27   double alpha5 = GTO_values(4, 0);
28   double alpha6 = GTO_values(5, 0);
29   double alpha7 = GTO_values(6, 0);
30   double alpha8 = GTO_values(7, 0);
31   double alpha9 = GTO_values(8, 0);
32
33   double c1 = GTO_values(0, 1);
34   double c2 = GTO_values(1, 1);
35   double c3 = GTO_values(2, 1);
36   double c4 = GTO_values(3, 1);
37   double c5 = GTO_values(4, 1);
38   double c6 = GTO_values(5, 1);
39   double c7 = GTO_values(6, 1);
40   double c8 = GTO_values(7, 1);
41   double c9 = GTO_values(8, 1);
42
43   double K1 = GTO_coef_values(0, Kp_set);
44   double K2 = GTO_coef_values(1, Kp_set);
45   double K3 = GTO_coef_values(2, Kp_set);
46   double K4 = GTO_coef_values(3, Kp_set);
47   double K5 = GTO_coef_values(4, Kp_set);
48   double K6 = GTO_coef_values(5, Kp_set);
49   double K7 = GTO_coef_values(6, Kp_set);
50   double K8 = GTO_coef_values(7, Kp_set);
51   double K9 = GTO_coef_values(8, Kp_set);
52
53   psi = K1*(c1*G_func(alpha1, particle, 0, 0, 0) + c2*G_func(alpha2, particle, 0, 0, 0) + c3
54     *G_func(alpha3, particle, 0, 0, 0))
55   + K2*(c4*G_func(alpha4, particle, 0, 0, 0) + c5*G_func(alpha5, particle, 0, 0, 0))
56   + K3*(c6*G_func(alpha6, particle, 0, 0, 0))
57   + K4*(c7*G_func(alpha7, particle, 1, 0, 0) + c8*G_func(alpha8, particle, 1, 0, 0))
58   + K5*(c9*G_func(alpha9, particle, 1, 0, 0))
59   + K6*(c7*G_func(alpha7, particle, 0, 1, 0) + c8*G_func(alpha8, particle, 0, 1, 0))
60   + K7*(c9*G_func(alpha9, particle, 0, 1, 0))
61   + K8*(c7*G_func(alpha7, particle, 0, 0, 1) + c8*G_func(alpha8, particle, 0, 0, 1))
62   + K9*(c9*G_func(alpha9, particle, 0, 0, 1));
63 }
64 return psi;
65
66 // Compute G in GTO orbitals for given parameters
67 double VMCSSolver::G_func(double GTO_alpha, int particle, int i, int j, int k){
68   double x, y, z, r, N;
69   x = rNew(particle, 0);
70   y = rNew(particle, 1);
71   z = rNew(particle, 2);
72   N = Normalization_factor(GTO_alpha, i, j, k);
73   double G = N*pow(x, i)*pow(y, j)*pow(z, k)*exp(-GTO_alpha*(x*x + y*y + z*z));
74   return G;
75 }
76 // General expression for computing the normalization factor in GTO orbitals

```

```

77 double VMCSolver::Normalization_factor(double GTO_alpha, int i, int j, int k){
78     int fac_i = factorial_func(i);
79     int fac_j = factorial_func(j);
80     int fac_k = factorial_func(k);
81     int fac_2i = factorial_func(2*i);
82     int fac_2j = factorial_func(2*j);
83     int fac_2k = factorial_func(2*k);
84     double frac_over = pow(8.0*GTO_alpha, i+j+k) * (fac_i*fac_j*fac_k);
85     double frac_under = fac_2i*fac_2j*fac_2k;
86     double N = pow(((2.0*GTO_alpha)/pi), -(3.0/4.0)) * sqrt(frac_over/frac_under);
87     return N;
88 }
89
90 // Computes factorial of a given number
91 double VMCSolver::factorial_func(int number){ int factorial = 1;
92     for(int i=2; i<=number; i++){
93         factorial *= i;
94     }
95     return factorial;
96 }
```

## 6.9 Matlab code to perform blocking on data set

```

1 for i=1:block_trials
2     blocks = data_size/(i);
3     energy_trial_sum = 0;
4     energySquared_trial_sum = 0;
5     for j=1:blocks
6         block_size = data_size/blocks;
7         index_start = floor(1 + (j-1)*block_size);
8         index_stop = floor(j*block_size);
9         n = (index_stop - index_start) + 1;
10
11        energy_mean_block = sum(energy(index_start:index_stop))/n;
12        energy_trial_sum = energy_trial_sum + energy_mean_block;
13        energySquared_trial_sum = energySquared_trial_sum+ energy_mean_block*energy_mean_block;
14    end
15
16    energy_mean_blockSize = energy_trial_sum/blocks;
17    energySquared_mean_blockSize = energySquared_trial_sum/blocks;
18    variance_mean_blockSize = (energySquared_mean_blockSize - (energy_mean_blockSize*
19        energy_mean_blockSize))/blocks;
20    block_size_values(i) = floor(block_size);
21    block_size_variance(i) = variance_mean_blockSize;
22 end
23 SD = sqrt(block_size_variance);
24 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 Initio 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] Filippi Singh and Umringar, J. Chemical Physics **105**, 123 (1996). Results on  $Be_2$ .
- [15] Orbital illustrations, [http://catalog.flatworldknowledge.com/bookhub/4309?e=averill\\_1.0-ch06\\_s05#](http://catalog.flatworldknowledge.com/bookhub/4309?e=averill_1.0-ch06_s05#), (June 10, 2015).
- [16] Coulomb potential, <http://cps-www.bu.edu/Wasser/robert/work/node9.html>, (June 10, 2015).
- [17] History of the discovery of atoms, <http://www.nobeliefs.com/atom.htm>, (June 10, 2015).
- [18] Steepes decent algorithm, [http://en.wikipedia.org/wiki/Gradient\\_descent](http://en.wikipedia.org/wiki/Gradient_descent), (June 10, 2015).
- [19] M. Hjorth-Jensen, *Conjugate gradient methods and other optimization methods*, lecture notes in FYS4411, University of Oslo, (2015).
- [20] Newtons Method algorithm, [http://en.wikipedia.org/wiki/Newton%27s\\_method](http://en.wikipedia.org/wiki/Newton%27s_method), (June 10, 2015).
- [21] Memory Bandwidth: [http://en.wikipedia.org/wiki/Memory\\_bandwidth](http://en.wikipedia.org/wiki/Memory_bandwidth), (June 10, 2015).