Quantum Monte-Carlo Studies of Generalized Many-body Systems

by

Jørgen Høgberget

THESIS
for the degree of
MASTER OF SCIENCE

(Master in Computational Physics)

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

June 2013

Preface

blah blah

Contents

1 Introduction **7**

I Theory **9**

II Results **11**

2 Results **13**

2.1 Optimization Results 13

2.2 The Non-interacting Case 17

2.3 Quantum Dots 20

2.3.1 Ground State Energies 20

2.3.2 One-body Densities 22

2.3.3 Lowering the frequency 25

2.3.4 Simulatings in a Double-well 28

2.4 Atoms 29

2.4.1 Ground State Energies 29

2.4.2 One-body densities 29

2.5 Homonuclear Diatomic Molecules 32

2.5.1 Ground State Energies 32

2.5.2 One-body densities 32

2.5.3 Parameterizing Forces 34

3 Conclusions **37**

A Dirac Notation **41**

B DCViz: Visualization of Data **43**

B.1 Basic Usage 43

B.1.1 The Terminal Client 47

B.1.2 The Application Programming Interface (API) 47

C Auto-generation with SymPy **51**

C.1 Usage 51

C.1.1 Symbolic Algebra 51

C.1.2 Exporting C++ and Latex Code 52

C.1.3 Calculating Derivatives 52

C.2 Using the auto-generation Script 54

C.2.1 Generating Latex code 54

C.2.2 Generating C++ code 56

D Harmonic Oscillator Orbitals 2D **59**

E Harmonic Oscillator Orbitals 3D **67**

F Hydrogen Orbitals **73**

Bibliography **79**

1 Introduction

Studies of general systems demand a general solver. The process of developing code aimed at a specific task is fundamentally different from the process of developing a general solver, simply due to the fact that the general equations need to be implemented *independent* of any specific properties a modelled system may contain. This is most commonly achieved through object oriented programming, which allows for the code to be structured into general implementations and specific implementations. The specific - and general implementations can then be interfaced through a functionality referred to as *polymorphism*. The aim of this thesis is to use object oriented C++ to build a general and efficient Quantum Monte-Carlo (QMC) solver, which can tackle several many-particle systems, from confined electron systems, i.e. quantum dots, to bosons.

A constraint put on the QMC solver in this thesis is that the ansatz for the *trial wave function* consists of a single term, i.e. a single *Slater determinant*. This opens up the possibility to study systems consisting of a large number of particles due to efficient optimizations in the single determinant. A simplistic trial wave function will also significantly ease the implementation of different systems, and thus make it easier to develop a general framework.

Variational Monte-Carlo (VMC) is expected to suffer due to the fact that the ansatz is simple, however, Diffusion Monte-Carlo (DMC) is supposed to withstand the problems introduced by a simplistic trial wave function and thus yield a good final estimate nevertheless. To study this purposed power of DMC is another main focus of this thesis, in addition to pushing the limits regarding optimization of the code, and thus run ab-initio simulations of a vast amount of particles.

The two dimensional quantum dot was chosen as the system of reference around which the code was planned. The reason for this is that all the current Master students are studying quantum dots at some level, which means that we can help each other reach a collective understanding of the system at hand. Additionally, Sarah Reimann were studying two dimensional quantum dots for up to 42 particles in her thesis, and later also 56, using a non-variational method called *Similarity Renormalization Group theory*. Providing her with precise variational DMC benchmarks were considered to be of utmost importance. Coupled Cluster Singles and Doubles (CCSD) results are done up to 56 particles by Christoffer Hirth [1], however, for the lower frequencies, i.e. for higher correlations, CCSD struggles with convergence.

Depending on the success of the implementation, various additional systems could be implemented and studied in detail, such as atomic systems, three dimensional - and double-well quantum dots.

Apart from benchmarking DMC ground state energies, the specific aim in the case of quantum dots is to study their behavior as the frequency is lowered. A lower frequency implies a higher correlation in the system. Understanding these correlated systems of electrons are of great importance to many-body theory in general. The effect of adding a third dimension is also of high interest. The advantage of DMC

compared to other methods is that the distribution is relatively easy to obtain.

Ground state energies for atomic systems can be benchmarked against experimental results [2–5], which yields an excellent opportunity to test the limits of DMC given a simplistic trial wave function. Going further to molecular systems, an additional aim is to explore the transition between QMC and molecular dynamics by parameterizing simple force field potentials [6].

Benchmarking the results of quantum dots, many former Master students, such as Christoffer Hirth [1] and Veronica K.B. Olsen [7], have studied the two dimensional system in the past, and has thus generated ground state energies with which the DMC energies can be compared. For three dimensional quantum dots, little results are available for benchmarking.

The structure of the thesis

- The first chapter introduces the concept of object oriented programming, with focus on the methods used to develop the code for this thesis. The reader is assumed to have some background in programming, hence the very fundamentals of programming are not presented. A full documentation of the code is available in Ref. [8]. The code will thus not be covered in full detail. In addition to concepts from C++ programming, Python scripting will be introduced. General strategies regarding planning and structuring of code will also be covered in detail. The two most important Python scripts used in this thesis are documented in Appendix C and Appendix B.
- The second chapter serves as a theoretical introduction to QMC, discussing the necessary many-body theory in detail. Important theory which is required to understand the concepts introduced in later chapters are given the primary focus. The reader is assumed to have a basic understanding of Quantum wave mechanics. An introduction to the commonly used Dirac notation is given in Appendix A.
- Chapter **??** presents all the assumptions regarding the systems modelled in this thesis together with the aims regarding the generalization and optimization of the code. The strategies applied to achieve these aims will then be covered in high detail.
- Chapter **??** introduces the systems modelled in this thesis, that is, the quantum dots and atomic systems. The single particle wave functions used to generate the trial wave functions for the different systems are presented together with the respective Hamiltonians.
- The results, along with the discussions and the conclusions mark the final part of this thesis. Results for up to 56 electrons in the two dimensional quantum dot are presented and comparisons are made between two - and three dimensional quantum dots for high and low frequency ranges. A brief display of a double-well quantum dot is then given before the atomic results are presented. The ground state energies of atoms up to Krypton and molecules up to O_2 are then compared to experimental values. Concluding the results section, the molecular energies as a function of the separation of cores are compared to the Lennard-Jones 12-6 potential [9, 10]. Final remarks are then made regarding further work expanding on what was achieved in the thesis.

[11]

Part I

Theory

Part II

Results

2

Results

The results were produced using atomic units, i.e $\hbar = e = m_e = 4\pi\epsilon_0 = 1$, where m_e and e_0 is the electron mass and vacuum permittivity respectively. This implies that all listed energies are given in Hartrees, i.e. scaled with $\hbar^2/m_e a_0^2$, and all lengths are given in Bohr radii, i.e. scaled with $a_0 = 4\pi\epsilon_0\hbar^2/m_e e^2$.

2.1 Optimization Results

The optimization results listed in this section are estimated using a 30 particle two-dimensional quantum dot as reference system.

Profiling the code revealed, not surprisingly, that 99% of the runtime was spent diffusing the particles, i.e. spent in `QMC::diffuse_walker`, regardless of whether Variational Monte-Carlo (VMC), Diffusion Monte-Carlo (DMC) or Adaptive Stochastic Gradient Descent (ASGD) was run. Optimizing the code then solely involved optimizing this function.

The profiling tool of choice was *KCacheGrind*, available at the Ubuntu Software Center. KCacheGrind lists relative time spent in functions graphically in blocks, whose size is proportional to the time spent inside the function, much like standard hard drive listing software does with files and file size.

Optimizations discussed in chapter **??** which are not mentioned in the following sections are optimizations implemented prior to the optimization process. These optimizations were considered "standard optimizations" and was thus implemented from start.

Storing the Slater matrix

This optimization is described in detail in Section **??**. In addition to storing the Slater matrix, the calculation of $\tilde{I}$ from the inverse updating algorithm in Eq. (**??**) was done outside the main loops.

The percentages listed in the following table is the total time spent inside this specific function relative to all other functions.

`Orbitals::phi`	
Relative runtime used prior to optimization	80.88%
Relative runtime used after optimization	8.2%
Relative function speedup	9.86

The speedup is not a result of optimizations within the function itself, but a result of far less calls to the function. If $\tilde{I}$ was calculated outside the main loops in the first place, the speedup would be far less significant.

Optimized Jastrow gradient

This optimization described in this Section is discussed in detail in Section **??**.

The percentages listed in the following table is the total time spent inside this specific function relative to all other functions.

`Jastrow::get_grad` & `Jastrow::calc_dJ`	
Relative runtime used prior to optimization	40%
Relative runtime used after optimization	5.24%
Relative function speedup	7.63

Exploiting the symmetries of the Padé Jastrow gradient, in addition to calculating the new gradient based on the old, is in other words extremely efficient. Keep in mind that these results are for a high number of particles. For e.g. two particles, this optimization would not matter at all.

Storing the orbital derivatives

This optimization is covered in detail in Section **??**. Much like for the Slater matrix, the optimization in this case comes from the fact that the function itself is called fewer times, rather than being faster.

The percentages listed in the following table is the total time spent inside this specific function relative to all other functions.

`Orbitals.dell_phi`	
Relative runtime used prior to optimization	56.27%
Relative runtime used after optimization	7.31%
Relative function speedup	7.70

Storing quantum number independent terms

This optimization is covered in detail in Section **??**. The result of the optimization is a reduction in the number of exponential function calls, which means a more efficient calculation of single particle states, their gradients and Laplacians.

The percentages listed in the following table is the total time spent inside this specific function relative to all other functions.

`Orbitals::phi` & `Orbitals::dell_phi`	
Relative runtime used prior to optimization	5.85%
Relative runtime used after optimization	0.13%
Relative function speedup	45

This result is not surprisingly equal to $15 \cdot 3$, since a 30 particle quantum dot has 15 unique quantum numbers. One set is used by the orbitals, and two by their gradients (the Laplacian is not a part of the diffusion). Prior to this optimization, 45 exponential calls were needed to fill a row in the Slater matrix and the derivative matrix; this has been reduced to one.

Overall optimization

Combining all the optimizations listed in this chapter, the final runtime was reduced to 5% of the original. The final scaling is presented in Figure 2.1.

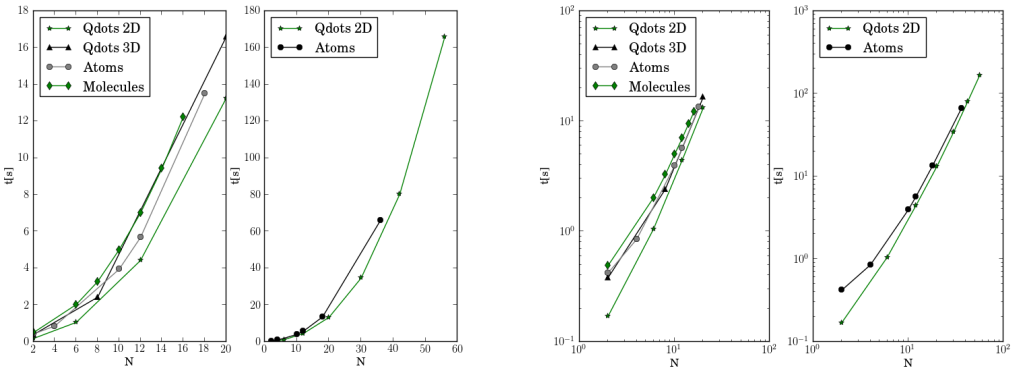


Figure 2.1: Scaling of the code with respect to the number of particles N based on VMC calculations with 10^6 cycles with 10^5 thermalization steps run on eight processors. The figures are split into a low N region and a high N region. Only two dimensional quantum dots and atoms are displayed in the high N figure. The figures to the right contain the same data as the figures to the left, however, displayed using logarithmic axes. As expected, the two-dimensional quantum dots (denoted Qdots 2D) are lowest on CPU-time and the homonuclear diatomic molecules are highest (denoted Molecules). The logarithmic figures clearly show a linear trend, implying a underlying power law.

The following power laws are deduced based on linear regression of the above figures for $N > 2$

System	Scaling
Two dimensional quantum dots	$N^{2.1038}$
Three dimensional quantum dots	$N^{2.1008}$
Atoms	$N^{1.8119}$
Homonuclear diatomic molecules	$N^{1.8437}$

As the number of particles increase, the spatial dimensions contribution to the scaling becomes negligible compared to the number of particles, rendering two dimensional quantum dots and atoms similar in runtime. This is expected since there are far more matrices in the code of dimensions $N \times N$ than $N \times d$, where d denotes the dimension.

The Jastrow factor, inverse updating, etc., all involve the same computations for all systems, hence the reason why the atomic systems scale better than the quantum dots has to originate from the efficiency of the single particle wave functions. Consider for example the third single particle wave function for the hydrogen-like orbitals:

$$\phi_3 = x. \tag{2.1}$$

The corresponding expression for two dimensional quantum dots is

$$\phi_3 = 2k^2y^2 - 1. \tag{2.2}$$

It is obvious that the orbital for quantum dots contain a higher computational cost for the processor than

the one for atoms. Comparing the expressions listed for quantum dots in Appendix E and Appendix D with those for atoms in Appendix F, it is apparent that this trend is consistent.

The fact that the difference in the cost of the single particle wave functions govern the scaling demonstrates the efficiency in the general framework. Moreover, having the molecular system scaling almost identically to the atomic one demonstrates the efficiency of the system's implementation.

2.2 The Non-interacting Case

In the case of non-interacting particles, that is, the case with no electron-electron interaction, the trial wave function is the exact wave function both in the case of quantum dots and atoms. For molecules and the double-well quantum dot, the additional requirement that $R \to \infty$ should also be applied, where R is the distance between the atoms in the case of molecules, and the distance between the well centers in the case of the quantum dot. All of the presented systems are covered in detail in Chapter **??**.

Exact solutions serve as a powerful guide, since results can be benchmarked against these, that is, the code can be validated. In the non-interacting case, Adaptive Stochastic Gradient Descent (ASGD) should always provide a variational parameter equal to unity, i.e. $\alpha = 1$. Variational Monte-Carlo (VMC) and Diffusion Monte-Carlo (DMC) should reproduce the exact solutions from Eq. (**??**) in the case of quantum dots and Eq. (**??**) in the case of atomic systems to machine precision.

In Table 2.1, validation runs for the three lowest lying closed-shell quantum dots are run for both two and three dimensions. Figure 2.2 shows ASGD finding the exact minima. Table 2.3 shows similar results for atoms.

DMC should in the case of an exact wave function be perfectly stable. The trial energy should equal the ground state energy through all time steps and zero fluctuations in the number of walkers should occur. This trend is shown for the Neon atom in figure 2.3.

A final non-interacting case is run for DMC without the exact wave function. As discussed in Chapter **??**, DMC should result in a better estimate of the ground state energy than VMC in the case of a trial wave function which is different from the exact ground state. A test case is presented in figure 2.4.

	2D					3D				
ω	N	E_{VMC}	E_{DMC}	α	E_0	N	E_{VMC}	E_{DMC}	α	E_0
0.5	2	1.0	1.0	1.0	1	2	3.0	3.0	1.0	3
1.0		2.0	2.0	1.0	2		1.5	1.5	1.0	1.5
0.5	6	5.0	5.0	1.0	5	8	18.0	18.0	1.0	18
1.0		10.0	10.0	1.0	10		9.0	9.0	1.0	9
0.5	12	14.0	14.0	1.0	14	20	60.0	60.0	1.0	60
1.0		28.0	28.0	1.0	28		30.0	30.0	1.0	30

Table 2.1: Validation results for N-particle quantum dots with no Coulomb interaction and frequency ω. The left-hand side shows the results for two dimensions, while the results for three dimensions are listed on the right-hand side. The last column for each dimension lists the exact energies E_0 calculated from Eq. (**??**). The exact solution to α is unity. As requires, all methods reproduce the exact results. The variance is zero to machine precision for all listed results.

ω	N	E_{VMC}	E_{DMC}	α	$E_0(R \to \infty)$
0.5		4.0	4.0	1.0	4
1	4	2.0	2.0	1.0	2
0.5		20.0	20.0	1.0	20
1	12	10.0	10.0	1.0	10
0.5		28.0	28.0	1.0	28
1	24	56.0	56.0	1.0	56

Table 2.2: Validation results for N-particle double-well quantum dots with no Coulomb interaction and frequency ω. The exact energy E_0, calculated from Eq. (**??**), is listed in the last column. The calculations are performed with the wells separated at a distance $R = 20$ in the x-direction. The exact solution to α is unity. As for the single-well quantum dots in Table 2.1, all methods reproduce the exact solution. The variance is zero to machine precision for all listed results.

Atom	N	E_{VMC}	E_{DMC}	α	E_0
He	2	-4.0	-4.0	1.0	-4
Be	4	-20.0	-20.0	1.0	-20
Ne	10	-200.0	-200.0	1.0	-200

Table 2.3: Validation results for different atoms consisting of N electrons with no electron-electron Coulomb interaction. The exact energies E_0 are calculated from Eq. (**??**). The exact solution of the variational parameter α is unity. As required, all methods reproduce the exact solutions. The variance is zero to machine precision for all listed results.

Molecule	N	R	E_{VMC}	E_{DMC}	$E_0(R \to \infty)$
He_2	4	10	-8.403	-8.398	-8
		100	-8.085	-8.067	
		325	-8.032	-8.030	
Be_2	8	10	-41.596	-41.608	-40
		100	-40.298	-40.231	
		325	-40.123	-40.112	
Ne_2	20	10	-409.999	-410.010	-400
		100	-401.390	-401.049	
		325	-	-	

Table 2.4: Validation results for homonuclear diatomic molecules separated at a distance R with no electron-electron Coulomb interaction. The last column lists the exact energies E_0 calculated from Eq. (**??**) for $R \to \infty$. The energies converge to the exact energies equal to two times those listed in Table 2.3. Choosing R too high results in a singular slater determinant due to finite machine precision. This happened already for $R = 325$ in the case of Ne_2. It is apparent that increasing R brings the solutions closer to the exact energy. The statistical error is skipped.

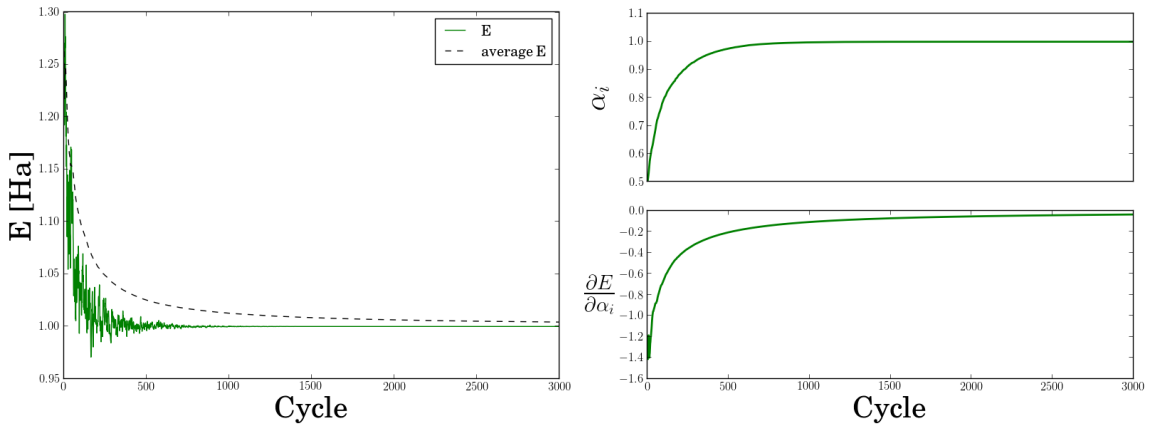


Figure 2.2: Adaptive Stochastic Gradient Descent (ASGD) results for a two-particle quantum dot with frequency $\omega = 0.5$ and no electron-electron interaction. The exact energy $E_0 = 1$ is reached after approximately 1000 cycles, where the variational parameter α has converged close to unity. Due to enormous fluctuations the variational derivative is plotted as an accumulated average. The gradient is approximately zero after 1000 cycles, in agreement with the behavior of the energy. The variational principle described in Section **??** is governing the trend of the energy convergence, however, a lot of statistical noise is present in the first 1000 cycles due to a high variance and a small number of samples.

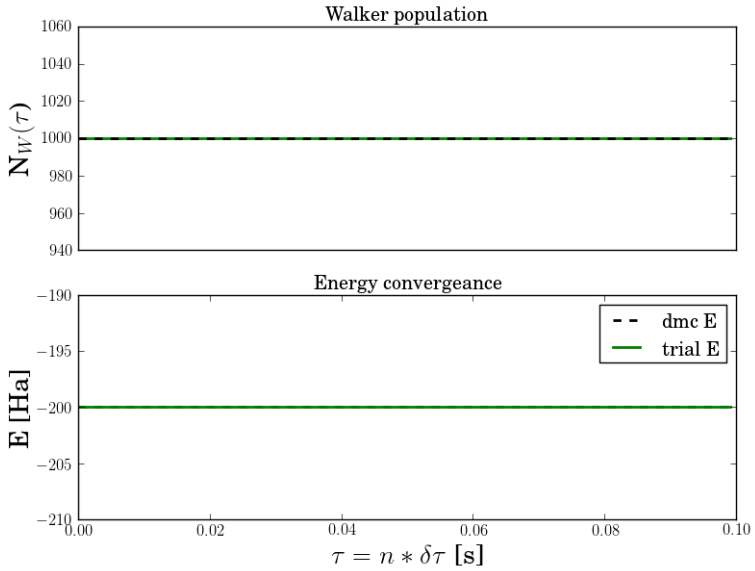


Figure 2.3: Illustration of the Diffusion Monte-Carlo (DMC) energy convergence for the Neon atom listed in Table 2.3. The trial energy is fixed at the exact ground state energy as required. The number of walkers are constant, implying an approximately zero variance in the samples.

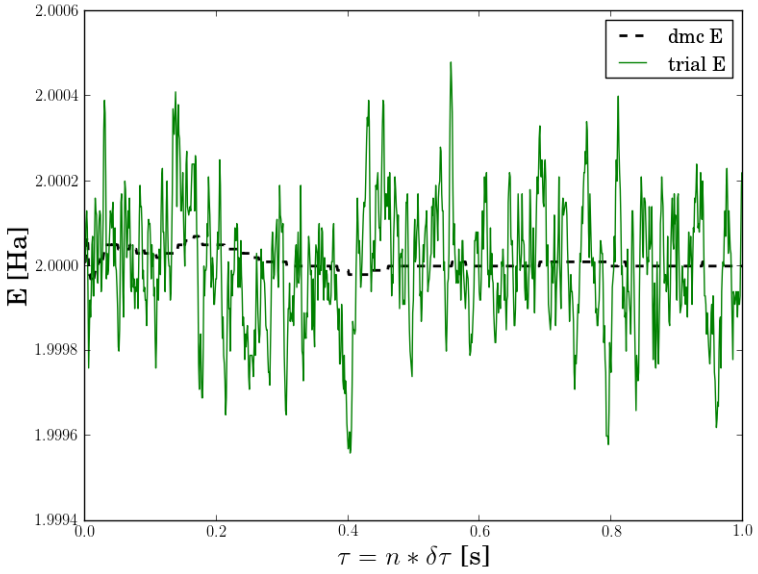


Figure 2.4: Illustration of the Diffusion Monte-Carlo (DMC) energy convergence for a two-particle quantum dot with frequency $\omega = 1$. The calculations are done with a variational parameter $\alpha = 0.75$, where as the exact wave function is given for $\alpha = 1$. Unlike the case with the exact wave function presented in figure 2.3, the trial energy oscillates around the exact value $E_0 = 2.0$. The final result reveals a DMC energy of 2.00000(2), where the original Variational Monte-Carlo (VMC) energy was 2.0042(3). This illustrates the power of DMC contra VMC in the interesting cases where the exact wave function is unknown. The calculation was done using 10000 random walkers.

2.3 Quantum Dots

The focus regarding quantum dots has been on studying the distribution of electrons as a function of the level of confinement. In addition, ground state energies are provided and compared to other many-body methods to demonstrate the efficiency and precision of Variational Monte-Carlo (VMC) and Diffusion Monte-Carlo (DMC). In the case of two dimensional quantum dots, there are multiple published results, however, for three dimensions this is not the case. An introduction to quantum dots is given in Section ??.

The double-well quantum dot has not been a focus in this thesis, however, some simple results are provided to demonstrate the flexibility of the code.

2.3.1 Ground State Energies

Two dimensional quantum dots

Table 2.5 presents the calculated ground state energies for two-dimensional quantum dots in addition to corresponding results from methods such as Similarity Renormalization Group theory (SRG), Coupled Cluster Singles and Doubles (CCSD) and Full Configuration Interaction (FCI). In addition, some previously published DMC results are supplied. The references are listed in the table caption.

In light of the variational nature of DMC and VMC, the results show that DMC provides a more precise estimate for the ground state energy than VMC, both in terms of lower energies and lower errors. The exact energy in the case of two electrons with $\omega = 1$ has been calculated in Ref. [12] and is $E_0 = 3$, which is in excellent agreement with the presented results.

The DMC energies calculated in this thesis has a lower error than those provided in Ref. [13]. This may only be due to the fact that the calculations in this thesis have been run on a super computer. Running smaller simulations on fewer processors results in larger errors. Both the implementations successfully agree with the FCI result for two particles, which strongly indicates that the disagreements in results are a result of systematic errors.

In the case of two particles, DMC and FCI agree up to five decimals, which leads to the conclusion that DMC indeed is a very precise method. The SRG method is not variational in the sense that it can undershoot the exact energy. The DMC result should thus not be read as less precise in the cases where SRG provides a lower energy estimate. Diffusion Monte-Carlo and SRG are in excellent agreement for a large number of particles compared to FCI and CCSD, which drift away from the DMC results as their basis sizes shrink.

For high frequencies, the VMC energy is higher than the CCSD energy. The fact that both the methods are variational implies that CCSD performs better than VMC in this frequency range. However, looking at the results the lower frequency range, it is clear that VMC performs better than CCSD. This is due to the fact that CCSD struggles with convergence as the correlations within the system increase, indicated by the decrease in number of shells used to perform the calculations.

The DMC energy is overall less than the CCSD energy, which, due to the variational nature of the methods, implies that DMC performs better than CCSD. The results for 56 particles are in excellent agreement with each other.

N	ω	E_{VMC}	E_{DMC}	$E_{\mathrm{ref}}^{(a)}$	$E_{\mathrm{ref}}^{(b)}$	$E_{\mathrm{ref}}^{(c)}$	$E_{\mathrm{ref}}^{(d)}$
2	0.01	0.07406(5)	0.073839(2)	-	-	0.0738 {23}	0.07383505 {19}
	0.1	0.44130(5)	0.44079(1)	-	-	0.4408 {23}	0.44079191 {19}
	0.28	1.02215(5)	1.02164(1)	-	0.99263 {19}	1.0217 {23}	1.0216441 {19}
	0.5	1.66021(5)	1.65977(1)	1.65975(2)	1.643871 {19}	1.6599 {23}	1.6597723 {19}
	1.0	3.00030(5)	3.00000(1)	3.00000(3)	2.9902683 {19}	3.0002 {23}	3.0000001 {19}
6	0.1	3.5690(3)	3.55385(5)	-	3.49991 {18}	3.5805 {22}	3.551776 {9}
	0.28	7.6216(4)	7.60019(6)	7.6001(1)	7.56972 {18}	7.6254 {22}	7.599579 {6}
	0.5	11.8103(4)	11.78484(6)	11.7888(2)	11.76228 {18}	11.8055 {22}	11.785915 {6}
	1.0	20.1902(4)	20.15932(8)	20.1597(2)	20.14393 {18}	20.1734 {22}	20.160472 {8}
12	0.1	12.3162(5)	12.26984(8)	-	12.2253 {17}	12.3497 {21}	12.850344 {3}
	0.28	25.7015(6)	25.63577(9)	-	25.61084 {17}	25.7095 {21}	26.482570 {2}
	0.5	39.2343(6)	39.1596(1)	39.159(1)	39.13899 {17}	39.2194 {21}	39.922693 {2}
	1.0	65.7905(7)	65.7001(1)	65.700(1)	65.68304 {17}	65.7399 {21}	66.076116 {3}
20	0.1	30.0729(8)	29.9779(1)	-	29.95345 {16}	30.2700 {8}	34.204867 {1}
	0.28	62.0543(8)	61.9268(1)	61.922(2)	61.91368 {16}	62.0676 {20}	67.767987 {1}
	0.5	94.0236(9)	93.8752(1)	93.867(3)	93.86145 {16}	93.9889 {20}	100.93607 {1}
	1.0	156.062(1)	155.8822(1)	155.868(6)	155.8665 {16}	155.9569 {20}	164.61280 {1}
30	0.1	60.584(1)	60.4205(2)	-	60.43000 {15}	61.3827 {9}	-
	0.28	124.181(1)	123.9683(2)	-	123.9733 {15}	124.2111 {9}	-
	0.5	187.294(1)	187.0426(2)	-	187.0408 {15}	187.2231 {19}	-
	1.0	308.858(1)	308.5627(2)	-	308.5536 {15}	308.6810 {19}	-
42	0.1	107.881(1)	107.6389(2)	-	-	111.7170 {8}	-
	0.28	220.161(1)	219.8426(2)	-	219.8836 {14}	222.1401 {8}	-
	0.5	331.002(1)	330.6306(2)	-	330.6485 {14}	331.8901 {8}	-
	1.0	544.2(8)	542.9428(8)	-	542.9528 {14}	543.1155 {18}	-
56	0.1	176.269(2)	175.9553(7)	-	-	186.1034 {9}	-
	0.28	358.594(2)	358.145(2)	-	-	363.2048 {9}	-
	0.5	538.5(6)	537.353(2)	-	-	540.3430 {9}	-
	1	880.2(7)	879.3986(6)	-	-	879.6386 {17}	-

Table 2.5: Ground state energy results for two dimensional N-electron quantum dots with frequency ω. Refs. (a): F. Pederiva [13] (DMC), (b): S. Reimann [14] (Similarity Renormalization Group theory), (c): C. Hirth [1] (Coupled Cluster Singles and Doubles), (d): V. K. B. Olsen [7] (Full Configuration Interaction). The numbers inside curly brackets denote the number of shells used above *Fermi-level* to construct the basis for the corresponding methods.

N	ω	E_{VMC}	E_{DMC}	E_{ref}
2	0.01	0.07939(3)	0.079206(3)	-
	0.1	0.50024(8)	0.499997(3)	0.5
	0.28	1.20173(5)	1.201725(2)	-
	0.5	2.00005(2)	2.000000(2)	2.0
	1.0	3.73032(8)	3.730123(3)	-
8	0.1	5.7130(6)	5.7028(1)	-
	0.28	12.2040(8)	12.1927(1)	-
	0.5	18.9750(7)	18.9611(1)	-
	1.0	32.6842(8)	32.6680(1)	-
20	0.1	27.316(2)	27.2717(2)	-
	0.28	56.440(2)	56.3868(2)	-
	0.5	85.714(2)	85.6555(2)	-
	1.0	142.951(2)	142.8875(2)	-

Table 2.6: Ground state energy results for three-dimensional N-electron quantum dots with frequency ω. The values in the fifth column is exact calculations taken from Ref. [12]. The VMC result is as expected always higher than the corresponding DMC result.

Three dimensional quantum dots

The results for three dimensional quantum dots are presented in Table 2.6. Three dimensional quantum dots do not have the same foothold in literature as the two dimensional ones, hence no results are listed except for some exact solutions taken from Ref. [12].

As expected, DMC reproduces the exact results for two particles. Compared to the exact results for two dimensions, which was reproduced with five digit precision, the exact results are reproduced with six decimal precision for three dimensions. This strongly indicates that for two electrons, DMC behaves better for three dimensions than for two. For higher number of particles, however, the errors are of the same order of magnitude as for two dimensions, leading to the conclusion that DMC performs equally good in either dimension for quantum dots.

2.3.2 One-body Densities

The one-body densities are calculated using the methods described in Section **??**.

Figure 2.5 presents the one-body densities for two-dimensional quantum dots. It is clear that the distribution is following a clear trend: Odd number of closed shells ($N = 2$, 12, 30) are all similar in shape. It is almost as if the $N = 2$ case is a zoom-in of the $N = 12$ top, which in turn is a zoom-in of the $N = 30$ top. A physical explanation is that the shapes are conserved due to the fact that they represent energetically favorable configurations.

The same trend is present for the even numbered shells ($N = 6$, 20, 42) as well. The density for $N = 56$, which is not included in the figure, further demonstrates this trend. Viewing the distributions as a sequence of images, from the lowest number of particles to the highest, it is apparent that the shape propagates very much like water ripples. It is remarkable how the solutions to the most complex of equations can come in the form of simple patterns found all around nature.

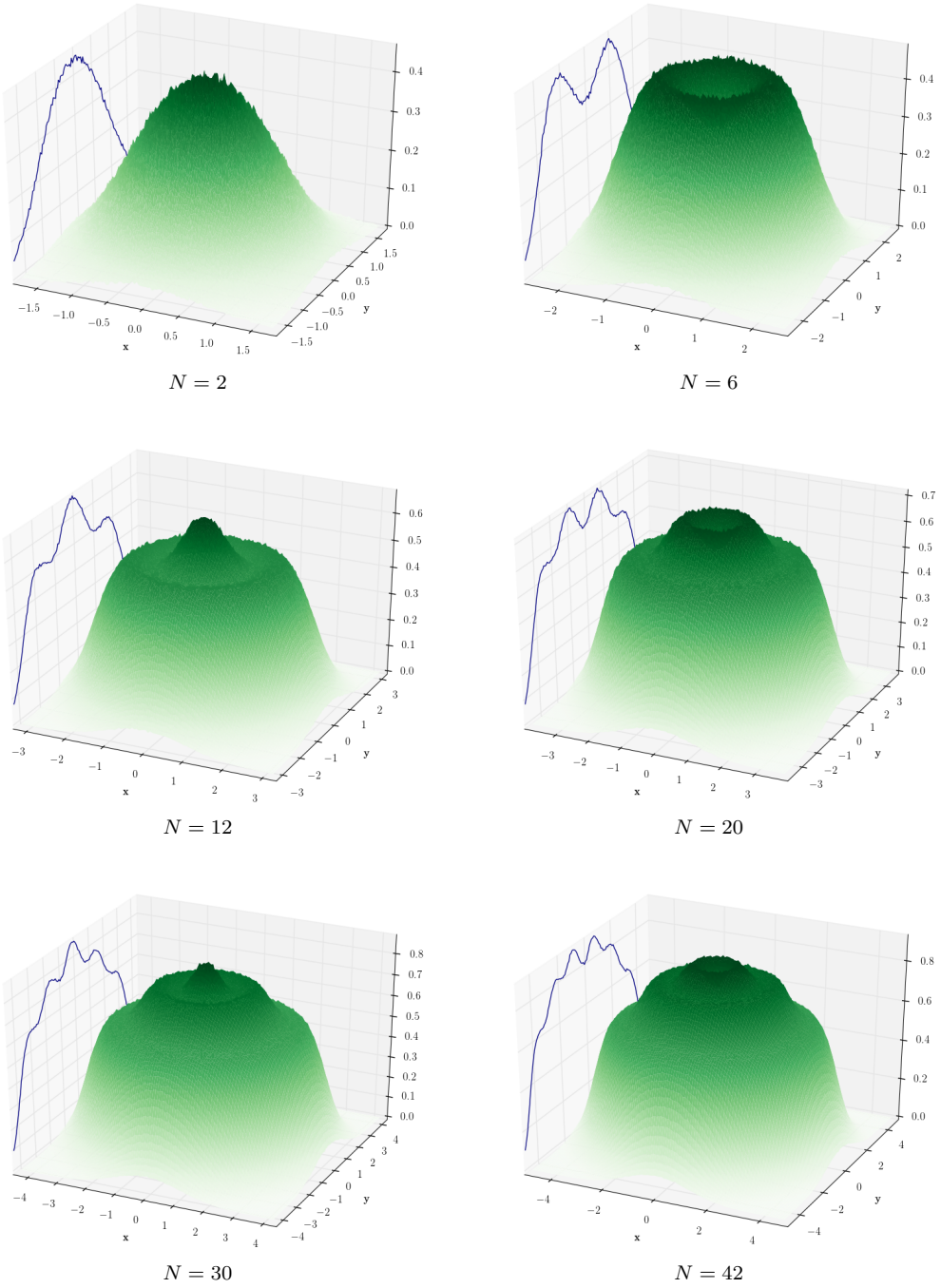


Figure 2.5: Diffusion Monte-Carlo one-body densities for two dimensional quantum dots with frequency $\omega = 1$. The number of particles N are listed below each density. It is apparent that the density behaves much like water ripples as the number of particles increase, conserving the shape in an oscillatory manner.

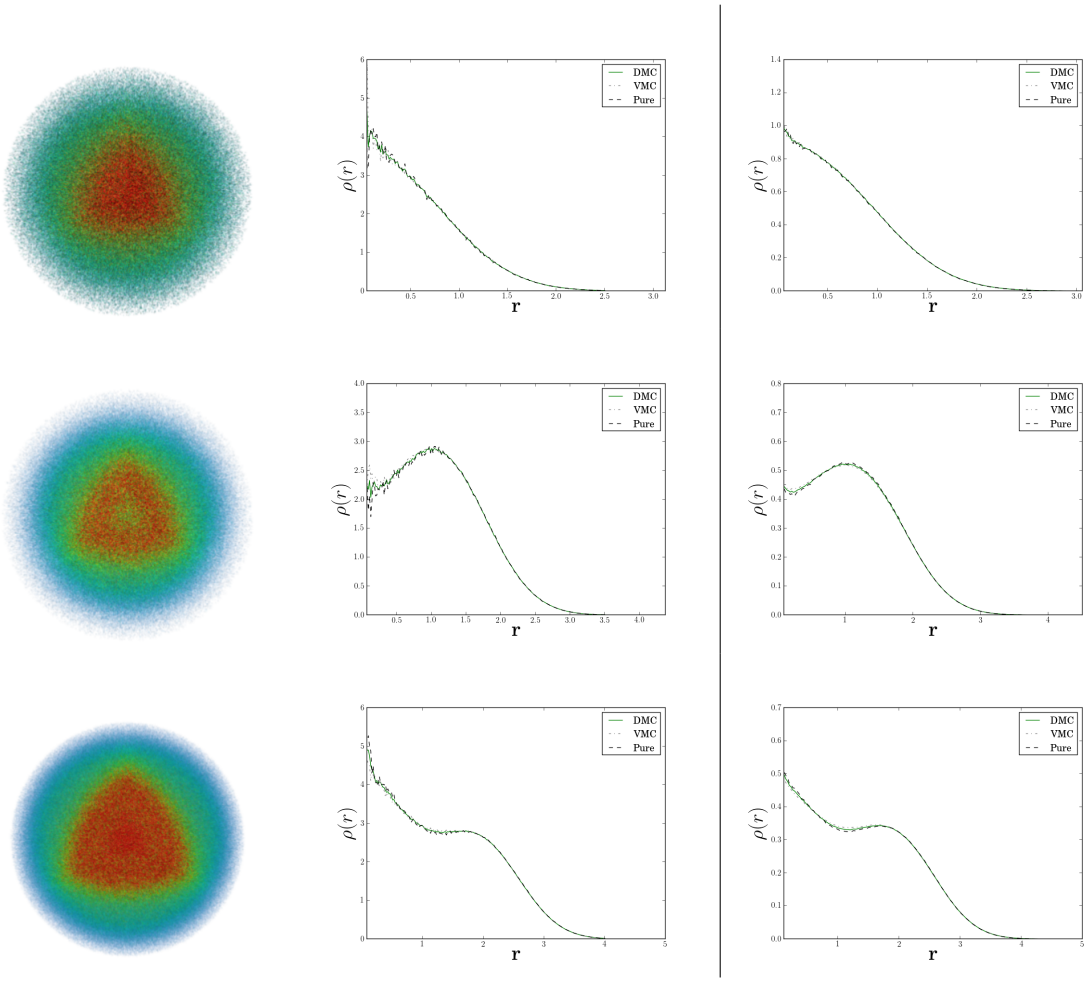


Figure 2.6: Left and middle column: One-body densities for quantum dots in three dimensions with frequency $\omega = 1$. A quarter of the spherical density is removed to present a better view of the core. Red and blue color implies a low and high electron density respectively. From top to bottom, the number of particles are 2, 8 and 20. Right column: One-body densities for two dimensional quantum dots for $N = 2$, 6 and 12 electrons (from the top) with $\omega = 1$. It is apparent that the shape of the density is conserved as the third dimension is added. The radial densities are not normalized. Normalizing the densities would only change the vertical extent.

Adding a third positional dimension changes the shape of the Schrödinger equation, which due to the Coulomb interaction is not separable in Cartesian coordinates. In other words, by going to three dimensions, the general shape of the solutions are expected to change. Nevertheless, by looking at the one-body densities for three dimensions in Figure 2.6, it is apparent that the general density profile is unchanged with respect to the number of closed shells. However, by comparing the two dimensional density for $N = 20$ electrons from Figure 2.5 with the three dimensional one for the same number of particles, it is clear that the shape of the densities are not conserved with respect to N alone.

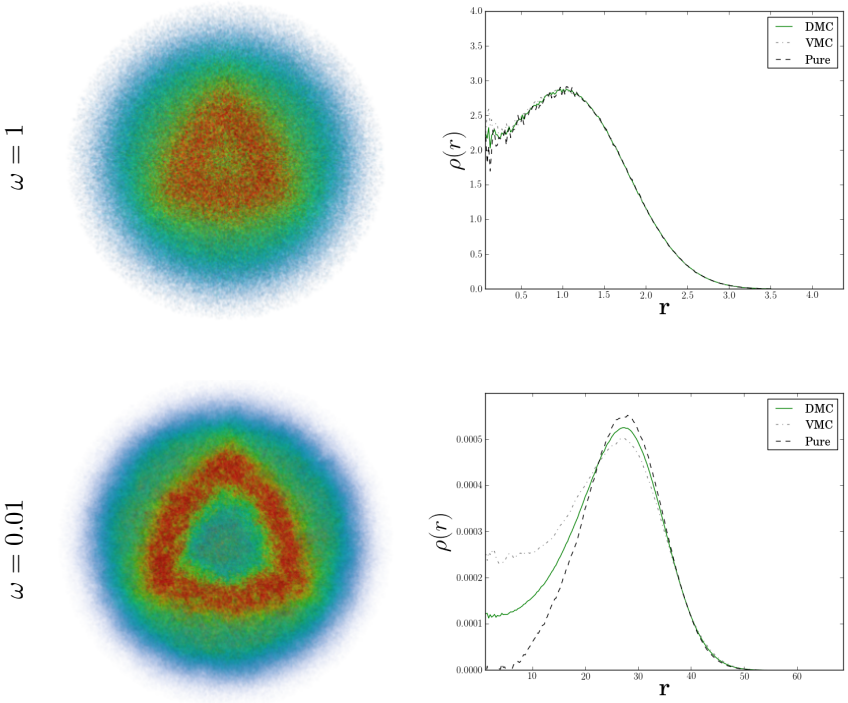


Figure 2.7: Comparison of the one-body densities for quantum dots in three dimensions for $N = 8$ electrons for high and low frequency ω. It is apparent that the distribution becomes more narrow as the frequency is reduced. Red and blue color implies a low and high electron density respectively. A quarter of the spherical density is removed to present a better view of the core.

2.3.3 Lowering the frequency

From Figure 2.7 it is apparent that lowering the frequency shifts the electrons away from the center of the potential. Moreover, Figure 2.8 reveals that the outer lying shells become just as populated as the inner lying shells, which means that all of the shells contain the same amount of particles on average.

This behavior is expected since the lower the frequency is set, the lower the energy cost of populating a higher shell becomes.

An equally populated shell structure implies that it is not energetically efficient for the electrons to occupy several shells simultaneously, indicating that crystallization becomes the preferred positional state[1]. This effect is called *Wigner Crystallization*, and is expected for confined electrons with a potential energy which dominates the kinetic energy [15]. This effect has strong experimental evidence [16, 17].

The fact that the correlation energy dominates the kinetic energy is clear by looking at Figure 2.9. The kinetic energy contribution is almost constant with respect to the frequency, while the Coulomb energy contribution rapidly increases.

Another very interesting result is that the symmetries for the two - and three dimensional closed shells solutions discussed in the previous section are broken at lower frequencies. The radial density for $N = 8$ in Figure 2.7 does in no way resemble the $N = 6$ density in Figure 2.8.

[1] Unless at least one particle is frozen in the QMC simulations, the Quantum Dot densities should always be rotationally symmetric. Crystallization in a QMC perspective comes thus not in the form of actual "crystals", but rather as a rotated crystallized state.

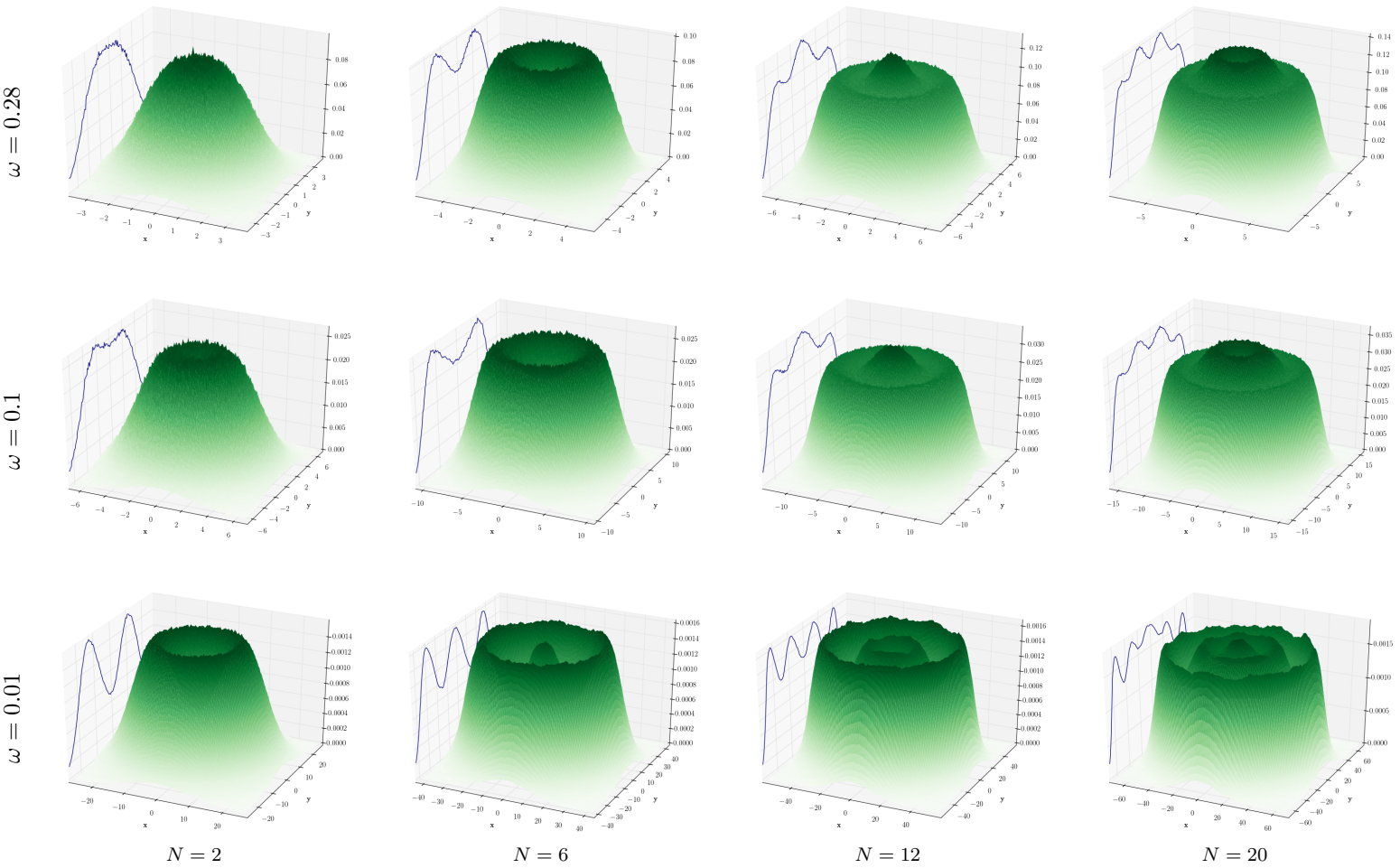
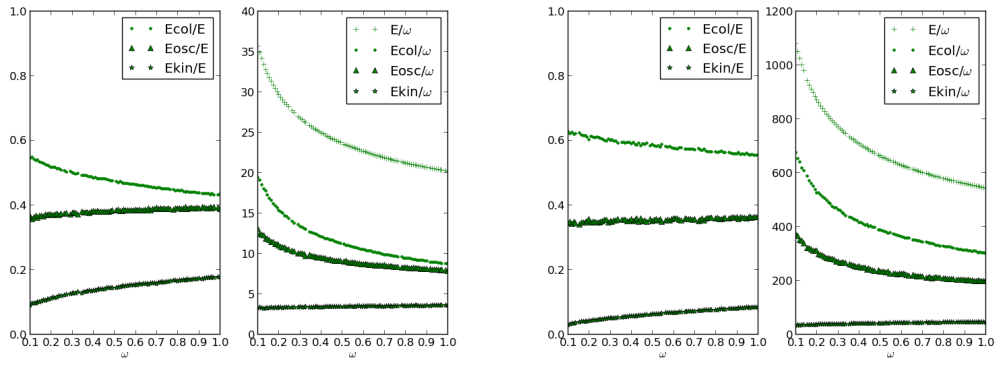


Figure 2.8: DMC One-body densities for Quantum Dots for decreasing oscillator frequencies ω and increasing number of particles N. Each row represents a given ω, and each column represents a given N. Notice that the densities for $\omega = 1$ (from figure 2.5) are indistinguishable from those of $\omega = 0.28$ except for their radial extent. This trend has been verified in the case of $N = 30$, 42 and 56 electrons as well as for $\omega = 0.5$, however, for the sake of transparency, these results are left out of the current figure. Results for $N = 30$, 42 and 56 particles for $\omega = 0.01$ has too wide radial extent to converge with the computational resources available at the present time and has thus not been computed.



$N = 6$ $N = 42$

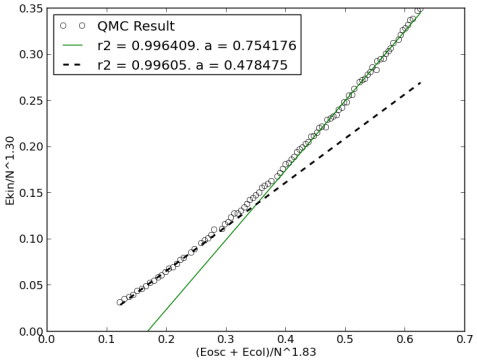
Figure 2.9: The relative magnitude of the different energy sources as a function of the frequency ω (left) together with the magnitude of the sources' energy contributions scaled with the oscillator frequency (right). The plots are supplied with legends to increase the readability. The different energy sources are kinetic (kin), oscillator (osc) and Coulomb (col). It is apparent that the kinetic contribution is constant in both cases and the potential contribution is constant for the relative energies. The values are calculated using two dimensional quantum dots. The number of electrons N are displayed beneath each respective plot.

In other words, quantum dots fall into a crystallized state at low frequencies. In light of the previous discussion regarding Wigner crystallization, this follows as a consequence of the kinetic energy being dominated by the Coulomb energy. Since the Coulomb energy in turn dominates the oscillator energy (the left-hand side parts of the plots in Figure 2.9), the interesting quantity to study is the kinetic energy as a function of the total potential energy.

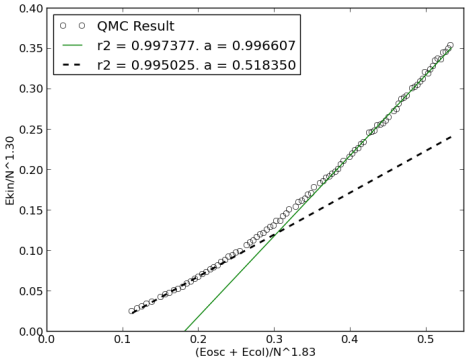
This is closely related to the *virial theorem* from classical mechanics. The quantum mechanical version of the virial theorem was proven by Fock in 1930 [18] and reads

$$V(\mathbf{r}) \propto r^{\gamma} \quad \longrightarrow \quad \langle \widehat{\mathbf{T}} \rangle = \frac{\gamma}{2} \langle \widehat{\mathbf{V}} \rangle, \tag{2.3}$$

where $\widehat{\mathbf{T}}$ and $\widehat{\mathbf{V}}$ denote the kinetic - and potential energy operators respectively. The important conclusion which can be drawn from this is that a constant slope in the kinetic - vs. potential energy plot implies that the systems behave identically, that is, they follow the same effective potential and thus have similar eigenstates. From Figure 2.10 it is apparent that there is a remarkably constant slope for two different regions: High - and low kinetic energy, which by looking at Figure 2.9 corresponds to high - and low frequencies. This proves that the system changes characteristics as the frequency changes, which explains the sudden change in the density profiles.



$N = 6$



$N = 42$

Figure 2.10: The total kinetic energy vs. the total potential energy of two dimensional quantum dots. The number of electrons N are displayed beneath each respective plot. The axes are scaled with a power of N to collapse the data to the same axis span. Once the kinetic energy drops below a certain energy dependent on the number of particles, the slope changes, which in light of the virial theorem from Eq. (2.3) indicates that the overall system changes properties. The data is fitted with linear lines with slopes a. The parameter $r2$ indicates how well the data fits a linear line. An exact fit yields $r2 = 1$.

2.3.4 Simulatings in a Double-well

awesome

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 2.7: 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. [4] for He through Ar, and [5] 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.

2.4 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 **??**.

2.4.1 Ground State Energies

Table 2.7 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.

As expected, VMC performs rather poorly compared to DMC. In comparison to quantum dots, where the two methods produced significantly similar energies, atoms have an addition approximation in the trial wave function due to the fact that the solid harmonics are used instead of the spherical harmonics. This is covered in detail in Section **??**. This fact further demonstrates that DMC, unlike VMC, has the power to perform well even in situations where the trial wave function is not optimal.

2.4.2 One-body densities

The one-body densities for the *noble gases*, that is, the closed shell atoms, are presented in Figure 2.12. Comparing these to the one-body densities for the alkaline earth metals, i.e. Be, Mg, etc., in Figure 2.11, it is clear that the noble gases have a more confined electron distribution. This corresponds well to the

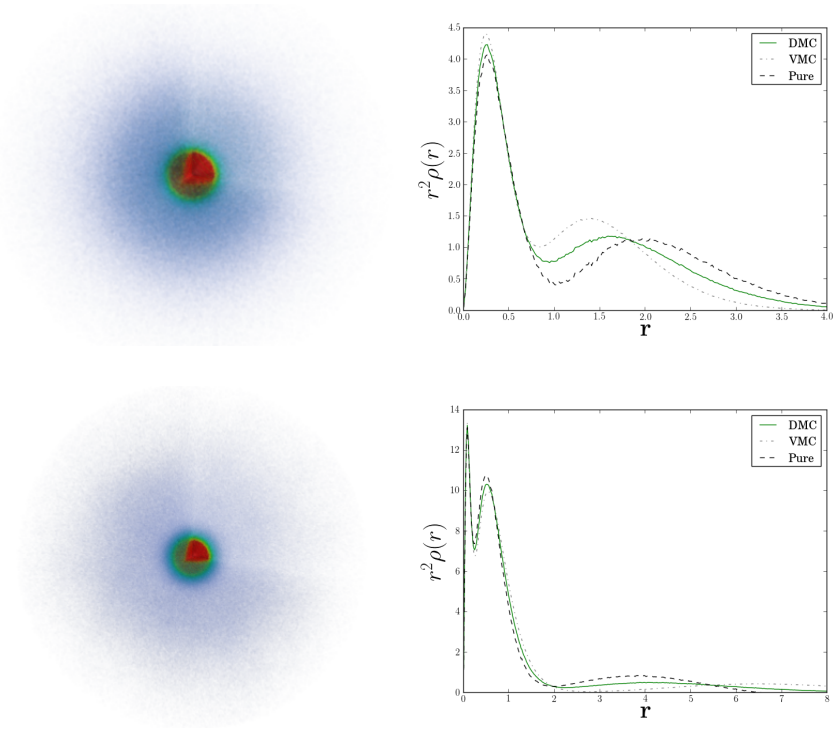


Figure 2.11: Three dimensional one-body density (left column) and angular averaged radii (right column) for alkaline earth metals; Beryllium (top) and Magnesium (bottom). The color bar shows increasing values from left to right. Notice that it is not the radial one-body density (which is too steep to reveal characteristics) in the right column, but the electron. Compared to the noble gases in Figure 2.12, 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 [19]. Red and blue color implies a low and high electron density respectively.

fact that noble gases do not form compound materials, i.e. molecules [19]. 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 **??**.

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.

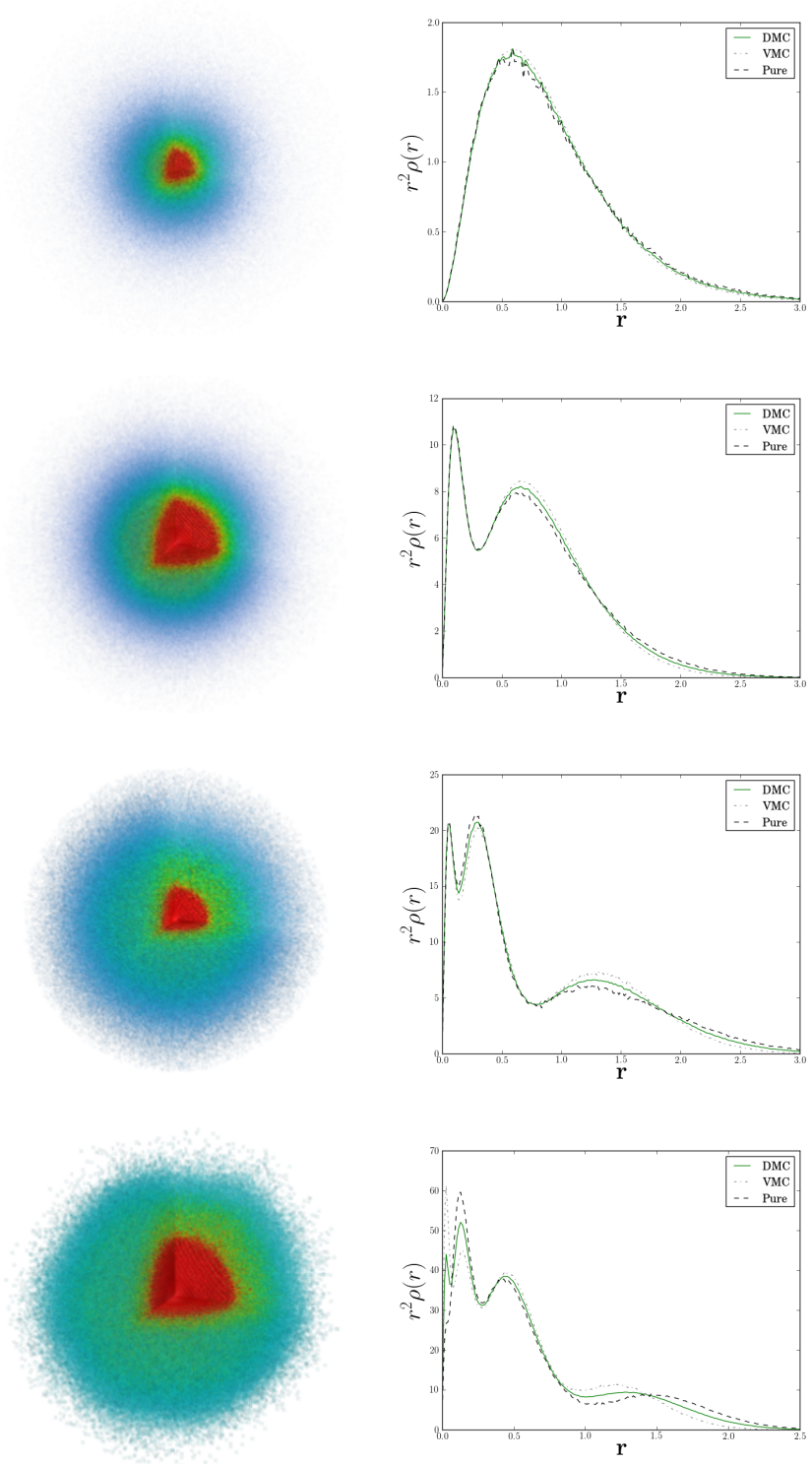


Figure 2.12: One-body densities for noble gases. Counting top to bottom: Helium, Neon, Argon and Krypton. A quarter of the spherical density is removed to present a better view of the core. Red and blue color implies a low and high electron density respectively. Note that the radial densities are scaled 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 2.8: Ground state energies for homonuclear diatomic molecules calculated using VMC and DMC. The distance between the atoms R are taken from Ref. [2] for H_2 and from Ref. [20] for Li_2 to O_2. The experimental energies are taken from Ref. [2] for H_2 and from Ref. [3] for Li_2 to O_2. As expected DMC is closer to the experimental energy than VMC. $\epsilon_{\mathrm{rel}} = |E_{\mathrm{DMC}} - \mathrm{Expt.}|/|\mathrm{Expt.}|$ denotes the relative error between the DMC result and the experimental value. As expected, this error increase with the atomic number.

2.5 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 **??**.

2.5.1 Ground State Energies

Table 2.8 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 2.7. 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.

2.5.2 One-body densities

Figure 2.13 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 [19]. 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.

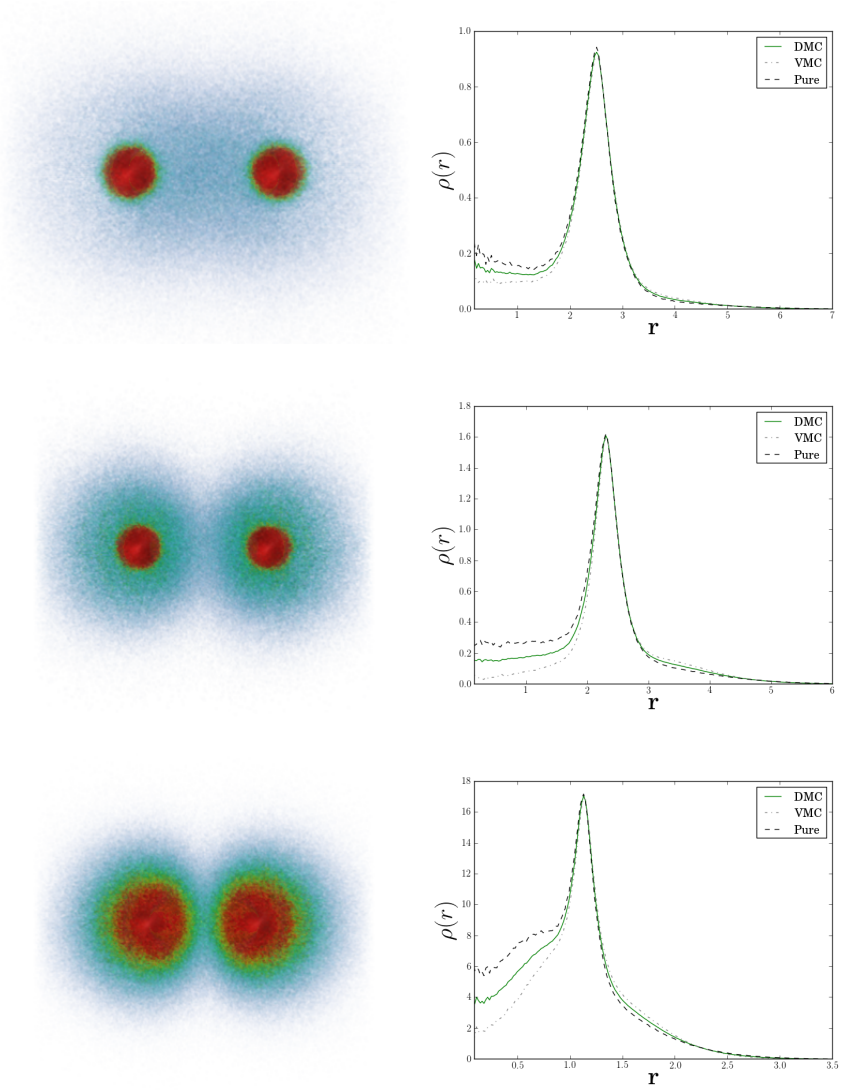


Figure 2.13: One-body densities of Li_2 (top), Be_2 (middle) and O_2 (bottom). The figures to the left are spherical densities sliced through the middle to reveal the core structure. The figures to the right are radial one-body densities projected on the nucleus-nucleus axis. Red and blue color indicates a high and low electron density respectively. The right-hand figures are symmetric around the origin.

2.5.3 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 [9, 10]

$$V(R) = 4\epsilon \left(\left(\frac{\sigma}{R} \right)^{12} - \left(\frac{\sigma}{R} \right)^{6} \right), \tag{2.4}$$

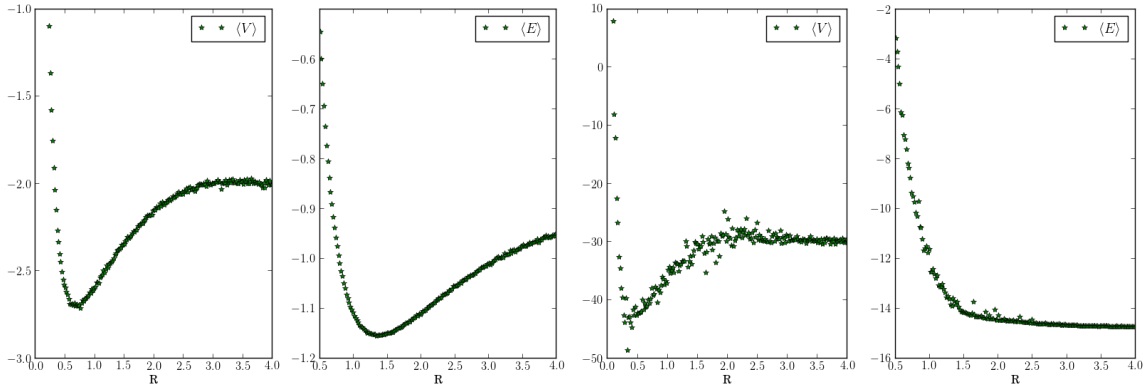
where ϵ and σ are parameters which can be fit to a given system.

However, the force field can be parameterized in greater detail using QMC calculations, resulting in a more precise molecular dynamics simulation [6]. 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

$$F_{\mathrm{MD}} = \frac{\mathrm{d}\langle E \rangle}{\mathrm{d}R}. \tag{2.5}$$

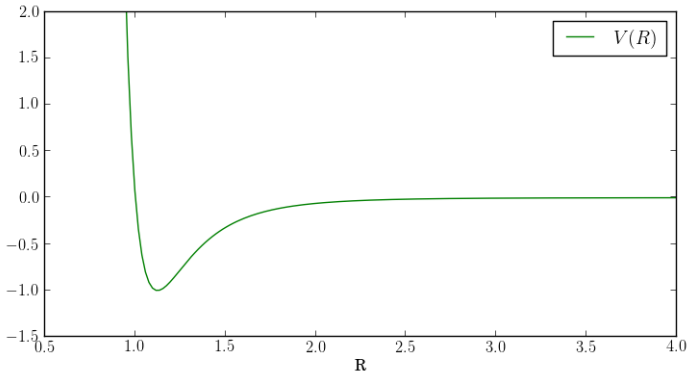
Expressions for this derivative can be obtained in ab-initio methods by using the Hellmann-Feynman theorem [6]. However, the derivative can be approximated by the slope of the energy in Figure 2.14. The figure shows that there are clear similarities between the widely used Lennard-Jones 12-6 potential 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 [21].



(a) H_2 (b) Li_2

Figure 2.14: 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 [19]. 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.

3

Conclusions

The focus of this thesis was to develop an efficient and general Quantum Monte-Carlo (QMC) solver which could simulate systems with a large number of particles. This was achieved by using object oriented C++, resulting in 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 alone, 112 of these expressions were needed to simulate a 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, to construct 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, it sets up the proper environment for the simulation, that is, makes sure the simulation output is stored in folders whose names reflect the specific simulation. Without this script it would be impossible not to loose track of the data.

With an increase in data came an increase in the time needed to analyze it. To counter this, a script which automatically converted the output from different simulations to a Latex table was implemented, however, this did not solve the problem of efficiency, as convergence still had to be verified manually. To solve the problem permanently, the Christmas holiday was used to implement a visualization tool which made real-time visualization of the simulations possible. Moreover, implementing new data was made extremely easy. This script is described in 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 [22], and VMC calculations of atoms up to Silicon [23]. 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 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 implemented five different systems demonstrates the code's ability to tackle many different systems. The implementation of molecules and the double-well were done by adding no more than 200 lines of code. Additionally, fully implementing the three dimensional quantum dots were 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 [7] 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 the 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 momentum space QMC. This has the potential of describing nuclear interactions in great detail [24].

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 [25] or using a Coupled Cluster wave function [11] 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 [26], are necessarily orthogonal [27]. 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* [28] with a graphical user interface coded using *PySide* [29]. 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:

```
1 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 is 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 2.6 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.

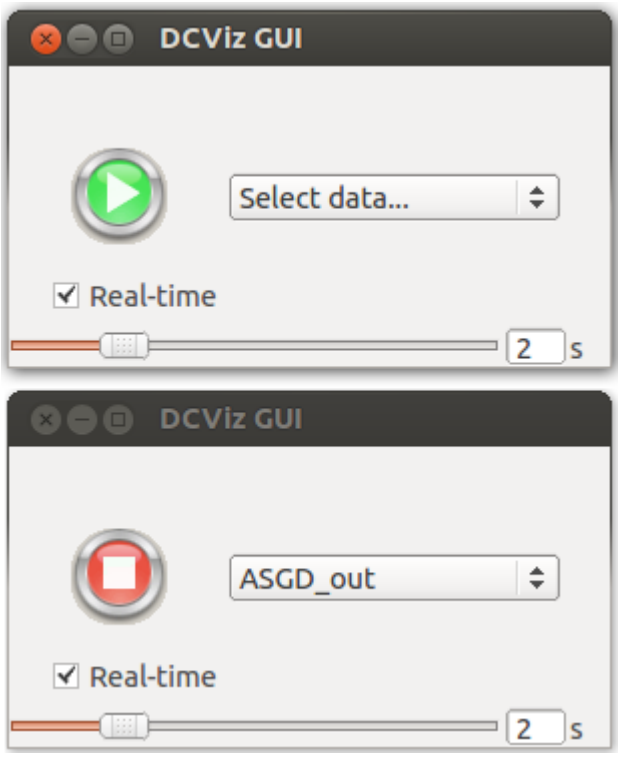


Figure B.1: Two consequent screen shots of the GUI. The first (top) is taken directly after the detector is finished loading files into the drop-down menu. The second is taken directly after the job is started.

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 [30]

This appendix will be focused 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 not human, given that the complexity of the expressions is proportional to the magnitude of the quantum number, which again scales with the number of particles. 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

```
...

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]: from sympy import factor, Symbol, printing

In [8]: R = Symbol('r') #Creates a symbolic equivalent of the mathematical r

In [9]: diff(phi, x).factor() #Factors out common factors
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))'

In [10]: diff(phi, x).factor().subs(r, R) #replaces (x^2 + y^2 + z^2)^(1/2) with r
Out[10]: '-Z*x*(Z*r - 4)*exp(-Z*r/2)/(2*r)'

In [11]: print printing.latex(diff(phi, x).factor().subs(r, R))
- \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 a 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. nml 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

A convenient function for returning generic parts of the expressions, i.e. the exponential parts. A set of quantum numbers are supplied on input in case the generic expression is dependent of 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 be taken out of the Latex expression 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__

__preCalc__
    return __return__

}
"""
```

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:

```
double dell_hydrogenic_9_y::eval(const Walker* walker, int i) {

    y = walker->r(i, 1);
    z = walker->r(i, 2);

    z2 = z*z;

    //-y*(k*(-r^2 + 3*z^2) + 6*r)*exp(-k*r/3)/(3*r)

    psi = -y*((*k)*(-walker->get_r_i2(i) + 3*z2) + 6*walker->get_r_i(i))/(3*walker->
        get_r_i(i));
    return psi*(*exp_factor);

}
```

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

```
def getCReturn(self, expr, i):
     return "psi*(*exp_factor);"

def getCPre(self, expr, i):
     qNums = self.stateMap[i]
     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

```
basis_functions[0] = new hydrogenic_0(k, k2, exp_factor_n1);
basis_functions[1] = new hydrogenic_1(k, k2, exp_factor_n2);
//...
basis_functions[5] = new hydrogenic_5(k, k2, exp_factor_n3);
//..
basis_functions[14] = new hydrogenic_14(k, k2, exp_factor_n4);
//..
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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

$\phi_4 \to \phi_{2,1,-1}$	
$\phi(\vec{r})$	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.

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

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

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

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

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

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] C. Hirth, "Studies of quantum dots: Ab initio coupled-cluster analysis using OpenCL and GPU programming," Master's thesis, University of Oslo, 2012.

[2] M. Kalos, "A new Look at Correlations in Atomic and Molecular Systems. Application of Fermion Monte Carlo Variational Method," *Int. Journal of Quantum Chemistry*, 1981.

[3] S. Datta, S. A. Alexander, and R. L. Coldwell, "Properties of selected diatomics using variational Monte Carlo methods," *The Journal of Chemical Physics*, vol. 120, no. 8, pp. 3642–3647, 2004. [Online]. Available: http://link.aip.org/link/?JCP/120/3642/1

[4] M. Degroote, "Faddeev random phase approximation applied to molecules," *The European Physical Journal Special Topics*, vol. 218, no. 1, pp. 1–70, 2013.

[5] H. Partridge, "Near Hartree–Fock quality GTO basis sets for the first- and third-row atoms," *The Journal of Chemical Physics*, vol. 90, no. 2, pp. 1043–1047, 1989. [Online]. Available: http://link.aip.org/link/?JCP/90/1043/1

[6] A. Badinski, P. D. Haynes, J. R. Trail, and R. J. Needs, "Methods for calculating forces within quantum Monte Carlo simulations," *Journal of Physics: Condensed Matter*, vol. 22, no. 7, p. 074202, 2010. [Online]. Available: http://stacks.iop.org/0953-8984/22/i=7/a=074202

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

[8] J. Høgberget, *Git Repository: LibBorealis*, 2013. [Online]. Available: http://www.github.com/jorgehog/QMC2

[9] M. Barisik and A. Beskok, "Equilibrium molecular dynamics studies on nanoscale-confined fluids," vol. 11, no. 3, pp. 269–282, Sep. 2011.

[10] K. P. Travis and K. E. Gubbins, "Poiseuille flow of Lennard-Jones fluids in narrow slit pores," *The Journal of Chemical Physics*, vol. 112, no. 4, pp. 1984–1994, 2000. [Online]. Available: http://link.aip.org/link/?JCP/112/1984/1

[11] A. Roggero, F. Pederiva, and A. Mukherjee, arXiv:1304.1549 [nucl-th].

[12] M. Taut, "Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem," *Phys. Rev. A*, vol. 48, pp. 3561–3566, Nov 1993. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevA.48.3561

[13] M. Pedersen Lohne, G. Hagen, M. Hjorth-Jensen, S. Kvaal, and F. Pederiva, "*Ab initio* computation of the energies of circular quantum dots," *Phys. Rev. B*, vol. 84, p. 115302, Sep 2011. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevB.84.115302

[14] S. Reimann, "Quantum-mechanical systems in traps and Similarity Renormalization Group theory," Master's thesis, University of Oslo, 2013.

[15] C. Yannouleas and U. Landman, "Symmetry breaking and quantum correlations in finite systems: studies of quantum dots and ultracold Bose gases and related nuclear and chemical methods," *Reports on Progress in Physics*, vol. 70, no. 12, p. 2067, 2007. [Online]. Available: http://stacks.iop.org/0034-4885/70/i=12/a=R02

[16] E. Wigner, "On the Interaction of Electrons in Metals," *Phys. Rev.*, vol. 46, pp. 1002–1011, Dec 1934. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRev.46.1002

[17] ——, "Effects of the electron interaction on the energy levels of electrons in metals," *Trans. Faraday Soc.*, vol. 34, pp. 678–685, 1938. [Online]. Available: http://dx.doi.org/10.1039/TF9383400678

[18] V. Fock, "Bemerkung zum Virialsatz," *Zeitschrift für Physik*, vol. 63, no. 11-12, pp. 855–858, Nov. 1930.

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

[20] C. Filippi and C. J. Umrigar, "Multiconfiguration wave function for quantum Monte Carlo calculations of first-row diatomic molecules," *The Journal of Chemical Physics*, vol. 105, Jul. 1996.

[21] A. C. T. van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, "ReaxFF: A Reactive Force Field for Hydrocarbons," *The Journal of Physical Chemistry A*, vol. 105, no. 41, pp. 9396–9409, 2001. [Online]. Available: http://pubs.acs.org/doi/abs/10.1021/jp004368u

[22] L. E. Lervåg, "VMC CALCULATIONS OF TWO-DIMENSIONAL QUANTUM DOTS," Master's thesis, University of Oslo, 2010.

[23] H. Sandsdalen, "Variational Monte Carlo studies of Atoms," Master's thesis, University of Oslo, 2010.

[24] A. Bulgac and M. M. Forbes, "Use of the discrete variable representation basis in nuclear physics," *Phys. Rev. C*, vol. 87, p. 051301, May 2013. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevC.87.051301

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

[26] D. Griffiths, *Introduction to Quantum Mechanics*, 2nd ed. Pearson, 2005.

[27] G. Golub and C. Van Loan, *Matrix computations*. Johns Hopkins Univ Press, 1996, vol. 3.

[28] (2013, May) Matplotlib website. [Online]. Available: http://matplotlib.org

[29] (2013, May) PySide website. [Online]. Available: http://qt-project.org/wiki/Category:LanguageBindings::PySide

[30] (2013, May) SymPy website. [Online]. Available: http://sympy.org