Part I

Theory

1

Modelled Systems

The systems modelled in this thesis are exclusively systems which have closed form solutions when the Coulomb interaction is removed, i.e. in the non-interacting case. As discussed in Section **??**, these closed form solutions are used to construct an optimal trial wave function in the form of a single Slater determinant. Without such an optimal basis, a single Slater determinant is not sufficient in Quantum Monte-Carlo (QMC) simulations. It is therefore not random that the focus of this thesis has been on various kinds of *quantum dots* and *atomic systems*, resembling the analytically solvable harmonic oscillator and the hydrogen atom respectively.

In this chapter atomic units will be used, that is, $\hbar = e = m_e = 4\pi\epsilon_0 = 1$, where m_e and e_0 is the electron mass and vacuum permittivity respectively.

1.1 Atomic Systems

Atomic systems are described as a number of electrons surrounding oppositely charged nuclei. As an approximation, the position of the nucleus is fixed. Due to the fact that the mass of the core is several orders of magnitude larger than the mass of the electrons, this serves as a very good approximation. In literature this is referred to as the *Born-Oppenheimer Approximation* [1].

1.1.1 The Single Particle Basis

The single particle basis used to construct the trial wave functions for atomic systems originate from the closed form solutions for the hydrogen atom, i.e. one electron surrounding a single nucleus.

Given a nucleus with charge Z, the external potential between the electron and the core is

$$\widehat{\mathbf{v}}_{\text{ext}}(\mathbf{r}) = -\frac{Z}{r}, \tag{1.1}$$

which results in the following single particle Hamiltonian:

$$\widehat{\mathbf{h}}_0(\mathbf{r}) = -\frac{1}{2}\nabla^2 - \frac{Z}{r}. \tag{1.2}$$

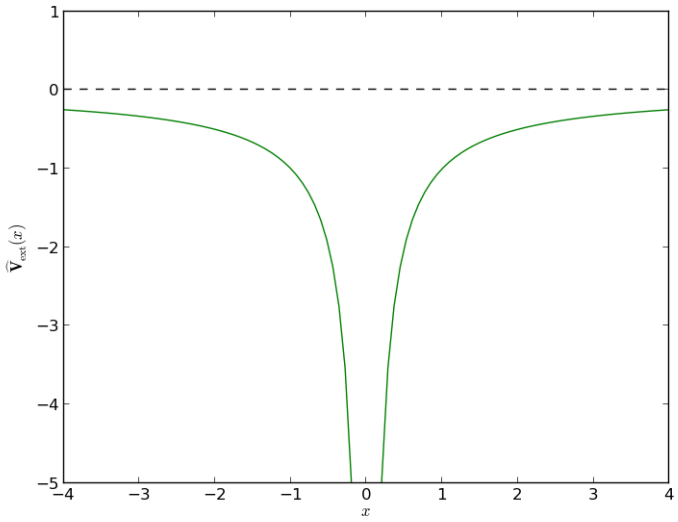


Figure 1.1: The one dimensional version of the single particle potential of hydrogen from Eq. (1.1). The potential is spherically symmetric for three dimensions and can be visualized by rotating the figure around all axes. The potential has a strong singularity at the origin, which originates from the fact that the nucleus (located at the origin) and the electrons have opposite charges, that is, they feel an attractive force.

The external potential is displayed in figure 1.1. The eigenfunctions of the Hamiltonian are [2]

$$\phi_{nlm}(r,\theta,\phi;Z) \propto r^l e^{-Zr/n}\left[L_{n-l-1}^{2l+1}\left(\frac{2r}{n}Z\right)\right]Y_l^m(\theta,\phi), \tag{1.3}$$

where $L_{q-p}^{p}(x)$ are the *associated Laguerre polynomials* and $Y_l^m(\theta,\phi)$ are the *spherical harmonics*. The spherical harmonics are related to the *associated Legendre functions* P_l^m in the following manner:

$$Y_l^m(\theta,\phi) \propto P_l^m(\cos\theta)e^{im\phi}, \tag{1.4}$$

In the current model, the principal quantum number n together with Z control the energy level of the atom,

$$E(n;Z) = -\frac{Z^2}{2n^2} \tag{1.5}$$

which implies that the energy levels are degenerate for all combinations of l and m. For a given value of n, the allowed levels of the *azimuthal* quantum number l and the *magnetic* quantum number m is

$$\begin{aligned} n &= 1, 2, ... \\ l &= 0, 1, ..., n-1 \\ m &= -l, -l+1, ..., l-1, l \end{aligned}$$

which defines the shell structure of the hydrogen atom.

A problem with the single particle basis of hydrogen is the complex terms in Eq. (1.4), i.e. the spherical harmonics. The introduction of the *solid harmonics* $S_l^m(r,\theta,\phi)$ allows for using real-valued eigenfunctions in the QMC simulations. The solid harmonics are related to the spherical harmonics through [3]

$$S_l^m(r,\theta,\phi) \propto r^l \left[Y_l^m(\theta,\phi) + (-1)^m Y_l^{-m}(\theta,\phi)\right] \tag{1.6}$$

$$\propto r^l P_l^{|m|}(\cos\theta) \begin{cases} \cos m\phi & m \geq 0 \\ \sin |m|\phi & m < 0 \end{cases}, \tag{1.7}$$

which results in the following real eigenfunctions

$$\phi_{nlm}(r,\theta,\phi;k) \propto e^{-kr/n} \left[L_{n-l-1}^{2l+1}\left(\frac{2r}{n}k\right)\right] S_l^m(r,\theta,\phi) \equiv \phi_{nlm}^{\mathrm{H}}(\mathbf{r}), \tag{1.8}$$

where $k = \alpha Z$ is a scaled charge with α as a variational parameter chosen by methods described in Section **??**.

A set of quantum numbers nlm is mapped to a single index i. A listing of all the single particle wave functions and their closed form derivatives are given in Appendix F.

1.1.2 Atoms

An atom is described as N electrons surrounding a fixed nucleus of charge $Z = N$. The Hamiltonian consists of N single particle hydrogen Hamiltonians in addition to the Coulomb interaction, which results in

$$\widehat{\mathbf{H}}_{\mathrm{Atoms}}(\mathbf{r}) = \sum_{i=1}^{N} \widehat{\mathbf{h}}_0(\mathbf{r}_i) + \sum_{i<j} \frac{1}{r_{ij}} \tag{1.9}$$

$$= \sum_{i=1}^{N} \left[-\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i}\right] + \sum_{i<j} \frac{1}{r_{ij}}, \tag{1.10}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$. Excluding the Coulomb term, the Hamiltonian can be decoupled into single particle terms with a total ground state energy

$$E_0 = -\frac{Z^2}{2} \sum_{i=1}^{N} \frac{1}{n_i^2}. \tag{1.11}$$

The Slater determinant is set up to fill the N lowest lying states, that is, the N states with lowest n without breaking the Pauli principle, using the single particle orbitals from Eq. (1.8). The degeneracy of level n in an atom is given by the following expression:

$$g(n) = 2 \sum_{l=0}^{n-1} \sum_{m=-l}^{l} 1 = 2n^2. \tag{1.12}$$

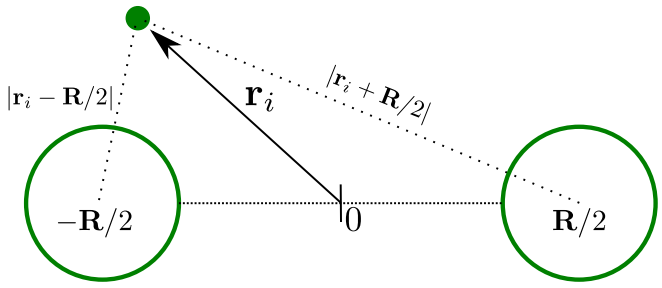


Figure 1.2: The model for the diatomic molecule used in this thesis. The two largest circles represents the atoms. An electron at position $\mathbf{r}_i$ gets a potential energy contribution from both the cores equal to $Z/|\mathbf{r}_i+\mathbf{R}/2|$ and $Z/|\mathbf{r}_i-\mathbf{R}/2|$, where Z is the charge of the nuclei (homonuclear). The diatomic molecular wave functions are set up as a superposition of two hydrogen-like wave functions, one at position $\mathbf{r}_i+\mathbf{R}/2$ and the second at position $\mathbf{r}_i - \mathbf{R}/2$.

1.1.3 Homonuclear Diatomic Molecules

A homonuclear diatomic molecule consists of two atoms (diatomic molecule) of the same element (homonuclear) with charge $Z = N/2$, separated by a distance R. The origin is set between the atoms, which are fixed at positions $\pm\mathbf{R}/2$. An electron at position $\mathbf{r}_i$ gets a contribution from both the cores as displayed in figure 1.2. In addition, there is a repulsive Coulomb potential between the two cores equal to Z^2/R. The resulting Hamiltonian becomes

$$\widehat{\mathbf{H}}_{\text{Mol.}}(\mathbf{r},\mathbf{R}) = \sum_{i=1}^{N}\left[-\frac{1}{2}\nabla_i^2 + \frac{Z}{|\mathbf{r}_i+\mathbf{R}/2|} + \frac{Z}{|\mathbf{r}_i-\mathbf{R}/2|}\right] + \frac{Z^2}{R} + \sum_{i<j}\frac{1}{r_{ij}}. \tag{1.13}$$

In order to transform the hydrogen eigenstates $\phi^{\text{H}}_{nlm}(\mathbf{r})$, which are symmetric around a single nucleus, into molecular single particle states $\phi^{\pm}_{nlm}(\mathbf{r}_i)$, a superposition of the two mono-nucleus wave functions are used

$$\phi^{+}_{nlm}(\mathbf{r}_i,\mathbf{R}) = \phi^{\text{H}}_{nlm}(\mathbf{r}_i+\mathbf{R}/2) + \phi^{\text{H}}_{nlm}(\mathbf{r}_i-\mathbf{R}/2), \tag{1.14}$$
$$\phi^{-}_{nlm}(\mathbf{r}_i,\mathbf{R}) = \phi^{\text{H}}_{nlm}(\mathbf{r}_i+\mathbf{R}/2) - \phi^{\text{H}}_{nlm}(\mathbf{r}_i-\mathbf{R}/2), \tag{1.15}$$

which reads “electron surrounding first nucleus combined with electron surrounding second nucleus”. See figure 1.2 for a better view.

As seen from the equations above, there are necessarily two ways of doing this superposition: Adding and subtracting the states. It is easy to show that

$$\left\langle \phi^{-}_{n'l'm'} | \phi^{+}_{nlm} \right\rangle = 0, \tag{1.16}$$

which implies that these states form an expanded complete set of single particle states for the molecular system, resulting in a four-fold degeneracy in each set of quantum numbers nlm. It is necessary to use

both the positive and negative states in order to fit e.g. four electrons into $n = 0$ for the case of the lithium molecule ($N = 6$). Using only the positive or only the negative states would result in a singular Slater determinant.

Using $\mathbf{R} = (R_x, R_y, R_z)$ as the vector separating the atoms, $\mathbf{j} = (0, 1, 0)$ as the unit vector in the y-direction, $\mathbf{r}_i = (x_i, y_i, z_i)$ as the electron position, and the chain rule of derivation, the gradient in the $\mathbf{j}$-direction becomes

$$\begin{aligned}
\mathbf{j} \cdot \nabla_i \phi^{\pm}_{nlm}(\mathbf{r}_i, \mathbf{R}) &= \underbrace{\frac{\partial(y_i + R_y/2)}{\partial y_i}}_{1} \frac{\partial \phi^{\mathrm{H}}_{nlm}(\mathbf{r}_i + \mathbf{R}/2)}{\partial(y_i + R_y/2)} \\
&\pm \underbrace{\frac{\partial(y_i - R_y/2)}{\partial y_i}}_{1} \frac{\partial \phi^{\mathrm{H}}_{nlm}(\mathbf{r}_i - \mathbf{R}/2)}{\partial(y_i - R_y/2)} \\
&= \frac{\partial \phi^{\mathrm{H}}_{nlm}(\mathbf{r}_i + \mathbf{R}/2)}{\partial(y_i + R_y/2)} \pm \frac{\partial \phi^{\mathrm{H}}_{nlm}(\mathbf{r}_i - \mathbf{R}/2)}{\partial(y_i - R_y/2)} \\
&= \frac{\partial \phi^{\mathrm{H}}_{nlm}(\tilde{\mathbf{R}}^{+}_{\mathbf{i}})}{\partial \tilde{Y}^{+}_i} \pm \frac{\partial \phi^{\mathrm{H}}_{nlm}(\tilde{\mathbf{R}}^{-}_{\mathbf{i}})}{\partial \tilde{Y}^{-}_i},
\end{aligned} \tag{1.17}$$

where $\tilde{\mathbf{R}}^{\pm}_{\mathbf{i}} = \mathbf{r}_i \pm \mathbf{R}/2 = (\tilde{X}^{\pm}_i, \tilde{Y}^{\pm}_i, \tilde{Z}^{\pm}_i)$ represents the electron position in the reference frame of the two nuclei. Equation (1.17) demonstrates that the closed form expressions used in simulations of single atoms can be reused in the case of diatomic molecules. In other words, the functions in Appendix F can simply be called with $\tilde{\mathbf{R}}^{\pm}_{\mathbf{i}}$ instead of $\mathbf{r}_i$ and then be either subtracted or added. This result holds for the Laplacian as well.

The non-interacting energy is equal to that of the regular atoms in the limit $R \to \infty$, however, now with a four-fold degeneracy and a charge equal to $N/2$. This four-foulding also implies that the degeneracy of level n becomes $g(n) = 4n^2$.

1.2 Quantum Dots

Quantum dots are electrons confined within a potential well. This potential well can be broadened and narrowed in such a way that the material properties of the quantum dot can be tuned to desired values. Such manufactured quantum dots have practical applications in solar cells [4], lasers [5], medical imaging [6], and quantum computing [7], however, the focus on quantum dots in this thesis has been purely academic.

Understanding the physics behind strongly and weakly confined electrons are of great importance when it comes to understanding many-body theory in general. The purpose of studying quantum dots from an academic point of view is to investigate the behavior of the system a function of the level of confinement and the number of electrons.

The model for the quantum dot used in this thesis is electrons trapped in a harmonic potential with frequency ω, which in the case of no Coulomb interaction can be solved analytically.

1.2.1 The Single Particle Basis

Just as the hydrogen potential was used to describe atoms, the *harmonic oscillator* potential is used as the single particle Hamiltonian describing quantum dots, that is, an external one-body potential on the

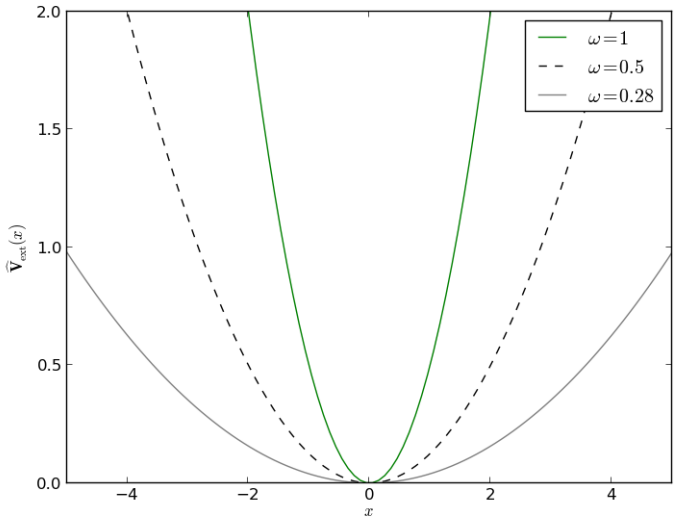


Figure 1.3: A one dimensional version of the single particle potential of quantum dots. In two - and three dimensions, the potential is rotationally/spherically symmetric and equal along all axes. Just as the hydrogen potential in figure 1.1, the electrons are drawn to the center.

form

$$\widehat{\mathbf{v}}_{\text{ext}}(\mathbf{r}) = \frac{1}{2}\omega^2 r^2, \tag{1.18}$$

where ω is the oscillator frequency representing the strength for the confinement. The potential for different ω is presented in figure 1.3. The single particle Hamiltonian is thus

$$\widehat{\mathbf{h}}_0(\mathbf{r}) = -\frac{1}{2}\nabla^2 + \frac{1}{2}\omega^2 r^2. \tag{1.19}$$

The eigenfunctions of the Hamiltonian for two and three dimensions are [1]

$$\phi_{n_x,n_y}(\mathbf{r}) = H_{n_x}(\sqrt{w}x)H_{n_y}(\sqrt{w}y)e^{-\frac{1}{2}wr^2} \tag{1.20}$$

$$\phi_{n_x,n_y,n_z}(\mathbf{r}) = H_{n_x}(\sqrt{w}x)H_{n_y}(\sqrt{w}y)H_{n_z}(\sqrt{w}z)e^{-\frac{1}{2}wr^2}, \tag{1.21}$$

where $H_n(x)$ is the n'th level Hermite polynomial. The shell structures in quantum dots arise from different combinations of n_x, n_y, and for three dimensions n_z, which sums up to the same n.

The variational parameter α is introduced by letting $\omega \to \alpha\omega$, just as $Z \to \alpha Z$ for atoms. Defining $k \equiv \sqrt{\alpha\omega}$, the eigenfunctions which are used as the single particle orbitals for quantum dots in this thesis are

$$\phi_{n_x,n_y}(\mathbf{r}) = H_{n_x}(kx)H_{n_y}(ky)e^{-\frac{1}{2}k^2r^2}, \tag{1.22}$$

$$\phi_{n_x,n_y,n_z}(\mathbf{r}) = H_{n_x}(kx)H_{n_y}(ky)H_{n_z}(kz)e^{-\frac{1}{2}k^2r^2}. \tag{1.23}$$

As for the hydrogen states, a set of quantum numbers are mapped to an integer i. A listing of all the single particle wave functions and their closed form derivatives are given in Appendix D for two dimensions and Appendix E for three dimensions.

1.2.2 Two - and Three Dimensional Quantum Dots

A quantum dot is described as N electrons confined in an oscillator potential with frequency ω. The Hamiltonian is

$$\widehat{\mathbf{H}}_{\text{Q.D.}}(\mathbf{r}) = \sum_{i=1}^{N} \widehat{\mathbf{h}}_0(\mathbf{r}_i) + \sum_{i<j} \frac{1}{r_{ij}} \tag{1.24}$$

$$= \sum_{i=1}^{N} \left[-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right] + \sum_{i<j} \frac{1}{r_{ij}}, \tag{1.25}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$. Excluding the Coulomb term, the Hamiltonian can be decoupled into single particle terms with a total ground state energy [2]

$$E_0 = \omega \sum_{i=1}^{N} \left(n_i + \frac{d}{2} \right), \tag{1.26}$$

where d is the number of dimensions, $n_i = n_x + n_y + n_z \geq 0$ for three dimensions and $n_i = n_x + n_y \geq 0$ for two dimensions. The degeneracy of level n in a quantum dot is $g(n) = 2n$ for two dimensions and $g(n) = (n+2)(n+1)$ for three dimensions.

The Slater determinant is set up to fill the N lowest lying states, that is, the N states with lowest n without breaking the Pauli principle, using the single particle orbitals from Eq. (1.8).

1.2.3 Double-well Quantum Dots

The same strategy used to transform an atomic system into a homonuclear diatomic molecular system can be applied to two dimensional quantum dots, resulting in a double-well quantum dot. Double-well quantum dots are used as a practical quantum dot system in experiments [8].

The model for the double-well potential used in this thesis is [9]

$$\widehat{\mathbf{v}}_{ext}(\mathbf{r}) = \frac{1}{2} m^* \omega_0^2 \left[r^2 + \frac{1}{4} R^2 - R|x| \right], \tag{1.27}$$

where R is the distance between the wells, m^* is a material parameter and ω_0 is the confinement strength. For simplicity, the wells are separated in the x-direction. The potential is presented in figure 1.4.

The full Hamiltonian becomes

$$\widehat{\mathbf{H}}_{\text{QDDW}}(\mathbf{r}, \mathbf{R}) = \sum_{i=1}^{N} \left(-\frac{1}{2m^*}\nabla_i^2 + \frac{1}{2} m^* \omega_0^2 \left[r^2 + \frac{1}{4} R^2 - R|x_i| \right] \right) + \sum_{i<j} \frac{1}{r_{ij}}, \tag{1.28}$$

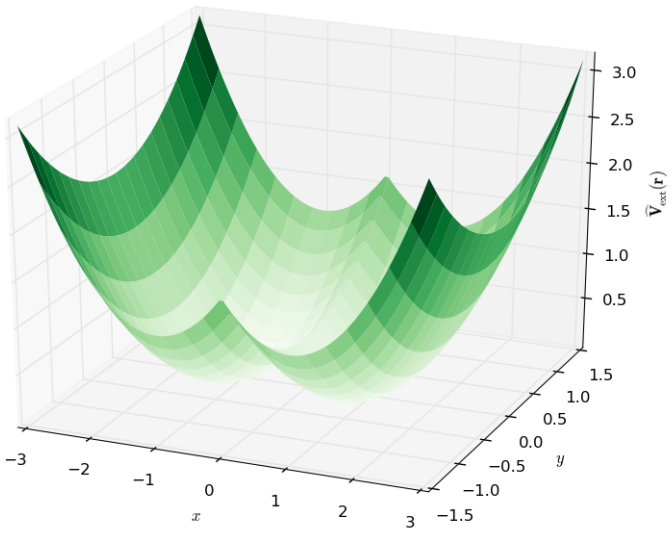


Figure 1.4: The external potential for a double-well quantum dot from Eq. (1.27) with $R = 2$ and $m^*\omega_0^2 = 1$, where R is the distance between the well centers and m^* and ω_0 are material constants.

which can be simplified to fit the standard form of the previous Hamiltonians by letting $r_i \to \sqrt{m^*}r_i$. Applying this transformation of coordinates yields

$$\widehat{\mathbf{H}}_{\text{QDDW}}(\mathbf{r}, \mathbf{R}) \to \widehat{\mathbf{H}}_{\text{QDDW}}(\sqrt{m^*}\mathbf{r}, \sqrt{m^*}\mathbf{R}) \tag{1.29}$$

$$= \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega_0^2 \left[r^2 + \frac{1}{4}R^2 - R|x_i| \right] \right) + \sqrt{m^*} \sum_{i<j} \frac{1}{r_{ij}}. \tag{1.30}$$

The eigenstates are, as for the homonuclear diatomic molecules in Eq. (1.14) and (1.15), given as positive and negative superpositions of the standard harmonic oscillator eigenstates. As shown for molecules in Eq. (1.17), the closed form expressions for the single-well quantum dot can be reused in the case of a double-well quantum dot.

The degeneracy of the n'th level is $g(n) = 4n$. The non-interacting single particle energies are identical to those of the single-well in the limit $R \to \infty$, that is, in the case of two decoupled potential wells.

Part II

Results

Atom	E_{VMC}	E_{DMC}	Expt.	ϵ_{rel}
He	-2.8903(2)	-2.9036(2)	-2.9037	$3.44 \cdot 10^{-5}$
Be	-14.145(2)	-14.657(2)	-14.6674	$7.10 \cdot 10^{-4}$
Ne	-127.853(2)	-128.765(4)	-128.9383	$1.34 \cdot 10^{-3}$
Mg	-197.269(3)	-199.904(8)	-200.054	$7.50 \cdot 10^{-4}$
Ar	-524.16(7)	-527.30(4)	-527.544	$4.63 \cdot 10^{-4}$
Kr	-2700(5)	-2749.9(2)	$-2752.054976*$	$7.83 \cdot 10^{-4}$

Table 1.1: Ground state energies for Atoms calculated using Variational - and Diffusion Monte-Carlo. Experimental energies are listed in the last column. As we see, DMC is rather close to the experimental energy. $\epsilon_{\mathrm{rel}} = |E_{\mathrm{DMC}} - \mathrm{Expt.}|/|\mathrm{Expt.}|$. Experimental, i.e. "exact" energies are taken from Ref. [10] for He through Ar, and [11] for Kr. (*) The referenced Krypton result is not a direct experimental result, but obtained by Hartree-Fock calulcations with optimized single particle wave functions, and is thus believed to be a very good approximation to the exact ground state.

1.3 Atoms

The focus regarding atoms has been on simulating heavy atoms using a simple ansatz for the trial wave function, and thus test its limits. Due to the important nature of atoms in nature, precise experimental data which can be used to benchmark the results exist. Due to having an non-negligible relativistic nature, atoms which are heavier than Krypton are not simulated. The specifics regarding the model used for atoms are covered in Section 1.1.

1.3.1 Ground State Energies

Table 1.1 presents the ground state energy results for different atoms together with the experimental results. As expected, the helium ground state has the highest overlap with the corresponding experimental result. The relative precision of the heavier atoms are in the range 10^{-3} - 10^{-4}, indicating that DMC performs equally well in all cases. However, the error in the calculations increase as the atoms become heavier. The calculations were done on a single node; running the calculations on several nodes with an increased number of walkers could reduce the error somewhat.

In comparison to quantum dots, where VMC and DMC produced significantly similar energies, it is evident that VMC performs rather poorly compared to DMC for atoms. Unlike quantum dots, the atomic systems allow for unbound states. This implies that the atomic systems in this thesis have an addition approximation in the trial wave function due to the fact that all the orbitals represent bound states. Nevertheless, this only further demonstrate the strengths of DMC to predict accurate results without much knowledge of the system at hand.

1.3.2 One-body densities

The one-body densities for the *noble gases*, that is, the closed shell atoms, are presented in Figure 1.6. Comparing these to the one-body densities for the alkaline earth metals, i.e. Be, Mg, etc., in Figure 1.5,

it is clear that the noble gases have a more confined electron distribution. This corresponds well to the fact that noble gases do not form compound materials, i.e. molecules [12]. The alkaline earth metals on the other hand are found naturally as parts of compound materials. The one-body densities of the alkaline earth metals spreading out in space are thus in excellent agreement with nature.

The attractive nature of the nuclear potential is demonstrated by the core having an extremely high electron density, independent of the number of electrons. The radial density is thus not very insightful, however, multiplied with the radius squared, the difference between atoms become noticeable.

It is apparent that the VMC distribution and the pure distribution differ more in the case of alkaline earth metals than for noble gases. This implies that the trial wave function is better in the case of noble gases. To explain this phenomenon, it is important to realize that for closed shell systems, there is only one configuration with the lowest possible non-interacting energy. For open shell systems, there are several ways of configuring the electrons which produce the same non-interacting energy. This is due to the fact that the single particle energy is independent of the angular and azimuthal quantum numbers l and m introduced in Section 1.1.

For instance, Beryllium has two electrons in the $n = 1$ state and two in the $n = 2$ state. The trial wave function used in this thesis uses the $m = l = 0$ states only for beryllium, however, in light of the previous paragraph, the wave function suffers from the fact that the $l = 1$ states are equally energy efficient. In order to have an equally optimal ansatz in the case of alkaline earth metals as in the case of noble gases, the contributions from these equally efficient states must be included.

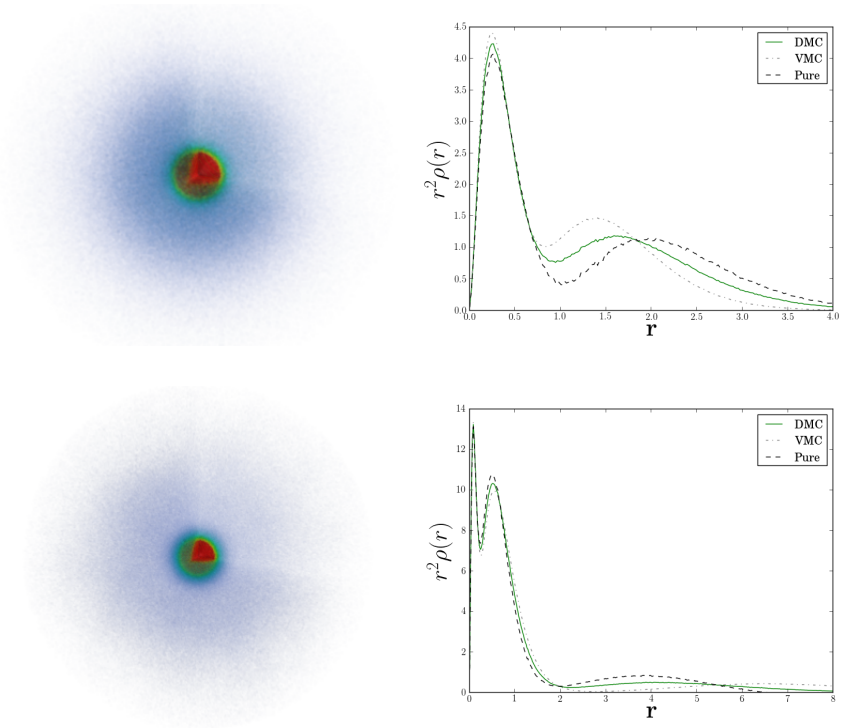


Figure 1.5: Three dimensional one-body density (left column) and angular averaged radii (right column) for alkaline earth metals; Beryllium (top) and Magnesium (bottom). The color bar shows increasing values from left to right. Notice that it is not the radial one-body density (which is too steep to reveal characteristics) in the right column, but the electron. Compared to the noble gases in Figure 1.6, the alkaline earth metals have a surrounding dispersed probability cloud due to the broken closed shell symmetry. The element is thus more unstable and potent for chemical reactions and molecular formations through covalent - and ionic bonds [12]. Red and blue color implies a low and high electron density respectively.

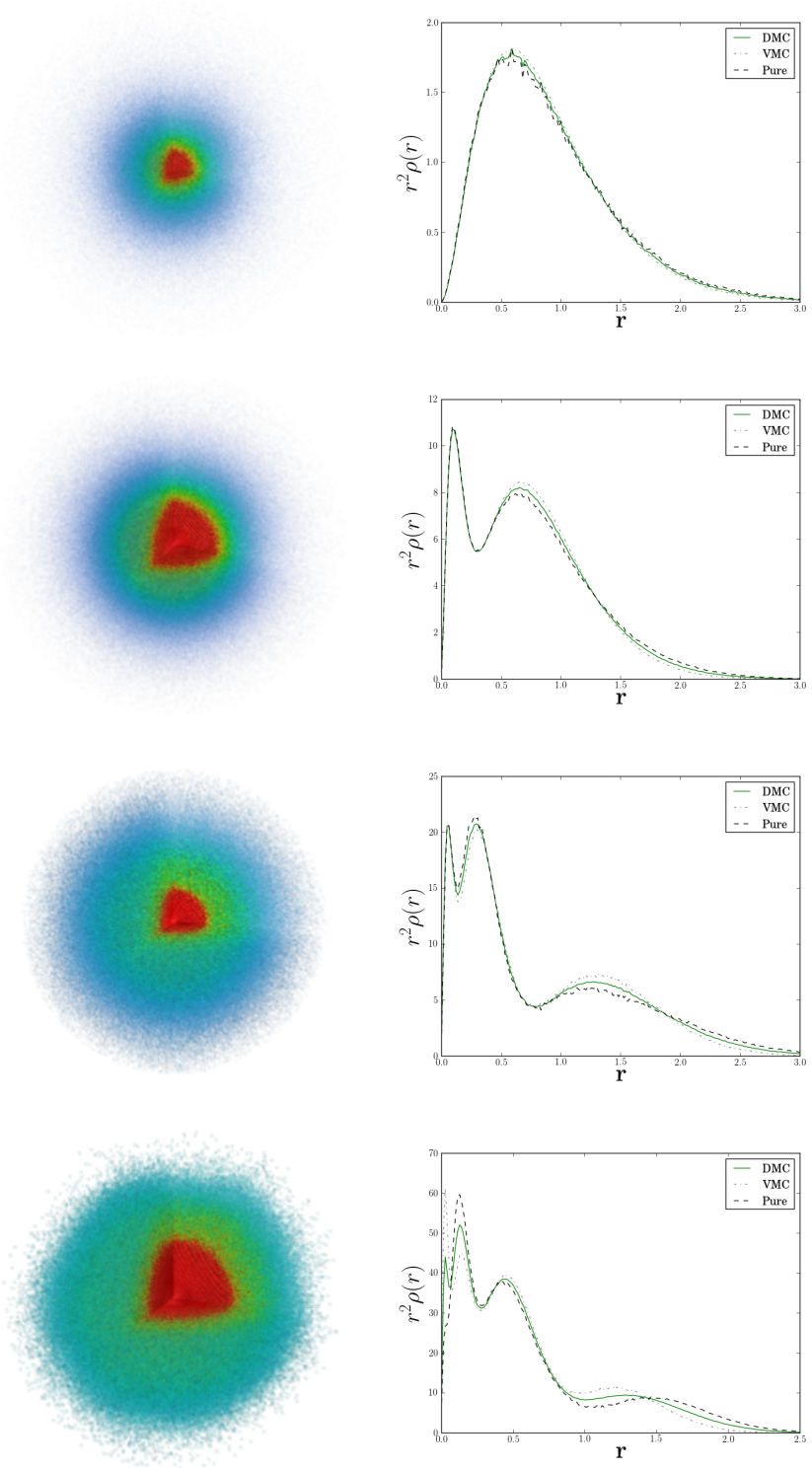


Figure 1.6: One-body densities for noble gases. Counting top to bottom: Helium, Neon, Argon and Krypton. A quarter of the spherical density is removed to present a better view of the core. Red and blue color implies a low and high electron density respectively. Note that the radial densities are scaled with r^2 in order to reveal their differences. Unlike previous figures, the left and right column should thus not agree.

Molecule	R	E_{VMC}	E_{DMC}	Expt.	ϵ_{rel}
H_2	1.4	-1.1551(3)	-1.1745(3)	-1.1746	$8.51 \cdot 10^{-5}$
Li_2	5.051	-14.743(3)	-14.988(2)	-14.99544	$4.96 \cdot 10^{-4}$
Be_2	4.63	-28.666(5)	-29.301(5)	$-29.33854(5)$	$1.28 \cdot 10^{-3}$
B_2	3.005	-47.746(7)	-49.155(5)	-49.4184	$5.33 \cdot 10^{-3}$
C_2	2.3481	-72.590(8)	-74.95(1)	$-75.923(5)$	$1.28 \cdot 10^{-2}$
N_2	2.068	-102.78(1)	-106.05(2)	-109.5423	$3.19 \cdot 10^{-2}$
O_2	2.282	-143.97(2)	-148.53(2)	-150.3268	$1.2 \cdot 10^{-2}$

Table 1.2: Ground state energies for homonuclear diatomic molecules calculated using VMC and DMC. The distance between the atoms R are taken from Ref. [13] for H_2 and from Ref. [14] for Li_2 to O_2. The experimental energies are taken from Ref. [13] for H_2 and from Ref. [15] for Li_2 to O_2. As expected DMC is closer to the experimental energy than VMC. $\epsilon_{\text{rel}} = |E_{\text{DMC}} - \text{Expt.}|/|\text{Expt.}|$ denotes the relative error between the DMC result and the experimental value. As expected, this error increase with the atomic number.

1.4 Homonuclear Diatomic Molecules

The focus regarding homonuclear diatomic molecules, from here on referred to as molecules, has been similar to the focus on atoms, with the exception of parameterizing atomic forces which can be applied in molecular dynamics simulations. The implementation of molecular systems were achieved by adding 200 lines of code. This fact by it self represents a successful result regarding the code structure. For details regarding the transformation from atomic - to molecular systems, see Section 1.1.3.

1.4.1 Ground State Energies

Table 1.2 lists the VMC and DMC results with the corresponding experimental energies for H_2 through O_2. As expected, the two particle result is very close to the experimental value with the same precision as the result for the Helium atom in Table 1.1. The relative error from the experimental energy increase with the atomic number, and is far higher than the errors in the case of pure atoms. This is a result of the trial wave function being worse due to the fact that it does not account for the nucleus-nucleus interaction term in the molecular Hamiltonian. Nevertheless, taking the simple nature of the trial wave function into consideration, the calculated energies are satisfyingly close to the experimental ones.

As with atoms, these energies were calculated on a single node, resulting in a rather big statistical error in DMC. Doing the calculations on a supercomputer with an increase in the number of walkers could decrease this error.

1.4.2 One-body densities

Figure 1.7 presents the one-body densities of Li_2, Be_2 and O_2. The densities have strong peaks located at a distance equal to half of the listed core separation R, indicating that the nucleus interaction still

dominates the general shape of the distributions. From the figure it is clear that most of the electrons are on the side facing the opposite core, leading to the conclusion that the molecules share a covalent bond [12]. This is especially clear in the case of the oxygen molecule, where there is a small formation of electrons on the inner side of the nuclei.

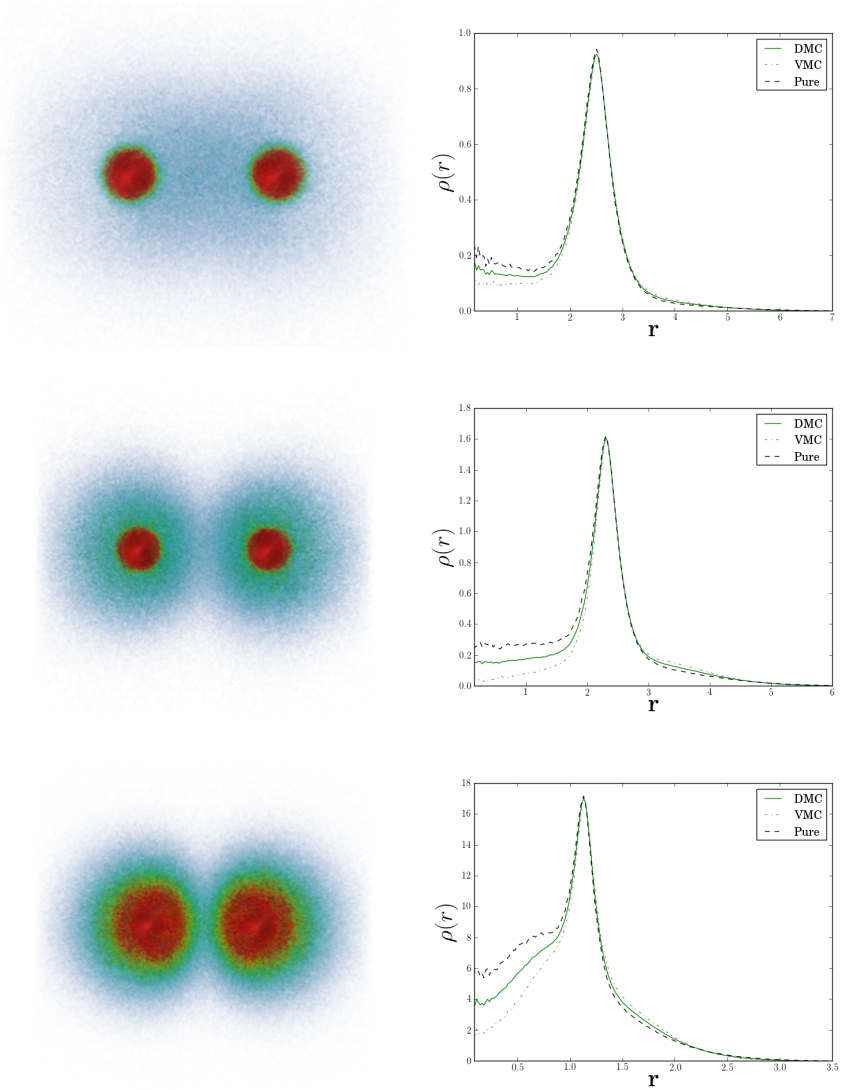


Figure 1.7: One-body densities of Li_2 (top), Be_2 (middle) and O_2 (bottom). The figures to the left are spherical densities sliced through the middle to reveal the core structure. The figures to the right are radial one-body densities projected on the nucleus-nucleus axis. Red and blue color indicates a high and low electron density respectively. The right-hand figures are symmetric around the origin.

1.4.3 Parameterizing Forces

In molecular dynamics, it is custom to use the *Lennard Jones 12-6 potential* as an ansatz to the interaction between pairs of atoms [16, 17]

$$V(R) = 4\epsilon\left(\left(\frac{\sigma}{R}\right)^{12} - \left(\frac{\sigma}{R}\right)^{6}\right), \tag{1.31}$$

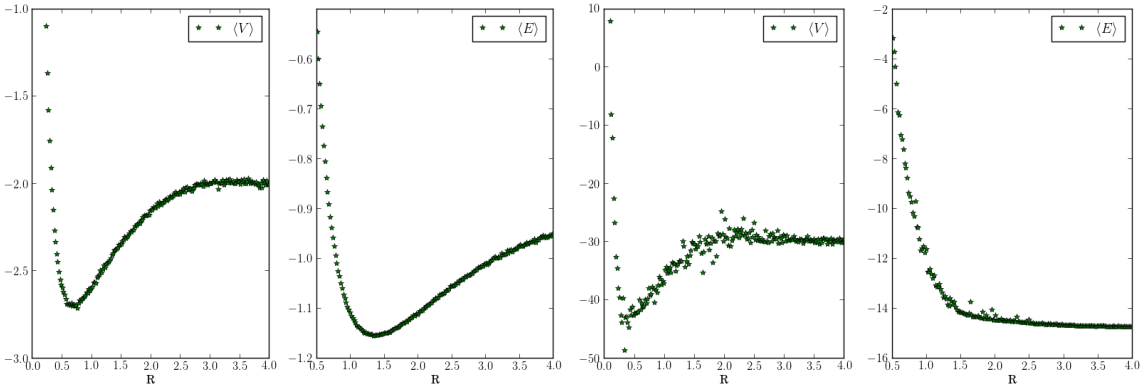
where ϵ and σ are parameters which can be fit to a given system.

However, the force field can be parameterized in greater detail using QMC calculations, resulting in a more precise molecular dynamics simulation [18]. The quantity of interest is the *force*, that is, the gradient of the potential. The classical potential in molecular dynamics does not correspond to the potential in the Schrödinger equation, due to the fact that the kinetic energy contribution from the electrons is not counted as part of the total kinetic energy in the molecular dynamics simulation. Hence it is the total energy of the Schrödinger equation which corresponds to the potential energy in molecular dynamics. In the case of diatomic molecules this means that

$$F_{\mathrm{MD}} = \frac{\mathrm{d}\langle E\rangle}{\mathrm{d}R}. \tag{1.32}$$

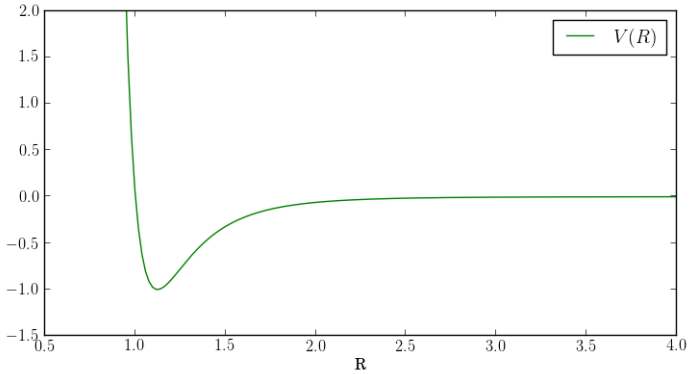
Expressions for this derivative can be obtained in ab-initio methods by using the Hellmann-Feynman theorem [18]. However, the derivative can be approximated by the slope of the energy in Figure 1.8. The figure shows that there are clear similarities between the widely used Lennard-Jones 12-6 potential and the results of QMC calculations done in this thesis, leading to the conclusion that the current state of the code can in fact produce approximations to atomic forces for use in molecular dynamics.

For more complicated molecules, modelling the force using a single parameter R does not serve as a good approximation. However, the force can be found as a function of several angles, radii, etc., which in turn can be used to parameterize a more complicated molecular dynamics potential. An example of such a potential is the *ReaxFF* potential for hydrocarbons [19].



(a) H_2 (b) Li_2

Figure 1.8: Top figures: The distance between the atoms R vs. the potential and total energy calculated using QMC. To the left: H_2. To the right: Li_2. It is evident that there exist a well-defined minima in the energy in the case of Hydrogen. For Lithium this is not the case, which is expected since Lithium does not appear naturally in a diatomic gas phase, but rather as an ionic compound in other molecules [12]. Bottom figure: The general shape of the Lennard-Jones potential commonly used in molecular dynamics simulations as an approximation to the potential between atoms. The top figures clearly resemble the Lennard-Jones potential, leading to the conclusion that QMC calculations can be used to parameterize a more realistic potential.



(c) The Lennard-Jones 12-6 potential.

2

Conclusions

The focus of this thesis was to develop an efficient and general Quantum Monte-Carlo (QMC) solver which could simulate various systems with a large number of particles. This was achieved by using object oriented C++, resulting in $\sim$ 15000 lines of code. The code was redesigned a total of four times. The final structure was carefully planned in advance of the coding process.

It became apparent that in order to maintain efficiency for a high number of particles, closed form expressions for the single particle wave function derivatives were necessary. For two dimensional quantum dots, 112 of these expressions were needed to simulate the 56-particle system. Needless to say, this process had to be automated. This was achieved by using *SymPy*, an open source symbolic algebra package for Python, wrapped in a script which generated all the C++ necessary code within seconds. This task is described in detail in Appendix C. A total of 252 expressions were generated.

As the solver became more and more flexible, the dire need of a control script arose. Hence the current state of the code is 100% controlled by a Python script. The script translates configuration files to instructions which are fed to the pre-compiled C++ program. In addition, the script sets up the proper environment for the simulation, that is, it makes sure the simulation output is stored in a folder stamped with the current date and time, as well as with a supplied name tag. The current version of the code and the configuration file(s) used are copied into the folder. All in all, this made it possible to run an immense amount of different simulations without loosing track of the data.

In order to handle the increase in data, a script which automatically converted the output files to Latex tables were written. However, this would not uncover whether or not the simulations had converged. An additional script was thus written, which made the process of visualizing immense amounts of data very easy. Moreover, the data could be analyzed real-time, which meant that the failing simulations could be aborted before they were complete, saving a lot of time. Expanding the script to handle new output files was by design unproblematic. This script is covered in high detail in Appendix B.

Several Master projects over the years have involved QMC simulations of some sort, like Variational Monte-Carlo (VMC) calculations of two dimensional quantum dots up to 42 particles [20], and VMC calculations of atoms up to Silicon (14 particles) [21]. In this thesis, the step was taken to full Diffusion Monte-Carlo (DMC) studies of both systems, adding three dimensional - and double-well quantum dots in addition to homonuclear diatomic molecules. Additionally, the simulation sizes were increased to 56 electrons and Krypton (36 particles) for two dimensional quantum dots and atoms, respectively.

The optimization of the code was done by profiling the code, focusing on the parts which took the most time. Due to the general structure of the code, the function responsible for diffusing the particles used almost all the runtime. This function is at the core of Adaptive Stochastic Gradient Descent (ASGD), VMC and DMC. In other words, the task of optimization the full code decreased down to the task of

optimizing this specific function. By optimizing one part of the diffusion process at the time, the final runtime was successfully reduced to 5% of the original.

Having successfully implemented five different systems demonstrates the code's generality. The implementation of molecules and the double-well was done by adding no more than 200 lines of code. Additionally, implementing the three dimensional quantum dots was done in an afternoon. Overall the code has lived up to every single one of the initial aims, with the exception of simulating bosons. Studying new fermionic systems were considered more interesting, hence specific implementations for bosons were abandoned.

For quantum dots, both VMC and DMC perform very well. The results from Full Configuration Interaction [22] for two particles, which are believed to be very close to the exact solution, are reproduced to five digits using DMC, with the VMC result being a little higher (2-3 digits). For atomic systems, the gap between VMC and DMC increase. This is expected since the trial wave function uses real-valued solid harmonics instead of the complex spherical harmonics. Nevertheless, DMC performs very well, reproducing the experimental results for two particles with an error at the level of 10^{-4}. For heavier atoms, the error increases somewhat, however, taking into consideration the simple trial wave function, the results are remarkably good.

Moreover, it was found that the radial distributions of two - and three dimensional quantum dots with the same number of closed shells were remarkably similar. This, however, is only the case at high frequencies. For lower frequencies, both systems transitioned into a Wigner crystallized state, however, the distributions were no longer similar. This breaking of symmetry is an extremely interesting phenomenon, which can be studied in greater detail in order to say something general about how the number of spatial dimensions affect systems of confined electrons.

It is clear that the energy of H_2 graphed as a function of the core separation resembles the Lennard-Jones 12-6 potential. This demonstrates that the code can be used to parameterize potential energies for use in molecular dynamics simulations, however, to produce realistic potentials, support for more complicated molecules must be implemented.

Prospects and future work

Shortly after handing in this thesis, I will focus my effort on studying the relationship between two - and three dimensional quantum dots in higher detail. The goal is to publish my findings, and possibly say something general regarding how the number of dimensions affect a system of confined electrons.

Additionally, the double-well quantum dot will be studied in higher detail using realistic values for the parameters. These results can then be benchmarked with the results of Sigve Bøe Skattum and his *Multi-configuration Time Dependent Hartree-Fock* solver.

My supervisor and I will at the same time work with implementing a momentum space version of QMC. This has the potential of describing nuclear interactions in great detail [23].

I will continue my academic career as a PhD student in the field of *multi-scale physics*. The transition from QMC to molecular dynamics will thus be of high interest. The plan is to expand the code to general molecules. However, in order to maintain a reasonable precision, the single particle wave functions needs to be optimized. Hence implementing a Hartree-Fock solver [24] or using a Coupled Cluster wave function [25] will be prioritized.

Appendices

A

Dirac Notation

Calculations involving sums over inner products of orthogonal states are common in Quantum Mechanics. This due to the fact that eigenfunctions of Hermitian operators, which is the kind of operators which represent observables [2], are necessarily orthogonal [26]. These inner-products will in many cases result in either zero or one, i.e. the *Kronecker-delta* function δ_{ij}; explicitly calculating the integrals is unnecessary.

Dirac notation is a notation in which quantum states are represented as abstract components of a *Hilbert space*, i.e. an inner product space. This implies that the inner-product between two states are represented by these states alone, without the integral over a specific basis, which makes derivations a lot cleaner and general in the sense that no specific basis is needed.

Extracting the abstract state from a wave function is done by realizing that the wave function can be written as the inner product between the position basis eigenstates $|x\rangle$ and the abstract quantum state $|\psi\rangle$

$$\psi(x) = \langle r, \psi\rangle \equiv \langle x|\psi\rangle = \langle x| \times |\psi\rangle \,.$$

The notation is designed to be simple. The right hand side of the inner product is called a *ket*, while the left hand side is called a *bra*. Combining both of them leaves you with an inner product bracket, hence Dirac notation is commonly referred to as *bra-ket* notation.

To demonstrate the simplicity introduced with this notation, imagine a coupled two-level spin-$\frac{1}{2}$ system in the following state

$$|\chi\rangle = N\Big[\,|\uparrow\downarrow\rangle - i\,|\downarrow\uparrow\rangle\,\Big] \tag{A.1}$$

$$\langle\chi| = N\Big[\,\langle\uparrow\downarrow| + i\,\langle\downarrow\uparrow|\,\Big] \tag{A.2}$$

Using the fact that both the $|\chi\rangle$ state and the two-level spin states should be orthonormal, the normalization factor can be calculated without explicitly setting up any integrals

$$\begin{aligned}
\langle\chi|\chi\rangle &= N^2\Big[\langle\uparrow\downarrow| + i\langle\downarrow\uparrow|\Big]\Big[|\uparrow\downarrow\rangle - i|\downarrow\uparrow\rangle\Big] \\
&= N^2\Big[\langle\uparrow\downarrow|\uparrow\downarrow\rangle + i\langle\downarrow\uparrow|\uparrow\downarrow\rangle - i\langle\uparrow\downarrow|\downarrow\uparrow\rangle + \langle\downarrow\uparrow|\downarrow\uparrow\rangle\Big] \\
&= N^2\Big[1 + 0 - 0 + 1\Big] \\
&= 2N^2 \\
&= 1,
\end{aligned}$$

This implies the trivial solution $N = 1/\sqrt{2}$. With this powerful notation at hand, important properties such as the *completeness relation* of a set of states can be shown. A standard strategy is to start by expanding one state $|\phi\rangle$ in a complete set of different states $|\psi_i\rangle$:

$$\begin{aligned}
|\phi\rangle &= \sum_i c_i |\psi_i\rangle \\
\langle\psi_k|\phi\rangle &= \sum_i c_i \underbrace{\langle\psi_k|\psi_i\rangle}_{\delta_{ik}} \\
&= c_k \\
|\phi\rangle &= \sum_i \langle\psi_i|\phi\rangle |\psi_i\rangle \\
&= \left[\sum_i |\psi_i\rangle\langle\psi_i|\right]|\phi\rangle
\end{aligned}$$

which implies that

$$\sum_i |\psi_i\rangle\langle\psi_i| = \mathbb{1} \tag{A.3}$$

for any complete set of orthonormal states $|\psi_i\rangle$. Calculating the corresponding identity for a continuous basis like e.g. the position basis yields

$$\int |\psi(x)|^2 dx = 1 \tag{A.4}$$

$$\begin{aligned}
\int |\psi(x)|^2 dx &= \int \psi^*(x)\psi(x)\mathrm{d}x \\
&= \int \langle\psi|x\rangle\langle x|\psi\rangle\,\mathrm{d}x \\
&= \langle\psi|\left[\int |x\rangle\langle x|\,dx\right]|\psi\rangle\,.
\end{aligned} \tag{A.5}$$

Combining eq. A.4 and eq. A.5 with the fact that $\langle\psi|\psi\rangle = 1$ yields the identity

$$\int |x\rangle\langle x|\,dx = \mathbb{1}. \tag{A.6}$$

Looking back at the introductory example, this identity is exactly what is extracted when a wave function is described as an inner product instead of an explicit function.

B

DCViz: Visualization of Data

With a code framework increasing in complexity comes an increasing need for tools to ease the interface between the code and the developer(s). In computational science, a must-have tool is a tool for efficient visualization of data; there is only so much information a single number can hold. To supplement the QMC code, a visualization tool named DCViz (**D**ynamic **C**olumn data **Vi**sualiz**e**r) has been developed.

The tool is written in Python, designed to plot data stored in columns. The tool is not designed explicitly for the QMC framework, and has been successfully applied to codes by several Master students at the time of this thesis. The plot library used is *Matplotlib* [27] with a graphical user interface coded using *PySide* [28]. The data can be plotted dynamically at a specified interval, and designed to be run parallel to the main application, e.g. DMC.

DCViz is available at `https://github.com/jorgehog/DCViz`

B.1 Basic Usage

The application is centered around the `mainloop()` function, which handles the extraction of data, the figures and so on. The virtual function `plot()` is where the user specifies how the data is transformed into specified figures by creating a subclass which overloads it. The superclass handles everything from safe reading from dynamically changing files, efficient and safe re-plotting of data, etc. automatically. The tool is designed to be simple to use by having a minimalistic interface for implementing new visualization classes. The only necessary members to specify in a new implementation is described in the first three sections, from where the remaining sections will cover additional support.

The figure map

Represented by the member variable `figMap`, the figure map is where the user specifies the figure setup, that is, the names of the main-figures and their sub-figures. Consider the following figure map:

```
figMap = {"mainFig1": ["subFig1", "subFig2"], "mainFig2": []}
```

This would cause DCViz to create two main-figures `self.mainFig1` and `self.mainFig2`, which can be accessed in the plot function. Moreover, the first main-figure will contain two sub-figures accessible through `self.subFig1` and `self.subFig2`. These sub-figures will be stacked vertically if not `stack="H"` is specified, in which they will be stacked horizontally.

The name tag

Having to manually combine a data file with the correct subclass implementation is annoying, hence DCViz is designed to automate this process. Assuming a dataset to be specified by a unique pattern of characters, i.e. a *name tag*, this name tag can be tied to a specific subclass implementation, allowing DCViz to automatically match a specific filename with the correct subclass. Name tags are

```
nametag = "DMC_out_\d+\.dat"
```

The name tag has *regular expressions* (regex) support, which in the case of the above example allows DCViz to recognize any filename starting with "DMC_out_" followed by any integer and ending with ".dat" as belonging to this specific subclass. This is a necessary functionality, as filenames often differ between runs, that is, the filename is specified by e.g. a time step, which does not fit an absolute generic expression. The subclasses must be implemented in the file `DCViz_classes.py` in order for the automagic detection to work.

To summarize, the name tag invokes the following functionality

```
import DCvizWrapper, DCViz_classes

#DCViz automagically executes the mainloop for the
#subclass with a nametag matching 'filename'
DCVizWrapper.main(filename)

#This would be the alternative, where 'specific_class' needs to be manually selected.
specificClass = DCViz_classes.myDCVizClass #myDCVizClass matches 'filename'
specificClass(filename).mainloop()
```

The plot function

Now that the figures and the name tag has been specified, all that remains for a fully functional DCViz instance is the actual plot function

```
def plot(self, data)
```

where `data` contains the data harvested from the supplied filename. The columns can then be accessed easily by e.g.

```
col1, col2, col3 = data
```

which can then in turn be used in standard Matplotlib functions with the figures from `figMap`.

Additional (optional) support

Additional parameters can be overloaded for additional functionality

`nCols`	The number of columns present in the file. Will be automatically detected unless the data is stored in binary format.
`skipRows`	The number of initial rows to skip. Will be automatically detected unless the data is stored as a single column.
`skipCols`	The number of initial columns to skip. Defaults to zero.
`armaBin`	Boolean flag. If set to true, the data is assumed to be stored in Armadillo's binary format (doubles). Number of columns and rows will be read from the file header.
`fileBin`	Boolean flag. If set to true, the data is assumed to be stored in binary format. The number of columns must be specified.

The LaTeXsupport is enabled if the correct packages are installed.

An example

```
#DCViz_classes.py

from DCViz_sup import DCVizPlotter #Import the superclass

class myTestClass(DCVizPlotter): #Create a new subclass
    nametag =  'testcase\d\.dat' #filename with regex support

    #1 figure with 1 subfigure
    figMap = {'fig1': ['subfig1']}

    #skip first row (must be supplied since the data is 1D).
    skipRows = 1

    def plot(self, data):
        column1 = data[0]

        self.subfig1.set_title('I have $\LaTeX$ support!')

        self.subfig1.set_ylim([-1,1])

        self.subfig1.plot(column1)

        #exit function
```

Families

A specific implementation can be flagged as belonging to a family of similar files, that is, files in the same directory matching the same name tag. Flagging a specific DCViz subclass as a family is achieved by setting the class member variable `isFamilyMember` to true. When a family class is initialized with a file, DCViz scans the file's folder for additional matches to this specific class. If several matches are found, all of these are loaded into the `data` object given as input to the plot function. In this case `data[i]` contains the column data of file i.

To keep track of which file a given data-set was loaded from, a list `self.familyFileNames` is created, where element i is the filename corresponding to `data[i]`. To demonstrate this, consider the following example

```
isFamilyMember = True
def plot(self, data)

    file1, file2 = data
    fileName1, fileName2 = self.familyFileNames

    col1_1, col_2_1 = file1
    col1_2, col_2_2 = file2
    #...
```

A class member string `familyName` can be overridden to display a more general name in the auto-detection feedback.

Families are an important functionality in the cases where the necessary data is spread across several files. For instance, in the QMC library, the radial distributions of both VMC and DMC are needed in order to generate the plots shown in figure **??** of the results chapter. These results may be generated in separate runs, which implies that they either needs to be loaded as a family, or be concatenated beforehand. Which dataset belongs to VMC and DMC can be extracted from the list of family file names.

All the previous mentioned functionality is available for families.

Family example

```
#DCViz_classes.py

from DCViz_sup import DCVizPlotter

class myTestClassFamily(DCVizPlotter):
    nametag =  'testcaseFamily\d\.dat' #filename with regex support

    #1 figure with 3 subfigures
    figMap = {'fig1': ['subfig1', 'subfig2', 'subfig3']}

    #skip first row of each data file.
    skipRows = 1

    #Using this flag will read all the files matching the nametag
    #(in the same folder.) and make them aviable in the data arg
    isFamilyMember = True
    familyName = "testcase"

    def plot(self, data):

        mainFig = self.fig1
        mainFig.suptitle('I have $\LaTeX$ support!')
        subfigs = [self.subfig1, self.subfig2, self.subfig3]

        #Notice that fileData.data is plotted (the numpy matrix of the columns)
        #and not fileData alone, as fileData is a 'dataGenerator' instance
        #used to speed up file reading. Alternatively, data[:] could be sent
        for subfig, fileData in zip(subfigs, data):
            subfig.plot(fileData.data)
            subfig.set_ylim([-1,1])
```

loading e.g. `testcaseFamily0.dat` would automatically load `testcaseFamily1.dat` etc. as well.

Dynamic mode

Dynamic mode in DCViz is enabled on construction of the object

```
DCVizObj = myDCVizClass(filename, dynamic=True)
DCVizObj.mainloop()
```

This flag lets the mainloop know that it should not stop after the initial plot is generated, but rather keep on reading and plotting the file(s) until the user ends the loop with either a keyboard-interrupt (which is caught and safely handled), or in the case of using the GUI, with the stop button.

In order to make this functionality more CPU-friendly, a `delay` parameter can be adjusted to specify a pause period in between re-plotting.

Saving figures to file

The generated figures can be saved to file by passing a flag to the constructor

```
DCVizObj = myDCVizClass(filename, toFile=True)
DCVizObj.mainloop()
```

In this case, dynamic mode is disabled and the figures will not be drawn on screen, but rather saved in a subfolder of the supplied filename's folder called `DCViz_out`.

B.1.1 The Terminal Client

The `DCVizWrapper.py` script is designed to be called from the terminal with the path to a datafile specified as command line input. From here it automatically selects the correct subclass based on the filename:

```
jorgen@teleport:~$ python DCVizWrapper.py ./ASGD_out.dat

[ Detector ] Found subclasses 'myTestClass', 'myTestClassFamily', 'EnergyTrail',
                              'Blocking', 'DMC_OUT', 'radial_out', 'dist_out',
                              'R_vs_E', 'E_vs_w', 'testBinFile', 'MIN_OUT'
[  DCViz   ] Matched [ASGD_out.dat] with [MIN_OUT]
[  DCViz   ] Press any key to exit
```

If the option `-d` is supplied, dynamic mode is activated:

```
jorgen@teleport:~$ python DCVizWrapper.py ./ASGD_out.dat -d

[ Detector ] Found subclasses ......
[  DCViz   ] Matched [ASGD_out.dat] with [MIN_OUT]
[  DCViz   ] Interrupt dynamic mode with CTRL+C
^C[  DCViz   ] Ending session...
```

Saving figures through the terminal client is done by supplying the flag `-f` to the command line together with a folder `aDir`, whose content will then be traversed recursively. For every file matching a DCViz name tag, the file data will be loaded and its figure(s) saved to `aDir/DCViz_out/`. In case of family members, only one instance needs to be run (they would all produce the same image), hence "family portraits" are taken only once:

```
jorgen@teleport:~$ python DCVizWrapper.py ~/scratch/QMC_SCRATCH/ -f

[ Detector ] Found subclasses ......
[  DCViz   ] Matched [ASGD_out.dat] with [MIN_OUT]
[  DCViz   ] Figure(s) successfully saved.
[  DCViz   ] Matched [dist_out_QDots2c1vmc_edge3.05184.arma] with [dist_out]
[  DCViz   ] Figure(s) successfully saved.
[  DCViz   ] Matched [dist_out_QDots2c1vmc_edge3.09192.arma] with [dist_out]
[  DCViz   ] Family portait already taken, skipping...
[  DCViz   ] Matched [radial_out_QDots2c1vmc_edge3.05184.arma] with [radial_out]
[  DCViz   ] Figure(s) successfully saved.
[  DCViz   ] Matched [radial_out_QDots2c1vmc_edge3.09192.arma] with [radial_out]
[  DCViz   ] Family portait already taken, skipping...
```

The terminal client provides extremely efficient and robust visualization of data. When e.g. blocking data from 20 QMC runs, the automated figure saving functionality is gold.

B.1.2 The Application Programming Interface (API)

DCViz has been developed to interface nicely with any Python script. Given a path to the data file, all that is needed in order to visualize it is to include the wrapper function used by the terminal client:

```
import DCVizWrapper as viz
dynamicMode = False #or true

...
#Generate some data and save it to the file myDataFile (including path)

#DCVizWrapper.main() automatically detects the subclass implementation
#matching the specified file. Thread safe and easily interruptable.
viz.main(myDataFile, dynamic=dynamicMode, toFile=toFile)
```

If on the other hand the data needs to be directly visualized without saving it to file, the pure API function `rawDataAPI` can be called directly with a numpy array `data`. If the plot should be saved to file, this can be enabled by supplying an arbitrary file-path (e.g. `/home/me/superDuper.png`) and setting `toFile=True`.

```
from DCViz_classes import myDCVizClass

#Generate some data
myDCVizObj = myDCVizClass(saveFileName, toFile=ToFile)
myDCVizObj.rawDataAPI(data)
```

The GUI

The script `DCVizGUI.py` sets up a GUI for visualizing data using DCViz. The GUI is implemented using PySide (python wrapper for Qt), and is designed to be simple. Data files are loaded from an open-file dialog (`Ctrl+s` for entire folders or `Ctrl+o` for individual files), and will appear in a drop-down menu once loaded labeled with the corresponding class name. The play button executes the main loop of the currently selected data file. Dynamic mode is selected though a check-box, and the pause interval is set by a slider (from zero to ten seconds). Dynamic mode is interrupted by pressing the stop button. Warnings can be disabled through the configuration file. See figure B.1 for a screenshot of the GUI in action.

The GUI can be opened from any Python script by calling the `main` function (should be threaded if used as part of another application). If a path is supplied to the function, this path will be default in all file dialogues. Defaults to the current working directory.

The following is a tiny script executing the GUI for a QMC application. If no path is supplied at the command line, the default path is set to the scratch path.

```
import sys, os
from pyLibQMC import paths #contains all files specific to the QMC library

#Adds DCVizGUI to the Python path
sys.path.append(os.path.join(paths.toolsPath, "DCViz", "GUI"))

import DCVizGUI

if __name__ == "__main__":

    if len(sys.argv) > 1:
        path = sys.argv[1]
    path = paths.scratchPath

    sys.exit(DCVizGUI.main(path))
```

The python script responsible for starting the QMC program and setting up the environments for simulations in this thesis automatically starts the GUI in the simulation main folder, which makes the visualizing the simulation extremely easy.

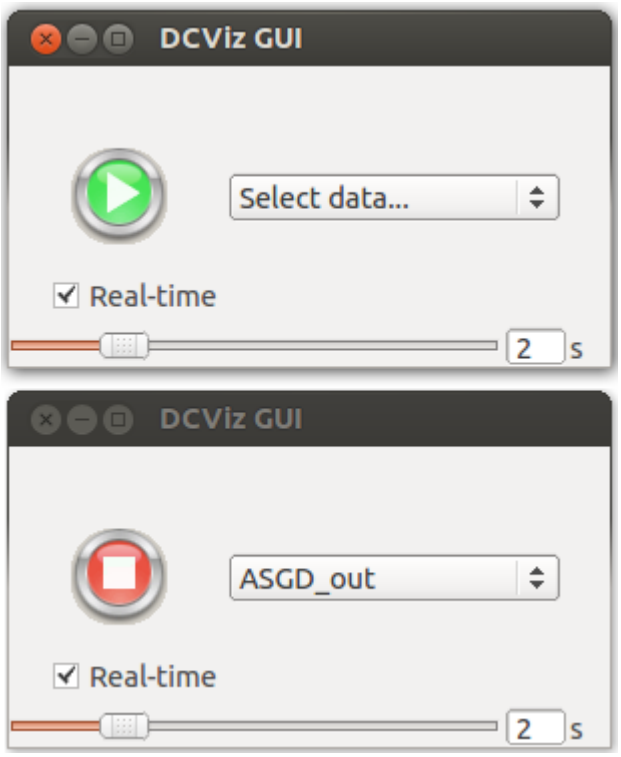


Figure B.1: Two consequent screen shots of the GUI. The first (top) is taken directly after the detector is finished loading files into the drop-down menu. The second is taken directly after the job is started.

Alternatively, `DCVizGUI.py` can be executed directly from the terminal with an optional default path as first command line argument.

The following is the terminal feedback supplied from opening the GUI

```
 .../DCViz/GUI$ python DCVizGUI.py
[ Detector ]:  Found subclasses 'myTestClass', 'myTestClassFamily', 'EnergyTrail',
'Blocking', 'DMC_OUT', 'radial_out', 'dist_out', 'testBinFile', 'MIN_OUT'
[   GUI    ]:  Data reset.
```

Selecting a folder from the open-folder dialog initializes the detector on all file content

```
[ Detector ]:  matched [  DMC_out.dat  ] with [    DMC_OUT    ]
[ Detector ]:  matched [  ASGD_out.dat ] with [    MIN_OUT    ]
[ Detector ]:  matched [blocking_DMC_out.dat] with [    Blocking   ]
[ Detector ]:  'blocking_MIN_out0_RAWDATA.arma' does not match any DCViz class
[ Detector ]:  'blocking_DMC_out_RAWDATA.arma' does not match any DCViz class
[ Detector ]:  matched [blocking_VMC_out.dat] with [    Blocking   ]
```

Executing a specific file selected from the drop-down menu starts a threaded job, hence several non-dynamic jobs can be ran at once. The limit is set to one dynamic job pr. application due to the high CPU cost (in case of a low pause timer).

The terminal output can be silenced through to configuration file to not interfere with the standard output of an application. Alternatively, the GUI thread can redirect its standard output to file.

C

Auto-generation with SymPy

"*SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.*"

- The SymPy Home Page [29]

Thie aim of this appendix will be on using SymPy to calculate closed form expressions for single particle wave functions needed to optimize the calculations of the Slater gradient and Laplacian. For systems of many particles, it is crucial to have these expressions in order for the code to remain efficient.

Calculating these expressions by hand is a waste of time, given that the complexity of the expressions is proportional to the magnitude of the quantum number, which again scales with the number of particles, and little new insights are gained from doing the calculations. In the case of a 56 particle Quantum Dot, the number of unique derivatives involved in the simulation is 112.

C.1 Usage

SymPy is, as described in the introductory quote, designed to be simple to use. This section will cover the basics needed to calculate gradients and Laplacians, auto-generating C++ - and Latex code.

C.1.1 Symbolic Algebra

In order for SymPy to recognize e.g. `x` as a symbol, that is, a *mathematical variable*, special action must be made. In contrast to programming variables, symbols are not initialized to a value. Initializing symbols can be done in several ways, the two most common are listed below

```
In [1]: from sympy import Symbol, symbols

In [2]: x = Symbol('x')

In [3]: y, z = symbols('y z')

In [4]: x*x+y
Out[4]: 'x**2 + y'
```

The `Symbol` function handles single symbols, while `symbols` can initialize several symbols simultaneously. The string argument might seem redundant, however, this represents the *label* displayed using print functions, which is neat to control. In addition, key word arguments can be sent to the symbol functions, flagging variables as e.g. positive, real, etc.

```
In [1]: from sympy import Symbol, symbols, im

In [2]: x2 = Symbol('x^2', real=True, positive=True) #Flagged as real. Note the label.

In [3]: y, z = symbols('y z') #Not flagged as real

In [4]: x2+y #x2 is printed more nicely given a describing label
Out[4]: 'x^2 + y'

In [5]: im(z) #Imaginary part cannot be assumed to be anything.
Out[5]: 'im(z)'

In [6]: im(x2) #Flagged as real, the imaginary part is zero.
Out[6]: 0
```

C.1.2 Exporting C++ and Latex Code

Exporting code is extremely simple: SymPy functions exist in the `sympy.printing` module, which simply takes a SymPy expression on input and returns the requested code-style equivalent. Consider the following example

```
In [1]: from sympy import symbols, printing, exp

In [2]: x, x2 = symbols('x x^2')

In [3]: printing.ccode(x*x*x*x*exp(-x2*x))
Out[3]: 'pow(x, 4)*exp(-x*x^2)'

In [4]: printing.ccode(x*x*x*x)
Out[4]: 'pow(x, 4)'

In [5]: print printing.latex(x*x*x*x*exp(-x2))
\frac{x^{4}}{e^{x^{2}}}
```

The following expression is the direct output from line five compiled in Latex

$$\frac{x^4}{e^{x^2}}$$

C.1.3 Calculating Derivatives

The $2s$ orbital from hydrogen (not normalized) is chosen as an example for this section

$$\phi_{2s}(\vec{r}) = (Zr - 2)e^{-\frac{1}{2}Zr} \tag{C.1}$$

$$r^2 = x^2 + y^2 + z^2 \tag{C.2}$$

Calculating the gradients and Laplacian is very simply by using the `sympy.diff` function

```
In [1]: from sympy import symbols, diff, exp, sqrt

In [2]: x, y, z, Z = symbols('x y z Z')

In [3]: r = sqrt(x*x + y*y + z*z)

In [4]: r
Out[4]: '(x**2 + y**2 + z**2)**(1/2)'

In [5]: phi = (Z*r - 2)*exp(-Z*r/2)

In [6]: phi
Out[6]: '(Z*(x**2 + y**2 + z**2)**(1/2) - 2)*exp(-Z*(x**2 + y**2 + z**2)**(1/2)/2)'

In [7]: diff(phi, x)
Out[7]: '-Z*x*(Z*(x**2 + y**2 + z**2)**(1/2) - 2)*exp(-Z*(x**2 + y**2 + z**2)**(1/2)/2)
    /(2*(x**2 + y**2 + z**2)**(1/2)) + Z*x*exp(-Z*(x**2 + y**2 + z**2)**(1/2)/2)/(x**2 +
    y**2 + z**2)**(1/2)'
```

Now, this looks like a nightmare. However, SymPy has great support for simplifying expressions through factorization, collecting, substituting etc. The following code demonstrated this quite nicely

```
1  ...
2
3  In [6]: phi
4  Out[6]: '(Z*(x**2 + y**2 + z**2)**(1/2) - 2)*exp(-Z*(x**2 + y**2 + z**2)**(1/2)/2)'
5
6  In [7]: from sympy import factor, Symbol, printing
7
8  In [8]: R = Symbol('r') #Creates a symbolic equivalent of the mathematical r
9
10 In [9]: diff(phi, x).factor() #Factors out common factors
11 Out[9]: '-Z*x*(Z*(x**2 + y**2 + z**2)**(1/2) - 4)*exp(-Z*(x**2 + y**2 + z**2)**(1/2)/2)
       /(2*(x**2 + y**2 + z**2)**(1/2))'
12
13 In [10]: diff(phi, x).factor().subs(r, R) #replaces (x^2 + y^2 + z^2)^(1/2) with r
14 Out[10]: '-Z*x*(Z*r - 4)*exp(-Z*r/2)/(2*r)'
15
16 In [11]: print printing.latex(diff(phi, x).factor().subs(r, R))
17 - \frac{Z x \left(Z r -4\right)}{2 r e^{\frac{1}{2} Z r}}
```

This version of the expression is much more satisfying to the eye. The output from line 11 compiled in Latex is

$$-\frac{Zx\left(Zr-4\right)}{2re^{\frac{1}{2}Zr}}$$

SymPy has a general method for simplifying expressions `sympy.simplify`, however, this function is extremely slow and does not behave well on general expressions. SymPy is still young, so nothing can be expected to work perfectly. Moreover, in contrast to *Wolfram Alpha* and *Mathematica*, SymPy is open source, which means that much of the work, if not all of the work, is done by ordinary people on their spare time. The ill behaving simplify function is not really a great loss; full control for a Python programmer is never considered a bad thing, whether it is enforced or not.

Estimating the Laplacian is just a matter of summing double derivatives

```
...

In [12]: (diff(diff(phi, x), x) +
    ....:  diff(diff(phi, y), y) +
    ....:  diff(diff(phi, z), z)).factor().subs(r, R)
Out[12]: 'Z*(Z**2*x**2 + Z**2*y**2 + Z**2*z**2 - 10*Z*r + 16)*exp(-Z*r/2)/(4*r)'

In [13]: (diff(diff(phi, x), x) + #Not quite satisfying.
    ....:  diff(diff(phi, y), y) + #Let's collect the 'Z' terms.
    ....:  diff(diff(phi, z), z)).factor().collect(Z).subs(r, R)
Out[13]: 'Z*(Z**2*(x**2 + y**2 + z**2) - 10*Z*r + 16)*exp(-Z*r/2)/(4*r)'

In [14]: (diff(diff(phi, x), x) + #Still not satisfying.
    ....:  diff(diff(phi, y), y) + #The r^2 terms needs to be substituted as well.
    ....:  diff(diff(phi, z), z)).factor().collect(Z).subs(r, R).subs(r**2, R**2)
Out[14]: 'Z*(Z**2*r**2 - 10*Z*r + 16)*exp(-Z*r/2)/(4*r)'

In [15]: (diff(diff(phi, x), x) + #Let's try to factorize once more.
    ....:  diff(diff(phi, y), y) +
    ....:  diff(diff(phi, z), z)).factor().collect(Z).subs(r, R).subs(r**2, R**2).factor()
Out[15]: 'Z*(Z*r - 8)*(Z*r - 2)*exp(-Z*r/2)/(4*r)'
```

Getting the right factorization may come across as tricky, but with minimal training this poses no real problems.

C.2 Using the auto-generation Script

The superclass `orbitalsGenerator` aims to serve as an interface with the QMC C++ `BasisFunctions` class, automatically generating the C++ code containing all the implementations of the derivatives for the given single particle states. The single particle states are implemented in the generator by subclasses overloading system specific virtual functions which will be described in the following sections.

C.2.1 Generating Latex code

The following methods are user-implemented functions used to calculate the expressions which are in turn automagically converted to Latex code. Once they are implemented, the following code can be executed in order to create the latex output

```
orbitalSet = HO_3D.HOOrbitals3D(N=40) #Creating a 3D harm. osc. object
orbitalSet.closedFormify()
orbitalSet.TeXToFile(outPath)
```

The constructor

The superclass constructor takes on input the maximum number of particles for which expressions should be generated and the name of the orbital set, e.g. `hydrogenic`. Calling a superclass constructor from a subclass constructor is done in the following way

```
class hydrogenicOrbitals(orbitalGenerator):

    def __init__(self, N):

        super(hydrogenicOrbitals, self).__init__(N, "hydrogenic")
        #...
```

makeStateMap

This function takes care of the mapping of a set of quantum numbers, e.g. nlm to a specific index i. The Python dictionary `self.stateMap` must be filled with values for every unique set of quantum numbers (not counting spin) in order for the Latex and C++ files to be created successfully. For the three-dimensional harmonic oscillator wave functions, the state map looks like this

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
n_x	0	0	0	1	0	0	0	1	1	2	0	0	0	0	1	1	1	2	2	3
n_y	0	0	1	0	0	1	2	0	1	0	0	1	2	3	0	1	2	0	1	0
n_z	0	1	0	0	2	1	0	1	0	0	3	2	1	0	2	1	0	1	0	0

setUpOrbitals

Within this function, the orbital elements corresponding to the quantum number mappings made in `makeStateMap` needs to be implemented in a matching order. The quantum numbers from `self.stateMap` are calculated prior to this function being called, and can thus be accessed in case they are needed, as is the case for the n-dependent exponential factor of the hydrogen-like orbitals.

The i'th orbital needs to be implemented in `self.orbitals[i]`, using the `x`, `y` and `z` variables defined in the superclass. For the three-dimensional harmonic oscillator, the function is simply

```
1 def setupOrbitals(self):
2
3     for i, stateMap in self.stateMap.items():
4         nx, ny, nz = stateMap
5
6         self.orbitals[i] = self.Hx[nx]*self.Hy[ny]*self.Hz[nz]*self.expFactor
```

where `self.Hx` and the exponential factor are implemented in the constructor. After the orbitals are created, the gradients and Laplacians cam be calculated by calling the `closedFormify()` function, however, unless the following member function is implemented, they are going to look messy.

simplifyLocal

As demonstrated in the previous example, SymPy expressions are messy when they are fresh out of the derivative functions. Since every system needs to be treated differently when it comes to cleaning up their expressions, this function is available. For hydrogen-like wave functions, the introductory example's strategy can be applied up to the level of Neon. Going higher will require more advanced strategies for cleaning up the expressions.

The expression and the corresponding set of quantum numbers are given on input. In addition, there is an input argument `subs`, which if set to false should make the function return the expression in terms of `x`, `y` and `z` without substituting e.g. $x^2 + y^2 = r^2$.

genericFactor

The method serves as convenient function for describing generic parts of the expressions, e.g. the exponentials, which are often reused. A set of quantum numbers are supplied on input in case the generic expression depends on these. In addition, a flag `basic` is supplied on input, which if set to true should, as in the simplify function, return the generic factor in Cartesian coordinates. This generic factor can

then easily be taken out of the Latex expressions and mentioned in the caption in order to clean up the expression tables.

__str__

This method is invoked by calling `str(obj)` on an arbitrary Python object `obj`. In the case of the orbital generator class, this string will serve as an introductory text to the latex output.

C.2.2 Generating C++ code

A class `CPPbasis` is constructed to supplement the orbitals generator class. This objects holds the empty shells of the C++ constructors and implementations. After the functions described in this section are implemented, the following code can be executed to generate the C++ files

```
orbitalSet = HO_3D.HOOrbitals3D(N=40) #Creating a 3D harm. osc. object
orbitalSet.closedFormify()
orbitalSet.TeXToFile(outPath)
orbitalSet.CPPToFile(outPath)
```

initCPPbasis

Sets up the variables in the `CPPbasis` object needed in order to construct the C++ file, such as the dimension, the name, the constructor input variables and the C++ class members. The following function is the implementation for the two-dimensional harmonic oscillator

```
def initCPPbasis(self):

    self.cppBasis.dim = 2

    self.cppBasis.setName(self.name)

    self.cppBasis.setConstVars('double* k',          #sqrt(k2)
                               'double* k2',         #scaled oscillator freq.
                               'double* exp_factor') #The exponential

    self.cppBasis.setMembers('double* k',
                             'double* k2',
                             'double* exp_factor',
                             'double H',             #The Hermite polynomial part
                             'double x',
                             'double y',
                             'double x2',            #Squared Cartesian coordinates
                             'double y2')
```

getCPre and getCreturn

The empty shell of the `BasisFunctions::eval` functions in the `CPPbasis` class is implemented as below

```
self.evalShell = """
double __name__::eval(const Walker* walker, int i) {

__necessities__

    //__simpleExpr__
```

```
7
8  __preCalc__
9      return __return__
10
11 }
12 """
```

where `__preCalc__` is a generated C++ expression returned from `getCpre()`, and `__return__` is the returned C++ expression from `getCreturn()`. The commented `__simpleExpr__` will be replaced by the expression in nicely formatted SymPy output code. `__necessities__` is automatically detected by the script, and represents the Cartesian variable expressions needed by the expressions.

The functions take a SymPy generated expression on input, i.e. an orbital, gradient or Laplacian, and the corresponding index of the expression i. The reason these functions are split into a precalculation and a return expression is purely cosmetic. Consider the following example output for the hydrogen-like wave functions:

```
1  double dell_hydrogenic_9_y::eval(const Walker* walker, int i) {
2
3      y = walker->r(i, 1);
4      z = walker->r(i, 2);
5
6      z2 = z*z;
7
8      //-y*(k*(-r^2 + 3*z^2) + 6*r)*exp(-k*r/3)/(3*r)
9
10     psi = -y*((*k)*(-walker->get_r_i2(i) + 3*z2) + 6*walker->get_r_i(i))/(3*walker->
           get_r_i(i));
11     return psi*(*exp_factor);
12
13 }
```

The `*exp_factor` is the precalculated $n = 3$ exponential which is then simply multiplied by the non-exponential terms before being returned. The commented line is a clean version of the full expression. The required Cartesian components are retrieved prior to the evaluation.

The full implementation of `getCpre()` and `getCreturn()` for the hydrogen-like wave functions are given below

```
1  def getCReturn(self, expr, i):
2       return "psi*(*exp_factor);"
3
4  def getCPre(self, expr, i):
5       qNums = self.stateMap[i]
6       return "    psi = %s;" % printing.ccode(expr/self.genericFactor(qNums))
```

makeOrbConstArg

Loading the generated `BasisFunctions` objects into the `Orbitals` object in the QMC code is rather a dull job, and is not designed to be done manually. The function `makeOrbConstArg` is designed to automate this process. This is best demonstrated by an example: Consider the following constructor of the hydrogen-like wave function's orbital class

```
1  basis_functions[0] = new hydrogenic_0(k, k2, exp_factor_n1);
2  basis_functions[1] = new hydrogenic_1(k, k2, exp_factor_n2);
3  //...
4  basis_functions[5] = new hydrogenic_5(k, k2, exp_factor_n3);
5  //..
```

```
6 basis_functions[14] = new hydrogenic_14(k, k2, exp_factor_n4);
7 //..
8 basis_functions[17] = new hydrogenic_17(k, k2, exp_factor_n4);
```

where `exp_factor_nk` represents $\exp(-Zr/k)$, which is saved as a pointer reference for reasons explained in Section **??**. The same procedure is applied to the gradients and the Laplacians as well, leaving a total of 90 sequential initializations. Everything needed in order to auto-generate the code is the following implementation

```
1 def makeOrbConstArgs(self, args, i):
2     n = self.stateMap[i][0]
3     args = args.replace('exp_factor', 'exp_fa-ctor_n%d' % n)
4     return args
```

which ensures that the input arguments to e.g. element 1 is `(k, k2, exp_factor_n2)`, since the single particle orbital `self.phi[1]` has a principle quantum number $n = 2$. The input argument `args` is the default constructor arguments set up the the `initCPPbasis`, and is in the case of hydrogen-like wave functions `(k, k2, exp_factor)`.

The tables listed in Appendix D, E and F are all generated within seconds using this framework. The generated C++ code for these span 8975 lines not counting blank ones.

D

Harmonic Oscillator Orbitals 2D

Orbitals are constructed in the following fashion:

$$\phi(\vec{r})_{n_x,n_y} = H_{n_x}(kx)H_{n_y}(ky)e^{-\frac{1}{2}k^2r^2}$$

where $k = \sqrt{\omega\alpha}$, with ω being the oscillator frequency and α being the variational parameter.

$H_0(kx)$	1
$H_1(kx)$	$2kx$
$H_2(kx)$	$4k^2x^2 - 2$
$H_3(kx)$	$8k^3x^3 - 12kx$
$H_4(kx)$	$16k^4x^4 - 48k^2x^2 + 12$
$H_5(kx)$	$32k^5x^5 - 160k^3x^3 + 120kx$
$H_6(kx)$	$64k^6x^6 - 480k^4x^4 + 720k^2x^2 - 120$
$H_0(ky)$	1
$H_1(ky)$	$2ky$
$H_2(ky)$	$4k^2y^2 - 2$
$H_3(ky)$	$8k^3y^3 - 12ky$
$H_4(ky)$	$16k^4y^4 - 48k^2y^2 + 12$
$H_5(ky)$	$32k^5y^5 - 160k^3y^3 + 120ky$
$H_6(ky)$	$64k^6y^6 - 480k^4y^4 + 720k^2y^2 - 120$

Table D.1: Hermite polynomials used to construct orbital functions

$\phi_0 \to \phi_{0,0}$	
$\phi(\vec{r})$	1
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 x$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 y$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 2\right)$

Table D.2: Orbital expressions HOOrbitals : 0, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_1 \to \phi_{0,1}$	
$\phi(\vec{r})$	y
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 xy$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\left(ky - 1\right)\left(ky + 1\right)$
$\nabla^2\phi(\vec{r})$	$k^2 y\left(k^2r^2 - 4\right)$

Table D.3: Orbital expressions HOOrbitals : 0, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_2 \to \phi_{1,0}$	
$\phi(\vec{r})$	x
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\left(kx - 1\right)\left(kx + 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 xy$
$\nabla^2\phi(\vec{r})$	$k^2 x\left(k^2r^2 - 4\right)$

Table D.4: Orbital expressions HOOrbitals : 1, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_3 \to \phi_{0,2}$	
$\phi(\vec{r})$	$2k^2y^2 - 1$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 x\left(2k^2y^2 - 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 y\left(2k^2y^2 - 5\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 6\right)\left(2k^2y^2 - 1\right)$

Table D.5: Orbital expressions HOOrbitals : 0, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_4 \to \phi_{1,1}$	
$\phi(\vec{r})$	xy
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-y\left(kx - 1\right)\left(kx + 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-x\left(ky - 1\right)\left(ky + 1\right)$
$\nabla^2\phi(\vec{r})$	$k^2 xy\left(k^2r^2 - 6\right)$

Table D.6: Orbital expressions HOOrbitals : 1, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_5 \rightarrow \phi_{2,0}$	
$\phi(\vec{r})$	$2k^2x^2 - 1$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2x\left(2k^2x^2 - 5\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2y\left(2k^2x^2 - 1\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 6\right)\left(2k^2x^2 - 1\right)$

Table D.7: Orbital expressions HOOrbitals : 2, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_6 \rightarrow \phi_{0,3}$	
$\phi(\vec{r})$	$y\left(2k^2y^2 - 3\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2y^2 - 3\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-2k^4y^4 + 9k^2y^2 - 3$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2 - 8\right)\left(2k^2y^2 - 3\right)$

Table D.8: Orbital expressions HOOrbitals : 0, 3. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_7 \rightarrow \phi_{1,2}$	
$\phi(\vec{r})$	$x\left(2k^2y^2 - 1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\left(kx - 1\right)\left(kx + 1\right)\left(2k^2y^2 - 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2y^2 - 5\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2 - 8\right)\left(2k^2y^2 - 1\right)$

Table D.9: Orbital expressions HOOrbitals : 1, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_8 \rightarrow \phi_{2,1}$	
$\phi(\vec{r})$	$y\left(2k^2x^2 - 1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2 - 5\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\left(ky - 1\right)\left(ky + 1\right)\left(2k^2x^2 - 1\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2 - 8\right)\left(2k^2x^2 - 1\right)$

Table D.10: Orbital expressions HOOrbitals : 2, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_9 \rightarrow \phi_{3,0}$	
$\phi(\vec{r})$	$x\left(2k^2x^2 - 3\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-2k^4x^4 + 9k^2x^2 - 3$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2 - 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2 - 8\right)\left(2k^2x^2 - 3\right)$

Table D.11: Orbital expressions HOOrbitals : 3, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{10} \rightarrow \phi_{0,4}$	
$\phi(\vec{r})$	$4k^4y^4 - 12k^2y^2 + 3$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2x\left(4k^4y^4 - 12k^2y^2 + 3\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2y\left(4k^4y^4 - 28k^2y^2 + 27\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 10\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$

Table D.12: Orbital expressions HOOrbitals : 0, 4. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{11} \rightarrow \phi_{1,3}$	
$\phi(\vec{r})$	$xy\left(2k^2y^2 - 3\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-y\left(kx - 1\right)\left(kx + 1\right)\left(2k^2y^2 - 3\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-x\left(2k^4y^4 - 9k^2y^2 + 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2 - 10\right)\left(2k^2y^2 - 3\right)$

Table D.13: Orbital expressions HOOrbitals : 1, 3. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{12} \rightarrow \phi_{2,2}$	
$\phi(\vec{r})$	$\left(2k^2x^2 - 1\right)\left(2k^2y^2 - 1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2x\left(2k^2x^2 - 5\right)\left(2k^2y^2 - 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2y\left(2k^2x^2 - 1\right)\left(2k^2y^2 - 5\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 10\right)\left(2k^2x^2 - 1\right)\left(2k^2y^2 - 1\right)$

Table D.14: Orbital expressions HOOrbitals : 2, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{13} \rightarrow \phi_{3,1}$	
$\phi(\vec{r})$	$xy\left(2k^2x^2 - 3\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-y\left(2k^4x^4 - 9k^2x^2 + 3\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-x\left(ky - 1\right)\left(ky + 1\right)\left(2k^2x^2 - 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2 - 10\right)\left(2k^2x^2 - 3\right)$

Table D.15: Orbital expressions HOOrbitals : 3, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{14} \rightarrow \phi_{4,0}$	
$\phi(\vec{r})$	$4k^4x^4 - 12k^2x^2 + 3$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2x\left(4k^4x^4 - 28k^2x^2 + 27\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2y\left(4k^4x^4 - 12k^2x^2 + 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 10\right)\left(4k^4x^4 - 12k^2x^2 + 3\right)$

Table D.16: Orbital expressions HOOrbitals : 4, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{15} \to \phi_{0,5}$	
$\phi(\vec{r})$	$y\left(4k^4y^4 - 20k^2y^2 + 15\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(4k^4y^4 - 20k^2y^2 + 15\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-4k^6y^6 + 40k^4y^4 - 75k^2y^2 + 15$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2 - 12\right)\left(4k^4y^4 - 20k^2y^2 + 15\right)$

Table D.17: Orbital expressions HOOrbitals : 0, 5. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{16} \to \phi_{1,4}$	
$\phi(\vec{r})$	$x\left(4k^4y^4 - 12k^2y^2 + 3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\left(kx - 1\right)\left(kx + 1\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(4k^4y^4 - 28k^2y^2 + 27\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2 - 12\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$

Table D.18: Orbital expressions HOOrbitals : 1, 4. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{17} \to \phi_{2,3}$	
$\phi(\vec{r})$	$y\left(2k^2x^2 - 1\right)\left(2k^2y^2 - 3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2 - 5\right)\left(2k^2y^2 - 3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\left(2k^2x^2 - 1\right)\left(2k^4y^4 - 9k^2y^2 + 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2 - 12\right)\left(2k^2x^2 - 1\right)\left(2k^2y^2 - 3\right)$

Table D.19: Orbital expressions HOOrbitals : 2, 3. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{18} \to \phi_{3,2}$	
$\phi(\vec{r})$	$x\left(2k^2x^2 - 3\right)\left(2k^2y^2 - 1\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\left(2k^2y^2 - 1\right)\left(2k^4x^4 - 9k^2x^2 + 3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2 - 3\right)\left(2k^2y^2 - 5\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2 - 12\right)\left(2k^2x^2 - 3\right)\left(2k^2y^2 - 1\right)$

Table D.20: Orbital expressions HOOrbitals : 3, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{19} \to \phi_{4,1}$	
$\phi(\vec{r})$	$y\left(4k^4x^4 - 12k^2x^2 + 3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(4k^4x^4 - 28k^2x^2 + 27\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\left(ky - 1\right)\left(ky + 1\right)\left(4k^4x^4 - 12k^2x^2 + 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2 - 12\right)\left(4k^4x^4 - 12k^2x^2 + 3\right)$

Table D.21: Orbital expressions HOOrbitals : 4, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{20} \to \phi_{5,0}$	
$\phi(\vec{r})$	$x\left(4k^4x^4 - 20k^2x^2 + 15\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-4k^6x^6 + 40k^4x^4 - 75k^2x^2 + 15$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(4k^4x^4 - 20k^2x^2 + 15\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2 - 12\right)\left(4k^4x^4 - 20k^2x^2 + 15\right)$

Table D.22: Orbital expressions HOOrbitals : 5, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{21} \to \phi_{0,6}$	
$\phi(\vec{r})$	$8k^6y^6 - 60k^4y^4 + 90k^2y^2 - 15$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2x\left(8k^6y^6 - 60k^4y^4 + 90k^2y^2 - 15\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2y\left(8k^6y^6 - 108k^4y^4 + 330k^2y^2 - 195\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 14\right)\left(8k^6y^6 - 60k^4y^4 + 90k^2y^2 - 15\right)$

Table D.23: Orbital expressions HOOrbitals : 0, 6. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{22} \to \phi_{1,5}$	
$\phi(\vec{r})$	$xy\left(4k^4y^4 - 20k^2y^2 + 15\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-y\left(kx - 1\right)\left(kx + 1\right)\left(4k^4y^4 - 20k^2y^2 + 15\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-x\left(4k^6y^6 - 40k^4y^4 + 75k^2y^2 - 15\right)$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2 - 14\right)\left(4k^4y^4 - 20k^2y^2 + 15\right)$

Table D.24: Orbital expressions HOOrbitals : 1, 5. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{23} \to \phi_{2,4}$	
$\phi(\vec{r})$	$\left(2k^2x^2 - 1\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2x\left(2k^2x^2 - 5\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2y\left(2k^2x^2 - 1\right)\left(4k^4y^4 - 28k^2y^2 + 27\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 14\right)\left(2k^2x^2 - 1\right)\left(4k^4y^4 - 12k^2y^2 + 3\right)$

Table D.25: Orbital expressions HOOrbitals : 2, 4. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{24} \to \phi_{3,3}$	
$\phi(\vec{r})$	$xy\left(2k^2x^2 - 3\right)\left(2k^2y^2 - 3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-y\left(2k^2y^2 - 3\right)\left(2k^4x^4 - 9k^2x^2 + 3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-x\left(2k^2x^2 - 3\right)\left(2k^4y^4 - 9k^2y^2 + 3\right)$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2 - 14\right)\left(2k^2x^2 - 3\right)\left(2k^2y^2 - 3\right)$

Table D.26: Orbital expressions HOOrbitals : 3, 3. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{25} \to \phi_{4,2}$	
$\phi(\vec{r})$	$\left(2k^2y^2-1\right)\left(4k^4x^4-12k^2x^2+3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2x\left(2k^2y^2-1\right)\left(4k^4x^4-28k^2x^2+27\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2y\left(2k^2y^2-5\right)\left(4k^4x^4-12k^2x^2+3\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2-14\right)\left(2k^2y^2-1\right)\left(4k^4x^4-12k^2x^2+3\right)$

Table D.27: Orbital expressions HOOrbitals : 4, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{26} \to \phi_{5,1}$	
$\phi(\vec{r})$	$xy\left(4k^4x^4-20k^2x^2+15\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-y\left(4k^6x^6-40k^4x^4+75k^2x^2-15\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-x\left(ky-1\right)\left(ky+1\right)\left(4k^4x^4-20k^2x^2+15\right)$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2-14\right)\left(4k^4x^4-20k^2x^2+15\right)$

Table D.28: Orbital expressions HOOrbitals : 5, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{27} \to \phi_{6,0}$	
$\phi(\vec{r})$	$8k^6x^6-60k^4x^4+90k^2x^2-15$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2x\left(8k^6x^6-108k^4x^4+330k^2x^2-195\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2y\left(8k^6x^6-60k^4x^4+90k^2x^2-15\right)$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2-14\right)\left(8k^6x^6-60k^4x^4+90k^2x^2-15\right)$

Table D.29: Orbital expressions HOOrbitals : 6, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

E

Harmonic Oscillator Orbitals 3D

Orbitals are constructed in the following fashion:

$$\phi(\vec{r})_{n_x,n_y,n_z} = H_{n_x}(kx)H_{n_y}(ky)H_{n_z}(kz)e^{-\frac{1}{2}k^2r^2}$$

where $k = \sqrt{\omega\alpha}$, with ω being the oscillator frequency and α being the variational parameter.

$H_0(kx)$	1
$H_1(kx)$	$2kx$
$H_2(kx)$	$4k^2x^2 - 2$
$H_3(kx)$	$8k^3x^3 - 12kx$
$H_0(ky)$	1
$H_1(ky)$	$2ky$
$H_2(ky)$	$4k^2y^2 - 2$
$H_3(ky)$	$8k^3y^3 - 12ky$
$H_0(kz)$	1
$H_1(kz)$	$2kz$
$H_2(kz)$	$4k^2z^2 - 2$
$H_3(kz)$	$8k^3z^3 - 12kz$

Table E.1: Hermite polynomials used to construct orbital functions

$\phi_0 \to \phi_{0,0,0}$	
$\phi(\vec{r})$	1
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 x$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 y$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2 z$
$\nabla^2\phi(\vec{r})$	$k^2\left(k^2r^2 - 3\right)$

Table E.2: Orbital expressions HOOrbitals3D : 0, 0, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_1 \to \phi_{0,0,1}$	
$\phi(\vec{r})$	z
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 xz$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 yz$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\left(kz - 1\right)\left(kz + 1\right)$
$\nabla^2\phi(\vec{r})$	$k^2 z\left(k^2r^2 - 5\right)$

Table E.3: Orbital expressions HOOrbitals3D : 0, 0, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_2 \to \phi_{0,1,0}$	
$\phi(\vec{r})$	y
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2 xy$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\left(ky - 1\right)\left(ky + 1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2 yz$
$\nabla^2\phi(\vec{r})$	$k^2 y\left(k^2r^2 - 5\right)$

Table E.4: Orbital expressions HOOrbitals3D : 0, 1, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_3 \to \phi_{1,0,0}$	
$\phi(\vec{r})$	x
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\left(kx - 1\right)\left(kx + 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2 xy$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2 xz$
$\nabla^2\phi(\vec{r})$	$k^2 x\left(k^2r^2 - 5\right)$

Table E.5: Orbital expressions HOOrbitals3D : 1, 0, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_4 \to \phi_{0,0,2}$	
$\phi(\vec{r})$	$4k^2z^2 - 2$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-2k^2 x\left(2k^2z^2 - 1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-2k^2 y\left(2k^2z^2 - 1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-2k^2 z\left(2k^2z^2 - 5\right)$
$\nabla^2\phi(\vec{r})$	$2k^2\left(k^2r^2 - 7\right)\left(2k^2z^2 - 1\right)$

Table E.6: Orbital expressions HOOrbitals3D : 0, 0, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_5 \rightarrow \phi_{0,1,1}$	
$\phi(\vec{r})$	yz
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2xyz$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-z\left(ky-1\right)\left(ky+1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-y\left(kz-1\right)\left(kz+1\right)$
$\nabla^2\phi(\vec{r})$	$k^2yz\left(k^2r^2-7\right)$

Table E.7: Orbital expressions HOOrbitals3D : 0, 1, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_6 \rightarrow \phi_{0,2,0}$	
$\phi(\vec{r})$	$4k^2y^2-2$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-2k^2x\left(2k^2y^2-1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-2k^2y\left(2k^2y^2-5\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-2k^2z\left(2k^2y^2-1\right)$
$\nabla^2\phi(\vec{r})$	$2k^2\left(k^2r^2-7\right)\left(2k^2y^2-1\right)$

Table E.8: Orbital expressions HOOrbitals3D : 0, 2, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_7 \rightarrow \phi_{1,0,1}$	
$\phi(\vec{r})$	xz
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-z\left(kx-1\right)\left(kx+1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2xyz$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-x\left(kz-1\right)\left(kz+1\right)$
$\nabla^2\phi(\vec{r})$	$k^2xz\left(k^2r^2-7\right)$

Table E.9: Orbital expressions HOOrbitals3D : 1, 0, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_8 \rightarrow \phi_{1,1,0}$	
$\phi(\vec{r})$	xy
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-y\left(kx-1\right)\left(kx+1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-x\left(ky-1\right)\left(ky+1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2xyz$
$\nabla^2\phi(\vec{r})$	$k^2xy\left(k^2r^2-7\right)$

Table E.10: Orbital expressions HOOrbitals3D : 1, 1, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_9 \rightarrow \phi_{2,0,0}$	
$\phi(\vec{r})$	$4k^2x^2-2$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-2k^2x\left(2k^2x^2-5\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-2k^2y\left(2k^2x^2-1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-2k^2z\left(2k^2x^2-1\right)$
$\nabla^2\phi(\vec{r})$	$2k^2\left(k^2r^2-7\right)\left(2k^2x^2-1\right)$

Table E.11: Orbital expressions HOOrbitals3D : 2, 0, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{10} \rightarrow \phi_{0,0,3}$	
$\phi(\vec{r})$	$z\left(2k^2z^2-3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xz\left(2k^2z^2-3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2yz\left(2k^2z^2-3\right)$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-2k^4z^4+9k^2z^2-3$
$\nabla^2\phi(\vec{r})$	$k^2z\left(k^2r^2-9\right)\left(2k^2z^2-3\right)$

Table E.12: Orbital expressions HOOrbitals3D : 0, 0, 3. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{11} \rightarrow \phi_{0,1,2}$	
$\phi(\vec{r})$	$y\left(2k^2z^2-1\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(2k^2z^2-1\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\left(ky-1\right)\left(ky+1\right)\left(2k^2z^2-1\right)$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-k^2yz\left(2k^2z^2-5\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2-9\right)\left(2k^2z^2-1\right)$

Table E.13: Orbital expressions HOOrbitals3D : 0, 1, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{12} \rightarrow \phi_{0,2,1}$	
$\phi(\vec{r})$	$z\left(2k^2y^2-1\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xz\left(2k^2y^2-1\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2yz\left(2k^2y^2-5\right)$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-\left(kz-1\right)\left(kz+1\right)\left(2k^2y^2-1\right)$
$\nabla^2\phi(\vec{r})$	$k^2z\left(k^2r^2-9\right)\left(2k^2y^2-1\right)$

Table E.14: Orbital expressions HOOrbitals3D : 0, 2, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{13} \rightarrow \phi_{0,3,0}$	
$\phi(\vec{r})$	$y\left(2k^2y^2-3\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(2k^2y^2-3\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-2k^4y^4+9k^2y^2-3$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-k^2yz\left(2k^2y^2-3\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2-9\right)\left(2k^2y^2-3\right)$

Table E.15: Orbital expressions HOOrbitals3D : 0, 3, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{14} \rightarrow \phi_{1,0,2}$	
$\phi(\vec{r})$	$x\left(2k^2z^2-1\right)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\left(kx-1\right)\left(kx+1\right)\left(2k^2z^2-1\right)$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-k^2xy\left(2k^2z^2-1\right)$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-k^2xz\left(2k^2z^2-5\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2-9\right)\left(2k^2z^2-1\right)$

Table E.16: Orbital expressions HOOrbitals3D : 1, 0, 2. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{15} \rightarrow \phi_{1,1,1}$	
$\phi(\vec{r})$	xyz
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-yz\,(kx-1)\,(kx+1)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-xz\,(ky-1)\,(ky+1)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-xy\,(kz-1)\,(kz+1)$
$\nabla^2\phi(\vec{r})$	$k^2xyz\left(k^2r^2-9\right)$

Table E.17: Orbital expressions HOOrbitals3D : 1, 1, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{16} \rightarrow \phi_{1,2,0}$	
$\phi(\vec{r})$	$x\left(2k^2y^2-1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\left(kx-1\right)\left(kx+1\right)\left(2k^2y^2-1\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2y^2-5\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2xz\left(2k^2y^2-1\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2-9\right)\left(2k^2y^2-1\right)$

Table E.18: Orbital expressions HOOrbitals3D : 1, 2, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{17} \rightarrow \phi_{2,0,1}$	
$\phi(\vec{r})$	$z\left(2k^2x^2-1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2xz\left(2k^2x^2-5\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2yz\left(2k^2x^2-1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\left(kz-1\right)\left(kz+1\right)\left(2k^2x^2-1\right)$
$\nabla^2\phi(\vec{r})$	$k^2z\left(k^2r^2-9\right)\left(2k^2x^2-1\right)$

Table E.19: Orbital expressions HOOrbitals3D : 2, 0, 1. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{18} \rightarrow \phi_{2,1,0}$	
$\phi(\vec{r})$	$y\left(2k^2x^2-1\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2-5\right)$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\left(ky-1\right)\left(ky+1\right)\left(2k^2x^2-1\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2yz\left(2k^2x^2-1\right)$
$\nabla^2\phi(\vec{r})$	$k^2y\left(k^2r^2-9\right)\left(2k^2x^2-1\right)$

Table E.20: Orbital expressions HOOrbitals3D : 2, 1, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

$\phi_{19} \rightarrow \phi_{3,0,0}$	
$\phi(\vec{r})$	$x\left(2k^2x^2-3\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-2k^4x^4+9k^2x^2-3$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-k^2xy\left(2k^2x^2-3\right)$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-k^2xz\left(2k^2x^2-3\right)$
$\nabla^2\phi(\vec{r})$	$k^2x\left(k^2r^2-9\right)\left(2k^2x^2-3\right)$

Table E.21: Orbital expressions HOOrbitals3D : 3, 0, 0. Factor $e^{-\frac{1}{2}k^2r^2}$ is omitted.

F

Hydrogen Orbitals

Orbitals are constructed in the following fashion:

$$\phi(\vec{r})_{n,l,m} = L_{n-l-1}^{2l+1}\Big(\frac{2r}{n}k\Big)S_l^m(\vec{r})e^{-\frac{r}{n}k}$$

where n is the principal quantum number, $k = \alpha Z$ with Z being the nucleus charge and α being the variational parameter.

$$l = 0,\ 1,\ ...,\ (n-1)$$

$$m = -l,\ (-l+1),\ ...,\ (l-1),\ l$$

$\phi_0 \to \phi_{1,0,0}$	
$\phi(\vec{r})$	1
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kx}{r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{ky}{r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kz}{r}$
$\nabla^2\phi(\vec{r})$	$\frac{k(kr-2)}{r}$

Table F.1: Orbital expressions hydrogenicOrbitals : 1, 0, 0. Factor e^{-kr} is omitted.

$\phi_1 \to \phi_{2,0,0}$	
$\phi(\vec{r})$	$kr-2$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kx(kr-4)}{2r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{ky(kr-4)}{2r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kz(kr-4)}{2r}$
$\nabla^2\phi(\vec{r})$	$\frac{k(kr-8)(kr-2)}{4r}$

Table F.2: Orbital expressions hydrogenicOrbitals : 2, 0, 0. Factor $e^{-\frac{1}{2}kr}$ is omitted.

$\phi_2 \to \phi_{2,1,0}$	
$\phi(\vec{r})$	z
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz}{2r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kyz}{2r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$\frac{-kz^2+2r}{2r}$
$\nabla^2\phi(\vec{r})$	$\frac{kz(kr-8)}{4r}$

Table F.3: Orbital expressions hydrogenicOrbitals : 2, 1, 0. Factor $e^{-\frac{1}{2}kr}$ is omitted.

$\phi_3 \to \phi_{2,1,1}$	
$\phi(\vec{r})$	x
$\vec{i} \cdot \nabla\phi(\vec{r})$	$\frac{-kx^2+2r}{2r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kxy}{2r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz}{2r}$
$\nabla^2\phi(\vec{r})$	$\frac{kx(kr-8)}{4r}$

Table F.4: Orbital expressions hydrogenicOrbitals : 2, 1, 1. Factor $e^{-\frac{1}{2}kr}$ is omitted.

$\phi_4 \rightarrow \phi_{2,1,-1}$	
$\phi(\vec{r})$	y
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kxy}{2r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$\frac{-ky^2+2r}{2r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kyz}{2r}$
$\nabla^2\phi(\vec{r})$	$\frac{ky(kr-8)}{4r}$

Table F.5: Orbital expressions hydrogenicOrbitals : 2, 1, -1. Factor $e^{-\frac{1}{2}kr}$ is omitted.

$\phi_5 \rightarrow \phi_{3,0,0}$	
$\phi(\vec{r})$	$2k^2r^2 - 18kr + 27$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kx\left(2k^2r^2-30kr+81\right)}{3r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{ky\left(2k^2r^2-30kr+81\right)}{3r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kz\left(2k^2r^2-30kr+81\right)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{k(kr-18)\left(2k^2r^2-18kr+27\right)}{9r}$

Table F.6: Orbital expressions hydrogenicOrbitals : 3, 0, 0. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_6 \rightarrow \phi_{3,1,0}$	
$\phi(\vec{r})$	$z\,(kr-6)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz(kr-9)}{3r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kyz(kr-9)}{3r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$\frac{3kr^2-kz^2(kr-9)-18r}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{kz(kr-18)(kr-6)}{9r}$

Table F.7: Orbital expressions hydrogenicOrbitals : 3, 1, 0. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_7 \rightarrow \phi_{3,1,1}$	
$\phi(\vec{r})$	$x\,(kr-6)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$\frac{3kr^2-kx^2(kr-9)-18r}{3r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kxy(kr-9)}{3r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz(kr-9)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{kx(kr-18)(kr-6)}{9r}$

Table F.8: Orbital expressions hydrogenicOrbitals : 3, 1, 1. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_8 \rightarrow \phi_{3,1,-1}$	
$\phi(\vec{r})$	$y\,(kr-6)$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\frac{kxy(kr-9)}{3r}$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$\frac{3kr^2-ky^2(kr-9)-18r}{3r}$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-\frac{kyz(kr-9)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{ky(kr-18)(kr-6)}{9r}$

Table F.9: Orbital expressions hydrogenicOrbitals : 3, 1, -1. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_9 \rightarrow \phi_{3,2,0}$	
$\phi(\vec{r})$	$-r^2+3z^2$
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\frac{x\left(k\left(-r^2+3z^2\right)+6r\right)}{3r}$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\frac{y\left(k\left(-r^2+3z^2\right)+6r\right)}{3r}$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-\frac{z\left(k\left(-r^2+3z^2\right)-12r\right)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{k\left(-r^2+3z^2\right)(kr-18)}{9r}$

Table F.10: Orbital expressions hydrogenicOrbitals : 3, 2, 0. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_{10} \rightarrow \phi_{3,2,1}$	
$\phi(\vec{r})$	xz
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\frac{z\left(kx^2-3r\right)}{3r}$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\frac{kxyz}{3r}$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-\frac{x\left(kz^2-3r\right)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{kxz(kr-18)}{9r}$

Table F.11: Orbital expressions hydrogenicOrbitals : 3, 2, 1. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_{11} \rightarrow \phi_{3,2,-1}$	
$\phi(\vec{r})$	yz
$\vec{i}\cdot\nabla\phi(\vec{r})$	$-\frac{kxyz}{3r}$
$\vec{j}\cdot\nabla\phi(\vec{r})$	$-\frac{z\left(ky^2-3r\right)}{3r}$
$\vec{k}\cdot\nabla\phi(\vec{r})$	$-\frac{y\left(kz^2-3r\right)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{kyz(kr-18)}{9r}$

Table F.12: Orbital expressions hydrogenicOrbitals : 3, 2, -1. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_{12} \to \phi_{3,2,2}$	
$\phi(\vec{r})$	$x^2 - y^2$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{x\left(k\left(x^2-y^2\right)-6r\right)}{3r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{y\left(k\left(x^2-y^2\right)+6r\right)}{3r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kz\left(x^2-y^2\right)}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{k\left(x^2-y^2\right)(kr-18)}{9r}$

Table F.13: Orbital expressions hydrogenicOrbitals : 3, 2, 2. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_{13} \to \phi_{3,2,-2}$	
$\phi(\vec{r})$	xy
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{y\left(kx^2-3r\right)}{3r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{x\left(ky^2-3r\right)}{3r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kxyz}{3r}$
$\nabla^2\phi(\vec{r})$	$\frac{kxy(kr-18)}{9r}$

Table F.14: Orbital expressions hydrogenicOrbitals : 3, 2, -2. Factor $e^{-\frac{1}{3}kr}$ is omitted.

$\phi_{14} \to \phi_{4,0,0}$	
$\phi(\vec{r})$	$k^3r^3 - 24k^2r^2 + 144kr - 192$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kx\left(k^3r^3-36k^2r^2+336kr-768\right)}{4r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{ky\left(k^3r^3-36k^2r^2+336kr-768\right)}{4r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kz\left(k^3r^3-36k^2r^2+336kr-768\right)}{4r}$
$\nabla^2\phi(\vec{r})$	$\frac{k(kr-32)\left(k^3r^3-24k^2r^2+144kr-192\right)}{16r}$

Table F.15: Orbital expressions hydrogenicOrbitals : 4, 0, 0. Factor $e^{-\frac{1}{4}kr}$ is omitted.

$\phi_{15} \to \phi_{4,1,0}$	
$\phi(\vec{r})$	$z\left(k^2r^2 - 20kr + 80\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz(kr-20)(kr-8)}{4r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kyz(kr-20)(kr-8)}{4r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$\frac{4k^2r^3-80kr^2-kz^2(kr-20)(kr-8)+320r}{4r}$
$\nabla^2\phi(\vec{r})$	$\frac{kz(kr-32)\left(k^2r^2-20kr+80\right)}{16r}$

Table F.16: Orbital expressions hydrogenicOrbitals : 4, 1, 0. Factor $e^{-\frac{1}{4}kr}$ is omitted.

$\phi_{16} \to \phi_{4,1,1}$	
$\phi(\vec{r})$	$x\left(k^2r^2 - 20kr + 80\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$\frac{4k^2r^3 - 80kr^2 - kx^2(kr-20)(kr-8) + 320r}{4r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$-\frac{kxy(kr-20)(kr-8)}{4r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kxz(kr-20)(kr-8)}{4r}$
$\nabla^2\phi(\vec{r})$	$\frac{kx(kr-32)\left(k^2r^2-20kr+80\right)}{16r}$

Table F.17: Orbital expressions hydrogenicOrbitals : 4, 1, 1. Factor $e^{-\frac{1}{4}kr}$ is omitted.

$\phi_{17} \to \phi_{4,1,-1}$	
$\phi(\vec{r})$	$y\left(k^2r^2 - 20kr + 80\right)$
$\vec{i} \cdot \nabla\phi(\vec{r})$	$-\frac{kxy(kr-20)(kr-8)}{4r}$
$\vec{j} \cdot \nabla\phi(\vec{r})$	$\frac{4k^2r^3 - 80kr^2 - ky^2(kr-20)(kr-8) + 320r}{4r}$
$\vec{k} \cdot \nabla\phi(\vec{r})$	$-\frac{kyz(kr-20)(kr-8)}{4r}$
$\nabla^2\phi(\vec{r})$	$\frac{ky(kr-32)\left(k^2r^2-20kr+80\right)}{16r}$

Table F.18: Orbital expressions hydrogenicOrbitals : 4, 1, -1. Factor $e^{-\frac{1}{4}kr}$ is omitted.

Bibliography

[1] J. J. Sakurai. *Modern Quantum Mechanics*. Addison-Wesley, New York, Revised ed edition, 1994.

[2] David Griffiths. *Introduction to Quantum Mechanics*. Pearson, 2nd edition edition, 2005.

[3] Jaime Fernández Rico, Rafael López, Ignacio Ema, and Guillermo Ramírez. Translation of real solid spherical harmonics. *Int. J. Quant. Chem.*, **113**:1544, 2013.

[4] A. Kongkanand, K. Tvrdy, K. Takechi, M. Kuno, and P. V. Kamat. Quantum dot solar cells. Tuning photoresponse through size and shape control of CdSe-TiO2 architecture. *J. Am. Chem. Soc.*, **130**:4007, March 2008.

[5] G. Park, O.B. Shchekin, D.L. Huffaker, and D.G. Deppe. Low-threshold oxide-confined 1.3- micrometer quantum-dot laser. *IEEE Photon. Technol. Lett.*, **12**:230, 2000.

[6] E.T. Ben-Ari. Nanoscale quantum dots hold promise for cancer applications. *J. Natl. Cancer Inst.*, **90**:502, 2003.

[7] D. Loss and D. P. Vincenzo. Quantum computation with quantum dots. *Phys. Rev. A*, **57**:120, 1998.

[8] R. V. Shenoi, J. Hou, Y. Sharma, J. Shao, T. E. Vandervelde, and S. Krishna. Low strain quantum dots in a double well infrared detector. pages 708207–708207–6, 2008.

[9] Yang Min Wang. Coupled-Cluster Studies of Double Quantum Dots. Master's thesis, University of Oslo, 2011.

[10] Matthias Degroote. Faddeev random phase approximation applied to molecules. *Eur. Phys. J. ST*, **218**:1, 2013.

[11] Harry Partridge. Near Hartree–Fock quality GTO basis sets for the first- and third-row atoms. *J. Chem. Phys.*, **90**:1043, 1989.

[12] H. D. Young, R.A. Freedman, and L.A. Ford. *Sears and Zemansky's University Physics*. Pearson, Addison-Wesley, 12 edition, 2008.

[13] Moskowitz Kalos. A new Look at Correlations in Atomic and Molecular Systems. Application of Fermion Monte Carlo Variational Method. *Int. J. Quant. Chem.*, **XX**:1107, 1981.

[14] Claudia Filippi and C. J. Umrigar. Multiconfiguration wave function for quantum Monte Carlo calculations of first-row diatomic molecules. *J. Chem. Phys*, **105**:213, July 1996.

[15] S. Datta, S. A. Alexander, and R. L. Coldwell. Properties of selected diatomics using variational Monte Carlo methods. *J. Chem. Phys.*, **120**:3642, 2004.

[16] Murat Barisik and Ali Beskok. Equilibrium molecular dynamics studies on nanoscale-confined fluids. *Microfluid Nanofluid*, **11**(3):269, September 2011.

[17] Karl P. Travis and Keith E. Gubbins. Poiseuille flow of Lennard-Jones fluids in narrow slit pores. *J. Chem. Phys*, **112**:1984, 2000.

[18] A Badinski, P D Haynes, J R Trail, and R J Needs. Methods for calculating forces within quantum Monte Carlo simulations. *J. Phys.: Condens. Matter*, **22**:074202, 2010.

[19] Adri C. T. van Duin, Siddharth Dasgupta, Francois Lorant, and William A. Goddard. ReaxFF: A Reactive Force Field for Hydrocarbons. *J. Phys. Chem. A*, **105**:9396, 2001.

[20] Lars Eivind Lervåg. VMC CALCULATIONS OF TWO-DIMENSIONAL QUANTUM DOTS. Master's thesis, University of Oslo, 2010.

[21] Håvard Sandsdalen. Variational Monte Carlo studies of Atoms. Master's thesis, University of Oslo, 2010.

[22] Veronica K. B. Olsen. Full Configuration Interaction Simulation of Quantum Dots. Master's thesis, University of Oslo, 2012.

[23] Aurel Bulgac and Michael McNeil Forbes. Use of the discrete variable representation basis in nuclear physics. *Phys. Rev. C*, **87**:051301, May 2013.

[24] I. Shavitt and R. J. Bartlett. *Many-Body Methods in Chemistry and Physics*. Cambridge University Press, Cambridge, 2009.

[25] A. Roggero, F. Pederiva, and A. Mukherjee. Quantum Monte Carlo with Coupled-Cluster wave functions. arXiv:1304.1549 [nucl-th].

[26] G.H. Golub and C.F. Van Loan. *Matrix computations*, volume 3. Johns Hopkins Univ Press, 1996.

[27] Matplotlib website, May 2013. http://matplotlib.org.

[28] PySide website, May 2013.
http://qt-project.org/wiki/Category:LanguageBindings::PySide.

[29] SymPy website, May 2013. http://sympy.org.