Studies of Quantum Dots using Machine Learning

by

Even Marius Nordhagen

THESIS

for the degree of

MASTER OF SCIENCE

Faculty of Mathematics and Natural Sciences
University of Oslo

October 2019

All illustrations in this thesis are created using the TikZ package [1] if nothing else is specified. The plots are produced using a controversial combination of Matplotlib [2] and PGFPlots [1].

Effort was made to follow the ISO 80000-2:2009 standard for mathematical signs and symbols [3], and the ISO 80000-9:2009 standard for quantities and units in physical chemistry [4].

The LaTeX document preparation system was applied for typesetting.

Abstract

Being able to solve the many-body Schrödinger equation accurately, in principle all physics and chemistry could be derived from first principles. However, exact wave functions of realistic and interesting systems are in general unavailable because they are non-deterministic polynomial-hard to compute [5], implying that we need to rely on approximations. The variational Monte Carlo (VMC) method is widely used for ground state studies, but requires a trial wave function ansatz which must trade off between efficiency and accuracy. The method has also many common features with machine learning algorithms, and as neural networks have shown impressive power as function approximators, the idea is to use a neural network as the trial wave function guess. For fermionic systems, like electronic structure systems, the wave function needs to obey Fermi-Dirac statistics, which typically is achieved using a Slater determinant. As a neural network hardly can model this feature, our approach is to replace the single-particle functions in the Slater determinant with restricted Boltzmann machines (RBM). To model the cusp correctly, correlation factors in the forms of a simple Jastrow factor and the well-known Padé-Jastrow factor were added, abbreviated RBM+SJ and RBM+PJ, respectively.

Our primary focus is on closed-shell circular quantum dots, where we compute the ground-state energy and electron density for two-dimensional systems with up to $N = 90$ electrons and three-dimensional systems with up to $N = 70$ electrons. The energy obtained by the RBM was reasonably close to experimental results, and it gradually got closer as we added more complex correlation factors. For the RBM+PJ, the energy was found to be lower than the VMC-energy for small dots, but for larger dots it was slightly higher. However, the one-body density profile reveals that the RBM gives more distinctly located electrons compared to the VMC method, which can be explained by the way the RBM models the correlations. From the two-body density profile, we also observe that the repulsive interactions get more significant as we add a Jastrow factor. Based on the electron densities and the energy distribution between kinetic and potential energy, it is certain that the various methods provide different electron configurations. For low-frequency dots, the electron density undergoes a phase transformation with an additional radial peak compared to high-frequency dots, indicating "incipient" Wigner localization.

The computational time consumption was found to be favorable for the RBM for small systems and VMC for large systems, which can be explained by the exploding number of variational parameters in the RBM as the system sizes increase. RBMs with Jastrow factors were significantly more computationally expensive than the other methods, and apparently, there is no point in adding a simple Jastrow factor when we can add a more complicated Jastrow factor.

Acknowledgements

After five exciting years at Blindern my time here has come to an end, and on that occasion I would like to acknowledge some people who have been influential throughout the studies. First, I would like to thank my excellent supervisor, Morten Hjorth-Jensen, whom I luckily got to know three years ago. From day one, you took me under your wing and enthralled me with your eager, massive knowledge and work ethic. Few things make me more motivated than talking with you, be it in real-life at Blindern or through video conversations from whatever place you happen to be at.

I would also like to acknowledge the support I have got from my parents and my sister. Albeit the schedule has been filled up and I have not spent as much time with you as I wanted, you have always been supportive and stand up for me anytime I need some help. Thanks to my friends (you know who you are) who are always ready for a beer (or ten) whenever I seek to disconnect from the studies. Those moments filled with lively discussions and dark humor have kept me motivated throughout the studies, which has been absolutely crucial.

The computational physics group is a funny composition of different people with a shared predilection for physics and programming. There are so many talented guys in the group, and I have really appreciated spending early mornings and late nights with you. Last, but not least, I would also like to thank my high school teacher, Jens Otto Opaker, who with his enthusiasm and dedication, got me hooked on science in the first place.

Contents

CHAPTER 1

Introduction

The properties and behavior of quantum many-body systems are determined by the laws of quantum physics which have been known since the 1930s. The time-dependent Schrödinger equation describes the binding energy of atoms and molecules, as well as the interaction between particles in a gas. Besides, it has been used to determine the energy of artificial structures like quantum dots, nanowires and ultracold condensates. As the quantum theory is the most precisely tested in the history of science [6], computer experiments are in principle capable of obtaining the energy as precise as laboratory experiments, and can in that sense replace laboratory experiments.

Although we know the laws of quantum mechanics, many challenges are encountered when calculating real-world problems. First, interesting systems often involve a large number of particles, which causes expensive calculations. Second, the correct wave functions are seldom known for complex systems, which is vital for measuring the observable accurately. Together, they are known as the many-body problem, which we in this thesis will attempt to solve using machine learning.

1.1 The many-body problem

In quantum mechanics, a many-body system contains three or more interacting particles. These interactions create so-called quantum correlations, which makes the wave function of the system a complicated object holding a large amount of information. As a consequence, exact or analytical calculations become impractical or even impossible, which is known as the many-body problem. Indeed, Paul Dirac recognized those problems already in 1929:

> "The general theory of quantum mechanics is now almost complete... ...The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble."
>
> Paul M. Dirac, *Quantum Mechanics of Many-electron Systems*, [7].

There are numerous approaches to solve this problem where approximative methods often are used to reduce the sometimes extreme computational cost. Popular methods include the Hartree-Fock method, which replaces the interaction by a mean-field, and methods like Full Configuration Interaction (FCI) and Coupled Cluster which seek to solve the problem by approximating the wave function with expansions. Lastly, quantum Monte Carlo (QMC) methods are different approaches attempting to solve the problem directly using a stochastic evaluation of the integrals occurring from the Schrödinger equation.

What all these methods have in common, is that they require significant amounts of physical intuition in order to work. In general, prior knowledge of the wave function, covering all the interactions and the cusps, is required. This knowledge is often unavailable, especially for complex systems, which also make accurate estimates of the observable unavailable. In this thesis, we will try to bypass this problem by inventing more flexible and robust methods that allow a relatively bad wave function guess. A natural base for this is the machine learning algorithms, as they can ”learn” themselves and thus hopefully find reasonable estimates through a training process. The apparent link between machine learning and quantum mechanics are the QMC methods, since they both are based on minimizing a *cost function* in order to obtain optimal configurations. The connection between machine learning and QMC methods, in particular, variational Monte Carlo (VMC), will be discussed thoroughly throughout this thesis

1.2 Machine learning

Machine learning has recently achieved immense popularity in fields such as computer vision, economics, autonomy - and science, due to its ability to learn without being explicitly programmed. As a branch of artificial intelligence, machine learning is based on studies of the human brain and attempts to recreate the way neurons in the brain process information.

Especially the artificial neural networks have experienced significant progress over the past decade, which can be attributed to an array of innovations. Most notably, the convolutional neural network (CNN) AlexNet [8] managed to increase the top-5 test error rate of image recognition with a remarkable 11.1% compared to the second-best back in 2012! Today, the CNNs have been further improved, and they are even able to beat humans in recognizing images [9]. Also, speech recognition algorithms have lately been revolutionized, thanks to recurrent neural networks (RNNs), and exceptionally long short-term memory (LSTM) networks. Their ability to recognize sequential (time-dependent) data made the technology good enough for entry to millions of people’s everyday-life through services such as Google Translate [10], Apple’s Siri [11] and Amazon Alexa [12]. It is also interesting to see how machine learning has made computers eminent tacticians using reinforcement learning. The Google-developed program AlphaGo demonstrated this by beating the 9-dan professional Sedol in the board game Go [13], before an improved version, AlphaZero, beat the at that time highest-rated chess computer, Stockfish, in chess [14]. Both these scenarios were unbelievable just a couple of decades ago.

Even though all these branches are both exciting and promising, they will not be discussed further in this work, since they will not work for our purposes. The reason is that they initially require a data set with known outputs in order to be trained; they obey so-called *supervised* learning. For our quantum mechanical systems, we do not have those targets and need to rely on *unsupervised* learning with the focus on restricted Boltzmann machines (RBMs). Lately, some effort has been put into this field, known as quantum machine learning. Carleo & Troyer [15] demonstrated the link between RBMs and QMC and named the states *neural-network quantum states* (NQS). They used the technique to study the Ising model and the Heisenberg model. Pfau *et al.* [16] went further and predicted the dissociation curves of the nitrogen molecule using a so-called fermionic neural network, and Flugsrud [17] investigated ground-state properties of circular quantum dots, also using RBMs. We will extend the work she did to larger quantum dots.

1.3 Quantum dots

Quantum dots are often called artificial atoms because of their common features to real atoms, where both are electron structure systems with electrons confined in an external potential. Quan-

tum dots are interesting both from a theoretical, technological and experimental point of view, and are therefore covered in different fields of research. Theoretically, quantum dot are interesting as they are relatively simple structures that can model a long range of phenomena. An example on this is the "incipient" Wigner localization, which has been observed in strongly interacting quantum dots where the potential energy dominates the kinetic energy [18, 19]. As those systems are more strongly correlated than, for instance, atoms, other challenges are encountered which might require other approaches. Research indicate that quantum Monte Carlo methods are the best suited techniques for studies of those systems [18].

Over the past decade, the popularity of quantum dots has increased, thanks to technological progress in semiconductor research. In particular, they are expected to be the next big thing in display technology due to their ability to emit photons of specific wavelengths, a smoother transition between colors and enhancing the control of the diodes [20]. The quantum displays have also proven to be 30% more energy efficient than the current generation of LED displays [21], and Samsung already claim that they use this technology in their displays [20]. Another reason why we are interested in simulating quantum dots is the fact that there exist experiments that can be used as benchmarks. Due to very strong confinement in the z-direction, the experimental dots, made by patterning GaAs/AlGaAs heterostructures, become essentially two-dimensional [22, 23]. For that reason, our main focus in this work is on two-dimensional dots, but also dots of three dimensions will be investigated.

1.4 Computer experiments

The advent of computer technology has offered us a new window of opportunity for studies of quantum (and many other) problems. It serves as a third way of doing science which is based on simulations, in contrast to laboratory experiments and analytical approaches. In a broad sense, by simulations, we mean computational models of reality based on physical laws, such as the fundamental Schrödinger equation. Such models have value when they enable them to make predictions or to provide new information which is otherwise impossible or too costly to obtain. In this respect, QMC methods represent an illustration and an example of what is the potential of such methodologies.

The use and popularity of QMC methods have increased as personal computers and computer clusters have been more powerful. With today's reliable computers, we see those methods as a natural choice when ground state properties of quantum mechanical systems are investigated. Even the most straightforward method, VMC, does typically yield excellent results, and the more complicated diffusion Monte Carlo (DMC) is in principle capable of employing exact results. They both appear to have among the highest performance-to-cost ratios out of all the quantum many-body methods.

Albeit the fact that the QMC methods relatively recently have been applied to large-scale calculations, some of the ideas go back to the time before the invention of the electronic computer. Already in the 1940's, Enrico Fermi revealed the similarities between the imaginary time Schrödinger equation and stochastic processes in statistical mechanics. The first attempt to use this link on actual calculations was performed by a group of scientists at Los Alamos National Laboratory in the early 1950's when they tried to compute the ground state energy of the hydrogen molecule using a simple version of VMC [24]. Around at the same time, Metropolis & Ulam [25] introduced the original Metropolis algorithm, which estimates the integrals by moving particles randomly in an ergodic scheme and rejecting inappropriate moves. This method was further improved in the early 1960s when Kalos [26] laid down the statistical and mathematical framework for the Green's function in QMC methods, and Hastings [27] developed an efficient algorithm based on the theory, where the particles are moved after an educated guess.

The use of the QMC methods on many-fermion systems was first done by Ceperley & Alder [28] in the 1980s and started a new era of stochastic methods applied to electronic structure problems.

1.5 Ethics in science

In science as an entirety, there are some general guidelines that we all should follow in order to maintain ethical behavior. Firstly, one should always have respect for other's work, and the authors should be credited whenever one uses other's work, no matter the scope. With work, we mean illustrations, text, code, methods, algorithms *et cetera*, which are protected by the Copyright Act (in Norway, åndsverkloven). Secondly, all the research that one does should always be detailed in a such way that others can reproduce the experiments and results. This means that all the factors which possibly have a significant impact on the experiments should be described, and computer experiments are no exception. Lastly, there are unfortunately many examples of misuse of knowledge throughout history, which obviously has to be avoided.

In our specific work, most of the ethical aspects related to the use of machine learning, which can cause fatal consequences if it is not used correctly and carefully. The fact that machine learning allows the computers to learn things themselves have made profiled people like Stephen Hawking [29] and Elon Musk [30] warn us that they can be greatly misused if they are set on learning the wrong things. Every person who develops machine learning algorithms should take this warning seriously, remember that in the end, there is one of us who end up creating the multi-headed monster *Hydra*.

1.6 Our goals and contributions

The goals of this thesis are mainly related to the development and investigation of a quantum many-body method that requires less physical intuition. In order to do this, the software implementation is essential, which obviously is a significant part of the work and thereby the goals. This also includes a VMC code for benchmark purposes, in addition to the RBM code. Further, the next goal is naturally to obtain some results, which will mostly come from the quantum dots. In the end, we will provide a thorough comparison of the VMC and RBM results, inclusive wave function studies. We can summarize the goals in a few points:

- Develop a VMC code for the study of large fermionic systems.
- Implement RBMs as flexible trial wave function guesses.
- Study ground state properties of quantum dots and atoms, including energy, variance and electron densities.
- Make a critical evaluation of the RBMs compared with VMC studies.

We believe that machine learning-based methods are the next big thing in many-body quantum mechanics, and our contribution to the field is therefore to investigate one of the many such approaches. There are also some fascinating similarities between the typical sampling process behind Boltzmann machines and the QMC methods, and by revealing the actual mechanisms, we might be able to develop more robust methods.

1.7 Our developed code

There exist plenty of commercial software programs for solving the quantum many-body problem, and they are often efficiency optimized. In general, it is wise to use already existing code

and not try to reinvent the wheel, but in our case, we will investigate a new approach, which forces us to write the code from scratch.

Our VMC solver is written in object-orientated C++, inspired by the example implementation created by Ledum [31], and our single-particle functions are assumed to be given in Cartesian coordinates. The goal is not to compete with the performance of commercial software, but we will still make significant efforts to develop an efficient code. As far as it is possible, all operations are vectorized using the open-source template library Eigen, which again is based on the tremendously fast packages BLAS and LAPACK. We used the profiling tool Valgrind to analyze which functions that require most computer power and thus which parts of the code that should be made more efficient. Additionally, the message passing interface (MPI) lets us parallelize the code and thereby run on computer clusters. For implementation details, see chapter 10 and 11.

The outcome of this work can in its entirety be found on https://github.com/evenmn/, where the source code is found in the directory VMC/src under MIT license [32]. Python scripts used for less costly operations like plotting, linear regression and symbolic integration are found in the directory VMC/scripts, and all the scripts that we refer to throughout the thesis can be found there. The thesis-related files, including TEX scripts and figures, are found in the repository https://github.com/evenmn/Master-thesis/. We also provide the raw data, including electron density source files, energy expectation value files, files containing final parameters and the raw slurm output files from the Abel computer cluster on [33].

1.8 Structure of the thesis

This thesis is divided into six parts, where each part again is divided into respective chapters. Part I announces the quantum theory, included an introduction (chapter 2), many-body quantum mechanics (chapter 3) and systems (chapter 4). Thereafter, the machine learning theory is addressed in Part II, where the supervised learning (chapter 5) is used as a motivation for the Boltzmann machines (chapter 6). The method is described in Part III, and the actual implementation, with focus on efficiency, flexibility and readability, is detailed in Part IV. Part V aims the presentation and discussion of the results, and lastly, we summarize the work and observations in Part VI.

Items that are relevant, but do not belong to the main text are moved to the appendix. This includes a brief description of Dirac formalism (appendix A), derivation of natural units (appendix B), derivation of a general restricted Boltzmann machine with Gaussian-binary units (appendix C) and an extended collection of results (appendix D).

Part I

Quantum Theory

CHAPTER 2

Preliminary Quantum Mechanics

I do not like it, and I am sorry I ever had anything to do with it.

Erwin Schrödinger, [34]

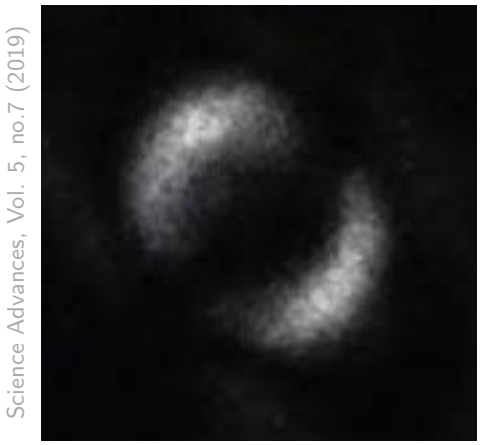


Figure 2.1: The first ever image of quantum entanglement, where two groups of particles are sharing the physical state $\Psi(\boldsymbol{x})$, was published by Moreau *et al.* [35] under the title *Imaging Bell-type nonlocal behavior* in July 2019.

The quantum theory is a fundamental theory that describes nature at the smallest scales, typically used to explain the behavior of atoms and subatomic particles. Although the theory in principle holds for systems consisting of a large number of particles as well, the calculations become both comprehensive and expensive as the system size increases, and exact computations are therefore reserved small systems. For larger systems, we in practice need to rely on approximative estimates of the observable, which also fast becomes infeasible when the system size increase.

In the most general form, the theory is based on the time-dependent Schrödinger equation,

$$i\hbar\frac{\partial}{\partial t}\left|\Psi(\boldsymbol{x},t)\right\rangle = \hat{\mathcal{H}}\left|\Psi(\boldsymbol{x},t)\right\rangle \tag{2.1}$$

which describes the state, or the wave function, $\Psi(\boldsymbol{x},t)$, of a system with Hamilton operator $\hat{\mathcal{H}}$, known as the Hamiltonian. $\boldsymbol{x} = (\boldsymbol{r},\sigma)$ specifies the coordinates and the spin of a particle, $\hbar$ is

the reduced Planck's constant and i is the solution of the equation $x^2 = -1$. As we later will see, the quantum mechanical system is completely specified by the wave function, so by solving the equation we will in principle know anything about the system.

Schrödinger [36] introduced the now-called Schrödinger equation in a 1926 paper as one of many contributions to the quantum mechanics at that time. Other contributors include Born [37] who suggested now-standard interpretation of the probability density function as $|\Psi(\boldsymbol{x}, t)|^2$ in 1926 and his companion Heisenberg [38], who formulated the matrix mechanics representation in 1925. However, many physicists were skeptical about the new theory, including Albert Einstein and Schrödinger himself. The theory differed from the classical mechanics in the sense that it was based on a statistical interpretation, with some strange consequences. For instance, the theory allowed negative kinetic energies to occur with the consequence that a particle could *go through* a potential wall, known as quantum tunneling. Later, Einstein [39] also pointed out that when pairs or groups of particles are generated in ways such that the wave function of each particle cannot be described independently of the other state, they will be affected by each other even at large distances. He called this *"spooky action at a distance"*, and used the observation to argument that the quantum theory had to be incorrect. However, the observation was later proven to be correct and as late as in July 2019 the first image of quantum entanglement, as the phenomenon is called, was captured by Moreau *et al.* [35]. See figure (2.1).

Today, there is wide agreement in the physics society on the quantum theory. Actually, the theory is the most precisely tested in the history of science, where computed observable of atoms agree perfectly with experiments. Most notably, quantum electrodynamical calculations of the fine structure constant α are, by Odom *et al.* [6], found to agree with experiments within ten parts in a billion, 10^{-8}.

In this chapter, we will first state the postulates of quantum mechanics, and thereafter we discuss the time-independent Schrödinger equation and challenges related to solving it. As we will see, every observable in quantum mechanics is associated with an operator. The consequence is that all obtained values are identified with standard errors which are also important to specify in order to present the accuracy of the value. In that manner, quantum mechanics is based on statistics, and in section 2.3 we discuss the statistical interpretation of the theory. The variational principle states that the ground state energy is the lowest possible energy calculated, so by minimizing the energy in a variational scheme, a ground state energy estimation can be obtained. Other things discussed are the quantum numbers, which are the values that give acceptable solutions to the Schrödinger equation, and the virial theorem, which relates kinetic and potential energy.

Albeit the quantum mechanics will make up the framework for this work, we will in this chapter only discuss the theory needed for the work, and this is therefore not meant as an encyclopedia to quantum mechanics. For a complete introduction to the topic, *Introduction to Quantum Mechanics* written by Griffiths [40] serves as an excellent read. Before we get started, we make a few assumptions in order to simplify our problem. The most important ones are specified below with an explanation of why they are valid.

- **Point-like particles:** First, all particles involved will be assumed to be point-like, i.e., they lack spatial extension. For electrons, this makes sense since they, as far as we know, do not extend. The assumption is also applied to the nucleus in atomic systems, but it still makes sense since the distance from the nucleus to the electrons is known to be much larger than the nucleus extension.

- **Non-relativistic spacetime:** Second, we operate in the non-relativistic spacetime, which is an excellent approximation as long as we do not approach the speed of light and we do

not involve strong forces. Applying classical physics, we can find that the speed of the electron in a hydrogen atom is about 1% of the speed of light. Even though the electrons achieve higher velocities in heavy atoms, we do not need to worry about it as we will stick to the lighter atoms. In the quantum dots, this assumption holds even for large systems, as the velocity and density will still be sufficiently low.

- For specific systems, we might make new assumptions and approximations. For instance, for atomic systems, we will assume that the nucleus is at rest. Those approximations will be discussed consecutively.

2.1 The postulates of quantum mechanics

Quantum mechanics is characterized by a set of fundamental axioms that make up the base of the theory. As we assume that they always hold, every statement in quantum mechanics is based on them, and it is, therefore, natural to list them now in the beginning. They can be formulated by six postulates, in the following way:

1. *"The state of a quantum mechanical system is completely specified by the wave function $\Psi(\boldsymbol{x}, t)$."*
2. *"To every observable in classical mechanics, there corresponds a linear, Hermitian operator in quantum mechanics."*
3. *"In any measurement of the observable associated with an operator $\hat{O}$, the only values that will ever be observed are the eigenvalues o which satisfy $\hat{O}\Psi(\boldsymbol{x}, t) = o\Psi(\boldsymbol{x}, t)$."*
4. *"The expectation value of the observable corresponding to operator $\hat{O}$ is given by*

$$\langle \hat{O} \rangle = \frac{\int d\tau \Psi^*(\boldsymbol{x}, t)\hat{O}\Psi(\boldsymbol{x}, t)}{\int d\tau \Psi^*(\boldsymbol{x}, t)\Psi(\boldsymbol{x}, t)}."$$

5. *"The wave function evolves in time according to the time-dependent Schrödinger equation,*

$$\hat{\mathcal{H}}\Psi(\boldsymbol{x}, t) = i\hbar\frac{\partial\Psi(\boldsymbol{x}, t)}{\partial t}."$$

6. *"The total wave function must be anti-symmetric concerning the interchange of all coordinates of one fermion with those of another. Electronic spin must be included in this set of coordinates."*

All the postulates will be used in this work, and they will be described in detail when we need them. Albeit we will look at stationary states only, even the time-dependent Schrödinger from the fifth postulate will be discussed due to the description of the diffusion Monte Carlo method in section 7.2. The other will be covered in the current and the next chapter. The formulation of the presented postulates are taken from [41].

2.2 The Schrödinger equation

We have already presented the time-dependent Schrödinger equation on several occasions, but as mentioned above, we will in this work study stationary systems only. If we also recall that the particles are assumed to be non-relativistic, the focus will be on solving the time-independent

non-relativistic Schrödinger equation. By defining $\Psi_n(\boldsymbol{x})$ as the wave function of a state n with energy ε_n, the equation can be expressed as

$$\hat{\mathcal{H}}\Psi_n(\boldsymbol{x}) = E_n\Psi_n(\boldsymbol{x}) \tag{2.2}$$

where $\hat{\mathcal{H}}$ is the aforementioned Hamiltonian, the total energy operator. By analogy with the classical mechanics, the total mechanical energy is the kinetic and potential energy summarized,

$$\hat{H} = \hat{T} + \hat{V} \tag{2.3}$$

with $\hat{T}$ and $\hat{V}$ as the kinetic and potential energy operators respectively.

Again from classical mechanics, the kinetic energy of a moving particle of mass m yields $T = p^2/2m$ where p is the (linear) momentum, such that the kinetic energy operator can be represented as

$$\hat{T} = \frac{\hat{p}^2}{2m}, \tag{2.4}$$

according to Ehrenfest's theorem. Further, the momentum operator is $\hat{p} = -i\hbar\hat{\nabla}$ with $\hat{\nabla}$ as the differential operator and the factor $i\hbar$ arising from the canonical commutator relation between the position operator and the momentum operator,

$$[\hat{x}, \hat{p}] = \hat{x}\hat{p} - \hat{p}\hat{x} = i\hbar, \tag{2.5}$$

which indicates that the momentum and the position do not *commute*. In other words, the order of the operators in an equation is not arbitrary. The potential, on the other hand, is obviously dependent on the system we want to study. For atomic systems, the potential as a function of the distance from the nucleus can be found from Coulomb's law and reads

$$V(r) = \frac{1}{4\pi\epsilon_0}\frac{Ze^2}{r} \tag{2.6}$$

where the nucleus is assumed to be at rest at the origin, $k_e = 1/4\pi\epsilon_0$ is the Coulomb constant, Z is the atomic number of the nucleus, and e is the elementary charge. For a general potential, the Hamiltonian can be expressed as

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}\nabla^2 + V(r) \tag{2.7}$$

which is the farthest we can go without specifying the external potential $V(r)$.

By setting up equation (2.2) with respect to the energies, we obtain an integral,

$$E_n = \frac{\int d\boldsymbol{r}\Psi_n^*(\boldsymbol{x})\hat{\mathcal{H}}\Psi_n(\boldsymbol{x})}{\int d\boldsymbol{r}\Psi_n^*(\boldsymbol{x})\Psi_n(\boldsymbol{x})}, \tag{2.8}$$

which not necessarily is trivial to solve. For almost[1] all many-electron systems, this becomes analytically infeasible due to a two-body interaction term. This will be covered in chapter 3.

As suggested by Max Born, we get the probability density function if we take the dot product between the complex conjugate wave function and the wave function itself,

$$P(\boldsymbol{r}) = \Psi_n^*(\boldsymbol{x})\Psi_n(\boldsymbol{x}) = |\Psi_n(\boldsymbol{x})|^2, \tag{2.9}$$

so the denominator in equation (2.8) is basically the integral over all the probabilities. If the wave function is normalized correctly, this should always give 1.

As a very brief example, we will below calculate the bounding energy of the Hydrogen atom.

[1] Exceptions include quantum dots of two electrons in two and three dimensions, where Taut has presented semi-analytical energies for a few oscillator frequencies [42, 43].

2.2.1 The Hydrogen atom

We presented the one-particle Hamiltonian in equation (2.7), and the potential from the nucleus in equation (2.6). By introducing the Bohr radius

$$a_0 = \frac{4\pi\epsilon_0\hbar^2}{m_e Z e^2} \tag{2.10}$$

we can set up the total Hamiltonian as

$$\hat{\mathcal{H}} \cdot \frac{(4\pi\epsilon_0)^2\hbar^2}{m_e Z^2 e^4} = -\frac{1}{2}a_0^2\nabla^2 - \frac{a_0}{r} \tag{2.11}$$

where we have multiplied all the terms by the factor $(4\pi\epsilon_0)^2\hbar^2/m_e Z^2 e^4$ in order to make the equation dimensionless. We can further scale the energy as $E \leftarrow E \cdot (4\pi\epsilon_0)^2\hbar^2/m_e Z^2 e^4$ and the distance as $r \leftarrow r/a_0$, which both are dimensionless, and obtain the Hamiltonian

$$\hat{\mathcal{H}} = -\frac{1}{2}\nabla^2 - \frac{1}{r} \tag{2.12}$$

where all quantities are in atomic units.

The Hydrogen ground state wave function is well-known and reads

$$\Psi(r) \propto e^{-r} \tag{2.13}$$

in atomic units. The binding energy in Hydrogen is found from the Schrödinger equation in equation (2.2), and gives

$$\hat{\mathcal{H}}\Psi(r) = \left(-\frac{1}{2}\nabla^2 - \frac{1}{r}\right)\Psi(r) = -\frac{1}{2}\Psi(r) \tag{2.14}$$

which indicates that $E_0 = -0.5$.

2.3 Statistical interpretation

In equation (2.8), we found the expectation value of the energy using the Hamiltonian, which is the energy operator. By the second postulate of quantum mechanics, every observable in the classical mechanics is associated with such a Hermitian operator $\hat{O}$ related to an expectation value $\langle\hat{O}\rangle$ is the same way,

$$\langle\hat{O}\rangle = \frac{\int d\boldsymbol{r}\Psi_n^*(\boldsymbol{x})\hat{O}\Psi_n(\boldsymbol{x})}{\int d\boldsymbol{r}\Psi_n^*(\boldsymbol{x})\Psi_n(\boldsymbol{x})}. \tag{2.15}$$

As a consequence, there is always an uncertainty associated with a quantum mechanical calculation, such that we can only tell the probability of measuring something. The uncertainty is usually described by the variance, which for a set of independent measurements is given by

$$\sigma^2 = \langle\hat{O}^2\rangle - \langle\hat{O}\rangle^2. \tag{2.16}$$

If there are correlations between the measurements, this expression underestimates the actual sampling variance as the *covariance* is not taken into account, which is detailed in section 7.4. If we take the square root of the variance, we obtain the standard deviation, which is the quantity given as the standard error in the results. From the standard deviation, a variety of mathematical inequalities follows, where Heisenberg's uncertainty principle is the most famous. It states that

the more precisely the position of some particle is determined, the less precisely its momentum can be known, and is mathematically presented as

$$\sigma_x \sigma_p \geq \frac{\hbar}{2} \tag{2.17}$$

where σ_x is the standard deviation of the position and σ_p is the standard deviation of the momentum.

The general expected value in equation (2.15) is expressed using the Schrödinger wave mechanics picture, but it can also be written in terms of, for instance, the Heisenberg picture applying matrices. However, the standard formalism today in most branches of the quantum mechanics is the Dirac notation, which relates the Schrödinger picture and the Heisenberg picture elegantly. Using this, the expectation value from equation (2.15) reads

$$\langle \hat{O} \rangle = \frac{\langle \Psi | \hat{O} | \Psi \rangle}{\langle \Psi | \Psi \rangle} \tag{2.18}$$

where $\langle \Psi |$ is called the *bra* and $| \Psi \rangle$ is called the *ket*. For that reason, the formalism is also called bracket formalism. Usually, the wave function is assumed to be normalized, which further simplifies the expected value to

$$\langle \hat{O} \rangle = \langle \Psi | \hat{O} | \Psi \rangle \,, \tag{2.19}$$

and will henceforth be assumed. More information about the Dirac formalism can be found in Appendix A.

2.4 The variational principle

In the equations above, the presented wave functions are assumed to be the exact eigenfunctions of the Hamiltonian. However, often we do not know the exact wave functions, and we need to guess what the wave functions might be. In those cases, we apply the variational principle, which states that only the exact ground-state wave function can give the ground state energy. All other wave functions that fulfill the required properties (see section 3.2) give higher energies, and mathematically we can express the statement as

$$E_0 \leq \langle \Psi | \hat{\mathcal{H}} | \Psi \rangle \tag{2.20}$$

where Ψ is an arbitrary wave function. The variational method is a way of finding approximations to the energy ground state, which is based on the variational principle. Most notably, the variational Monte Carlo method is named after the principle and attempts to solve the integrals directly by varying a trial wave function. The variational principle ensures that the obtained energy never goes below the ground state energy, as further described in chapter 7. Other methods that apply the variational method are the famous Hartree-Fock method and the infamous coupled cluster method.

2.5 Quantum numbers

Unlike in classical mechanics, all the observable in quantum mechanics are discrete or *quantized*, which means that the n associated with E_n above cannot take any number. n can only take positive integers and is named the principal quantum number. We also have other quantum numbers identified with the angular momentum and spin as described below.

Principal

The **principal** quantum number describes the electron shell, and can take the numbers $n \in [1, 2, 3, \ldots)$. As n increases, the electron excites to a higher shell such that also the energy increases. In general, $E_1 < E_2 < E_3 \ldots$ as long as all other quantum numbers are fixed. The electron shells can again be split up in subshells, requiring more quantum numbers.

Angular

An electron shell can possibly have more than one subshell, described by the **angular** quantum number l. l can take the values $0, 1, \ldots n-1$, such that the degeneracy of subshells in a shell is simply n. In atoms, the angular quantum number describes the shape of the shell, where $l = 0$ gives a spherical shape, $l = 1$ gives a polar shape while $l = 2$ gives a cloverleaf shape.

Magnetic

We also have a **magnetic** quantum number m_l, which has the range $-l, -l+1, \ldots, l-1, l$. If l describes the shape of a shell, m_l specifies its orientation in space. This quantum number was first observed under the presence of a magnetic field, hence the name.

Spin

The **spin** quantum number s gives the spin of a particle, which can just be seen as a particle's property. Particles are often divided into two groups depending on the spin because of their different behavior: **bosons** have integer spin, while **fermions** have half-integer spin. Electrons and protons have spin $s = 1/2$, which makes them fermions.

Spin projection

The last number we will discuss is the **spin projection** quantum number m_s. It has the range $-s, -s+1, \ldots, s-1, s$, and is therefore related to the spin quantum number in the same way as the magnetic number m_l is related to the angular number l. Electrons can for that reason take the values $m_s = +1/2$ or $m_s = -1/2$, such that there are two groups of electrons. The consequences will be discussed in the section 3.2.

2.6 The virial theorem

The virial theorem relates the kinetic energy to the potential energy, and makes it possible to find the (time) average of the kinetic energy even for complex systems. The classical statement of the theorem was formulated during the 19th century and named by Clausius [44] in 1870. It is in the most general form given by

$$\langle \hat{T} \rangle = -\frac{1}{2} \sum_{i=1}^{N} \langle \boldsymbol{F}_i \cdot \boldsymbol{r}_i \rangle_t, \tag{2.21}$$

where $\boldsymbol{F}_i$ represents the force on particle i at position $\boldsymbol{r}_i$. The quantum mechanical version was proven by Fock [45] in a 1930 paper, where the expectation value of the force is represented with $d\langle p \rangle / dt$. For potential sources in the form of $V_i = a r^{n_i}$, we can use Ehrenfest's theorem to express the virial theorem in a simpler fashion,

$$2\langle \hat{T} \rangle = \sum_i n_i \langle \hat{V}_i \rangle. \tag{2.22}$$

The same expression can be found in classical physics as well, using the relation $\boldsymbol{F} = -\nabla V$. The virial theorem requires that the system is in a bound state. An example on a system that does not fulfill the requirements, is a comet that has enough kinetic energy to escape a planet's gravitational field.

On the other hand, the Hydrogen atom, discussed in section 2.2.1, satisfies the virial theorem as the potential is in the form of $V(r) \propto r^{-1}$ and the electron is bounded. For that particular case, the virial theorem reads $2\langle\hat{T}\rangle = -\langle\hat{V}\rangle$, which means that $\langle\hat{H}\rangle = \langle\hat{V}\rangle/2$. However, if we add another electron and get a Hydrogen anion, the simplified virial theorem in equation (2.22) breaks down because of the interaction.

It is also important to emphasize that the expectation value discussed above is, in principle, the time average of the operator. However, if the ergodic hypothesis holds for the system, i.e., the ensemble average is equal to the time average, an ensemble average can also be taken [46].

CHAPTER 3

Many-body Quantum Mechanics

We have to remember that what we observe is not nature in itself but nature exposed to our method of questioning.

Werner Heisenberg, [47]

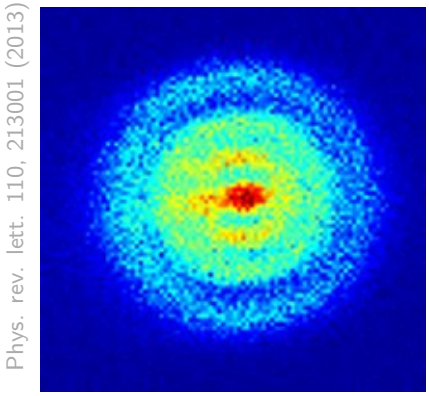


Figure 3.1: The first photograph of a Hydrogen atom was captured by an ultra sensitive camera in 2013. One can actually see the probability distribution $|\Psi(\boldsymbol{R})|^2$ with the naked eye. Published by Stodolna *et al.* [48] under the title *Hydrogen atoms under magnification*.

In the previous chapter, the quantum mechanics of single particles were discussed. We presented the time-independent Schrödinger equation, and from that, we obtained a general expression of the energy of a stationary particle. The energy expression of a stationary *system* is almost identical,

$$E_n = \langle \Psi_n(\boldsymbol{R}) | \hat{\mathcal{H}} | \Psi_n(\boldsymbol{R}) \rangle \,, \tag{3.1}$$

but we here use the many-body wave function $\Psi_n(\boldsymbol{R})$ of state n with $\boldsymbol{R} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} = \{(\boldsymbol{r}_1, \sigma_1), (\boldsymbol{r}_2, \sigma_2), \ldots, (\boldsymbol{r}_N, \sigma_N)\}$ denoting the collective coordinates and the spins of all the N particles in the system. The Hamiltonian, $\hat{\mathcal{H}}$, defines the system and is given explicitly in the next section. As noted before, the wave function for large systems needs to store an immense amount of information and is therefore expensive or even impossible to deal with. In this section, we will first look at how the many-body Hamiltonian is set up before we have an extensive discussion of how the many-body wave function is constructed.

3.1 The electronic Hamiltonian

We have already seen what the one-body Hamiltonian looks like, and the many-body Hamiltonian is not very different. We recall that it can be split in a kinetic and a potential term,

$$\hat{H} = \hat{T} + \hat{V} \tag{3.2}$$

where $\hat{T}$ is the kinetic energy and $\hat{V}$ is the potential energy. Nevertheless, as we study electrons, they are charged and will therefore interact with each other. For that reason, we need to add an interaction term to the Hamiltonian, which in general is included in the potential term $\hat{V} = V_{\text{ext}} + V_{\text{int}}$ with V_{ext} as the external potential and V_{int} as the interaction potential. In the same way as the nucleus potential, the interaction potential is given by Coulomb's law, for two electrons given by

$$V_{\text{int}} = k_e \frac{e^2}{r_{12}} \tag{3.3}$$

where r_{12} is the distance between the electrons. For a general system containing N electrons, the total Hamiltonian can therefore be expressed as

$$\hat{\mathcal{H}} = -\sum_{i=1}^{N} \frac{\hbar^2}{2m_e} \nabla_i^2 + \sum_{i=1}^{N} V_i + \sum_{i=1}^{N} \sum_{j>i}^{N} k_e \frac{e^2}{r_{ij}} \tag{3.4}$$

which is the farthest we can go without specifying the external potential V_i. r_{ij} is the relative distance between particle i and j, defined by $r_{ij} \equiv |\boldsymbol{r}_i - \boldsymbol{r}_j|$. From now on, we will use atomic units setting $\hbar = m_e = k_e = e = 1$, see appendix B for details.

By putting this Hamiltonian into equation (3.1), the integral can be split in three parts,

$$E_n = \sum_{i=1}^{N} \left[-\frac{1}{2} \langle \Psi_n(\boldsymbol{R}) | \nabla_i^2 | \Psi_n(\boldsymbol{R}) \rangle + \langle \Psi_n(\boldsymbol{R}) | V_i | \Psi_n(\boldsymbol{R}) \rangle + \sum_{j>i}^{N} \langle \Psi_n(\boldsymbol{R}) | \frac{1}{r_{ij}} | \Psi_n(\boldsymbol{R}) \rangle \right] \tag{3.5}$$

where the two former ones are the one-body integrals, or *matrix elements*, which we in many cases easily can solve. However, the last term is very tricky to solve, and in fact there are analytical solutions available for the two-particle case only. With other words, a precise evaluation of this integral can usually only be found using numerical methods, which we will have a closer look at in chapter 7 in conjunction with quantum Monte Carlo methods.

3.2 The many-body wave function

By the first postulate of quantum mechanics presented in section 2.1, the wave function contains all the information specifying the state of the system. This means that all observable in classical mechanics can in principle also be estimated from the wave function, which makes finding the wave function our aim. As we have seen, we can define the wave function for a single particle, known as the *single-particle function* (SPF), $\psi(\boldsymbol{r}, \sigma)$. Can we combine the SPFs of the electrons in a system and obtain the many-body wave function? Possibly, the most straight-forward way of doing this is to simply multiply all the SPFs,

$$\Psi(\boldsymbol{R}) = \psi(\boldsymbol{r}_1, \sigma_1)\psi(\boldsymbol{r}_2, \sigma_2) \dots \psi(\boldsymbol{r}_N, \sigma_N), \tag{3.6}$$

known as the *Hartree product*. However, this product is generally not correct, as it does not include the required symmetry properties of the many-body wave function. Instead, we can

take the symmetry into account by expressing the many-body wave function as a determinant or a permanent, as we will see in section 3.2.2. In addition to the symmetry properties, there is an array of requirements the wave function needs to meet in order to be physically correct. Some of them are:

1. **Normalizability:** The wave function needs to be normalizable in order to make physical sense. The total probability should always be 1, and a wave function that cannot be normalized will not have a finite total probability. The consequence is that the wave function goes to zero when the positions get large, $\Psi(x \to \pm\infty) \to 0$.

2. **Cusp condition:** The cusp condition (also called the Kato theorem) states that the wave function should have a cusp where the potential explodes. An example on this is when electrons come close to each other, i.e., the electron-electron cusp.

3. **Symmetry and anti-symmetry:** The wave function needs to be either symmetric or anti-symmetric under the exchange of two coordinates, dependent on whether the electrons are fermions or bosons. This is the statement of the sixth postulate, which will be further explained in the next section.

3.2.1 Anti-symmetry and the Pauli principle

Symmetry and anti-symmetry are central concepts in quantum mechanics, and often one can use symmetry arguments to simplify expressions and calculations. In this section, we will look at the symmetry and anti-symmetry properties of the wave function and discuss how it can be used to define a wave function in the next section.

Assume that we have a permutation operator, $\hat{P}(i \to j)$, which exchanges the coordinates of the particles i and j,

$$\hat{P}(i \to j)\Psi_n(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_j, \ldots, \boldsymbol{x}_M) = p\Psi_n(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_M), \tag{3.7}$$

where p is just a factor which comes from the transformation. If we again apply the $\hat{P}$ operator, we should switch the same coordinates back, and we expect to end up with the initial wave function. For that reason, p must be either +1 or -1. [1] The particles that have an anti-symmetric wave function under the exchange of two coordinates are called fermions, named after Enrico Fermi, and as discussed before they have half-integer spin. On the other hand, the particles that have a symmetric wave function under the exchange of two coordinates are called bosons, named after Satyendra Nath Bose, and have integer spin. A consequence of the anti-symmetric wave function is that two identical fermions cannot occupy the same state at the same time, known as the Pauli principle. This means that identical fermions even in the ground state (at zero temperature) spread over multiple states, and in the next section, we will see how this principle is baked into the wave function through a Slater determinant.

3.2.2 The Slater determinant

For a system of many particles, we can define a total wave function, which is a composition of all the single-particle wave functions (SPF) and contains all the information about the system as the first postulate requires. For fermions, we need to combine the SPFs such that the Pauli principle is fulfilled at all times, which can be accomplished by a determinant.

[1] Actually, in two-dimensional systems a third possibility is allowed which gives an *anyon*. The theory on this was developed by Leinaas & Myrheim [49] during the 1970s.

Consider a system of two identical fermions with SPFs $\psi_1(\boldsymbol{r},\sigma)$ and $\psi_2(\boldsymbol{r},\sigma)$ with coordinates and spin $\boldsymbol{r}_1,\sigma_1$ and $\boldsymbol{r}_2,\sigma_2$ respectively. The way we define the wavefunction of the system is then

$$\begin{aligned}\Psi(\boldsymbol{R}) &= \frac{1}{\sqrt{2}}\begin{vmatrix}\psi_1(\boldsymbol{r}_1,\sigma_1) & \psi_2(\boldsymbol{r}_1,\sigma_1)\\ \psi_1(\boldsymbol{r}_2,\sigma_2) & \psi_2(\boldsymbol{r}_2,\sigma_2)\end{vmatrix}\\ &= \frac{1}{\sqrt{2}}\Big[\psi_1(\boldsymbol{r}_1,\sigma_1)\psi_2(\boldsymbol{r}_2,\sigma_2)-\psi_2(\boldsymbol{r}_1,\sigma_1)\psi_1(\boldsymbol{r}_2,\sigma_2)\Big],\end{aligned} \tag{3.8}$$

which is automatically set to zero if the particles happen to be at the same position and have the same spin at the same time. If the particles, on the other hand, have different spins, they are allowed to be at the same position at the same time and the determinant will not cancel. For larger systems, the Slater determinant is constructed in the same way as above, and any pair of identical particles located in the same state will make the determinant collapse. A Slater determinant containing N electrons reads

$$\Psi(\boldsymbol{R}) = \frac{1}{\sqrt{N!}}\begin{vmatrix}\psi_1(\boldsymbol{r}_1,\sigma_1) & \psi_2(\boldsymbol{r}_1,\sigma_1) & \dots & \psi_N(\boldsymbol{r}_1,\sigma_1)\\ \psi_1(\boldsymbol{r}_2,\sigma_2) & \psi_2(\boldsymbol{r}_2,\sigma_2) & \dots & \psi_N(\boldsymbol{r}_2,\sigma_2)\\ \vdots & \vdots & \ddots & \vdots\\ \psi_1(\boldsymbol{r}_N,\sigma_N) & \psi_2(\boldsymbol{r}_N,\sigma_N) & \dots & \psi_N(\boldsymbol{r}_N,\sigma_N)\end{vmatrix} \tag{3.9}$$

where the $\psi(\boldsymbol{r},\sigma)$ is the tensor product between the radial part $\phi(\boldsymbol{r})$ and the spin part $\xi(\sigma)$,

$$\psi(\boldsymbol{r},\sigma) = \phi(\boldsymbol{r})\otimes\xi(\sigma). \tag{3.10}$$

In section 10.3.3, it is shown that the Slater determinant can be split in a spin-up part and a spin-down part such that the spin-dependency $\xi(\sigma)$ can be omitted. For that reason, we can define a basis set consisting of the spatial parts only, discussed in the next section.

As a note, we will in the rest of this thesis use Ψ as the many-particle wave function, ψ are the single-particle functions, and ϕ is the spatial part of the single-particle function. ξ will be reserved the spin part of the single-particle functions, but will often be omitted as the spin-part can be factorized out. Sometimes it is appropriate to split up the many-particle wave function, and we will, in that case, denote each part by Ψ_i where i is an index associated with the particular element. Lastly, the basis functions will be denoted by φ, which we will discuss in the next section.

3.2.3 Basis set

In quantum chemistry, a basis set is a set of basis functions, which often are called the atomic orbitals. Generally, a basis function should not be confused with a single-particle function, as a single-particle function is a linear combination of basis functions, and are thus often called the molecular orbitals. Commonly used atomic orbitals are Pople basis sets (in the form of x-yz G), correlation-consistent basis sets (in the form of cc-pVNZ) and Slater-type orbitals (in the form of STO-nG), where all are built on Gaussian functions. Gaussian functions are preferred as they allow efficient implementations of post Hartree-Fock methods.

In our work, however, the single-particle functions are often the basis functions, as we in most of the cases do not expand our single-particle functions $\psi(\boldsymbol{r},\sigma)$ in a basis. Our single-particle functions are typically the solution of the non-interacting system, and as the Jastrow factor is supposed to deal with the interaction, we use the same functions for the interacting case as well.

The standard notation is to use the Greek letter φ for the atomic orbitals and ψ for the molecular orbitals, such that single-particle functions can be obtained from an expansion of the N

basis functions $\{\varphi_1(\boldsymbol{r}), \varphi_2(\boldsymbol{r}), \ldots \varphi_N(\boldsymbol{r})\}$ in the manner of

$$\phi_i(\boldsymbol{r}) = \sum_{j=1}^{N} c_{ji}\varphi_j(\boldsymbol{r}). \tag{3.11}$$

where c_{ij} are the coefficients to be found. There are different approaches to obtain these coefficients, where the popular Hartree-Fock algorithm generates c_{ij}'s in order to find the optimal single Slater determinant. The larger basis, the more accurate results we will get. The actual functions used in this work are presented in the next chapter, 4, and linked to their respective systems.

3.2.4 Modeling the cusp

From electrostatics, we know that identical, charged particles will repulse each other. This means that the probability of finding two particles close to each other should be low, which needs to be baked into the wave function. To estimate the observable accurately, it is crucial to model the electron-electron cusp correctly.

Different methods attack this challenge in different ways. The Hartree-Fock method attempts to construct an optimal single Slater determinant by expanding the molecular orbitals in atomic orbitals, like shown in equation (3.11). The coefficients are determined such that the energy is minimized, which is performed by the Hartree-Fock algorithm. Then, we only need to deal with a Slater determinant, and as the correlations are not given explicitly, the Hartree-Fock theory is often called a mean-field theory. Further, we have a bunch of post Hartree-Fock methods, like configuration interaction and the coupled cluster method, which utilize the Hartree-Fock basis, but express the wave function as a linear combination of Slater determinants, where the correlations are determined by the coefficients. If a sufficient number of Slater determinants are included in the linear combination, both the methods are capable of providing exact results. However, since the scaling goes as $N!$ and N^6 respectively, this is possible only for very small systems.

The variational Monte Carlo (VMC) method, which we have implemented in this work, models the electron-electron cusp in a totally different way. We there define a *trial wave function*, which consists of one or more Slater determinants and a Jastrow factor, where the latter is assumed to account for the correlations. In that way, the Slater determinants are used only to account for the Pauli principle only. The Jastrow factor will be discuss further in section 7.1.2.

However, our primary focus in this work is to see if we can use machine learning to reduce the need of physical intuition. Our hope is that the method will be able to model the cusp correctly without prior knowledge about the correlations. In the first place, we will try to construct a flexible Slater determinant based on neural network, which also models the correlations. This is discussed in chapter **??**.

3.3 Electron density

In quantum many-body computations, the electron density is frequently calculated, and there are several reasons for that. Firstly, the electron density can be found experimentally, such that the calculations can be benchmarked. Secondly, the electron density is very informative, since information about all particles can be gathered in one plot. The P-body electron density is defined by the multi-dimensional integral over the probability density function of all the particles

but P, for a normalized wave function represented by

$$\rho_P(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_P) = N \int_{-\infty}^{\infty} d\boldsymbol{r}_{P+1} \ldots d\boldsymbol{r}_N |\Psi(\boldsymbol{r}_1, \ldots \boldsymbol{r}_N)|^2 \tag{3.12}$$

where $P \leq N$. This integral is in general really tricky so solve, and it can be found analytically for just a few realistic systems. For other systems, we need to solve the integral numerically, and we will in chapter 7 describe how this can be done using Monte Carlo integration. The density should not be normalized to unit both when it is calculated analytically and numerically, but to the number of particles, i.e,

$$\int_{-\infty}^{\infty} d\boldsymbol{r}_1 \ldots d\boldsymbol{r}_P \Big[\rho_P(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_P)\Big] = N. \tag{3.13}$$

As we cannot pinpoint a particle in quantum mechanics, i.e., it is impossible to distinguish two identical particles in any way, the electron density is the same no matter which particles we decide to leave out. In equations, we always leave out the first P particles, but we could also, for instance, have left out the P last particles and so forth.

So, what does the electron density tell us? The two most widely used electron densities are the one-body density, $\rho_1(\boldsymbol{r}_1)$ and the two-body density, $\rho_2(\boldsymbol{r}_1, \boldsymbol{r}_2)$. The former is the most applied electron density and is sometimes simply referred to as the electron density. It gives the probability density of finding an electron throughout the space, and give insight about how the particles are distributed in the system. The two-body density becomes a two-dimensional function and is therefore often expressed as a matrix. It gives the probability density of finding an electron throughout the space, given the position of another particle. It, therefore, contains information about how the particles distribute relative to each other and is essential when we want to study the pairwise interaction. In systems with strong forces, such as nuclear systems, the three-body interactions become important and then it also makes sense to look at the three-body density $\rho_3(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3)$. For closed-shell circular quantum dots the radial electron density profile is often preferred as the density then also is circular and independent of the angle.

For non-interacting systems, the wave function is trivially separable with respect to the different particles. In those cases one can easily find the electron density analytically, since the functions of coordinates which we do not integrate over can be factorized out of the integral,

$$\begin{aligned}
\rho_P(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_P) &= \int_{-\infty}^{\infty} d\boldsymbol{r}_{P+1} \ldots d\boldsymbol{r}_N |\Psi(\boldsymbol{r}_1), \ldots \Psi(\boldsymbol{r}_N)|^2 \\
&= \int_{-\infty}^{\infty} d\boldsymbol{r}_{P+1} \ldots d\boldsymbol{r}_N |\Psi(\boldsymbol{r}_1)|^2 \ldots |\Psi(\boldsymbol{r}_N)|^2 \\
&= |\Psi(\boldsymbol{r}_1)|^2 \ldots |\Psi(\boldsymbol{r}_P)|^2 \int_{-\infty}^{\infty} d\boldsymbol{r}_{P+1} |\Psi(\boldsymbol{r}_{P+1})|^2 \ldots \int_{-\infty}^{\infty} d\boldsymbol{r}_N |\Psi(\boldsymbol{r}_N)|^2 \\
&= |\Psi(\boldsymbol{r}_1)|^2 \ldots |\Psi(\boldsymbol{r}_P)|^2
\end{aligned} \tag{3.14}$$

where we have assumed that the wave functions are normalized. This result has no scientific importance, but will be used to validate the implementation of the electron density in the code, see section 12.3.

3.3.1 Wigner crystals

A Wigner crystal is a solid phase where electrons maximize the distance to each other in order to minimize the potential energy. As Coulomb's law gives the interaction potential, the potential

is minimized when the distances between the electrons are maximized. In one-dimensional systems, the electrons are thus found at discrete locations, such that they form an evenly spaced lattice. In two-dimensional systems, the Wigner crystals form triangular lattices which are known to be the configuration that maximizes the distance in a limited space, and in three-dimensional systems, the electrons form so-called body-centered cubics. The phenomenon occurs only when the potential energy dominates the kinetic energy since the electrons then are almost ”at rest” and external forces are not strong enough to push the electrons close to each other.

If the concept still is unclear, imagine Norwegians waiting for the metro. In the nature of the people, the Norwegians always want to maximize the distance to each other, meaning that they form triangular lattices on the metro station. In extreme cases, this is equivalent to the Wigner crystals.

CHAPTER 4

Systems

We must be clear that when it comes to atoms, language can be used only as in poetry.

Niels Bohr, [50]

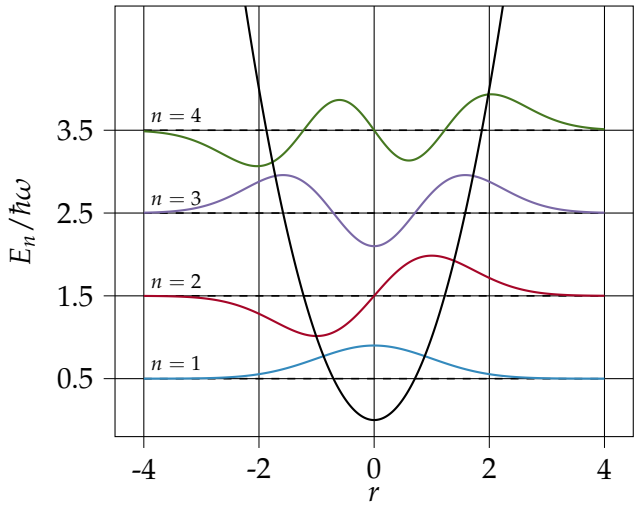


Figure 4.1: The harmonic oscillator potential (solid black line), with the exact single particle wave functions, the Hermite functions, represented up to the 4th order. The quantum harmonic oscillator has many applications, be it the binding potential in molecules, lattice vibrations (phonons) or any trapped particle [51].

In quantum physics, as in classical physics, there are many different systems with various complexities. If a system gets very complicated, it is often beneficial to approximate it with a more straightforward system, which hopefully catches the essential features in the system. Approximations of systems are especially advantageous in quantum mechanics, as just a few systems can be solved analytically. We have already seen that the atomic system can be solved analytically for the non-interacting case, and additionally harmonic systems can be solved analytically. Our circular quantum dots have a confinement represented by the harmonic potential, and for non-interacting quantum dots, we can thus also find analytical expressions.

We will in this chapter set up the explicit expressions of the Hamiltonians for the various systems and single-particle functions for various quantum mechanical systems. Although the quantum dots will be investigated in the first place, it is natural to include a brief discussion of the atom as there are many common features between the quantum dots. Also, double quantum

dots will be covered, as one can find analytical solutions by an expansion of single quantum dot functions.

4.1 Quantum dots

Quantum dots are tiny particles and consist of fermions or bosons trapped in an external potential which is not created by a nucleus, unlike atoms. However, they have discrete electronic states with a well-defined shell structure like atoms and are therefore often called artificial atoms. In this thesis, we will study circular quantum dots with electrons confined in a harmonic oscillator potential, sometimes called Hooke's atom.

The harmonic oscillator is often the first system that one is introduced to when attending an introductory course to quantum mechanics, for several reasons. Firstly, many systems can be approximated with the harmonic oscillator potential, including interactions in molecules and lattice vibrations. Secondly, the harmonic oscillator behaves nicely with absent of local minima and has neat equations for the non-interacting case. For an electron i, the potential reads

$$V_i(r) = \frac{1}{2}m\omega^2 r_i^2, \tag{4.1}$$

where m is the electron mass, ω is the oscillator frequency and r_i is the relative distance from particle i to the center of the dot. Using natural units as described in Appendix B, we can write the Hamiltonian as

$$\hat{\mathcal{H}} = \sum_{i=1}^{N}\Big(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2\Big) + \sum_{i<j}\frac{1}{r_{ij}} \tag{4.2}$$

where the energy is given in Hartrees and the lengths are scaled with respect to $\sqrt{\hbar/m}$.

The exact solutions of the non-interacting Hamiltonian are the Hermite functions,

$$\phi_n(x) = H_n(\sqrt{\omega}x)\exp\big(-\omega x^2/2\big) \tag{4.3}$$

which will also be used for the interacting atoms, though with a variational parameter detailed in section 10.3.2. $H_n(x)$ is the Hermite polynomial of n'th degree, and the first four Hermite functions are illustrated in figure (4.1). The energy of an electron with principal quantum number n in a certain dimension is given by the binomial coefficient

$$E_n = \omega\Big(n + \frac{1}{2}\Big) \qquad \forall \quad n \in \{0, 1, 2...\} \tag{4.4}$$

where we in multi-dimensional potentials need to summarize the contribution from all the dimensions.

We will study closed-shell systems only as the Slater determinant in that case is unambiguous, i.e, there is only one possible determinant. In open-shell systems, there are in general multiple possible Slater determinants, and the trial wave function is thus a linear combination of them. The number of particles in closed-shell systems are called magic numbers, which in two dimensions are $N = 2, 6, 12, \ldots$. In general, the magic numbers are given by

$$N = s\binom{n+d}{d} \qquad \forall \quad n \in \{0, 1, 2...\} \tag{4.5}$$

where s is the number of spin configurations (2 for electrons), n is the principal quantum number and d is the number of dimensions. This is a direct consequence of the Pauli principle, where the ground state can take two electrons with spatial single-particle functions $\phi_{n_x=0,n_y=0}(x,y)$, the first excited energy level can take four electrons with spatial single-particle functions $\phi_{n_x=1,n_y=0}(x,y)$ and $\phi_{n_x=0,n_y=1}(x,y)$ with degeneracy 2 and so on. See figure (4.2) for an illustration of the first few states in a two-dimensional quantum dot.

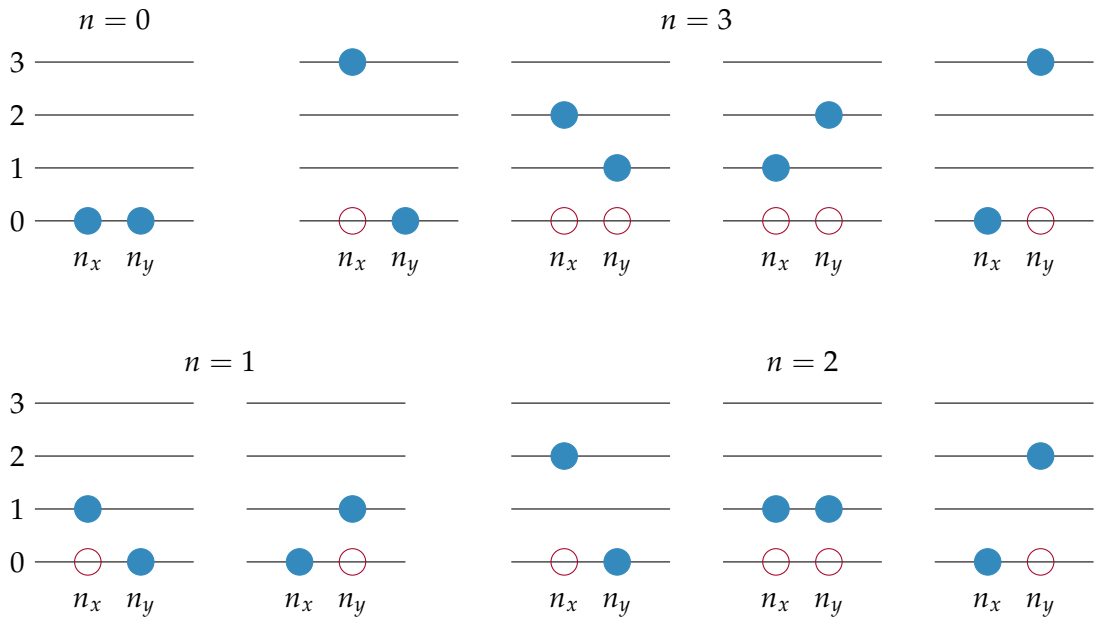


Figure 4.2: The possible states of a two-dimensional quantum dot for $n = n_x + n_y = 0, 1, 2, 3$. Recalling that the electrons can take spins $+1$ and -1, one can use this schematic to determine the maximal number of electrons in each shell and thus reveal the magic numbers.

4.2 Quantum double dots

Another historically important quantum system is the double dot, which similarly to the single dot can be solved analytically. For the same reason as the single dot often is called an artificial atom, the double dots are called artificial molecules.

The potential of symmetrical quantum dots can in principle take a variety of different shapes, but the most used one-dimensional symmetrical potentials can be derived from the formula

$$V_i(x) = \frac{1}{2}\omega^2\left[|x_i|^a - \left(\frac{b}{2}\right)^a\right]^2 \tag{4.6}$$

with b as the distance between the lowest points of the wells and a as an arbitrary integer [52]. Setting $a = 1$ gives two parabolic wells with a sharp local maximum at $x = 0$, while $a = 2$ gives a smoother but steeper well. In figure (4.3) the potential is plotted for $a = 1, 2$ and 3 and $b = 2$.

For reference and benchmark purposes, we will focus on the case with $a = 1$ and $b = 2$, which can be written out as

$$V_i(x) = \frac{1}{2}\omega^2\left[x_i^2 + \frac{1}{4}b^2 - b|x_i|\right], \tag{4.7}$$

still in one dimension. For more than one dimension, we assume that the double dot expands in the x-direction, which gives us the dimension-independent expression

$$V_i^{\text{DW}} = \frac{1}{2}\omega^2\left[r_i^2 + \frac{1}{4}b^2 - b|x_i|\right] = V_i^{\text{HO}} + \frac{1}{2}\omega^2\left[\frac{1}{4}b^2 - b|x_i|\right] \tag{4.8}$$

where HO means harmonic oscillator potential and DW means double-well potential. What we actually observe, is that the potential separates in a single-dot part and a double-dot part, which

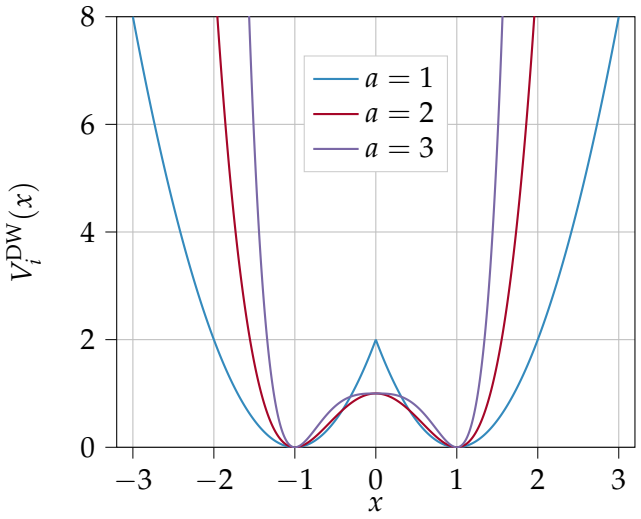


Figure 4.3: One-dimensional double-well potentials plotted with $a = 1, 2$ and 3, and $b = 2$. For $a = 1$ the potential was multiplied with 2 to make it comparable to the others.

makes the double-dot Hamiltonian similar to the single-dot Hamiltonian,

$$\hat{\mathcal{H}} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 + \frac{1}{2}\omega^2 \left(\frac{1}{4}b^2 - b|x_i| \right) \right) + \sum_{i<j} \frac{1}{r_{ij}}. \tag{4.9}$$

What is remaining is to find an appropriate set of single-particle functions. Based on the observations above, a reasonable set should be similar to the Hermite functions which we found to be the analytical single-particle functions of the single dot. For that reason, we will expand the double-dot single-particle functions in single-dots single-particle functions,

$$\left|\phi_n^{\mathrm{DW}}(x)\right\rangle = \sum_{\lambda=1}^{L} C_{n\lambda} \left|\phi_\lambda^{\mathrm{HO}}(x)\right\rangle, \tag{4.10}$$

where $L \in [1, \infty\rangle$ is the number of basis functions used and $C_{n\lambda}$ is the coefficient associated with the double-dot function n and the single-dot function λ, which is what we want to find. Inserting this into the double-dot Schrödinger equation and multiplying with $\langle\phi_\nu^{\mathrm{HO}}(x)|$ on the left-hand side gives

$$\sum_{\lambda=1}^{L} C_{n\lambda} \left\langle \phi_\nu^{\mathrm{HO}}(x) \middle| \hat{\mathcal{H}}_{\mathrm{DW}} \middle| \phi_\lambda^{\mathrm{HO}}(x) \right\rangle = E_n \sum_{\lambda=1}^{L} C_{n\lambda} \left\langle \phi_\nu^{\mathrm{HO}}(x) \middle| \phi_\lambda^{\mathrm{HO}}(x) \right\rangle \tag{4.11}$$

where the right-hand side sum collapses because the overlap is just the Kronecker delta, $\langle\phi_\nu^{\mathrm{HO}}(x)|\phi_\lambda^{\mathrm{HO}}(x)\rangle = \delta_{\nu\lambda}$. By defining the matrix elements

$$\hat{h}_{\nu\lambda} \equiv \left\langle \phi_\nu^{\mathrm{HO}}(x) \middle| \hat{\mathcal{H}}_{\mathrm{DW}} \middle| \phi_\lambda^{\mathrm{HO}}(x) \right\rangle, \tag{4.12}$$

we can set up equation (4.11) as an eigenvalue problem in the form of

$$\hat{h}\hat{C} = E\hat{C} \tag{4.13}$$

where our targets, $\hat{C}$, are just the eigenvectors of the $\hat{h}$-matrix. Now recall that the double dot Hamiltonian is just an extension of the single dot Hamiltonian, such that we can rewrite

$$\hat{h}_{\nu\lambda} = \left\langle \phi_\nu^{\mathrm{HO}}(x) \middle| \hat{\mathcal{H}}_{\mathrm{HO}} \middle| \phi_\lambda^{\mathrm{HO}}(x) \right\rangle + \left\langle \phi_\nu^{\mathrm{HO}}(x) \middle| \hat{\mathcal{H}}_{+} \middle| \phi_\lambda^{\mathrm{HO}}(x) \right\rangle \tag{4.14}$$

with $\hat{\mathcal{H}}_+ = (1/2)\omega \sum_{i=1}^{N} \Big((1/4)b^2 - b|x_i| \Big)$ as the contribution from the double-dot. The former integrals are just the harmonic oscillator energies, presented in equation (4.4), while the latter integrals are trivial to calculate. However, since we need to calculate a relatively large number of matrix elements, we decided to do it using numerical integration on the computer. The script doing this is given the creative name `doublewell_functions.py`.

4.3 Atoms

Although we will focus on quantum dots in this work, atoms are relevant as they behave similar to quantum dots and can be studied with the same code framework with few modifications. When defining the atomic Hamiltonian, one often freezes out the nucleonic degrees of freedom known as the Born-Oppenheimer approximation. The electrons will in fact affect the nucleus, but due to the mass difference this effect is negligible. We again have Coulomb interaction between the electrons and the nucleus, and since we assume the latter to be at rest at the origin, the external potential affecting particle i is

$$V_i(r) = -\frac{1}{2} k_e \frac{Ze^2}{r_i}, \tag{4.15}$$

where Z is the atomic number (number of protons in the nucleus). The total Hamiltonian,

$$\hat{\mathcal{H}} = \sum_{i=1}^{N} \Big(-\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i} \Big) + \sum_{i<j} \frac{1}{r_{ij}}, \tag{4.16}$$

is given in atomic units which also is discussed in Appendix B. For the non-interacting case, the energy of a particle in shell n is given by the Bohr formula

$$E_n = -\frac{Z^2}{2n^2}, \tag{4.17}$$

which means that to find the binding energy of an atom, we need to summarize the energy of all the electrons.

Also, for this system, we need to specify a basis set to use. For atoms where the electrons do not interact, the wave functions are given by the *Hydrogen-like orbitals*, which in spherical coordinates can be split in a radial part and an angular part,

$$\psi_{nlm_l}(r, \theta, \phi) = R_{nl}(r) Y_l^{m_l}(\theta, \phi), \tag{4.18}$$

where n again is the principal quantum number, l is the angular quantum number and m_l is the magnetic quantum number. Henceforth m_l will be simplified as m to make the expressions neater. Since we are physicists, we use θ as the polar angle and ϕ is the azimuthal angle.

The radial part can be presented as a function of the *associated Laguerre polynomials* or *generalized Laguerre polynomials*, $L_q^p(x)$ and the Hydrogen wave function presented in section 2.2.1,

$$R_{nl}(r) \propto r^l e^{-Zr/n} \Big[L_{n-l-1}^{2l+1}\Big(\frac{2r}{n} Z\Big) \Big]. \tag{4.19}$$

The angular part is given by the *spherical harmonics*,

$$Y_l^m(\theta, \phi) \propto P_l^m(\cos\theta) e^{im\phi} \tag{4.20}$$

Table 4.1: Degeneracy and naming conversations for $l = 0, 1, 2, 3, 4$.

Subshell label	l	Max electrons	Name
s	0	2	sharp
p	1	6	principal
d	2	10	diffuse
f	3	14	fundamental
g	4	18	*alphabetic hereafter*

where $P_l^m(x)$ are the *associated Legendre polynomials*. The complex part in the spherical harmonics can be avoided by introducing the real solid harmonics instead

$$S_l^m(\theta,\phi) \propto P_l^{|m|}(\cos\theta) \begin{cases} \cos(m\phi) & \text{if} \quad m \geq 0 \\ \sin(|m|\phi) & \text{if} \quad m < 0, \end{cases} \tag{4.21}$$

such that

$$\psi_{nlm}(r,\theta,\phi) = R_{nl}(r) S_l^m(\theta,\phi). \tag{4.22}$$

The real solid harmonics do not alter any physical quantities that are degenerate in the subspace consisting of opposite magnetic quantum numbers, and for that reason they will give the same energies as the spherical harmonics as long as there is no magnetic field present [53].

Alternatively, one can use a Hartree-Fock basis based on Gaussian functions, as discussed in section 3.2.3. More or less all electronic structure studies of atoms nowadays use these expansions, as they provide very precise results and high performance due to the evaluation properties of the Gaussian functions. Even though the Gaussian functions do not have the correct shape, an expansion can be fitted pretty well, as demonstrated by Hehre *et al.* [54].

As for the quantum dot systems, we will study closed shells only, but for atoms, we will introduce subshells as well which are dependent on l and m in addition to the principal quantum number n. Traditionally, the first few subshells are denoted by s, p, d and f, and the meaning can be found in table (4.1), together with number of electrons in each subshell. For Helium, we have two electrons with $n = 1$, which means that both have $l = 0$ and both electrons are in the s-subshell, or the so-called s-wave. We can thus write the electron configuration as $1s^2$. Similarly to the principal quantum number n, we can use the rule of tumble that the lower l, the lower energy, such that for Beryllium all four electrons are still in the s-subshell. Beryllium therefore has electron configuration $1s^2 2s^2$ or [He] $2s^2$. Since both subshells are fully occupied, Beryllium can be included in our closed-shell calculations. If we continue with the same rules, we see that the next closed-shell atom has a fully occupied p-subshell as well, which is Neon with 10 electrons. This is a noble gas, and we can write the electron configuration as [Be] $2p^6$. All noble gases have endings Xs^2Xp^6, which is the reason why they always have eight valence electrons.

We can now compare this to the periodic table, and observe that the first two rows agree with the theory presented above: The first row has two elements, and the second has eight. However, the third one also has eight elements, which does not fit our theory. It must be something we have overlooked. The reason is that the angular momentum contribution is not taken into account, i.e., we need to include the Hamiltonian term

$$V_L = \frac{l(l+1)}{2r^2} \tag{4.23}$$

as well. If we do so, we see that the rule of thumb defined above not always holds. Sometimes a low l in a higher n causes lower energy than a high l in a lower n.

Part II

Machine Learning Theory

CHAPTER 5

Supervised Learning

People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world.

Pedro Domingos, [55]

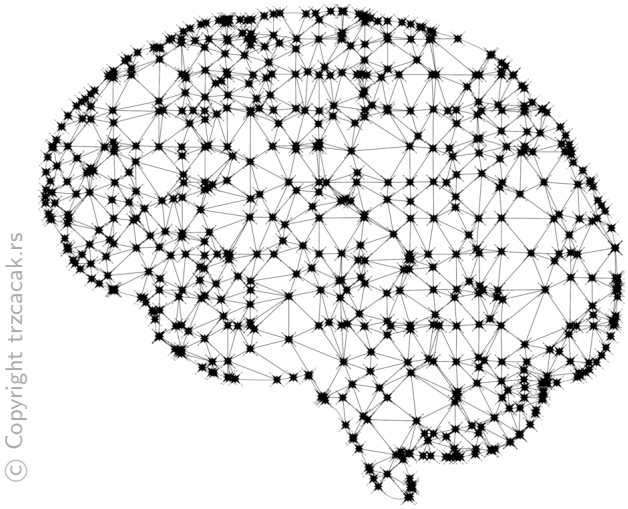


Figure 5.1: Artificial neural networks are inspired by neural networks in the brain.

The use of the term *machine learning* has exploded over the past years, and sometimes it even sounds like it is a new field. However, the truth is that many of the methods are relatively old, where for instance *linear regression* was known in the early 19th-century when Legendre [56] and Gauss [57] independently developed the concepts of mean square error (MSE). Those methods have just recently been taken under the machine learning umbrella, which is one of the reasons why the term is used more frequently than before. Another essential contributor to the booming popularity is the dramatically improvement of a majority of the machine learning algorithms, most notably neural networks, as discussed in the introduction.

Unlike traditional algorithms, machine learning algorithms are not explicitly told what to do, but they use optimization tools to minimize a *cost function* and fit a model to data sets. As a consequence, we often do not know precisely what the algorithms do and why they behave as they do. Because of this behavior and the fact that artificial neural networks are inspired by

the human brain, the processing is often called artificial intelligence. As a definition of the term machine learning, we use the definition by Stanford [58]:

> “Machine learning is the science of getting computers to act without being explicitly programmed.”

In our search for a technique to solve quantum mechanical problems where less physical intuition is needed, machine learning appears as a natural tool. First, machine learning algorithms are able to provide impressive results, even with absence of rules specifying what the network should do. This was, for instance, demonstrated by Silver *et al.* [14] who developed a chess engine which learned the rules by playing against itself thousands of times. Second, the fundamental goal of machine learning is to fit a model in order to minimize a cost function, which is exactly what we do in the popular variational Monte Carlo (VMC) method with the wave function as the model and with the energy as the cost function. However, in traditional VMC, the trial wave function is tailored to each particular system, which naturally requires a significant amount of physical intuition. With the trial wave function defined by a restricted Boltzmann machine (RBM), we attempt to find a model so flexible that it can approach the true wave function even with lack of physical intuition. This is important because the study of many urgent systems is out of reach for the existing methods due to lack of information about the wave function. Examples include bosonic systems, where expressions for the two-body correlations in general are unavailable, and nuclear systems, where the three-body correlations are crucial.

We typically classify the machine learning methods as either supervised or unsupervised, based on the way we define the cost function. Supervised models are provided with *targets*, which are outputs that the model should obtain with a certain input data set. We then define the cost function as the error between the output from the model and the targets, and the model is trained until it is able to reproduce the targets. On the other hand, unsupervised models are not provided with targets, and the cost function thus needs to be defined in another way. In our work, we aim to construct a robust method for the study of the ground state of complex systems, and as targets for those systems usually are unavailable, we will use unsupervised algorithms. For RBMs, the cost function is traditionally defined by a system energy which we want to minimize. As there are many common concepts in supervised and unsupervised learning, we will in this chapter stick to supervised learning where we introduce essential concepts like artificial neurons, weights and optimization schemes. The Boltzmann machines, which are the models that we actually use in the work, are discussed in chapter 6.

Supervised learning maps an input to an output based on example input-output pairs. To make this work, we need to train the model such that the error between the targets and the outputs is tolerable small. However, this is in general not sufficient as we also want to make new predictions from the same model. The two points that need to be satisfied in supervised learning are therefore:

1. The model needs to reproduce the targets
2. The model should be able to fit future observations.

In this chapter, we will examine how a model that satisfies both the requirements can be found, possible challenges and when the ansatz will break down. Subsequently, we will see that there is no guaranty that the second point is satisfied even when the first point is satisfied. First, the polynomial regression is presented to explain fundamental concepts of machine learning in an intuitive way, and thereafter we generalize the theory in form of linear regression. In the end, we go thorough neural networks which have many common features with the restricted Boltzmann machines discussed in the next chapter.

5.1 Polynomial regression

The polynomial regression is perhaps the most intuitive example on supervised learning, as it can be used to solve problems everyone is familiar with. In general, polynomial regression finds the p'th degree polynomial, $f(x;\boldsymbol{c}) = \sum_{i=0}^{p} c_i x^i$, that fits a set of points in the best possible way. In two dimensions, the data set consists of some n number of x- and y-coordinates,

$$\begin{aligned}\boldsymbol{x} &= (x_1, x_2, \ldots, x_n)\\ \boldsymbol{y} &= (y_1, y_2, \ldots, y_n),\end{aligned}$$

henceforth denoted by $\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$. The data set can for instance be fitted to a second-order polynomial,

$$f(x;a,b,c) = ax^2 + bx + c, \tag{5.1}$$

where the parameters a, b and c are our *estimators*. The polynomial is now our model, and by evaluating it on all values in the $\boldsymbol{x}$-vector we obtain a set of n equations

$$\begin{array}{ccccccc}\tilde{y}_1 & = & ax_1^2 & + & bx_1 & + & c\\ \tilde{y}_2 & = & ax_2^2 & + & bx_2 & + & c\\ \vdots & & \vdots & & \vdots & & \vdots\\ \tilde{y}_n & = & ax_n^2 & + & bx_n & + & c\end{array} \tag{5.2}$$

where $\tilde{y}_i = f(x_i)$ is the output from the model with x_i as the input. What we want to do is to determine the estimators a, b and c such that the mean squared error (MSE) of all these equations,

$$\min_{a,b,c} \frac{1}{n}\sum_{i=0}^{n-1}(y_i - f(x_i;a,b,c))^2, \tag{5.3}$$

is minimized. The MSE is then defined as our cost function $\mathcal{C}(\boldsymbol{\theta})$ (also called the loss function), which is always the function that we try to minimize in machine learning. For our choice of model, the cost function reads

$$\mathcal{C}(a,b,c) = \frac{1}{n}\sum_{i=0}^{n-1}\Big(y_i - (ax_i^2 + bx_i + c)\Big)^2, \tag{5.4}$$

which can be minimized in several ways. Before we proceed to the minimization, we will introduce a more general notation, where the estimators are collected in a column vector

$$\boldsymbol{\theta} \equiv (a,b,c)^T \tag{5.5}$$

and the x_i^j's are collected in a row vector

$$\boldsymbol{X}_i \equiv (x_i^2, x_i^1, x_i^0) = (x_i^2, x_i, 1). \tag{5.6}$$

By using this, the cost function can be written as

$$\begin{aligned}\mathcal{C}(\boldsymbol{\theta}) &= \frac{1}{n}\sum_{i=0}^{n-1}\Big(y_i - \sum_{j=0}^{2} X_{ij}\theta_j\Big)^2\\ &= \frac{1}{n}\sum_{i=0}^{n-1}\Big(y_i - \boldsymbol{X}_i\boldsymbol{\theta}\Big)^2\\ &= \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})\end{aligned} \tag{5.7}$$

where we in the last step have collected all the vectors $\boldsymbol{X}_i$ in a matrix $\boldsymbol{X} = [\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n]$. As the minimum of the cost function with respect to an estimator θ_k is found when the derivative is zero, we need to solve the equation

$$\begin{aligned}\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k}\left(\frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \sum_{j=0}^{2} X_{ij}\theta_j\right)^2\right) \\ &= \frac{2}{n}\sum_{i=0}^{n-1} X_{ik}\left(y_i - \sum_{j=0}^{2} X_{ij}\theta_j\right) = 0.\end{aligned} \tag{5.8}$$

We can go further and write it on matrix-vector form as

$$\frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{2}{n}\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = 0 \tag{5.9}$$

where the differentiating is done element-wise, $\partial \mathcal{C}(\boldsymbol{\theta})/\partial \theta_k$. This is satisfied if and only if

$$\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{5.10}$$

which is the equation we seek to solve to find the best fitting polynomial. Before we proceed to the general case, let us take a quick look at an example.

5.1.1 Example

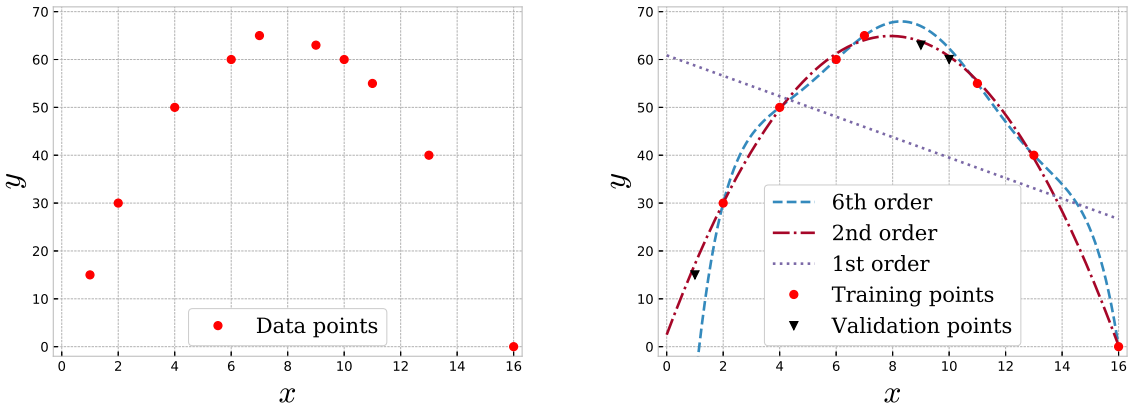
In this example, we will do the polynomial regression on an actual two-dimensional data set consisting of 10 points,

$$\begin{aligned}\boldsymbol{x} &= (1, 2, 4, 6, 7, 9, 10, 11, 13, 16) \\ \boldsymbol{y} &= (15, 30, 50, 60, 65, 63, 60, 55, 40, 0),\end{aligned} \tag{5.11}$$

which is nothing else than a second-order polynomial with some noise. The data points are plotted in figure (5.2 a), and we want to fit a p'th degree polynomial to the points. The first thing we need to realize is that in order to validate our models, we cannot use all points for the training. There is no strict rule on how much of the data set that should be used for training and validation, but at least the training data set should be larger than the validation data set. For this particular problem, we decide to leave out $\{(1, 15), (9, 63), (10, 60)\}$ from the training, which we later will use for validation.

Furthermore, we use equation (5.10) to find the best fitting first-, second- and sixth-order polynomials, and obtain the functions presented in table (5.1) with the respective training and prediction errors. The polynomials are also plotted in figure (5.2 b) together with the actual data points.

What we immediately observe, is that the more complex model (higher degree polynomial), the lower the training error. The polynomial of sixth-order reproduces the points entirely. The first-order polynomial is quite bad, while the second-order polynomial is intermediate. However, what is most important is the prediction error as it shows the ability to reproduce data that is not prior known, and for that, we can see that the sixth order polynomial performs terribly. When a model can reproduce the training set very well but is not able to reproduce the training set, we say that it overfits the data set. This means that the model is too complicated for the purpose. On the other hand, we see that the first-order polynomial also has a significant prediction error, which means that it is not able to reproduce the validation set either. We say that it is under fitted, and we are in need of a more complex model.



(a) Data set **(b)** Data set with fitted polynomials

Figure 5.2: Figure (a) presents the data points given in equation (5.11), while the figure (b) illustrates how a first-, second- and sixth order polynomial can be fitted to the training set in the best possible way.

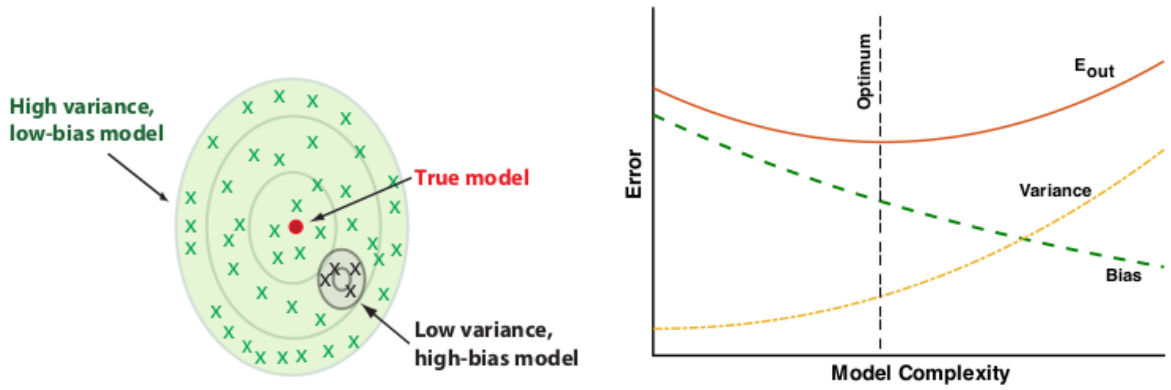
Table 5.1: Best fitting polynomials of first-, second- and sixth-order degree to the data set in equation (5.11). $f(x)$ gives the actual form of the polynomial, the training error is the MSE of the training data set and the prediction error is the MSE of the validation data set.

Order	$f(x)$	Training error	Prediction error
1st	$-2.14x + 60.87$	327.22	927.87
2nd	$-x^2 + 15.74x + 2.51$	0.47	2.04
6th	$-0.001x^6 + 0.04x^5 - 0.90x^4 + 9.04x^3 - 47.52x^2 + 129.74x - 98.67$	2.54E-11	187.53

Finally, we have the second-order polynomial, which is miles ahead of its competitors when it comes to the prediction error. It turns out that the second-order model has an appropriate complexity, which we could have guessed just by looking at the data points. The natural question now is *"How do we find a correct model complexity?"*. The answer is that there no easy way of doing this, which is one reason why machine learning is difficult. The trail and error method is the standard approach, where one examines various complexities and calculate the prediction error for each model. To find the prediction error precisely, one typically uses K cross-validation resampling, which evaluates K different choices of validation set to make the most use of the data. More about resampling analysis can be found in section 7.4.2. A deeper understanding of the prediction error and how to reveal if a model overfits or under its will hopefully be gained in the next section, on bias-variance tradeoff.

5.2 Bias-variance tradeoff

Up to this point, we have skipped some important terms in the statistics behind machine learning. First, we have the *bias*, which describes the best our model could do if we had an infinite amount of training data. We also have the *variance*, which is a measure of the fluctuations in the predictions. In figure (5.3 a), an example of high variance low-bias and a low variance high bias



(a) Illustration of bias and variance

(b) Bias-variance trade-off

Figure 5.3: Examples of high variance, low-bias and low variance high-bias (a) and illustration of the bias-variance trade-off (b). Figures are taken from Mehta *et al.* [59].

models are presented. What we actually want is a low variance low-bias model, but this model is normally infeasible, and we need to find the optimal tradeoff between bias and variance. This is known as the bias-variance tradeoff.

In figure (5.3 b), the bias-variance tradeoff is illustrated as a function of the model complexity. We observe that the prediction error is large when the model complexity is too low, which corresponds to a low variance. This substantiates what we discussed in the example in section 5.1.1, where we claimed that a too low model complexity under its data set. Therefore, a too low variance is associated with underfitting. On the other side of the plot, we can see that also a too complex model causes a large prediction error, which corresponds to a low bias. As discussed before, a too complex model overfits the model, which is associated with low bias.

To minimize the prediction error, we should therefore neither minimize the bias nor the variance. Instead, we should find the bias and variance which corresponds to the lowest error. To see how the error is distributed between the bias and variance, we can perform a so-called bias-variance decomposition. This is easiest if we assume that the true data is generated from a noisy model

$$y = f(x) + \epsilon \tag{5.12}$$

where ϵ is normally distributed with mean zero and standard deviation σ_ϵ. In that case, it can be shown that

$$\langle\langle(y_\epsilon - \tilde{y})^2\rangle\rangle_\epsilon = \underbrace{\sum_i (f(x_i) - \langle\tilde{y}\rangle)^2}_{\text{bias}^2} + \underbrace{\sum_i \langle(\tilde{y} - \langle\tilde{y}\rangle)^2\rangle}_{\text{variance}} + \underbrace{\sum_i \sigma_\epsilon^2}_{\text{noise}} \tag{5.13}$$

where the expectation values are over the data set $\mathcal{D}$ if nothing else is specified. This decomposition is shown carefully by Mehta *et al.* [59].

5.3 Linear regression

Polynomial regression, as already discussed, is an instance of linear regression and was meant as a motivation before we study linear regression in general. Instead of fitting a polynomial to a

set of points, we can fit a general function in the form of

$$f(x_i) = \sum_{j=0}^{p} X_{ij}(x_i)\theta_j, \tag{5.14}$$

where we have $p+1$ estimators θ_j. The matrix $\boldsymbol{X}$ is called the *design matrix*, and in the case where $X_{ij}(x_i) = x_i^j$ corresponds to polynomial regression, but it can in principle be an arbitrary function of x_i. The cost function for *the ordinary least square regression* (OLS) case is already found in equation (5.7), and we can recall it as

$$\mathcal{C}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{p} X_{ij}\theta_j\right)^2, \qquad \text{OLS} \tag{5.15}$$

which is minimized when

$$\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{5.16}$$

To solve this equation, we need to find the inverse of the matrix $\boldsymbol{X}^T\boldsymbol{X}$, which is typically done by *lower-upper* decomposition (LU) or *singular values decomposition* (SVD). However, quite often, when we deal with large data sets, the matrix above is singular, which means that the determinant of the matrix is zero. In those cases, we cannot find the inverse, and LU decomposition does not work. Fortunately, SVD *always* works, and in cases where the matrix is singular, it turns out to be a good idea to perform such decomposition.

5.3.1 Singular value decomposition

Singular value decomposition is a method which decomposes an $m \times n$ matrix $\boldsymbol{X}$ into a product of three matrices, written as

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{5.17}$$

where $\boldsymbol{U}$ is a unitary $m \times m$ matrix, $\boldsymbol{V}$ is a unitary $n \times n$ matrix and $\boldsymbol{\Sigma}$ is a diagonal $m \times n$ matrix. This might sounds like a bad idea, but especially for singular matrices this often makes life easier. The reason for this, is that only $\boldsymbol{\Sigma}$ is singular after the decomposition. For our case, we can thus write the matrix $\boldsymbol{X}^T\boldsymbol{X}$ as

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T \tag{5.18}$$

where we exploit that $\boldsymbol{U}^T\boldsymbol{U} = \mathbb{1}$ and $\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} = \boldsymbol{D}$ by definition. Further we can multiply by $\boldsymbol{V}$ on the right-hand-side

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{V} = \boldsymbol{V}\boldsymbol{D} \tag{5.19}$$

to get rid of the $\boldsymbol{V}^T$. A similar exercise can be done on $\boldsymbol{X}\boldsymbol{X}^T$, and we will obtain

$$(\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{U} = \boldsymbol{U}\boldsymbol{D}. \tag{5.20}$$

By using the former of the two expressions, one can show that

$$\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{y} \tag{5.21}$$

which is solvable even when $\boldsymbol{X}^T\boldsymbol{X}$ is singular.

5.3.2 Ridge regression

So, how can we avoid non-singular values in our matrix $\boldsymbol{X}^T\boldsymbol{X}$? We can remove them by introducing a penalty λ to ensure that all the diagonal values are non-zero, which can be accomplished by adding a small value to all diagonal elements. By doing this, all diagonal elements will get a non-zero value and the matrix is guaranteed to be non-singular. Still using the matrix-vector form, this can be written as

$$\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{5.22}$$

where $\boldsymbol{I}$ is the identity matrix. The penalty λ is also a *hyper-parameter*, which is a parameter that is specified before the training begins, in contrast to the estimators which are determined throughout training. This method is called Ridge regression, and has a cost function given by

$$\mathcal{C}(\boldsymbol{\theta}) = \sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p} X_{ij}\theta_j\right)^2 + \lambda\sum_{j=1}^{p}|\theta_j|^2, \qquad \text{Ridge} \tag{5.23}$$

where we in principle just add the L2-norm of the estimator vector to the OLS cost function. The link between equation (5.22) and (5.23) can be easiest found by going from the latter to the former, similarly to what we did for the polynomial regression in equations (5.7-5.10).

5.3.3 LASSO regression

Finally, we introduce the *least absolute shrinkage and selection operator* (LASSO) regression, which in the same way as Ridge regression is based on regularization. Instead of adding the L2-norm of the estimator matrix, we add the the L1-norm $|\theta_j|$, and the cost function expresses

$$\mathcal{C}(\boldsymbol{\theta}) = \sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p} X_{ij}\theta_j\right)^2 + \lambda\sum_{j=1}^{p}|\theta_j|. \qquad \text{Lasso} \tag{5.24}$$

For LASSO regression, we cannot set $\partial\mathcal{C}(\boldsymbol{\theta})/\partial\theta_k = 0$ and find a closed-form expression of $\boldsymbol{\theta}$, which means that we need to use an iterative optimization algorithm in order to obtain the optimal estimators. Such optimization methods are essential in non-linear problems such as deep neural networks and variational Monte-Carlo. They will therefore be familiar to the reader throughout this thesis. In section 5.6, we present various optimization methods, but for now on we will stick to one of the most basic methods, *gradient descent*, which can be written as

$$\theta_{k,t} = \theta_{k,t-1} - \eta\frac{\partial\mathcal{C}(\boldsymbol{\theta}_{t-1})}{\partial\theta_k}, \tag{5.25}$$

where $\theta_{k,t}$ is the parameter θ_k at iteration t and $\mathcal{C}(\boldsymbol{\theta})$ is an arbitrary cost function. Here we are introduced to a new hyper-parameter, η, known as the *learning rate*, which controls how much the estimators should be changed for each iteration. It has to be specified carefully, where a too large η will make the cost function diverge and a too small η will make the training too slow. Typically, to choose an η in the range 0.01-0.0001 is a good choice. For OLS, the vectorized iterative parameter update can be written as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}_{t-1}). \tag{5.26}$$

5.4 Logistic regression

So far, we have discussed polynomial regression and linear regression, which consist of models giving continuous outputs. However, what do we do if we want discrete outputs, for example, in the form of classification? This is what logistic regression is all about, where the name comes from the logistic function (sigmoid function) which is used to fire or not fire the neurons. As for the linear regression, we also need a cost function in logistic regression, which we will motivate in the following.

Consider a system that can take two possible energies ε_0 and ε_1. From elementary statistical mechanics, we know that the probability of finding a system in a state of a certain energy is given by the Boltzmann distribution, such that

$$P(y_i = 0) = \frac{\exp(-\varepsilon_0/k_BT)}{\exp(-\varepsilon_0/k_BT) + \exp(-\varepsilon_1/k_BT)} \tag{5.27}$$

$$= \frac{1}{1 + \exp(-(\varepsilon_1 - \varepsilon_0)/k_BT)} \tag{5.28}$$

which is the *sigmoid function*, in the most general given by

$$f(x) = \frac{1}{1 + \exp(-x)}. \tag{5.29}$$

The first denominator is known as the *partition function*,

$$Z = \sum_{i=0}^{1} \exp(-\varepsilon_i/k_BT) \tag{5.30}$$

where k_B is Boltzmann's constant and T is the system temperature. The probability of finding the system in the second state is given by

$$P(y_i = 1) = 1 - P(y_i = 0) \tag{5.31}$$

$$= \frac{1}{1 + \exp(-(\varepsilon_0 - \varepsilon_1)/k_BT)}. \tag{5.32}$$

Notice that the only thing we need is the energy difference between the two states rather than the energy itself. This is often the case in physics, where we for instance have no absolute potential energy. If we now assume that the energy difference can be written as a function of the coordinates that specify the state i stored in the row vector $\boldsymbol{X}_i$, and a column vector with parameters, $\boldsymbol{w}$, known as the *weights*, the difference can be written as

$$\varepsilon_1 - \varepsilon_0 = \boldsymbol{X}_i\boldsymbol{w} \equiv \tilde{y}_i, \tag{5.33}$$

giving the conditional probability

$$P(\boldsymbol{X}_i, y_i|\boldsymbol{w}) = \big(f(\boldsymbol{X}_i\boldsymbol{w})\big)^{y_i}\big(1 - f(\boldsymbol{X}_i\boldsymbol{w})\big)^{1-y_i}. \tag{5.34}$$

If we have a set of multiple states stored in a $\mathcal{D} = \{(\boldsymbol{X}_i, y_i)\}$, the joint probability yields

$$P(\mathcal{D}|\boldsymbol{w}) = \prod_{i=1}^{n} \big(f(\boldsymbol{X}_i\boldsymbol{w})\big)^{y_i}\big(1 - f(\boldsymbol{X}_i\boldsymbol{w})\big)^{1-y_i} \tag{5.35}$$

which is known as the *likelihood*. The *log-likelihood* function is simply the log of the likelihood, and is given by

$$l(\boldsymbol{w}) = \sum_{i=1}^{n} \Big[y_i \log f(\boldsymbol{X}_i\boldsymbol{w}) + (1 - y_i)\log(1 - f(\boldsymbol{X}_i\boldsymbol{w}))\Big]. \tag{5.36}$$

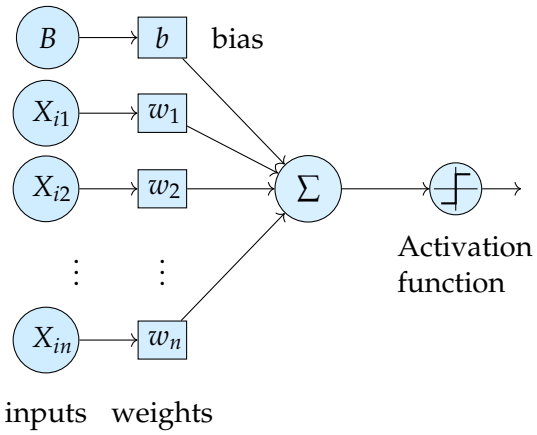


Figure 5.4: Logistic regression model with n inputs. Each input X_{ij} is multiplied with a weight w_j, and the contributions from all units are summarized. The output is obtained after the sum is activated by an activation function.

As in linear regression, we want to find a cost function which we can minimize in order to fit the model to the data set. Since the log-likelihood function is at its maximum at the highest probability, a natural choice is to set the cost function to the negative log-likelihood function,

$$\mathcal{C}(\boldsymbol{w}) = -l(\boldsymbol{w}) = -\sum_{i=1}^{n}\Big[y_i \log f(\boldsymbol{X}_i\boldsymbol{w}) + (1-y_i)\log(1-f(\boldsymbol{X}_i\boldsymbol{w}))\Big], \tag{5.37}$$

which is the *cross entropy*. To clarify things, we will try to illustrate how this works. In figure (5.4), we have an input set $\boldsymbol{X}_i$ where each unit is multiplied with a parameter from $\boldsymbol{w}$ and summarized. This corresponds to the inner product $\boldsymbol{X}_i\boldsymbol{w}$. Further, the sum (or the inner product) is *activated* by an *activation function*, which we above have assumed to be the sigmoid function. The output is then given by

$$a_i = f(\boldsymbol{X}_i\boldsymbol{w}). \tag{5.38}$$

where the bias node is included in the $\boldsymbol{X}_i$'s and the bias weights are included in the $\boldsymbol{w}$'s. The bias node is added in order to shift the activation function to the left or right, and works in the same way as a constant term in a function.

The output from the activation is used further in the cost function to calculate the cost. As for LASSO regression, the cost function is then minimized in an iterative scheme, where for example the gradient descent method gives the weight update

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta\boldsymbol{X}[\boldsymbol{y} - f(\boldsymbol{X}\boldsymbol{w}_{t-1})]. \tag{5.39}$$

where $\boldsymbol{X}$ is a matrix containing all the column vectors $\boldsymbol{X}_i$. This expression is extremely similar, not to say identical to the estimator update for ordinary least square presented in equation (5.26). The difference is that we now denote the parameters by $\boldsymbol{w}$ instead of $\boldsymbol{\theta}$ to prepare for the neural networks, but they are basically the same thing.

5.5 Neural networks

Now we know enough to dive into the field of artificial neural networks. Neural networks can give either continuous or discrete outputs and are therefore, competitors to both linear and

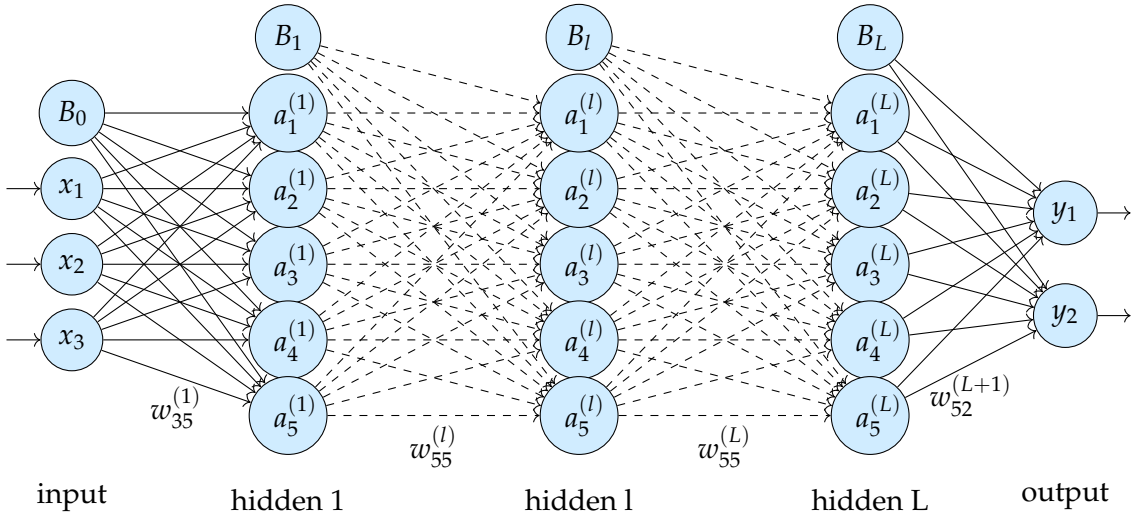


Figure 5.5: Neural network with 3 input units, L hidden layers with 5 hidden units each and two outputs. B_0, B_1, B_l and B_L are bias units for their respective layers, and the dashed lines indicate that it might be more layers between the two layers. We have labeled a few of the lines to relate them to the weights.

logistic regression. The great strength of neural networks is that one can add multiple *layers*, which potentially makes the model extremely flexible. According to **the universal approximation theorem**, a neural network with only one hidden layer with a finite number of units can approximate any continuous function [60]. However, often, multiple layers are used since those networks are in general known to be easier to train and work better for complex systems. Neural networks of more than one layer are called *deep* networks, and as more layers are added, the network gets *deeper*.

In figure (5.5), we have illustrated a deep neural network with an unspecified number of layers and five hidden units in each layer. It has some similarities with the logistic regression model in figure (5.4), but with multiple hidden layers and multiple outputs, this model is more complicated. We decided to drop the representation of the weights (apart from some selected labeled ones), but each line corresponds to a weight.

Without a hidden layer, we have seen that the update of weights is quite straight forward. For a neural network consisting of multiple layers, the question is: *How do we update the weights when we do not know the values of the hidden units?* This will be explained in section 5.5.3, where backward propagation, the most popular technique for weight update in a neural network, is discussed. Before that, we will generalize the forward phase presented in logistic regression.

5.5.1 Forward phase

In the previous section, we saw how the output is found for a single perceptron. For a neural network, the net output to the first layer is similar, and given by

$$z_j^{(1)} = \sum_{i=1}^{N_0} x_i w_{ij}^{(1)} = \boldsymbol{X}\boldsymbol{w}_j^{(1)}$$

where N_0 is the number of units in layer 0 (the input layer), $\boldsymbol{X}$ is the input vector which is assumed to be a row vector and $\boldsymbol{w}_j^{(1)}$ is the i'th column of the $\boldsymbol{w}$-matrix associated with the first layer. We have again assumed that the bias node is included in $\boldsymbol{X}$ and the first set of bias

weights are included in $\boldsymbol{w}^{(1)}$. The same applies for the other layers as well. If we let the activation function, $f(x)$, act on the net output, we get the real output given by

$$a_j^{(1)} = f(z_j^{(1)}) = f\Big(\sum_{i=1}^{N_0} x_i w_{ij}^{(1)}\Big).$$

This is then again the input to the next layer with N_1 units, so the output from the second layer is simply

$$a_j^{(2)} = f\Big(\sum_{i=1}^{N_1} a_i^{(1)} w_{ij}^{(2)}\Big).$$

For a neural network of multiple layers, the same procedure applies for all the layers and we can find a general formula for the output at a layer l. The net output to a node $z_j^{(l)}$ in layer l can be found to be

$$z_j^{(l)} = \sum_{i=1}^{N_{l-1}} a_i^{(l-1)} w_{ij}^{(l)} \tag{5.40}$$

where layer $l-1$ has N_{l-1} units and we need to be aware that $a_j^{(0)} = x_j$. After activation, the output is obviously found to be

$$a_j^{(l)} = f\Big(\sum_{i=1}^{N_{l-1}} a_i^{(l-1)} w_{ij}^{(l)}\Big) \tag{5.41}$$

which is the only formula needed for the forward phase. In practice, the operation is always implemented in a vectorized fashion, reading $\boldsymbol{a}^{(l)} = f(\boldsymbol{a}^{(l-1)}\boldsymbol{w}^{(l)})$. The activation function $f(x)$ is not explicitly defined, because it is often expedient having the chance to experiment with multiple activation functions.

5.5.2 Activation function

The task of the activation function is to define the output from a unit given an input. There are multiple reasons to do this, where the most important include filter the intensity of the output and make the output non-linear. Without an activation function, all of our layers would simply stack one affine transformation after another, resulting in a composition of transformations equal to such transformation. In other words, the deep networks would have lost their clout without the activation functions.

Yet, we have only discussed the sigmoid activation function, but there plenty of other activation functions available. The sigmoid function has lost its popularity, and is today superseded by the more advanced functions based on *rectified linear units* (ReLU). Some popular choices are the *leaky* ReLU and *exponential linear units* (ELU), which are linear for positive numbers. The pure linear activation function is still widely used, especially on the output layer. In figure (5.6), standard RELU, leaky RELU and ELU are plotted along with the sigmoid function.

5.5.3 Backward propagation

Backward propagation is the most robust technique for updating the weights in a neural network and is again based on the weight update presented for linear and logistic regression. The algorithm for this was presented in 1986, which made the deep neural networks able to solve relatively complicated problems for the first time [61]. To update the weights, one starts with

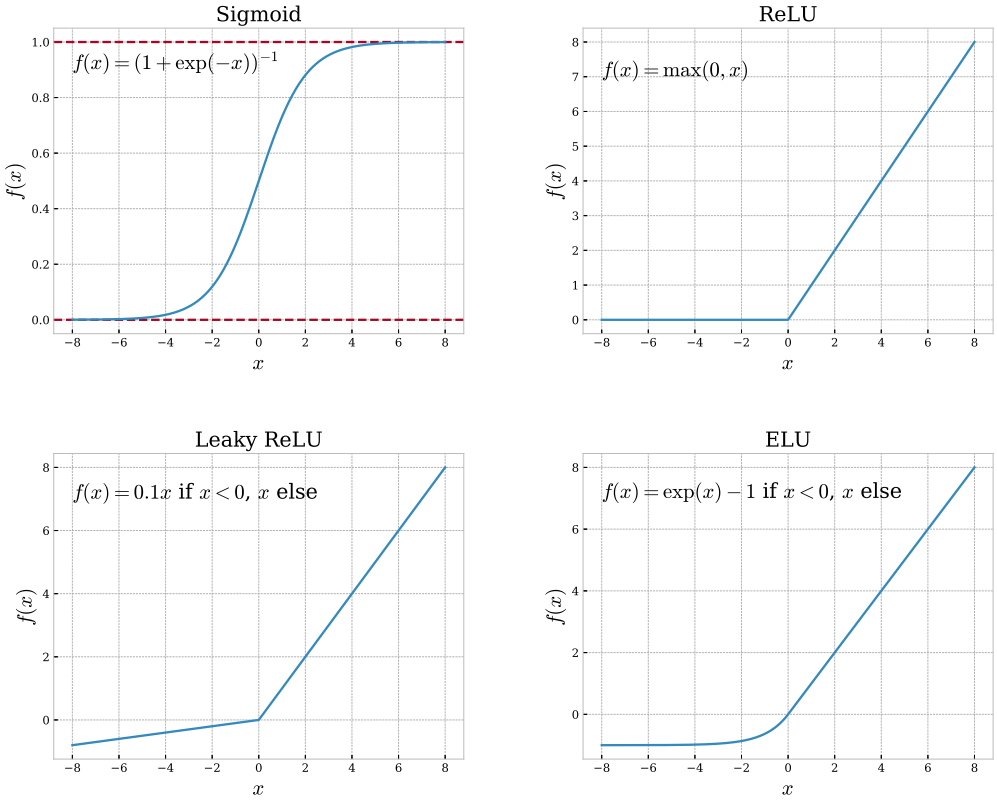


Figure 5.6: Some well-known activation functions. The sigmoid function stands out from the others since it maps between 0 and 1, and it is not linear for positive numbers.

the outputs and updates the weights layer-wise until one gets to the inputs, hence the backward propagation name.

As observed above, a node is dependent on all the units in the previous layers, and so are the weights. This means that the units are dependent on a large number of parameters, which makes the training scheme quite complex. Nevertheless, it is possible to generalize this to express the updating formulas on a relatively simple form, like the forward phase. From the linear and logistic regression, we know that we need the derivative of the cost function in order to implement the weight update regime. Again, we define the cost function as the mean square error,

$$\mathcal{C}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N_L}(y_i - a_i^{(L)})^2 \tag{5.42}$$

where we have $L+1$ layers (L is the last layer) and N_L output units. The derivative of this with respect to one of the weights between the $L-1$'th and L'th layer can be written as a sum using the chain rule

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_{jk}^{(L)}} = \frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}. \tag{5.43}$$

where $z_j^{(L)}$ and $a_j^{(L)}$ are found from equations (5.40) and (5.41) respectively. If we start with the

first factor, it can easily be obtained as

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial a_j^{(L)}} = -(y_j - a_j^{(L)}) \tag{5.44}$$

using the definition of the cost function. The second factor is the derivative of the activation function with respect to its argument, and is for the sigmoid function given by

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = a_j^{(L)}(1 - a_j^{(L)}). \tag{5.45}$$

Finally, the last factor is found from equation (5.40), and we obtain

$$\frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} = a_k^{(L-1)}. \tag{5.46}$$

Collecting all the factors, the update of the last set of weights can be found by

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_{jk}^{(L)}} = -(y_j - a_j^{(L)})a_j^{(L)}(1 - a_j^{(L)})a_k^{(L-1)}. \tag{5.47}$$

when the sigmoid function is used in the activation. In the next step, we can define

$$\delta_j^{(L)} = -a_j^{(L)}(1 - a_j^{(L)})(y_j - a_j^{(L)}) = f'(a_j^{(L)})\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial a_j^{(L)}} = \frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial z_j^{(L)}} \tag{5.48}$$

such that the weight update can be expressed on a neater form

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_{jk}^{(L)}} = \delta_j^{(L)} a_k^{(L-1)}. \tag{5.49}$$

For a general layer l, the derivative of the cost function with respect to a weight $w_{jk}^{(l)}$ is similar, and given by

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}. \tag{5.50}$$

Our goal is to find the general relation between layer l and $l+1$, and therefore we use the chain rule and sum over all the net outputs in layer $l+1$,

$$\delta_j^{(l)} = \frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial z_j^{(l)}} = \sum_k \frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}. \tag{5.51}$$

We now recognize that the first factor in the sum is just $\delta_k^{(l+1)}$ and the last factor can be found from equation (5.40). We obtain the final expression,

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)}) \tag{5.52}$$

where we use the expression of $\delta_j^{(L)}$ as our initial condition. As for several of the methods discussed above, a solution of the weight update does not exist in closed form and we need to rely on iterative optimization methods. Using gradient descent, a new set of weights $\boldsymbol{w}_t$ is found from

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta \frac{\partial \mathcal{C}(\boldsymbol{w}_{t-1})}{\partial \boldsymbol{w}_{t-1}}, \tag{5.53}$$

where the multi-dimensional differentiation is done element-wise. Other optimization methods will be discussed in the following section.

5.6 Optimization algorithms

We have above discussed the gradient descent optimization algorithm, which is among the most basic optimization methods available. That method is based on the gradient, which is the slope of the cost function. However, many other methods are also in need of the Hessian matrix, which gives the curvature of the cost function. We will barely scratch the surface of this field, limiting us to the gradient methods.

To have the method fresh in mind, we will start with reintroducing the gradient descent method before we move on to its stochastic brother. We will then take a look at how momentum can be added, and finally, we examine the stochastic and momentum-based ADAM optimizer. Given a cost function $\mathcal{C}(\boldsymbol{\theta})$, the gradient with respect to a parameter θ can be found from

$$\nabla_\theta \mathcal{C}(\boldsymbol{\theta}) \equiv \frac{\partial \mathcal{C}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{5.54}$$

where we henceforth use the short-hand notation with ∇_θ representing the multi-dimensional derivative $\partial/\partial\boldsymbol{\theta}$. Differentiating with respect to the vector implies that the operation shall be done element-wise.

5.6.1 Gradient descent

Perhaps the simplest and most intuitive method for finding the minimum is the gradient descent method, which reads

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \nabla_\theta \mathcal{C}(\boldsymbol{\theta}_{t-1}) \tag{5.55}$$

where $\boldsymbol{\theta}_t$ is the parameter vector at time step (iteration) t and η is the learning rate. $\nabla_\theta \mathcal{C}(\boldsymbol{\theta}_{t-1})$ is the gradient of the cost function with respect to all the parameters θ at time $t-1$.

The idea is to find the direction where the cost function $\mathcal{C}(\boldsymbol{\theta})$ has the steepest slope, and move in the direction which minimizes the cost function. For every time step, the cost function is thus minimized, and when the gradient approaches zero, the minimum is found. A possible, but basic, stop criterion is

$$\nabla_\theta \mathcal{C}(\boldsymbol{\theta}_t) < \epsilon. \tag{5.56}$$

where ϵ is a tolerance. More robust methods are based on comparing the value of the cost function for several past iterations. In cases where the cost function is not strictly decreasing, we will have both local and global minima. Often, it is hard to say whether we are stuck in a local or global minimum, and this is where the stochasticity enters the game.

5.6.2 Stochastic gradient descent

Stochastic gradient descent is closely related to the gradient descent method, but the method uses randomly selected batches to evaluate the gradients, hence the stochasticity. By introducing

this randomness, the parameters will not always be updated in order to minimize the energy, which makes us less likely to be stuck in a local minimum.

In practice, one splits the data set in n batches, and select one of them to be used in the parameter update. Our hope is that this batch is representative of the entire data set, such that the new parameters provide a lower cost function. If that is the case, we have reduced the cost of an iteration significantly, since we only need to care about a batch. We are not guaranteed that updating the parameters with respect to a batch gives a lower cost function, and when it is not, we need to run more batches in order to minimize the cost function. Since each iteration is faster than for standard gradient descent, this is acceptable. As long as the batch is slightly representative of the entire data set, the cost function will be minimized in the end. After each batch in the data set has had an opportunity to update the internal parameters, we say that we have gone through an *epoch*. Mathematically, the method can be expressed as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \nabla_\theta \mathcal{C}_i(\boldsymbol{\theta}_{t-1}) \tag{5.57}$$

where we use the i'th batch in the parameter update. Standard gradient descent is just a special case of this, where we only have one batch (i includes the whole data set). If we still get stuck in local minima after adding the stochasticity, it might be a good idea to add momentum as well.

5.6.3 Adding momentum

If we now recall what we learned in an introductory mechanics course, we might remember that momentum is a quantity that maintains the motion of a body. Imagine a ball that rolls down a steep hill, but then there is a local minimum that it needs to escape to keep rolling. If it has enough momentum, it will be able to escape.

The same idea lies behind the momentum used in optimization algorithms; the momentum will try to maintain the motion towards the global minimum, which makes the system less likely to be stuck in a local minimum. Momentum can be added to most optimization algorithms, also gradient descent and stochastic gradient descent. The way we do it is to save the direction we were moving during the previous iteration, and use it as a contribution to the next gradient update. A typical implementation of the first-order momentum applied on gradient descent looks like

$$\begin{aligned} \boldsymbol{m}_t &= \gamma \boldsymbol{m}_{t-1} + \eta \nabla_\theta \mathcal{C}_i(\boldsymbol{\theta}_{t-1}) \\ \boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t \end{aligned} \tag{5.58}$$

where γ is the momentum parameter, which is just another hyper-parameter usually initialized to a small number. $\boldsymbol{m}_t$ is the momentum vector and can be initialized as the zero vector, corresponding to no initial momentum.

The optimization algorithm can be modified further in unlimited ways. A common improvement is to add higher-order momentum; another is to make the learning-rate adaptive. We have implemented the most basic version of this, with monotonic adaptivity. Many algorithms, such as the conjugate gradient method, also make use of the Hessian as discussed in the introductory word to this section, but that is another level of complexity. We will end this section by setting up the algorithm of a stochastic gradient descent optimization with momentum and monotonic adaptivity. The algorithm is found in algorithm 1.

5.6.4 ADAM

ADAM is a first-order stochastic optimization method which is widely used in machine learning. It was discovered by Kingma & Ba [62], and published in a 2014 paper. The article has already more than 25000 citations! So what makes this method so popular?

Algorithm 1: Adaptive stochastic gradient descent with momentum. See sections (5.6.2-5.6.3) for details. Robust default settings for the hyper-parameters are $\eta = 0.001$, $\gamma = 0.01$ and $\lambda = 0.1$. All the operations are element-wise.

Parameter: η: Learning rate
Parameter: γ: Momentum parameter
Parameter: λ: Monotonic decay rate
Require : $\mathcal{C}(\boldsymbol{\theta})$: Cost function
Data : $\boldsymbol{\theta}_0$: Initial parameters
$\boldsymbol{m}_0 \leftarrow 0$ (Initialize momentum vector);
$t \leftarrow 0$ (Initialize time step);
while $\boldsymbol{\theta}_t$ *not converged* **do**
  $t \leftarrow t + 1$ (Increase time for each iteration);
  $\boldsymbol{g}_t \leftarrow \nabla_\theta \mathcal{C}_t(\boldsymbol{\theta}_{t-1})$ (Get gradients from a given batch at time t);
  $\boldsymbol{m}_t \leftarrow \gamma \boldsymbol{m}_{t-1} + \eta \cdot \boldsymbol{g}_t$ (Update first momentum estimate);
  $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t / \lambda^t$ (Update parameters with monotonic, adaptive step);
end
Result: Converged parameters $\boldsymbol{\theta}_t$.

The main reason why it is widely used, is obviously that it provides good performance. The fact that it only requires the gradient makes it efficient, and the way the momentum is implemented still makes it capable of handle a large number of parameters. The optimization algorithm can be expressed as a set of equations

$$
\begin{aligned}
\boldsymbol{g}_t &= \nabla_\theta \mathcal{C}_t(\boldsymbol{\theta}_{t-1}) \\
\boldsymbol{m}_t &= \gamma_1 \boldsymbol{m}_{t-1} + (1-\gamma_1)\boldsymbol{g}_t \\
\boldsymbol{v}_t &= \gamma_2 \boldsymbol{v}_{t-1} + (1-\gamma_2)\boldsymbol{g}_t^2 \\
\hat{\boldsymbol{m}}_t &= \boldsymbol{m}_t / (1-\gamma_1^t) \\
\hat{\boldsymbol{v}}_t &= \boldsymbol{v}_t / (1-\gamma_2^t) \\
\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \eta \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \boldsymbol{\varepsilon})
\end{aligned}
\tag{5.59}
$$

where $\boldsymbol{m}_t$ is the biased first momentum estimate of the parameter vector $\boldsymbol{\theta}$ and $\boldsymbol{v}_t$ is the biased second raw moment estimate. The momentum parameters need to be in the range $\gamma_1, \gamma_2 \in [0, 1)$, and are often set to values close to 1. This makes the optimization adaptive, as the time goes the factors $1 - \gamma_1^t$ and $1 - \gamma_2^t$ approach 1 from below. η is the learning rate, and should be a small number. Finally, the parameter ε is added to avoid division by zero. We can set up the algorithm in a way similar to the adaptive stochastic gradient descent algorithm presented above, which gives the algorithm 2.

Algorithm 2: ADAM optimizer. Robust default settings for the hyper-parameters are $\eta = 0.001$, $\gamma_1 = 0.1$ and $\gamma_2 = 0.001$. All the operations are element-wise, and for in-depth information see the original paper by Kingma & Ba [62].

Parameter: η: Learning rate
Parameter: $\gamma_1, \gamma_2 \in [0, 1)$: Momentum parameters
Parameter: ε: Division parameter
Require : $\mathcal{C}(\boldsymbol{\theta})$: Cost function
Data : $\boldsymbol{\theta}_0$: Initial parameters

$\boldsymbol{m}_0 \leftarrow 0$ (Initialize 1st momentum vector);
$\boldsymbol{v}_0 \leftarrow 0$ (Initialize 2st momentum vector);
$t \leftarrow 0$ (Initialize time step);
while $\boldsymbol{\theta}_t$ *not converged* **do**
  $t \leftarrow t + 1$ (Increase time for each iteration);
  $\boldsymbol{g}_t \leftarrow \nabla_\theta \mathcal{C}_t(\boldsymbol{\theta}_{t-1})$ (Get gradients from a given batch at time t);
  $\boldsymbol{m}_t \leftarrow \gamma_1 \boldsymbol{m}_{t-1} + (1 - \gamma_1) \cdot \boldsymbol{g}_t$ (Update first momentum estimate);
  $\boldsymbol{v}_t \leftarrow \gamma_2 \boldsymbol{v}_{t-1} + (1 - \gamma_2) \cdot \boldsymbol{g}_t^2$ (Update second raw momentum estimate);
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \gamma_1^t)$ (Bias-corrected first momentum estimate);
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \gamma_2^t)$ (Bias-corrected second momentum estimate) ;
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \cdot \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \boldsymbol{\varepsilon})$ (Update parameters) ;
end
Result: Converged parameters $\boldsymbol{\theta}_t$.

CHAPTER 6

Boltzmann Machines

> Available energy is the main object at stake in the struggle for existence and the evolution of the world.
>
> Ludwig Boltzmann, [63]

Boltzmann machines are generative, energy-based neural network models that fall under the category unsupervised learning. In unsupervised learning, unlike supervised learning which we discussed in chapter 5, the network is fed with an input data set only, i.e., we do not have any targets for supervising the network during the training. The task is then to find structures in the data, comparing data sets to each other and categorize the data sets concerning their similarities and differences (clustering).

They were invented by Ackley *et al.* [64] in 1985, where Hinton often is referred to as "The Godfather of Deep Learning"[1], and are based on the Boltzmann distribution, hence the name. Boltzmann machines with constrained connectivity, known as restricted Boltzmann machines (RBM), have found applications in classification [66], feature learning [67] and many-body quantum mechanics in the form of the Ising model [15], as discussed above. The RBM is well-suited for simulating the Ising model for two reasons: The model takes binary spins and the system energy of a Boltzmann machine takes the same form as the Ising energy. On the other hand, problems in electronic structure require wave functions that obey Fermi-Dirac statistics, which is a challenging task for machine learning. Our approach is to let the statistics be controlled by a Slater determinant, like in traditional variational Monte Carlo (VMC), and then let the single-particle functions be determined by Boltzmann machines. This is similar to the approach of Pfau *et al.* [16], who invented a so-called fermionic neural network consisting of a Slater determinant with the multi-electron functions controlled by a deep neural network.

In this chapter, we will focus exclusively on the Boltzmann machines, and move the detailed description of how they actually are used to after the discussion of quantum Monte Carlo methods in chapter 7.

In the previous chapter, we saw how the weights in supervised learning can be adjusted using the backward propagation algorithm, but it does not work when we do not have prior known targets. Instead, a set of probabilities controls the weights, and we let the log-likelihood function define the cost function. This is known as Bayesian statistics and is presented in the next section.

[1] Hinton's contribution to machine learning can hardly be overstated. He was co-author of the paper popularizing the backpropagation algorithm [61], supervisor of Krizhevsky who designed AlexNet [8] and the main author of the paper introducing the regularization technique *dropout* [65].

6.1 Statistical foundation

In this section, we will use Bayesian statistics to exploit the link between some data $\boldsymbol{x}$, called the *hypothesis*, and some other data $\boldsymbol{y}$ called the *evidence*. We will first do it in a general way before we link it to machine learning in the next section. Bayesian statistics appear in many fields of science, as it is a basic and often useful probability theory. It is based on Bayes' theorem, which gives rise to some marginal and conditional distributions. The expressions can either be set up in the continuous space or the discrete space, but here we will stick to the latter as we in practice will deal with discrete data.

We start expressing the joint probability distribution of measuring both $\boldsymbol{x}$ and $\boldsymbol{y}$ using the general relation,

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y}) = P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x}), \tag{6.1}$$

which basically states that the probability of observing $\boldsymbol{x}$ *and* $\boldsymbol{y}$ is just the probability of observing $\boldsymbol{x}$ multiplied with the probability of observing $\boldsymbol{y}$ given $\boldsymbol{x}$. $p(\boldsymbol{x}|\boldsymbol{y})$ is the conditional distribution of $\boldsymbol{x}$ and gives the probability of $\boldsymbol{x}$ given that $\boldsymbol{y}$ is true. The opposite applies for $P(\boldsymbol{y}|\boldsymbol{x})$. $P(\boldsymbol{x})$ and $P(\boldsymbol{y})$ are called the marginal probabilities for $\boldsymbol{x}$ and $\boldsymbol{y}$, and by reordering equation (6.1), we obtain Bayes' theorem

$$P(\boldsymbol{x}|\boldsymbol{y}) = \frac{P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x})}{P(\boldsymbol{y})}. \tag{6.2}$$

The marginal probability of $\boldsymbol{y}$, $P(\boldsymbol{y})$, is given by the sum over all the possible joint probabilities when $\boldsymbol{y}$ is fixed,

$$P(\boldsymbol{y}) = \sum_i P(x_i, \boldsymbol{y}) = \sum_i P(\boldsymbol{y}|x_i)P(x_i), \tag{6.3}$$

and from this we observe that Bayes' theorem gives us the *posterior* probability, $P(\boldsymbol{x}|\boldsymbol{y})$, given the *prior* probability, $P(\boldsymbol{x})$, and the *likelihood*, $P(\boldsymbol{y}|\boldsymbol{x})$, seen from

$$P(\boldsymbol{x}|\boldsymbol{y}) = \frac{P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x})}{\sum_i P(\boldsymbol{y}|x_i)P(x_i)}. \tag{6.4}$$

However, the summation gets extremely expensive quickly, and is intractable even for small systems. This was a big problem for a long time, but with the advent of powerful computers, algorithms like Markov chain Monte-Carlo can be used to estimate the posterior without knowing the *normalization constant*, $P(\boldsymbol{y})$. More about that in chapter 7.

In the section on supervised learning, the cost function was an important concept, and so is the case in unsupervised learning. However, how do we define a cost function when we do not have any targets? We find the answer by revealing the similarities between logistic regression and Bayesian statistics. In logistic regression, we find the probability that a system is in a particular state and define the cost function as the log-likelihood. We can do the same in unsupervised learning, and define the cost function as

$$\mathcal{C}(\boldsymbol{y}) = \ln \prod_{i=1}^{l} P(\boldsymbol{x}_i|\boldsymbol{y}) = \sum_{i=1}^{l} \ln P(\boldsymbol{x}_i|\boldsymbol{y}), \tag{6.5}$$

which is the log-likelihood. Maximizing the likelihood is the same as maximizing the log-likelihood, which again corresponds to minimizing the distance between the unknown distribution Q underlying $\boldsymbol{x}$ and the distribution P of the Markov random field $\boldsymbol{y}$. This distance is expressed in terms of the Kullback-Leibler divergence (KL divergence), which for a finite state space Ω is given by

$$\mathrm{KL}(Q||P) = \sum_{\boldsymbol{x}\in\Omega} Q(\boldsymbol{x}) \frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})}. \tag{6.6}$$

The KL divergence is a measure of the difference between two *probability density functions* (PDFs), and is zero for two identical PDFs. The divergence is often called a distance, but that is an unsatisfying description as it is non-symmetric ($\mathrm{KL}(Q||P)) \neq \mathrm{KL}(P||Q)$) in general.

To proceed further, we will introduce latent variables in form of hidden units. Suppose we want to model an m-dimensional unknown probability distribution Q. Typically, not all the variables $\boldsymbol{s}$ are observed components, they can also be latent variables. If we split $\boldsymbol{s}$ into *visible* variables $\boldsymbol{x}$ and hidden variables $\boldsymbol{h}$, and under the assumption that $\boldsymbol{x}$ and $\boldsymbol{h}$ are variables in an energy function $E(\boldsymbol{x}, \boldsymbol{h})$, we can express the joint probability as the Boltzmann distribution

$$P(\boldsymbol{x}, \boldsymbol{h}) = \frac{\exp(-E(\boldsymbol{x}, \boldsymbol{h}))}{Z} \tag{6.7}$$

where Z is the partition function, which is the sum of the probability of all possible states, which was already introduced in equation (5.30). We have ignored the factor $k_B T$ by setting it to 1. Where the visible units correspond to components of an observation, the hidden units introduce the system to more degrees of freedom. This allows us to describe complex distributions over the visible variables by means of simple conditional distributions [68]. Those conditional distributions will be described later, but let us first take a look at the marginal distributions.

6.1.1 Marginal distributions

We have already used the term marginal distribution, which means that we get rid of a set of variables by integrating the joint probability over all of them. The marginal probability of $\boldsymbol{x}$ is given by

$$P(\boldsymbol{x}) = \sum_{\boldsymbol{h}} P(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{x}, \boldsymbol{h})). \tag{6.8}$$

The sum over the $\boldsymbol{h}$ vector is just a short-hand notation where we sum over all the possible values of all the variables in $\boldsymbol{h}$. Further, the marginal probability of $\boldsymbol{h}$ is expressed similarly, with

$$P(\boldsymbol{h}) = \sum_{\boldsymbol{x}} P(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{x}} \exp(-E(\boldsymbol{x}, \boldsymbol{h})). \tag{6.9}$$

$P(\boldsymbol{x})$ is important as it gives the probability of a particular set of visible units $\boldsymbol{x}$, while $P(\boldsymbol{h})$ will not be used in the same scope in this work.

6.1.2 Conditional distributions

The conditional distributions can be found from Bayes' theorem, and read

$$P(\boldsymbol{h}|\boldsymbol{x}) = \frac{P(\boldsymbol{x}, \boldsymbol{h})}{P(\boldsymbol{x})} = \frac{\exp(-E(\boldsymbol{x}, \boldsymbol{h}))}{\sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{x}, \boldsymbol{h}))} \tag{6.10}$$

and

$$P(\boldsymbol{x}|\boldsymbol{h}) = \frac{P(\boldsymbol{x}, \boldsymbol{h})}{P(\boldsymbol{h})} = \frac{\exp(-E(\boldsymbol{x}, \boldsymbol{h}))}{\sum_{\boldsymbol{x}} \exp(-E(\boldsymbol{x}, \boldsymbol{h}))}. \tag{6.11}$$

The conditional probabilities are especially important in Gibbs sampling, where we want to update the visible units $\boldsymbol{x}$ given the hidden units $\boldsymbol{h}$ and *vice versa*.

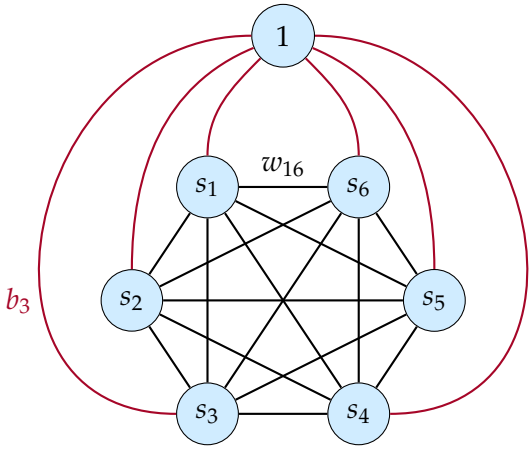


Figure 6.1: Unrestricted Boltzmann machine. Black lines are connections between all the units, where for instance the line between s_1 and s_6 is related to the weight w_{16}. The blue lines are related to the bias weights, and for instance, the line going from the bias unit to s_3 is related to b_3.

6.1.3 Maximum log-likelihood estimate

Now suppose that the energy function also is a function of some parameters $\boldsymbol{\theta}$. We have already expressed the log-likelihood function,

$$\ln P(\boldsymbol{x}|\boldsymbol{\theta}) = \ln\left[\frac{1}{Z}\sum_{\boldsymbol{h}}\exp(-E(\boldsymbol{x},\boldsymbol{h}))\right] = \ln\sum_{\boldsymbol{h}}\exp(-E(\boldsymbol{x},\boldsymbol{h})) - \ln\sum_{\boldsymbol{x},\boldsymbol{h}}\exp(-E(\boldsymbol{x},\boldsymbol{h})) \tag{6.12}$$

and by maximizing this we find the maximum log-likelihood estimate. This estimate is important in neural networks since we always seek to maximize the likelihood in the training process. The function is maximized when

$$\begin{aligned}\frac{\partial \ln P(\boldsymbol{x}|\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} &= \frac{\partial}{\partial\boldsymbol{\theta}}\left(\ln\sum_{\boldsymbol{h}}\exp(-E(\boldsymbol{x},\boldsymbol{h}))\right) - \frac{\partial}{\partial\boldsymbol{\theta}}\left(\ln\sum_{\boldsymbol{x},\boldsymbol{h}}\exp(-E(\boldsymbol{x},\boldsymbol{h}))\right)\\ &= -\sum_{\boldsymbol{h}}P(\boldsymbol{h}|\boldsymbol{x})\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial\boldsymbol{\theta}} + \sum_{\boldsymbol{x},\boldsymbol{h}}P(\boldsymbol{x},\boldsymbol{h})\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial\boldsymbol{\theta}} = 0.\end{aligned} \tag{6.13}$$

Similarly to the neural networks presented in chapter 5, we cannot find a closed-form expression for this, and we need to solve it iteratively.

6.2 Unrestricted Boltzmann machines

Unrestricted Boltzmann machines, or merely Boltzmann machines, are energy-based, generative neural networks based on the more primitive Hopfield network. They consist of a set of units, and similarly to the feed-forward neural networks presented in section 5.5, a weight matrix is connecting the units. However, in a standard unrestricted Boltzmann machine, we only have one layer where all the units connect to all other units, and a bias unit is commonly added to work as a constant term. In figure (6.1), we illustrate a plain Boltzmann machine consisting of $N = 6$ units and one bias unit.

By multiplying each unit with all the other units and the weight connecting them, one obtains the system energy, which should not be confused with the physical energy of a quantum state.

For the simplest case, the energy reads

$$E(\boldsymbol{s}) = -\sum_{i=1}^{N} s_i b_i - \sum_{i=1}^{N}\sum_{j=i}^{N} s_i w_{ij} s_j, \tag{6.14}$$

where $\boldsymbol{s}$ are the units and w_{ij} is the weight connecting the units s_i and s_j. The bias unit is fixed to 1, as always, and the weight between the bias unit and the unit s_i is denoted by b_i. In its most simple form, the units can only take binary values, and we, therefore, call it a binary-unit Boltzmann machine. The energy formula is then identical to the system energy of Hopfield networks, but what distinguishes a Boltzmann machine from a Hopfield network is that the units are *stochastic*. By stochastic, we mean that their values are randomly determined, introducing some randomness to the system. Also, the energy of an Ising model takes the same form as equation (6.14). Other architectures are also available, and for the restricted Boltzmann machine we will look at the Gaussian-binary unit model.

The reader has might already foreseen the next step, which is to use the Boltzmann distribution to define the probability of finding the system in a particular state $E(\boldsymbol{s}; \boldsymbol{w}, \boldsymbol{b})$, as discussed in the previous section. The probability distribution function (PDF) is then given by

$$P(\boldsymbol{s}) = \frac{1}{Z}\exp(-E(\boldsymbol{s})), \tag{6.15}$$

where Z again is the partition function. The PDF contains weights, which can be adjusted to change the distribution. In a supervised scheme, one can update the parameters in order to minimize the Kullback-Leibler divergence to a prior known distribution and in that manner reproduce the known distribution. In unsupervised learning, we cannot do this, but hopefully, we can obtain a reasonable distribution by minimizing the system energy.

A Boltzmann machine is also a Markov random field, as the stochastic processes satisfy the Markov property. Loosely speaking, this means that all the probabilities of going from one state to another are known, making it possible to predict the future of the process based solely on its present state. The property is also determined by "memorylessness", meaning that the next state of the system depends only on the current state and not on the sequence of events that preceded it [68]. The Markov chain is an essential part of the sampling methods that will we discuss in chapter 7.

6.3 Restricted Boltzmann machines

When there is an unrestricted guy, a restricted guy must exist as well. What the term restricted means in this context, is that we ignore all the connections between units in the same layer, and keep only the inter-layer ones. In a restricted Boltzmann machine (RBM), only the units in the first layer are the observable, while the units in the next layer are latent or hidden. In the same manner as in equation (6.14), we can look at the linear case and multiply each unit with the corresponding weight, but now we need to distinguish between a visible unit x_i and a hidden unit h_j. For the same reason, we divide all the bias weights into a group connected to the visible units, a_i, and a group connected to the hidden units, b_j. The system energy then reads

$$E(\boldsymbol{x}, \boldsymbol{h}) = -\sum_{i=1}^{F} x_i a_i - \sum_{j=1}^{H} h_j b_j - \sum_{i=1}^{F}\sum_{j=i}^{H} x_i w_{ij} h_j, \tag{6.16}$$

which is called a binary-binary unit or Bernoulli-Bernoulli unit RBM. H is the number of hidden units, and F is the number of visible units, later known as the degrees of freedom. In figure (6.2), a restricted Boltzmann machine with three visible units and three hidden units is illustrated.

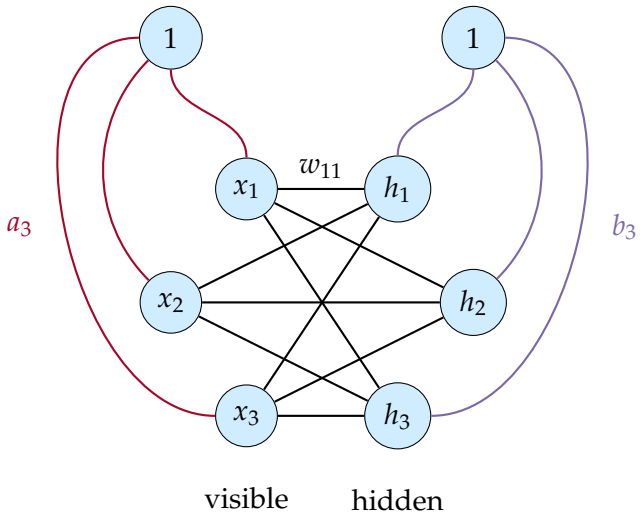


Figure 6.2: Restricted Boltzmann machine. Black lines represent the inter-layer connections, where for instance the line between x_1 and h_1 is related to the weight w_{11}. The red lines are related to the input bias weights, and for instance, the line going from the bias unit to x_3 is called a_3. Similarly, the blue lines are connections between the hidden units and the bias, and for instance, the line going from the bias unit to h_3 is denoted by b_3.

6.3.1 Gaussian-binary units

So far, we have discussed the linear models only, but as for feed-forward neural networks, we need non-linear models to solve non-linear problems. A natural next step is the model with Gaussian-binary units, which has a Gaussian mapping between the visible unit bias and the visible units and possibly also between the two layers. The energy expression of an architecture with Gaussian mapping between the visible units and the bias only takes the form

$$E(\boldsymbol{x}, \boldsymbol{h}) = \sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} h_j b_j - \sum_{i=1}^{F} \sum_{j=i}^{H} \frac{x_i w_{ij} h_j}{\sigma_i^2}, \tag{6.17}$$

where σ_i is the width of the Gaussian distribution, which can take an arbitrary number. Inserting the energy expression into equation (6.15), we obtain the Gaussian-binary joint probability distribution,

$$P(\boldsymbol{x}, \boldsymbol{h}) \propto \exp\Big(- \sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2} \Big) \prod_{j=1}^{H} \exp\Big(h_j b_j + \sum_{i=1}^{F} \frac{h_j w_{ij} x_i}{\sigma^2} \Big), \tag{6.18}$$

where the first factor (the exponential function) is actually the definition of a Gaussian function and the product has a complexity proportional to the number of hidden nodes. Generative sampling algorithms, as Gibbs sampling, use this distribution directly, while other sampling tools, like Metropolis sampling, need the marginal distribution. To find the marginal distribution of the visible units, we just need to take the sum over $h = 0$ and $h = 1$ as the hidden units are binary. The final expression is given by

$$P(\boldsymbol{x}) \propto \exp\Big(- \sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2} \Big) \prod_{j=1}^{H} \Big(1 + \exp\Big(b_j + \sum_{i=1}^{F} \frac{w_{ij} x_i}{\sigma^2} \Big) \Big), \tag{6.19}$$

were the transition from equation (6.17) is shown thoroughly in appendix C. Since the visible units take continuous values, we need to integrate to find the marginal distribution of the hidden

units, but since we never will use that distribution in this work, we will ignore the marginal distributions of the hidden units.

The conditional distributions are important in Gibbs sampling as they are used to determine the value of the hidden and visible nodes and update the weights. The distribution of h given x is used to update the hidden units and reads

$$P(\boldsymbol{h}|\boldsymbol{x}) = \frac{P(\boldsymbol{x},\boldsymbol{h})}{P(\boldsymbol{x})} = \prod_{j=1}^{H} \frac{\exp\left(h_j b_j + \sum_{i=1}^{F} x_i w_{ij} h_j/\sigma^2\right)}{1+\exp\left(b_j + \sum_{i=1}^{F} x_i w_{ij}/\sigma^2\right)}. \tag{6.20}$$

Similarly, the conditional distribution of x given h is used to update the visible units and turns out to be just the normal distribution,

$$P(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{a} + \boldsymbol{w}^T\boldsymbol{h}, \sigma^2), \tag{6.21}$$

with the width of the Gaussian distribution determining the variance. Note that the mean is $\boldsymbol{\mu} = \boldsymbol{a} + \boldsymbol{w}^T\boldsymbol{h}$, which is the vector obtained when going backwards in the restricted Boltzmann machine (multiplying the hidden units with the weights).

In Metropolis sampling, we only use the marginal distribution of the visible units, so the weights to the hidden units are additional variational parameters. For completeness reasons, we will discuss the Gibbs sampling, but we will in practice stick to the Metropolis sampling. More about the different sampling tools can be found in chapter 7. We also need the gradient of the log-likelihood function in order to train the network. The likelihood function is defined as the probability of $\boldsymbol{x}$ given a set of parameters $\boldsymbol{\theta}$, which relate to our problem as $P(\boldsymbol{x}|\boldsymbol{a},\boldsymbol{b},\boldsymbol{w})$. We therefore get three maximum log-likelihood estimates,

$$\begin{aligned}
\frac{\partial \ln P(\boldsymbol{x}|\boldsymbol{a},\boldsymbol{b},\boldsymbol{w})}{\partial \boldsymbol{a}} &= \frac{\boldsymbol{x}-\boldsymbol{a}}{\sigma^2} \\
\frac{\partial \ln P(\boldsymbol{x}|\boldsymbol{a},\boldsymbol{b},\boldsymbol{w})}{\partial \boldsymbol{b}} &= \boldsymbol{n} \\
\frac{\partial \ln P(\boldsymbol{x}|\boldsymbol{a},\boldsymbol{b},\boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\boldsymbol{x}\boldsymbol{n}^T}{\sigma^2}
\end{aligned} \tag{6.22}$$

where we have defined a vector $\boldsymbol{n}$ as the (element-wise) logistic function

$$\boldsymbol{n}(\boldsymbol{v}) \equiv \frac{1}{1+\exp(-\boldsymbol{v})} \tag{6.23}$$

with $\boldsymbol{v}$ as the vector containing all the elements in the last exponent in equation (6.19),

$$\boldsymbol{v} \equiv \boldsymbol{b} + \frac{\boldsymbol{w}^T\boldsymbol{x}}{\sigma^2}. \tag{6.24}$$

We decided to set up the vectorized expressions as that is what we will use in practice. In addition to $\boldsymbol{n}$, we will later introduce the its counterpart, $\boldsymbol{p}(v) = \boldsymbol{n}(-v)$, and the names make sense as $\boldsymbol{n}$ has a negative expression in the exponent, while $\boldsymbol{p}$ has a positive expression in the exponent. The expressions in equation (6.22) will later be used to maximize the likelihood with respect to the respective set of parameters.

6.4 Partly restricted Boltzmann machines

One can also imagine a partly restricted architecture, where we have intern connections between the visible units, but not the hidden units. This is what we have decided to call a partly restricted

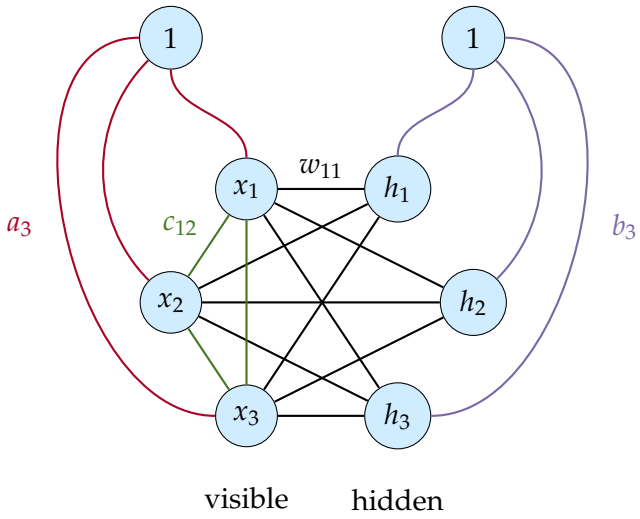


Figure 6.3: Partly restricted Boltzmann machine. Black lines represent inter-layer connections, where for instance the line between x_1 and h_1 is related to the weight w_{11}. The red lines are related to the input bias weights, and for instance, the line going from the bias unit to x_3 is related to a_3. Similarly, the blue lines are related to the hidden units bias weights, and for instance, the line going from the bias unit to h_3 is related to b_3. Finally, the purple lines are the intra-layer connections related to the intra-layer weights. The weight between unit x_1 and x_2 is called c_{12}.

Boltzmann machine, and are very similar to restricted Boltzmann machines but with another level of flexibility. A such neural network with three visible units and three hidden units is illustrated in figure (6.3). Compared to a standard restricted Boltzmann machine, we get an extra term in the energy expression where the visible units are connected. It is easy to see that the expression should be

$$E(\boldsymbol{x}, \boldsymbol{h}) = \sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{i=1}^{F}\sum_{j>i}^{F} x_i c_{ij} x_j - \sum_{j=1}^{H} h_j b_j - \sum_{i=1}^{F}\sum_{j=i}^{H} \frac{x_i w_{ij} h_j}{\sigma_i^2} \tag{6.25}$$

with c_{ij} as the weights between the visible units. In the rest of this project, we are interested in the marginal distribution of the visible units only, which becomes

$$P(\boldsymbol{x}) \propto \exp\Big(-\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2} + \sum_{i=1}^{F}\sum_{j>i}^{F} x_i c_{ij} x_j\Big) \prod_{j=1}^{H} \Big(1 + \exp\Big(b_j + \sum_{i=1}^{F} \frac{w_{ij} x_i}{\sigma^2}\Big)\Big) \tag{6.26}$$

by again using the approach detailed in appendix C. In chapter 11, we utilize that this marginal distribution can be split in a Gaussian part, a *partly restricted* part and a product part. Then we see that the expression of the gradient of the log-likelihood function becomes the same with respect to $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{w}$ compared to the restricted Boltzmann machine, which means that we only need to calculate the expression of the gradient of the log-likelihood with respect to $\boldsymbol{c}$. This is given by the outer product

$$\frac{\partial \ln P(\boldsymbol{x}|\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{w})}{\partial \boldsymbol{c}} = \boldsymbol{x}\boldsymbol{x}^T, \tag{6.27}$$

which also is written on a vectorized form.

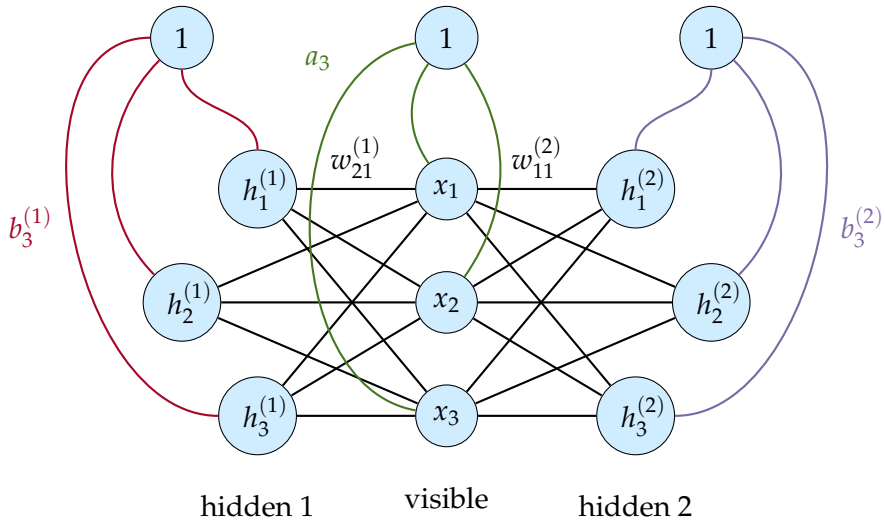


Figure 6.4: Deep restricted Boltzmann machine. Black lines represent the inter-layer connections, where for instance the line between $x_1^{(1)}$ and $h_1^{(1)}$ is related to the weight $w_{11}^{(1)}$ and similar for the other layer. The purple lines are related to the input bias weights, and for instance, the line going from the bias unit to x_3 is related to a_3. The red lines are related to the left-hand side hidden units bias weights, the line going from the bias unit to $h_3^{(1)}$ is related to $b_3^{(1)}$. The same applies for the right-hand side bias weights.

6.5 Deep Boltzmann machines

We can also construct deep Boltzmann machines, also known as deep belief networks, where we stack single-layer Boltzmann machines. There are many ways to construct those networks, where the number of layers, unit types, number of units, and the degree of restriction can be chosen as the programmer wants. The number of combinations is endless, but in order to make use of the dept, all the layers should have different configurations. Otherwise, the deep network can be reduced to a shallower network. In figure (6.4), a restricted Boltzmann machine of two hidden layers is illustrated. We have chosen three hidden units in each layer and three visible units. It should be trivial to imagine how the network can be expanded to more layers. As the main focus so far has been on restricted Boltzmann machines, also the deep networks will be assumed to be restricted, although both partly restricted and unrestricted can be constructed. The system energy of a deep restricted Boltzmann machine of L layers can be expressed as

$$E(\boldsymbol{x},\boldsymbol{h}) = \sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma_i^2} - \sum_{l=1}^{L}\sum_{j=1}^{H_L} h_j^{(l)} b_j^{(l)} - \sum_{l=1}^{L}\sum_{i=1}^{F}\sum_{j=i}^{H_L}\frac{x_i w_{ij}^{(l)} h_j^{(l)}}{\sigma_i^2} \tag{6.28}$$

where H_L is the number of hidden units in layer L. The marginal probability distribution of the visible units read

$$P(\boldsymbol{x}) \propto \exp\Big(-\sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma^2}\Big)\prod_{l=1}^{L}\prod_{j=1}^{H_L}\Big(1+\exp\Big(b_j^{(l)}+\sum_{i=1}^{F}\frac{w_{ij}^{(l)}x_i}{\sigma^2}\Big)\Big). \tag{6.29}$$

which again can be obtained from the general expressions in appendix C.

Part III

Methods

CHAPTER 7

Quantum Monte Carlo Methods

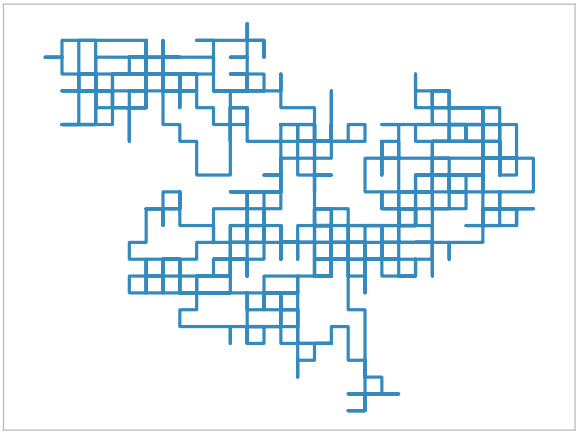


Figure 7.1: Random walker on a two-dimensional grid, 1000 moves.

Quantum Monte Carlo methods (QMC) is a bunch of *ab initio* methods that attempt to solve the Schrödinger equation using stochastic Monte Carlo integration. *Ab initio* reads "from first principles", which implies that the methods constitute a fundamental approach to the problem. They all seek to evaluate the multi-dimensional integral arising from the Schrödinger equation,

$$E_0 = \langle \Psi_0 | \hat{\mathcal{H}} | \Psi_0 \rangle = \frac{\int d\boldsymbol{R} \Psi_0(\boldsymbol{R})^* \hat{\mathcal{H}} \Psi_0(\boldsymbol{R})}{\int d\boldsymbol{R} \Psi_0(\boldsymbol{R})^* \Psi_0(\boldsymbol{R})}, \tag{7.1}$$

which provides the ground state energy expectation value for the exact ground state wave function $\Psi_0(\boldsymbol{R})$ with $\boldsymbol{R} = \{(\boldsymbol{r}_1, \sigma_1), (\boldsymbol{r}_2, \sigma_2), \ldots, (\boldsymbol{r}_N, \sigma_N)\}$ as the collective coordinates and spin of the N particles. As aforementioned, this integral is analytically infeasible for more or less all interesting systems, rising the need of numerical methods like QMC. As we will stick to ground state calculations, the wave function $\Psi(\boldsymbol{R})$ implies the many-body ground state wave function from this point.

In Monte Carlo integration, we use random numbers to evaluate integrals numerically. Typically, we want to estimate an expectation value $\langle \hat{O} \rangle$ by approximating the definition integral with a dense sum,

$$\langle \hat{O} \rangle \equiv \int_{-\infty}^{\infty} d\boldsymbol{x} P(\boldsymbol{x}) \hat{O}(\boldsymbol{x}) \approx \frac{1}{M} \sum_{i=1}^{M} \hat{O}(\boldsymbol{x}_i) \tag{7.2}$$

where M is the number of *Monte Carlo cycles* and the coordinates $\boldsymbol{x}_i$ are drawn randomly from the probability density function $P(\boldsymbol{x})$. A great advantage of the QMC methods, is that we obtain approximative ground state wave functions when solving equation (7.1), which by the fourth postulate of quantum mechanics allows estimations of the ground state expectation values associated with other operators as well.

Two widely popular QMC methods are the variational Monte Carlo method (VMC) and the diffusion Monte Carlo method (DMC), where the former is arguably the simplest of all the QMC methods. It attempts to solve the integrals in equation (7.1) directly by varying parameters, with the support of the variational principle presented in section 2.4. This makes VMC a comparably computationally cheap method, but the performance is usually not in the league of the best methods. DMC, on the other hand, is computationally expensive, but is also potentially numerically exact, making it a preferred method when high accuracy is needed. At first glance, it might seems like a tradeoff where VMC is used when computational time is more important than the accuracy and DMC is used when the opposite is true. However, DMC requires a wave function input which is close to the exact wave function, forcing us first to run VMC to obtain this wave function before the DMC machinery can be started.

As VMC is our main focus in this work, it will be explained thoroughly in this chapter. After that, we will briefly explain the idea behind the DMC method, but since this method is not implemented, it will not be our main priority. To reveal the uncertainty of our results, we will also discuss some methods to estimate the errors, in particular, the blocking method.

7.1 Variational Monte Carlo

The variational Monte Carlo method (hereafter the VMC method) is today widely used when it comes to the study of ground state properties of quantum many-body systems. It makes use of Markov chain Monte Carlo methods, often abbreviated MCMC, where the particles are assumed to be moving in Markov chains controlled by Monte Carlo simulations. Going back to the variational principle in equation (2.20), one observes that by choosing an approved wave function, one gets an energy larger or equal to the ground state energy.

Before we present the mathematical framework of the method, we will restate the two big challenges in many-body physics, mentioned in the introduction:

1. The correct many-body wave function is generally unavailable.

2. The many-body energy expectation value is analytically infeasible for most systems.

In this section, we will look at how the VMC method approaches those challenges. We start with discussing the trial wave function, and write up our trial wave function ansatz. Then, we define the local energy and explain how it is used to solve the energy integral using Monte Carlo integration. In the end, we will mention some common extensions to the VMC method.

7.1.1 The trial wave function

The trial wave function was mentioned in section 3.2.4, but what is actually the trial wave function? In VMC, we start from a wave function ansatz, which is our ground state wave function guess. This function is equipped with variational parameters, and in order to estimate the wave function accurately, the trial wave function needs to be able to approach the correct wave function as we vary the parameters. However, as the many-body wave function is NP-hard to calculate [5], we will only be able to approximate it and for that reason the wave function is by many considered as the root of all evil in many-body physics. For fermionic systems, the standard trial

wave function used in VMC, given a set of variational parameters $\boldsymbol{\theta}$, is the Slater-Jastrow wave function,

$$\Psi_T(\boldsymbol{R};\boldsymbol{\theta}) = |\hat{D}(\boldsymbol{R};\boldsymbol{\theta})|J(\boldsymbol{R};\boldsymbol{\theta}) \tag{7.3}$$

where $|\hat{D}(\boldsymbol{R};\boldsymbol{\theta})|$ is a Slater determinant used to bake in the anti-symmetry discussed in section 3.2.1 and $J(\boldsymbol{R};\boldsymbol{\theta})$ is a Jastrow factor used to model the electron-electron correlations. Recall that the general Slater determinant has the form

$$|\hat{D}(\boldsymbol{R};\boldsymbol{\theta})| = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\boldsymbol{r}_1,\sigma_1;\boldsymbol{\theta}) & \psi_2(\boldsymbol{r}_1,\sigma_1;\boldsymbol{\theta}) & \dots & \psi_N(\boldsymbol{r}_1,\sigma_1;\boldsymbol{\theta}) \\ \psi_1(\boldsymbol{r}_2,\sigma_2;\boldsymbol{\theta}) & \psi_2(\boldsymbol{r}_2,\sigma_2;\boldsymbol{\theta}) & \dots & \psi_N(\boldsymbol{r}_2,\sigma_2;\boldsymbol{\theta}) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\boldsymbol{r}_N,\sigma_N;\boldsymbol{\theta}) & \psi_2(\boldsymbol{r}_N,\sigma_N;\boldsymbol{\theta}) & \dots & \psi_N(\boldsymbol{r}_N,\sigma_N;\boldsymbol{\theta}) \end{vmatrix} \tag{7.4}$$

where the single-particle function $\psi(\boldsymbol{r},\sigma;\boldsymbol{\theta})$ can be decomposed in a spatial part, $\phi(\boldsymbol{r};\boldsymbol{\theta})$ and a spin part, $\xi(\sigma)$,

$$\psi(\boldsymbol{r},\sigma;\boldsymbol{\theta}) = \phi(\boldsymbol{r};\boldsymbol{\theta}) \otimes \xi(\sigma). \tag{7.5}$$

$\otimes$ denotes the tensor product, and the factorization is only possible if the orbital and spin angular momenta of the particle are separable in the Hamiltonian underlying the system's dynamics. We will skip the tensor product notation in the following, but it is implicit that it is there. We also drop the $\boldsymbol{\theta}$ argument of the single-particle functions. As we in the ground state have double degeneracy, the spatial part will be the same for pairwise spin-up and spin-down particles, and we arrange them as

$$\psi_j(\boldsymbol{r}_i,\sigma_i) = \begin{cases} \phi_j(\boldsymbol{r}_i)\xi_\uparrow(\sigma_i) & \text{if } j < N_\uparrow \\ \phi_j(\boldsymbol{r}_i)\xi_\downarrow(\sigma_i) & \text{if } j \geq N_\uparrow. \end{cases} \tag{7.6}$$

Since the first $N_\uparrow$ particles have spin up, $\sigma_i = \uparrow \quad \forall \quad i \in \{1,2,...,N_\uparrow\}$, and the remaining have spin down, we can now set up the Slater determinant in equation (7.4) where each single-particle function is split in a spatial part and a spin part,

$$|\hat{D}(\boldsymbol{R})| \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1)\xi_\uparrow(\uparrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_1)\xi_\uparrow(\uparrow) & \phi_1(\boldsymbol{r}_1)\xi_\downarrow(\uparrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_1)\xi_\downarrow(\uparrow) \\ \vdots & & \vdots & \vdots & & \vdots \\ \phi_1(\boldsymbol{r}_{N_\uparrow})\xi_\uparrow(\uparrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_{N_\uparrow})\xi_\uparrow(\uparrow) & \phi_1(\boldsymbol{r}_{N_\uparrow})\xi_\downarrow(\uparrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_{N_\uparrow})\xi_\downarrow(\uparrow) \\ \phi_1(\boldsymbol{r}_{N_\uparrow+1})\xi_\uparrow(\downarrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_{N_\uparrow+1})\xi_\uparrow(\downarrow) & \phi_1(\boldsymbol{r}_{N_\uparrow+1})\xi_\downarrow(\downarrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_{N_\uparrow+1})\xi_\downarrow(\downarrow) \\ \vdots & & \vdots & \vdots & & \vdots \\ \phi_1(\boldsymbol{r}_N)\xi_\uparrow(\downarrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_N)\xi_\uparrow(\downarrow) & \phi_1(\boldsymbol{r}_N)\xi_\downarrow(\downarrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_N)\xi_\downarrow(\downarrow) \end{vmatrix}.$$

We observe that the the spin-up particles sometimes occupy spin-down states, which they are not allowed to. Therefore, half of the elements become zero and the determinant can be further expressed as

$$|\hat{D}(\boldsymbol{R})| \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1)\xi_\uparrow(\uparrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_1)\xi_\uparrow(\uparrow) & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ \phi_1(\boldsymbol{r}_{N_\uparrow})\xi_\uparrow(\uparrow) & \dots & \phi_{N_\uparrow}(\boldsymbol{r}_{N_\uparrow})\xi_\uparrow(\uparrow) & 0 & \dots & 0 \\ 0 & \dots & 0 & \phi_1(\boldsymbol{r}_{N_\uparrow+1})\xi_\downarrow(\downarrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_{N_\uparrow+1})\xi_\downarrow(\downarrow) \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & \phi_1(\boldsymbol{r}_N)\xi_\downarrow(\downarrow) & \dots & \phi_{N_\downarrow}(\boldsymbol{r}_N)\xi_\downarrow(\downarrow) \end{vmatrix}$$

where the Slater matrix now is block diagonal! For a general block diagonal matrix, the determinant is given by the product of the determinant of each block

$$|\hat{D}(\boldsymbol{R})| \propto |\hat{D}_\uparrow(\boldsymbol{R}_\uparrow)| \cdot |\hat{D}_\downarrow(\boldsymbol{R}_\downarrow)| \tag{7.7}$$

which can be seen by writing the total matrix as a product over all the block diagonal matrix. $\hat{D}_\uparrow$ is the matrix containing all spin-up states with collective coordinates $\boldsymbol{R}_\uparrow$ and $\hat{D}_\downarrow$ is the matrix containing all spin-down states with collective coordinates $\boldsymbol{R}_\downarrow$. Since all elements in the respective matrices contain the same spin function, it can be factorized out,

$$\begin{aligned}|\hat{D}(\boldsymbol{R})| \propto &|\hat{D}_\uparrow(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_\uparrow})|\{\xi_\uparrow(\sigma_1^\uparrow)\xi_\uparrow(\sigma_2^\uparrow)\ldots\xi_\uparrow(\sigma_{N_\uparrow}^\uparrow)\} \\ &\times|\hat{D}_\downarrow(\boldsymbol{r}_{N_\uparrow+1}, \ldots, \boldsymbol{r}_{N-1}, \boldsymbol{r}_N)|\{\xi_\downarrow(\sigma_{N_\uparrow+1}^\downarrow)\ldots\xi_\downarrow(\sigma_{N-1}^\downarrow)\xi_\downarrow(\sigma_N^\downarrow)\}\end{aligned} \tag{7.8}$$

and omitted in the future study. We are then left with the Slater determinant

$$|\hat{D}(\boldsymbol{R})| \propto |\hat{D}_\uparrow(\boldsymbol{R}_\uparrow)| \cdot |\hat{D}_\downarrow(\boldsymbol{R}_\downarrow)| \tag{7.9}$$

which is independent of spin, i.e., the matrices now consist of the spatial functions $\phi_j(\boldsymbol{r}_i)$ as the elements,

$$|\hat{D}_\sigma(\boldsymbol{R})| \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1) & \phi_2(\boldsymbol{r}_1) & \ldots & \phi_N(\boldsymbol{r}_1) \\ \phi_1(\boldsymbol{r}_2) & \phi_2(\boldsymbol{r}_2) & \ldots & \phi_N(\boldsymbol{r}_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(\boldsymbol{r}_N) & \phi_2(\boldsymbol{r}_N) & \ldots & \phi_N(\boldsymbol{r}_N) \end{vmatrix}. \tag{7.10}$$

It is also worth to notice that the number of spin-up particles determines the size of the spin-up matrix, and the same applies to the spin-down matrix. In other words, we can change the total spin S by adjusting the relative sizes of the determinants. In the implementation, however, we stick to the magic quantum numbers, where $N_\uparrow = N_\downarrow$. Otherwise, we need to include a spin-dependent contribution in the Hamiltonian.

7.1.2 The Jastrow factors

From electrostatics, we know that identical, charged particles will repulse each other. This means that the probability of finding two particles close to each other should be low, which needs to be baked into the wave function. One way to do this is to simply multiply the wave function with the distance between the particles. This gives a lower probability when the distance between two electrons is small. However, since we are going to work in the logarithmic space, dealing with exponential functions will be much easier. This is the main idea behind the simple Jastrow factor.

7.1.2.1 Simple Jastrow

The simple Jastrow factor is just an exponential function with the sum over all particle distances. In addition, each distance r_{ij} is weighted by a parameter β_{ij}, and the factor becomes

$$J(\boldsymbol{r}; \boldsymbol{\beta}) = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N}\beta_{ij}r_{ij}\right). \tag{7.11}$$

All the β_{ij} are free variational parameters, which are expected to be symmetric since the distance matrix is symmetric. One problem with this Jastrow factor, is that it does not create the cusp around each particle correctly. Basically, the Jastrow factor increases faster than it should when a particle is moved away from another. To solve this, we need to introduce a more complex Jastrow factor, the Padé-Jastrow.

7.1.2.2 Padé-Jastrow

The Padé-Jastrow factor is closely related to the simple Jastrow above, but a denominator is added to make the cusp correct. It reads

$$J(\boldsymbol{r};\beta) = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{a_{ij}r_{ij}}{1+\beta r_{ij}}\right). \tag{7.12}$$

where β is a variational parameter. In addition, the fractions are multiplied with constants a_{ij} which depend on the particles i and j in the following way:

$$a_{ij} = \begin{cases} e^2/(d+1) & \text{if } i,j \text{ are particles of same spin} \\ e^2/(d-1) & \text{if } i,j \text{ are particles of opposite spin} \end{cases}, \tag{7.13}$$

for dimensions $d \in [2,3]$ where e is the elementary charge. We will later use natural and atomic units, and set $e = 1$, which for two dimensions gives $a_{ij} = 1/3$ (same spin) or $a_{ij} = 1$ (opposite spin) and for three dimensions $a_{ij} = 1/4$ (same spin) and $a_{ij} = 1/2$ (opposite spin) [19, 69].

This Jastrow factor is known to give accurate results for fermions and bosons because it gives the right cusp condition, and it is the one we are going to use in the standard variational Monte-Carlo simulations.

7.1.3 The local energy

We have now seen how we approximate the wave function by a trial wave function, but how do we actually find the energy? In chapter 2, we saw that the two-body interaction term makes the integral impossible to solve for many particles, such that we need to rely on numerical methods.

As we have seen, in VMC, we start with a trial wave function guess $\Psi_T(\boldsymbol{R};\boldsymbol{\theta})$ where the parameters $\boldsymbol{\theta}$ are varied to minimize the energy. According to the variational principle, the obtained energy will always be higher or equal to the true ground state energy, where the equality is the case if and only if the wave function is the exact ground state wave function. Denoting the exact ground state energy by E_0 and the obtained energy as E, we can summarize this by

$$E_0 \leq E = \frac{\int d\boldsymbol{R}\Psi_T(\boldsymbol{R})^*\hat{\mathcal{H}}\Psi_T(\boldsymbol{R})}{\int d\boldsymbol{R}\Psi_T(\boldsymbol{R})^*\Psi_T(\boldsymbol{R})}. \tag{7.14}$$

Furthermore, the integral can be written in the form of a general expectation value

$$E = \int d\boldsymbol{R}E_L(\boldsymbol{R})P(\boldsymbol{R}) \tag{7.15}$$

defining the local energy as

$$E_L(\boldsymbol{R}) \equiv \frac{1}{\Psi_T(\boldsymbol{R})}\hat{\mathcal{H}}\Psi_T(\boldsymbol{R}). \tag{7.16}$$

$P(\boldsymbol{R})$ is called the probability density function (PDF) and was first introduced in equation (2.9). In a more general scheme the PDF reads

$$P(\boldsymbol{R}) = \frac{|\Psi_T(\boldsymbol{R})|^2}{\int d\boldsymbol{R}|\Psi_T(\boldsymbol{R})|^2} \tag{7.17}$$

where the wave function not necessarily is normalized. Since the energy expectation value now is in the form of a general expectation value, we can use the approximation set up by Monte

Carlo integration in equation (7.2), yielding

$$E \approx \frac{1}{M}\sum_{i=1}^{M} E_L(\boldsymbol{R}_i). \tag{7.18}$$

where the local energies $E_L(\boldsymbol{R}_i)$ are still evaluated with particle configurations $\boldsymbol{R}_i$ drawn from $P(\boldsymbol{R})$. In this manner, the obtained energy is guaranteed to approach the exact energy as the number of Monte Carlo cycles, M, increases. Actually, the standard error goes as $\mathcal{O}(1/\sqrt{M})$, making the method pretty accurate for large M's. In the limit $M \to \infty$, the error goes to zero,

$$\langle E_L \rangle = \lim_{M\to\infty} \frac{1}{M}\sum_{i=1}^{M} E_L(\boldsymbol{R}_i), \tag{7.19}$$

indicating the desire of a large number of cycles. This is associated with a zero-variance property governing the VMC method, stating that the variance in the true ground state should be zero [70, 71]. Various sampling regimes are described in chapter 8.

7.1.4 Parameter update

Above we have discussed the sampling in detail, which is a central part of the VMC method. Another important part is the update of the parameters, which we need in order to find an approximative wave function. In section 5.6, we discussed various gradient-based optimization algorithms for iterative minimization of the cost function $\mathcal{C}(\boldsymbol{\theta})$ in machine learning, and the same methods can be used in VMC.

However, we need to define a cost function for VMC which is minimized when the ground state energy is obtained. According to the variational principle, the ground state energy is the lowest energy we can obtain, so an obvious cost function is the energy expectation value. We therefore set

$$\mathcal{C}(\boldsymbol{\theta}) = \langle E_L \rangle \tag{7.20}$$

since we get the expectation value of the local energy from the Metropolis sampling. Further, we use the definition of the gradient of an expectation value from Hjorth-Jensen [53] and obtain

$$\nabla_{\theta_j}\langle E_L \rangle = 2\Big(\langle E_L \nabla_{\theta_j} \ln \Psi_T \rangle - \langle E_L \rangle \langle \nabla_{\theta_j} \ln \Psi_T \rangle\Big) \tag{7.21}$$

which means that we need to calculate the expectation values $\langle E_L \nabla_{\theta_j} \ln \Psi_T \rangle$ and $\langle \nabla_{\theta_j} \ln \Psi_T \rangle$ in addition to the expectation value of the local energy. Those expectation values are found from the integrals

$$\langle \nabla_{\theta_j} \ln \Psi_T \rangle = \int_{-\infty}^{\infty} d\boldsymbol{R} \nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}) P(\boldsymbol{R}) \tag{7.22}$$

and

$$\langle E_L \nabla_{\theta_j} \ln \Psi_T \rangle = \int_{-\infty}^{\infty} d\boldsymbol{R} E_L(\boldsymbol{R}) \nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}) P(\boldsymbol{R}), \tag{7.23}$$

which can be found by Monte Carlo integration in the same way as the local energy:

$$\langle \nabla_{\theta_j} \ln \Psi_T \rangle \approx \frac{1}{M}\sum_{i=1}^{M} \nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}_i) \tag{7.24}$$

and

$$\langle E_L \nabla_{\theta_j} \ln \Psi_T \rangle \approx \frac{1}{M}\sum_{i=1}^{M} E_L(\boldsymbol{R}_i) \nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}_i). \tag{7.25}$$

Note that the only closed-form expression needed, in addition to the local energy, is the expression of $\nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}_i)$. This will later be found for the various wave functions.

We want to stress that the local energy is not the only possible choice of cost function. By taking advantage of the zero-variance property of the expectation value of the local energy in the minimum, one can also minimize the variance. Variance minimization requires the calculation of a few additional expectation values, but it is a fully manageable task to do. See for instance Bajdich & Mitas [24] for more information.

7.1.5 Estimate the electron density

In section 3.3, we introduced the electron density and defined the P-body density as an integral over all particles but P,

$$\rho_P(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_P) = N \int_{-\infty}^{\infty} d\boldsymbol{r}_{P+1} \ldots d\boldsymbol{r}_N |\Psi(\boldsymbol{r}_1, \ldots \boldsymbol{r}_N)|^2. \tag{7.26}$$

Not surprising, also this integral will be solved using Monte Carlo integration, but in a slightly different way than the integrals above since this integral returns a distribution rather than an expected value. First, we divide the space into bins which either have equal sizes or known sizes. After that, we sample the particles and count the number of times a particle occurs in each bin. We want to find the relative number of particles in each bin concerning the size of the bin, so if the bins are of different sizes, we need to standardize them all by dividing by their respective sizes. In the end, we normalize the density such that the sum of all densities becomes equal to N. This method and the particular implementation is detailed in section 10.8.

7.1.6 Common extensions

This finalizes the essential theory behind the VMC method. However, a sampling algorithm is needed to draw samples randomly from $P(\boldsymbol{R})$, and in chapter 8 some popular sampling techniques are described. Before we jump into this field, we will discuss some natural extensions and improvements to the VMC method.

Initially, the particle configuration $\boldsymbol{R}$ might not be representative for a configuration in equilibrium, making the first cycles a poor estimate of the expectation value. An easy fix to this problem is to basically ignore the first sampling period, known as the *burn-in period*. With this in mind, we can implement an equilibration fraction describing the fraction of the total number of cycles that are used in the burn-in period. When running multiple parallel processes, the burn-in period should be the same for all the processes.

We also have a technique called *thinning*, which means picking every n'th sample in the chain and ignore the rest. This is shown to give a more or less identical expectation value and makes the program less memory-requiring, but due to worse statistics, this should be avoided as long as there is no lack of memory.

7.2 Diffusion Monte Carlo

The second and last quantum Monte Carlo method we will discuss is the diffusion Monte Carlo (DMC) method. DMC belongs to a class of projection and Green's function approaches and provides in principle an exact solution of the Schrödinger equation. However, for fermionic systems, the method is plagued by the occurrence of a sign-problem, known as the fermion sign problem [5], and we need to use approximations to bypass the problem. The most common fix is the fixed-node approximation, which introduces errors to the computations. The problem is also attempted to solve using shadow wave functions, for instance, by Calcavecchia *et al.* [72],

which have many commonalities with our restricted Boltzmann machine (RBM) approach. The idea is that the RBMs can also contribute to solve the sign problem, but we leave this for later studies.

Since the DMC method is used as a reference for more or less all the produced results, it is natural to give a brief discussion of the idea behind the method. By the fifth postulate of quantum mechanics discussed in section 2.1, the motion of a particle that moves in a potential $V(\boldsymbol{R})$ is described by the time-dependent Schrödinger equation,

$$i\frac{\partial\Psi(\boldsymbol{R},t)}{\partial t} = \hat{\mathcal{H}}\Psi(\boldsymbol{R},t) \tag{7.27}$$

with t as the time and i as the solution of $x^2 = -1$. In natural units, the Hamiltonian is given by

$$\hat{\mathcal{H}} = -\frac{1}{2}\nabla^2 + V(\boldsymbol{R}). \tag{7.28}$$

Further, we assume that the potential becomes infinite as $x \to \pm\infty$, such that the particle motion is confined to a finite spatial domain. The formal solution of the time-dependent Schrödinger equation is then given by an expansion of the eigenfunctions of $\hat{\mathcal{H}}$, $\phi_n(\boldsymbol{r})$,

$$\Psi(\boldsymbol{R},t) = \sum_{n=0}^{\infty} c_n\phi_n(\boldsymbol{R})e^{-E_nt/\hbar} \tag{7.29}$$

with $\hbar$ as the reduced Planck's constant and where the coefficients can be obtained by

$$c_n = \int_{-\infty}^{\infty} d\boldsymbol{R}\phi_n(\boldsymbol{R})\Psi(\boldsymbol{R},0) \tag{7.30}$$

under the assumption that the eigenfunctions are orthonormal and that the energies E_n are sorted in increasing order. The DMC method is based on the time-imaginary Schrödinger equation, which is obtained by a **shift of energy scale**, setting $V(\boldsymbol{R}) \to V(\boldsymbol{R}) - E_R$ and $E_n \to E_n - E_R$, which does not change any physical quantity. By further using the **Wick rotation of time** setting $\tau = it$, we obtain the time-imaginary Schrödinger equation [73]

$$-\frac{\partial\Psi(\boldsymbol{R},\tau)}{\partial\tau} = -\frac{1}{2}\nabla^2\Psi(\boldsymbol{R},\tau) + [V(\boldsymbol{r}) - E_R]\Psi(\boldsymbol{R},\tau) \tag{7.31}$$

which gives the expansion

$$\Psi(\boldsymbol{R},\tau) = \sum_{n=0}^{\infty} c_n\phi_n(\boldsymbol{R})e^{-(E_n-E_R)\tau}. \tag{7.32}$$

The time-imaginary Schrödinger equation is similar to the Fokker-Planck equation presented in equation (8.6), which was noticed by Fermi already around 1945 [25, 28]. Since we have analytical solutions of the Fokker-Planck equation, the idea is to apply the same solutions to equation (7.31). To clarify this, consider the time-imaginary Schrödinger equation for non-interacting particles,

$$\frac{\partial\Psi(\boldsymbol{R},\tau)}{\partial\tau} = \frac{1}{2}\nabla^2\Psi(\boldsymbol{R},\tau) \tag{7.33}$$

which is really similar to the diffusion equation,

$$\frac{\partial\phi(\boldsymbol{R},t)}{\partial t} = D\nabla^2\phi(\boldsymbol{R},t) \tag{7.34}$$

where D is the diffusion constant set to $1/2$ in our calculations, no wonder why. We end our motivation of the DMC method here, and relegate the reader to Kosztin *et al.* [73] for a thorough and comprehensive explanation of the method. The last section of this chapter is about error estimation, which is very important in experimental physics, also when it comes to computer experiments.

7.3 Unifying Quantum Mechanics and Machine Learning

Now as we have introduced the necessary theory, both in the form of quantum theory and machine learning theory, in addition to detailing the variational Monte Carlo (VMC) method, we are ready to unify the machine learning and quantum mechanics. As hinted above, the way we do this is to let a restricted Boltzmann machine define our single-particle functions in the trial wave function, and then just update the function in a normal VMC scheme. This is very similar to the approach of Pfau *et al.* [16], but an essential difference is that they did not restrict the single-particle functions to take the coordinates of a single particle. Further, we investigate how the results change when Jastrow factors with gradually more physical intuition baked in are added. As the main goal is to find a method that requires less physical intuition in order to provide accurate results compared to the traditional methods, adding a simple Jastrow factor is also very interesting. We look at three different cases: a Slater determinant where the single-particle functions are given by a restricted Boltzmann machine (RBM), an RBM with a simple Jastrow added (RBM+SJ) and an RBM with a Padé-Jastrow factor added (RBM+PJ). They are detailed in respective sections below.

7.3.1 Restricted Boltzmann machine without Jastrow factor (RBM)

In chapter 7, we saw how the VMC method attempts to solve the time-independent Schrödinger equation directly by vary the parameters in a trial wave function. In order to satisfy the anti-symmetry properties of a fermionic many-body wave function, the trial wave function was composed as a Slater determinant,

$$\Psi_T(\boldsymbol{R}) \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1) & \phi_2(\boldsymbol{r}_1) & \dots & \phi_N(\boldsymbol{r}_1) \\ \phi_1(\boldsymbol{r}_2) & \phi_2(\boldsymbol{r}_2) & \dots & \phi_N(\boldsymbol{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{r}_N) & \phi_2(\boldsymbol{r}_N) & \dots & \phi_N(\boldsymbol{r}_N) \end{vmatrix} \equiv |\hat{D}(\boldsymbol{R})|. \tag{7.35}$$

Even though we want a method which requires as little physical intuition as possible, the wave function needs to obey Fermi-Dirac statistics, and we therefore also approximate the trial wave function with the Slater determinant for the machine learning method. Further, we use the same framework as we did for VMC, but the single-particle functions (SPFs) $\phi_j(\boldsymbol{r}_i)$ are given by the marginal distribution of the visible units from a restricted Boltzmann machine, $P(\boldsymbol{r})$. For quantum dots, we also add the Hermite polynomials, $H_n(\boldsymbol{r})$ to get an unique SPF for each state. We then have

$$\phi_j(\boldsymbol{r}_i) = H_j(\boldsymbol{r}_i)P(\boldsymbol{r}_i) \tag{7.36}$$

where the restricted Boltzmann machine takes all the coordinates as input nodes, i.e., the input vector is $\boldsymbol{x} = (x_1, y_1, x_2, y_2, \dots, x_N, y_N)$ for a two-dimensional system. With this, we end up with a trial wave function that contains more variational parameters than a standard VMC trial wave function, and which hopefully is able to provide a more correct wave function. Further, we will add a simple Jastrow factor to see how this affects the correlations.

7.3.2 RBM with a simple Jastrow factor (RBM+SJ)

This method, abbreviated RBM+SJ, is just an extension of the RBM method described in the previous section, where we add a simple Jastrow factor to help modeling the electron-electron cusp. When adding a Jastrow factor we also add some more physical intuition to the trial wave function, but the Jastrow factor used here still contains a minimum of physical intuition. It is

expressed as

$$J(\boldsymbol{R};\boldsymbol{\beta}) = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N}\beta_{ij}r_{ij}\right). \tag{7.37}$$

where r_{ij} is the radial distance between particle i and j and the β_{ij}'s are free variational parameters, which are expected to be symmetric since the distance matrix is symmetric. The trial wave function using this method thus takes the form

$$\Psi_T(\boldsymbol{R}) \propto |\hat{D}_\uparrow(\boldsymbol{R}_\uparrow)| \cdot |\hat{D}_\downarrow(\boldsymbol{R}_\downarrow)| J(\boldsymbol{R};\boldsymbol{\beta}) \tag{7.38}$$

7.3.3 RBM with a Padé-Jastrow factor (RBM+PJ)

Lastly, we add the well-known Padé-Jastrow factor to the RBM, abbreviated RBM+PJ, to see how much the results change when a more advanced correlation factor is added. As discussed in section 7.1.2, the Padé-Jastrow factor is constructed to model the electron-electron cusp correctly, and it is also interesting to see how more physical intuition in the trial wave function affects the results. The RBM+PJ trial wave function then reads

$$\Psi_T(\boldsymbol{R}) \propto |\hat{D}_\uparrow(\boldsymbol{R}_\uparrow)| \cdot |\hat{D}_\downarrow(\boldsymbol{R}_\downarrow)| J(\boldsymbol{R};\beta) \tag{7.39}$$

where we recall that the Padé-Jastrow factor contains the variational parameter β.

7.4 Error estimation

In experiments, we have two main classes of errors; systematical errors and statistical errors. The former is a result of external factors such as uncertainties in the apparatus or a person constantly takes an incorrect measurement, which is an error that is hard to estimate. The latter, however, can be found by estimating the variance of the sample mean, which we want to find accurately and efficiently. Monte Carlo simulations can be treated as computer experiments, and therefore we can use the same analyzing tools as we do for real experiments.

There are many ways to estimate the variance, where the cheapest ones ignore correlation effects between the measurements. More accurate standard error estimations can be performed by resampling methods like blocking, bootstrap, or jackknife. We will cover the blocking method only since we stick to that method in our particular implementations. To save computational time, we resample the final iteration only; for the others, we use the simple estimation method.

7.4.1 Central concepts of statistics

Before we go through the methods, we will give a brief introduction to some useful statistical quantities. We start with the *moments*, which are given by

$$\langle x^n \rangle = \int dx p(x) x^n$$

where $p(x)$ is the true probability density function. In order to make physical sense, this function needs to be normalized such that the integral over all possible outcomes gives a total probability of 1. This is associated with the zero'th moment, where we get $\int dx p(x) = 1$. The first moment is the *mean* of $p(x)$, and is often denoted by the letter μ,

$$\langle x \rangle = \mu = \int dx p(x) x. \tag{7.40}$$

Moreover, we can define the *central moments* given by

$$\langle (x - \langle x \rangle)^n \rangle = \int dx (x - \langle x \rangle)^n p(x), \tag{7.41}$$

which is centered around the mean. With $n = 0$ and $n = 1$, this is easy to find, but what is the central moment with $n = 2$? The central moment with $n = 2$ is what we call the *variance*, and is often denoted as σ^2 as we did in the equation (2.16). One can show that

$$\sigma^2 = \langle (x - \langle x \rangle)^2 \rangle = \langle x^2 \rangle - \langle x \rangle^2 \tag{7.42}$$

which was already stated in chapter 2. The *standard deviation* is defined as the square-root of the variance, σ, and will be presented as the measure of the uncertainty in the last digit of numerical results. By using the expressions above, we can calculate the mean and the variance of a probability density function $p(x)$, but what do we do if we have a set of data drawn from an unknown $p(x)$?

If the probability density function is unavailable, we cannot find the exact sample mean and sample mean variance. However, we can make an estimate, writing

$$\langle X \rangle \approx \frac{1}{n} \sum_{i=1}^{n} X_i \equiv \overline{X}, \tag{7.43}$$

for the sample mean assuming that X is our sample containing n points. Further, the variance of the sample mean, assuming that the measurements are *independent*, can be found from

$$\text{var}(\overline{X}) \equiv \frac{1}{n} \sum_{i=1}^{n} (X_i - \overline{X})^2 = \frac{\sigma^2}{n} \tag{7.44}$$

where σ^2 is the variance of the probability density function. The law of large numbers states that the estimated sample mean approaches the true sample mean as n goes to infinity. This is closely related to the central limit theorem, which says that the probability distribution of X, $p_X(x)$, can be approximated as a normal distribution $\mathcal{N}(\mu, \sigma)$ with $\mu = \overline{X}_n$ and $\sigma = \text{var}(\overline{X})$ as the number of sampling points n goes to infinity.

On the other hand, if the samples are *correlated*, equation (7.44) becomes an underestimation of the *sample error* err_X^2, and is thus more a guideline for the size of the uncertainty, more than an actual estimate of it. In that case, we need to calculate the more general *covariance* given by

$$\text{cov}(X_i, X_j) = (X_i - \overline{X})(X_j - \overline{X}), \tag{7.45}$$

which provides a measure of the correlations between X_i and X_j. Note that the variance is just the special case where X_i and X_j are independent. We define the sample error as

$$\text{err}_X^2 = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \text{cov}(X_i, X_j), \tag{7.46}$$

with an ability to be further expressed as a function of the *autocovariance*, defined as

$$\gamma(h) = \text{cov}(X_i, X_{i+h}) \tag{7.47}$$

and is thus just a measure of the correlation of a section of a time series with another section of the same time series. If $h = 0$, this is just the variance $\gamma(0) = \sigma^2/n$ and the sample error simplifies to

$$\text{err}_X^2 = \frac{\sigma^2}{n} + \frac{2}{n} \sum_{h=1}^{n-1} \Big(1 - \frac{h}{n}\Big) \gamma(h). \tag{7.48}$$

A problem with this definition of the sampling error is that it turns out to be very expensive to calculate as the number of samples gets large. We require a cheaper method, which is the task of the resampling methods. We will, in the following, discuss the blocking algorithm, which is the resampling method of choice in this work.

7.4.2 Blocking

Above, we have described the need for a proper uncertainty estimation in computational simulations, where the covariance was included in the calculation of σ^2. A quick and easy way to get a proper estimate of the uncertainty is by using the blocking method.

When the blocking method was made accessible by Flyvbjerg & Petersen [46] in 1989, the method required hyper-parameters which had to be carefully adjusted for each particular data set. In 2018, Jonsson [74] reinvented the algorithm and made it automated, with no need for external parameters. Despite this, no compromise was made on performance. The method scales as $12n + \mathcal{O}(\log_2 n)$ for small data sets, but reduces to $n + \mathcal{O}(1)$ for large data sets, which makes it preferred over bootstrap and jackknife for large data sets. We will now go through the idea behind the blocking method.

Consider a time series $\{x_1, x_2, \ldots, x_n\}$ with $n = 2^d$ data point for some integer $d > 1$. For this series, an autocovariance function $\gamma(h)$ is guaranteed to exist. We arrange the data in a vector

$$X = (x_1, x_2, \ldots, x_n), \tag{7.49}$$

which we assume to be asymptotically uncorrelated. The idea is to take the mean of subsequent pair of elements from X, and form a new vector X_1. We then repeat the operation on X_1 and form a new vector X_2 and so on. This is the reason why we require $n = 2^d$. If k denotes an element in vector X_i, we can write the procedure recursively as

$$\begin{aligned} (X_0)_k &\equiv (X)_k \\ (X_{i+1})_k &\equiv \frac{1}{2}\Big((X_i)_{2k-1} + (X_i)_{2k}\Big), \end{aligned} \tag{7.50}$$

where $1 \leq i \leq d-1$ and $1 \leq k \leq n/2^i$, which are known as the *blocking transformations*. According to equation (7.48), we can express the sample mean variance of X_k as

$$\mathrm{var}(\overline{X}_k) = \frac{\sigma_k^2}{n_k} + \frac{2}{n_k}\sum_{h=1}^{n_k-1}\Big(1 - \frac{h}{n_k}\Big)\gamma_k(h), \tag{7.51}$$

where we define the last term as the *truncation error*, e_k, as it is intractable. It can be shown that the sample variance of all pairs $(X_i)_k$ and $(X_j)_k$ after a while will be identical [46],

$$\mathrm{var}(\overline{X}_i) = \mathrm{var}(\overline{X}_j) \qquad \forall \quad i, j \in \{0, 1, ..., d-1\}, \tag{7.52}$$

with the consequence that the sample mean variance of the entire set of samples is also given by

$$\mathrm{var}(\overline{X}) = \frac{\sigma_k^2}{n_k} + e_k \qquad \forall \quad k \in \{0, 1, ..., d-1\}. \tag{7.53}$$

In the original (manual) blocking method, we had to know exactly where to stop the procedure in order to satisfy equation (7.52). If we do not continue long enough, the sample variance has not converged, while if we keep on going for too long, the standard error of $\mathrm{var}(\hat{\sigma}_k^2/n_k)$ get very large. If one plots the sample variance as a function of the iterations, one will see that the curve forms a plateau before it gets very noisy, but with Jonsson's automated method, we do not need to worry about this.

CHAPTER 8

The Metropolis Algorithm

Metropolis sampling is a Markov chain Monte Carlo method, which generates a Markov chain using a proposal density for new steps, and the method also rejects unsatisfying moves. In its most simple form, a particle is moved in a random direction, which was the original method invented by Metropolis *et al.* [75] back in 1953. Later, the method was improved by Hastings [27], giving rise to the more general Metropolis-Hastings algorithm. The genius of the Metropolis algorithms is that the acceptance of a move is not based on the probabilities themselves, but rather the ratio between the new and the old probabilities. In that way, we avoid calculating the normalizing factor, which is often computationally intractable.

We will first discuss Markov chains in general, before we connect them to the original Metropolis algorithm, henceforth called *brute-force sampling*, and then the Metropolis-Hastings algorithm, also called *importance sampling*. If we denote the current state by $\boldsymbol{R}$, and the proposed state by $\boldsymbol{R}'$, we have a transition rule $P(\boldsymbol{R}'|\boldsymbol{R})$ for going from $\boldsymbol{R}$ to $\boldsymbol{R}'$ and a transition rule $P(\boldsymbol{R}|\boldsymbol{R}')$ for going the opposite way. If we then assume that the rules satisfy *ergodicity* and *detailed balance*, we have the following relationship:

$$P(\boldsymbol{R}'|\boldsymbol{R})P(\boldsymbol{R}) = P(\boldsymbol{R}|\boldsymbol{R}')P(\boldsymbol{R}'), \tag{8.1}$$

which is actually a restatement of Bayes' theorem presented in section 6.1. The next step is to rewrite the transition rules in terms of a proposal distribution $T(\boldsymbol{R}'|\boldsymbol{R})$ and an acceptance probability $A(\boldsymbol{R}', \boldsymbol{R})$,

$$P(\boldsymbol{R}'|\boldsymbol{R}) = T(\boldsymbol{R}'|\boldsymbol{R})A(\boldsymbol{R}', \boldsymbol{R}). \tag{8.2}$$

In order to satisfy the detailed balance, we need to choose $A(\boldsymbol{R}', \boldsymbol{R})$ such that

$$A(\boldsymbol{R}', \boldsymbol{R}) = \min\left[1, \frac{T(\boldsymbol{R}|\boldsymbol{R}')P(\boldsymbol{R}')}{T(\boldsymbol{R}'|\boldsymbol{R})P(\boldsymbol{R})}\right], \tag{8.3}$$

where we limit A to maximum 1 as the probability should not exceed 1. We want to accept a move with probability $A(\boldsymbol{R}', \boldsymbol{R})$. One way to do that is to draw a number from a uniform distribution between 0 and 1. If this number is lower than the acceptance, the move should be accepted and rejected otherwise.

8.0.1 Brute-force sampling

In its simplest form, the move is proposed randomly both in magnitude and direction. Mathematically, we can write this as

$$\boldsymbol{R}' = \boldsymbol{R} + \Delta x d\boldsymbol{R} \tag{8.4}$$

where Δx is the step length and $d\boldsymbol{r}$ has a random magnitude and direction (typically which particle to move). We obtain the naïve acceptance probability when requiring $T(\boldsymbol{R}'|\boldsymbol{R}) = T(\boldsymbol{R}|\boldsymbol{R}')$,

such that the it simplifies to

$$A(\boldsymbol{R}', \boldsymbol{R}) = \min\left[1, \frac{P(\boldsymbol{R}')}{P(\boldsymbol{R})}\right]. \tag{8.5}$$

However, with this approach, an unsatisfying number of moves will be rejected as the particles can be moved in all directions, which results in a significant waste of computing power. A better method is **importance sampling**, since the particles are moved according to the quantum force.

8.0.2 Importance sampling

Importance sampling is a more intelligent sampling method than the brute-force sampling, since the new position is based on an educated guess. To understand how it works, we need to take a quick look at diffusion processes. We start from the Fokker-Planck equation,

$$\frac{\partial P(\boldsymbol{R}, t)}{\partial t} = D\nabla\left(\nabla - \boldsymbol{F}\right) P(\boldsymbol{R}, t) \tag{8.6}$$

which describes how a probability distribution $P(\boldsymbol{R}, t)$ evolves in appearance of a drift force $\boldsymbol{F}$. In the case $\boldsymbol{F} = \boldsymbol{0}$, the equation reduces to the diffusion equation with D as the diffusion constant. This simplifies to $D = 1/2$ in natural units.

The Langevin equation states that a diffusion particle tends to move parallel to the drift force in the coordinate space, with $\boldsymbol{\eta}$ introducing some random noise. The equation reads

$$\frac{\partial \boldsymbol{R}(t)}{\partial t} = D\boldsymbol{F}(\boldsymbol{R}(t)) + \boldsymbol{\eta}. \tag{8.7}$$

Given a position $\boldsymbol{R}$, the new position $\boldsymbol{R}'$ can be be found by applying forward-Euler on the Langevin equation, obtaining

$$\boldsymbol{R}' = \boldsymbol{R} + D\Delta t\boldsymbol{F}(\boldsymbol{R}) + \boldsymbol{\xi}\sqrt{\Delta t} \tag{8.8}$$

where Δt is a fictive time step and $\boldsymbol{\xi}$ is a Gaussian random variable. Further, we want to find an expression of the drift force $\boldsymbol{F}$ which makes the system converge to a stationary state. A stationary state is found when the probability density function, $P(\boldsymbol{R})$, is constant in time, i.e., when the left-hand side of the Fokker-Planck equation is zero. In that case, we can rewrite the equation as

$$\nabla^2 P(\boldsymbol{R}) = P(\boldsymbol{R})\nabla\boldsymbol{F}(\boldsymbol{R}) + \boldsymbol{F}(\boldsymbol{R})\nabla P(\boldsymbol{R}). \tag{8.9}$$

where the parenthesis are written out and we have moved the term that is independent of the force $\boldsymbol{F}$ to the left-hand side. Moreover, we assume that the drift force takes the form $\boldsymbol{F}(\boldsymbol{R}) = g(\boldsymbol{R})\nabla P(\boldsymbol{R})$ based on the fact that the force should point in the direction of the steepest slope. We can then go ahead and write

$$\nabla^2 P(\boldsymbol{R})\big(1 - P(\boldsymbol{R})g(\boldsymbol{R})\big) = \nabla\big(g(\boldsymbol{R})P(\boldsymbol{R})\big)\nabla P(\boldsymbol{R}) \tag{8.10}$$

where the quantity $\nabla^2 P(\boldsymbol{R})$ is factorized out from two of the terms. The equation is satisfied when $g(\boldsymbol{R}) = 1/P(\boldsymbol{R})$, such that the drift force evolves to the well-known form

$$\boldsymbol{F}(\boldsymbol{R}) = \frac{\nabla P(\boldsymbol{R})}{P(\boldsymbol{R})} = 2\frac{\nabla\Psi_T(\boldsymbol{R})}{\Psi_T(\boldsymbol{R})} = 2\nabla\ln\Psi_T(\boldsymbol{R}), \tag{8.11}$$

which is also known as the *quantum force*.

The remaining part concerns the acceptance of the moves. For this, we need to find the sampling distributions $T(\boldsymbol{R}'|\boldsymbol{R})$ from equation (8.3), which are just the solutions of the Fokker-Planck equation. The solutions read

$$G(\boldsymbol{R}', \boldsymbol{R}, \Delta t) \propto \exp\Big(- \big(\boldsymbol{R}' - \boldsymbol{R} - D\Delta t \boldsymbol{F}(\boldsymbol{R})\big)^2 / 4D\Delta t \Big), \tag{8.12}$$

which is categorized as a Green's function. In general, the essential property of any Green's function is that it provides a way to describe the response of an arbitrary differential equation solution. For our particular case, it correspond to $\mathcal{N}(\boldsymbol{R}'|\boldsymbol{R} + D\Delta t\boldsymbol{F}(\boldsymbol{R}), 2D\Delta t)$ which is the normal distribution with mean $\boldsymbol{\mu} = \boldsymbol{R} + D\Delta t\boldsymbol{F}(\boldsymbol{R})$ and variance $\sigma = 2D\Delta t$. By using this, the acceptance probability of the importance sampling can finally be written as

$$A(\boldsymbol{R}'|\boldsymbol{R}) = \min\left[1, \frac{G(\boldsymbol{R}, \boldsymbol{R}', \Delta t)P(\boldsymbol{R}')}{G(\boldsymbol{R}', \boldsymbol{R}, \Delta t)P(\boldsymbol{R})}\right], \tag{8.13}$$

where the marginal probabilities are still given by equation (7.17).

8.0.3 Gibbs sampling

Gibbs sampling has, throughout the years, gained high popularity in the machine learning community when it comes to training Boltzmann machines. There are probably many factors that contribute to this, where the performance is one of them. Another proper motivation is the absent of tuning parameters, which makes the method more comfortable to deal with compared to many of its competitors. The method is an instance of the Metropolis-Hastings algorithm and is therefore classified as another Markov chain Monte Carlo method. It differs from the Metropolis methods discussed above by the fact that all the moves are accepted, such that we do not waste computational time on rejected moves. Nevertheless, we should not use this argument alone to motivate the use of Gibbs sampling, as the algorithm usually and preferably rejects less than 1% of the moves in importance sampling.

We will in the following briefly describe the mathematical foundation of the method, before we, for the sake of clarity, connect it to the restricted Boltzmann machines. The method is built on the concept that, given a multivariate distribution, it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution. This is the reason why we do not need the marginal distributions in Gibbs sampling, but rather the conditional distributions. In the most general, Gibbs sampling proposes a rule for going from a sample $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$ to another sample $\boldsymbol{x}^{(i+1)} = (x_1^{(i+1)}, x_2^{(i+1)}, \dots, x_n^{(i+1)})$, similar to the Metropolis algorithm. However, in Gibbs sampling the transition from $x_j^{(i)}$ to $x_j^{(i+1)}$ is performed according to the conditional distribution specified by

$$P\big(x_j^{(i+1)}|x_1^{(i+1)}, \dots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \dots, x_n^{(i)}\big). \tag{8.14}$$

The marginal distribution of a subset of variables can then be approximated by simply considering the samples for that subset of variables, ignoring the rest.

For a restricted Boltzmann machine, this becomes a two-step sampling process as we have two layers, such that we use the conditional probabilities $P(\boldsymbol{x}|\boldsymbol{h})$ and $P(\boldsymbol{h}|\boldsymbol{x})$ to update the visible and hidden units respectively. For the restricted Boltzmann machine with Gaussian-binary units presented in section 6.3, the conditional probability of $h_j = 0$ given a set of visible nodes $\boldsymbol{x}$ is

$$P(h_j = 0|\boldsymbol{x}) = \frac{1}{1 + \exp\Big(+b_j + \sum_{i=1}^{N} x_i w_{ij}/\sigma^2\Big)} \tag{8.15}$$

and the corresponding conditional probability of $h_j = 1$ is

$$P(h_j = 1|\boldsymbol{x}) = \frac{1}{1 + \exp\Big(-b_j - \sum_{i=1}^{N} x_i w_{ij}/\sigma^2\Big)}, \tag{8.16}$$

which is by the way again our friend the sigmoid function. Note that $P(h_j = 0|\boldsymbol{x}) + P(h_j = 1|\boldsymbol{x}) = 1$, indicating that a hidden node h_j can only take 0 or 1, hence binary. When updating the hidden node, one typically calculate the sigmoid $P(h_j = 1|\boldsymbol{x})$ and set h_j to 1 if this probability is larger than $1/2$, and set it to 0 otherwise.

For the update of the visible nodes, we see from equation (6.21) that the visible nodes are chosen after the normal distribution,

$$P(x_i|\boldsymbol{h}) = \mathcal{N}\Big(a_i + \sum_{j=1}^{H} w_{ij} h_j, \sigma^2\Big), \tag{8.17}$$

which introduces some stochasticity to the system. By going back and forth in the Boltzmann machine multiple times (a round trip corresponds to an iteration), the hope is that the expectation values can be approximated by averaging over all the iterations.

As pointed out earlier, the Gibbs sampling will not be implemented in this work, but we describe it for completeness purposes. The reason for this is that the method has not shown promising results for our specific problem, and we will instead rely on Metropolis sampling. We have already tested the Gibbs sampling in another similar project on small quantum dots [76], and so have others like Flugsrud [17]. The results are matching and show poor performance compared to the Metropolis-Hastings algorithm.

However, the Gibbs sampling method should not be underestimated. Carleo & Troyer [15] showed its importance when solving the Ising model using a restricted Boltzmann machine and Gibbs sampling, and in traditional Boltzmann machines, the Gibbs sampling is the preferred tool.

Part IV

Implementation

CHAPTER 9

Scientific Programming

> Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
>
> John F. Woods, [77]

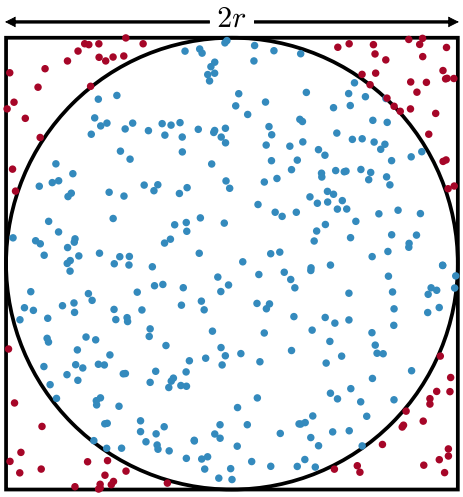


Figure 9.1: One can estimate the value of π using Monte Carlo integration. Here we approximate the ratio between the area of the circle and the area of the square by counting the number of random points occurring inside the circle and inside the square. The estimate of π is found from $\pi = 4 \cdot A_{\text{circle}} / A_{\text{square}}$.

Since we base this scientific work on solving equations numerically on the computer, software development is a significant part of the work. We will in this chapter go through the main concepts of scientific programming, and directly relate them to our code in the next chapters on implementation. As a motivation, we will provide a brief recap of some historical milestones in scientific computing.

Computers have long been used for solving complex scientific problems. Actually, for a long time, science was the primary motivation for developing computers. Already in 1929, Egil

Hylleraas used his mechanical desk calculator to calculate the ground state energy of the Helium atom, which remains a milestone in computational quantum chemistry as well as quantum chemistry in general [78]. Later, pioneers like Alan Turing and John von Neumann contributed to the invention of the first electronic computer, which made numerical solutions of ordinary differential equations possible [79].

Also on the software side, big breakthroughs have been made since the advent of the electronic computer. The programming language Fortran was released in 1957, which provided a new basis of scientific programming [80]. Later, Ole Johan Dahl and Kristen Nygaard developed the language SIMULA, which is considered as the first object-orientated language [81]. The language that we mostly will rely on in this work, C++, is an extension of the language C which was developed in the early 1970s and is, together with Fortran, one of few languages that have survived the ravages of time.

The computers used today, included both the hardware and the software, have become possible due to the heroic effort of an array of scientists, engineers, and mechanics over a long period. The computers are stronger than ever, are more memorious than ever, are compacter than ever and, last but not least, they are more user-friendly than ever. All this has made heavy computations possible that was unthinkable just decades ago and has contributed to the development of virtually all branches of science. Due to the user-friendliness, advanced simulations have also been available for young and curious students like the author, which we should not take for granted.

The computer's language itself is binary and is the lowest level. To translate commands to this language, we need a "translator", which is a language that fills the gap between the binary language and human commands. We categorize this language in levels based on how similar they are to the binary language. Low-level languages are more similar to the binary language than a high-level language, and the result is that low-level languages, in general, are more demanding to deal with, but they are typically faster than high-level languages. In our work, we use the low-level language C++ for the main code, but for scripting, including plotting and solving less expensive problems, the high-level language Python has been used.

In this chapter, we will explain the essentials behind scientific programming, using examples from both C++ and Python. We will begin from the very basic, so if the reader is an experienced programmer, this chapter will probably be perceived as old news. Programs are often either classified as *procedural programming* or *object-oriented programming*, based on whether the code is written in the same order as the program flow goes (procedural) or is based on objects (object-orientated). We will explain the basics of object-oriented programming in the following.

9.1 Object-orientated programming in Python

In everyday life, we are all the time surrounded by objects that we categorize based on properties like their function, behavior, or expression. For instance, a *circle* is an object with properties like area, circumference *et cetera*, and can be categorized as *shapes*. Other members of the class shapes might be square and triangle. In object-orientated programming, we create similar objects, or classes, to make the program flow more intuitive for the reader. The main class is called the parent class, while the sub-classes are called the children. The reason for this analogy is that the sub-classes always inherit from their main class, but not the other way around.

The example with the shapes can therefore be implemented with `Shapes` being the parent, and `Circle`, `Square` and `Triangle` being its children. In the high-level language Python, this can be implemented as

```
import numpy as np
```

```
class Shapes:
    def __init__(self, r):
        self.r = r

    def getArea(self):
        raise NotImplementedError("Class {} has no instance
            ↪ 'getArea'.".format(self.__class__.__name__))


class Circle(Shapes):
    def getArea(self):
        return np.pi*self.r**2

    def getCircumference(self):
        return 2*np.pi*self.r

    def getExtent(self, x, y):
        if np.linalg.norm([x,y]) < self.r:
            return True
        return False

class Square(Shapes):
    def getArea(self):
        return 4*self.r**2

    def getCircumference(self):
        return 8*self.r

    def getExtent(self, x, y):
        if abs(x) < self.r and abs(y) < self.r:
            return True
    return False

class Triangle(Shapes):
    def getCircumference(self):
        return 6*self.r
```

where we give the children the properties area, circumference and extent. The length `r` means the radius of a circle, half the side length of a square and also half a side length of the equilateral triangle, see figure (9.1). We can implement a circle, square and triangle of `r`=1, respectively named ”circ”, ”squr” and ”tria” by these three lines

```
circ = Circle(1)
squr = Square(1)
tria = Triangle(1)
```

Moreover, we can easily get their areas by calling the `getArea()` function,

```
>>> circ.getArea()
3.141592653589793

>>> squr.getArea()
4

>>> tria.getArea()
NotImplementedError: Class Triangle has no instance 'getArea'.
```

As we can see, the area of the circle and the square were calculated successfully, giving πr^2 and $4r$ respectively. On the other hand, the area of the triangle raised an error, which is because we do not have defined the area of the triangle! This example illustrates that the properties of the children overwrites the properties of the parent, but if the children do not have the called property, it will instead return the parent's property.

9.1.1 An example based on Monte Carlo simulations

Up to this point, we have talked a lot about Monte Carlo simulations without giving an illustrating example of what it is. Methods, where we draw numbers randomly from a function to reveal properties of the function, are in general classified as Monte Carlo simulations; in that manner, the name might sound more complicated than it is.

Suppose we did not know the value of π, what could we do to approximate its value? There are many ways to do this, where the most intuitive might be to measure the ratio between the diameter and the circumference in a circle. It is not possible to do this very accurate manually. A more accurate and robust solution would be to use Monte Carlo simulations, where we, for instance, can take the starting point on the relation

$$\pi = 4\frac{A_{\text{circle}}}{A_{\text{square}}}, \tag{9.1}$$

which is derived from the ratio between the circle area and the square area. By drawing random numbers from a uniform distribution and count the number of points that are in the square and the circle, we can approximate $A_{\text{circle}}/A_{\text{square}}$. We can do this by using the `getExtent()` function of the square class and circle class,

```
for i in range(1,10):                    # Iteration loop
    M = 10**i
    A_square = 0
    A_circle = 0
    for _ in range(int(M)):              # Monte Carlo loop
        x = np.random.uniform(-1,1)
        y = np.random.uniform(-1,1)
        if circ.getExtent(x,y):
            A_circle += 1
        if squr.getExtent(x,y):
            A_square += 1
        print("Pi: ", 4 * A_circle/A_square)
```

which gives a output similar to

```
Pi:  2.4
Pi:  3.32
Pi:  3.156
Pi:  3.1296
Pi:  3.14168
Pi:  3.140044
Pi:  3.1420708
Pi:  3.1416844
Pi:  3.141615628
```

Considering the fact that $\pi = 3.141592653589793...$, the obtained result is quite acceptable. However, to run the program with $M = 10^9$ Monte Carlo cycles, the program is quite slow. Is it possible to do this faster? The answer is yes, by switching entirely or partly to a low-level language.

9.2 Low-level programming with C++

Above, we looked at how inheritance can be implemented in the high-level language Python, and we observed the main weakness of high-level languages: the computational time. In this section, we will repeat the implementation and example above, using C++. Using a low-level language will, hopefully, provide a significant speed-up. A similar class structure in C++ looks like

```
class Shapes {
    public:
        Shapes();
        virtual double getArea() = 0;
        virtual double getCircumference() = 0;
        virtual bool getExtent(double x, double y) = 0;
        virtual ~Shapes();
};

class Circle : public Shapes {
    public:
        Circle(double r);
        double getArea() {
            return M_PI * m_r * m_r;
        }
        double getCircumference() {
            return 2 * M_PI * m_r;
        }
        bool getExtent(double x, double y) {
            if(sqrt(x*x+y*y) < m_r) {
                return true;
            }
            return false;
        }
    private:
        double m_r = 1;
};

Circle::Circle(double r) : Shapes() {
    m_r = r;
}

class Square: public Shapes {
    public:
        Square(double r);
        double getArea() {
            return 4 * m_r * m_r;
        }
        double getCircumference() {
            return 8 * m_r;
        }
        bool getExtent(double x, double y) {
            if(fabs(x) < m_r && fabs(y) < m_r) {
                return true;
            }
            return false;
        }
    private:
        double m_r = 1;
};

Square::Square(double r) : Shapes() {
    m_r = r;
}
```

which looks quite different from the implementation in Python. We will go thoroughly through the different part of the code. One of the first thing we observe, is that we here need to *declare* all the variables, which means that we need to specify what kind of variable it is. In the following we will have a look at some standard *data types*.

9.2.1 Built-in data types

In low-level languages, we need to specify the data types manually, giving us more control and flexibility. The most fundamental data types are `int` representing an integer number, `float` representing a floating number, `bool` representing a Boolean number and a `char` representing a

Table 9.1: Built-in data types in C++, with their memory occupation and range in a 64-bit processor, taken from [82]. Extensions in the parameters are optional.

Data types	Size [bytes]	Range
(signed) char	1	-2^7 to 2^7-1
(signed) short (int)	2	-2^{15} to $2^{15}-1$
(signed) int / long (int)	4	-2^{31} to $2^{31}-1$
(signed) long long (int)	8	-2^{63} to $2^{63}-1$
unsigned char	1	0 to 2^8
unsigned short (int)	2	0 to 2^{16}
unsigned int / unsigned long (int)	4	0 to 2^{32}
unsigned long long (int)	8	0 to 2^{64}
float	4	$\sim \pm$3.4E38 ($\sim$ 7 digits)
(long) double	8	$\sim \pm$1.7E308 ($\sim$ 15 digits)

character. Often, these four data types are not sufficient for the task we want to implement, and we need to introduce more data types. An especially common error is *overflow*, meaning that the number computed is out of the range of our data type. To avoid this, we can switch to data types of longer range, replacing `int` with a `long long int` (or just a `long long`), and replacing `float` with a `double`.

Some languages also deal with `long int` and `long double`, but in C++ an `int` and a `long int` is the same. Additionally, a `long double` is the same as a `double`. As lack of memory seldom is a problem on modern computers, it is common to declare all floating numbers as `doubles`. In table (9.1), the most common data types are listed with their range and memory occupation in C++. For integers, it is also possible to "move" the range from its zero-centered default to only include positive numbers. This is done by declaring an `unsigned` data type. By using unsigned data types, the range doubles in the positive direction, but it should not be done if one, by any chance, can get negative numbers, as one then will get *underflow*.

9.2.2 Access modifiers

Another thing that we observe from the code example above, is that we define the functions as `private` or `public`, which are called access modifiers. The access modifiers specify how much access the environment should have to the function: `private` means that the function can only be called from the particular class, while `public` means that also sub-classes (children) and other classes have access to the function. We also have a third access modifier in C++, called `protected`. `protected` members of a class A are not accessible outside of A's code, but any class that inherits from A has access to the protected member.

9.2.3 Pure virtual functions

In Python, we saw how a parent's function was overwritten by a function of the child. This works as long as the child's function has an identical name and identical arguments as the parent's function; otherwise it all will fail. In C++, a much safer method exists based on pure virtual functions, where a child is *forced* to have the same virtual functions as its parent. The virtual

functions of the parent therefore serves as a template of its children, such that all the children have the same features. This is often very useful, as the children tend to have the same features. Taking it back to the `Shapes` class, all the shapes ought to have features area and circumference, which can be forced by using virtual functions, as done in the example above.

9.2.4 Constructors and destructors

In the example above, we observe that the class `Shapes` has a function with the same name, `Shapes` and another virtual function `~Shapes`. The former one is used to create the object and is hence named a *constructor*. This function gets called automatically when the object of a class is created. Similarly, the latter one destructs an object when we are done evaluating the object. It is therefore named a *destructor* and is also called automatically when an object is deleted or goes out of scope.

9.2.5 Pointers

In the example above, we have not directly used any pointer. However, pointers are important in low-level languages, and we will here give a very brief introduction to pointers. For a C++ program, the memory of a computer is like a succession of memory cells, each with a unique address and of size one byte. When declaring a variable of size larger than one byte, the object will occupy memory cells that have consecutive addresses. Often, it is useful to know this address, which allows us to change the object directly in the memory. The address of a variable is given by the address-of operator, &, working as

```
address=&variable
```

By doing this, we say that we *point to the address*, and a variable that stores the address of another variable is thereby called a pointer. Furthermore, the dereference operator, *, gives the value of the address to where a pointer points. Thus, the following is true

```
variable=*address.
```

9.2.6 Back to the Monte Carlo example

We can now call the class above from the `main` function. We start defining the children `circ` and `squr`, and use the `getExtent()` function to approximate the area ratio between them. The code looks like

```
int main() {
    class Shapes* circ = new Circle(1);
    class Shapes* squr = new Square(1);

    for(int i=1; i<10; i++) {
        int M = int(pow(10,i));
        long long A_square = 0;
        long long A_circle = 0;
        for(int m=0; i<M; i++) {
            double x = uniform(gen);
            double y = uniform(gen);
            if(circ->getExtent(x, y)) {
                A_circle ++;
            }
            if(squr->getExtent(x, y)) {
                A_square ++;
            }
```

```
            }
            std::cout << std::fixed;
            std::cout << std::setprecision(10);
            std::cout << "Pi: " << 4 * double(A_circle)/A_square << std::endl;
        }
        return 0;
}
```

which gives about the same output as the Python script, but is much faster.

CHAPTER 10

Implementation: Variational Monte Carlo

> There are only two hard things in Computer Science: cache invalidation and naming things.
>
> Phil Karlton, [83]

In this chapter, de will describe the implemented variational Monte Carlo (VMC) code, which was developed from scratch in C++. As the code itself is around 7000 significant[1] lines of code, we will go through selected and often not obvious parts. As often said, *good planning is half the battle*, which largely relates to writing VMC code. The code was rewritten and restructured several times before we ended on the final version. As a starting point, we used the VMC framework implemented by Ledum [31], which was meant as an example implementation in the course *FYS4411 - Computational Physics II: Quantum Mechanical Systems*. The entire source code can be found on the authors GitHub account, http://www.github.com/evenmn/VMC.

We developed the code with a focus on three main goals: it should be efficient, flexible, and readable. It needs to be flexible in order to support the Boltzmann machines as our trial wave function guess, and since we will try out various Jastrow factors, it should be easy to add and remove wave function elements. Since quantum mechanical simulations, in general, are very expensive, it is crucial to develop efficient code to be able to study systems of some size. Lastly, we aim to write readable code such that others can reuse the code in its entirety or parts of it later. How we work to achieve the goals will be illustrated by code mainly picked from the WaveFunction class, which is the heart of the code.

For all matrix operations, we use the open-source template library for linear algebra Eigen throughout the code. Eigen provides an elegant interface, with support for all the needed matrix and vector operations. Additionally, Eigen is built on the standard software libraries for numerical linear algebra, BLAS, and LAPACK, which are incredibly fast. These contribute significantly to the performance of the code.

10.1 Flexibility and readability

We have done several things in order to keep the code as readable as possible. Firstly, the code was written in an object-orientated scheme which makes it more intuitive to a human as discussed in chapter 9. The Hamiltonians, optimizers, wave functions, sampling methods, and even the random number generator were treated as objects, making the code more or less as object-orientated as possible. An object-oriented code makes it also makes it straight-forward to

[1] Significant lines of code in this sense means lines that are not blank or commented. Counted by the *cloc* program [84].

define a system since we can specify the preferred object. Below, we define a two-dimensional quantum dot system of $N = 6$ electrons with frequency $\omega = 1.0$, learning rate $\eta = 0.1$, number of Metropolis cycles $M = 2^{20} = 1,048,576$ and max number of iterations set to 1000.

```
System *QD = new System();

QD->setNumberOfDimensions(2);
QD->setNumberOfParticles(6);
QD->setNumberOfMetropolisSteps(int(pow(2, 20)));
QD->setFrequency(1.0);
QD->setLearningRate(0.1);

QD->setBasis(new Hermite(QD));
QD->setHamiltonian(new HarmonicOscillator(QD));

QD->setWaveFunctionElement(new Gaussian(QD));
QD->setWaveFunctionElement(new SlaterDeterminant(QD));
QD->setWaveFunctionElement(new PadeJastrow(QD));

QD->runSimulation(numberOfIterations=1000);
```

Listing 10.1: Example on how a quantum dot system can be initialized.

We observe that one first needs to define an object to represent the system, and then the other settings are connected to the system in the form of subclasses. The reader might notice that we use the **lowerCamelCase** naming convention for function and variable names, which means that each word begins with a capital letter except the first word. For classes, we use the **UpperCamelCase** to distinguish from function names. Using the camel case is known to make the code readable, and apart from, for example, the popular **snake_case**, we do not need delimiters between the words, which saves some space. After the naming convention is decided, we are still responsible for giving reasonable names, which is not always an easy task, as Karlton points out. When one sees the name, one should know exactly what the variable/function/class is or does. Besides, as a code format, we use the ClangFormat, which provides a consequent way of formatting the code.

The snippet above also demonstrates how the code was made flexible when it comes to the wave function. One can construct a wave function consisting of various elements by simply calling the `setWaveFunctionElement` multiple times. This creates a wave function vector, `m_waveFunctionVector`, in the background containing all the elements, which makes it easy to compose the wave function in whatever preferred way. All the elements can be combined. The reader might stub on the use of the element `Gaussian`, is not the trial wave function defined by the Slater determinant multiplied with a Padé-Jastrow factor? It is, but as we will see later in section 10.3.1, the Gaussian part can be factorized out from the Slater determinant when using a Hermite basis. However, we will now start from the fundamental assumption that the trial wave function consists of a Slater determinant and a Jastrow factor, and take it from there.

10.2 Decompose the trial wave function

In our VMC implementation, the trial wave function, $\Psi_T(\boldsymbol{R})$, is assumed to consist of a single Slater determinant and a Jastrow factor to take care of the repulsive interactions. In section 7.1.1, we saw that the electronic Slater determinant can be split up in a spin-up part and a spin-down part, giving the trial wave function

$$\Psi_T(\boldsymbol{R}) = |\hat{D}_\uparrow(\boldsymbol{R})| \cdot |\hat{D}_\downarrow(\boldsymbol{R})| J(\boldsymbol{R}) \tag{10.1}$$

where $\boldsymbol{R}$ is the collective coordinates of all the particles, where we exclude the spin as it is assumed to not affect the energy. $J(\boldsymbol{R})$ is an arbitrary Jastrow factor, while the Slater determinant

is the determinant of the matrix $\hat{D}(\boldsymbol{R})$, henceforth the Slater matrix. To convince the reader that the Slater determinant and the Jastrow factor can be treated separately, we will consider a general trial wave function consisting of p wave function elements $\{\Psi_1, \Psi_2 \dots \Psi_p\}$,

$$\Psi_T(\boldsymbol{R}) = \prod_{i=1}^{p} \Psi_i(\boldsymbol{R}), \tag{10.2}$$

where $\Psi_i(\boldsymbol{R})$ in principle can be any function of the coordinates $\boldsymbol{R}$. However, we will later see that the parameter update simplifies if we restrict each variational parameter θ_j to appear in an element only. Before that, we will look at how the kinetic energy can be expressed in terms of independent factors.

10.2.1 Kinetic energy computations

The local energy computation is the heart of the VMC code, and the aim for an efficient and flexible code starts here. It was first defined in equation (7.16), and by inserting a general Hamiltonian $\hat{\mathcal{H}} = -1/2\nabla^2 + V$ with V covering all the potential energy, we obtain

$$E_L = \sum_{k=1}^{F} \Big[-\frac{1}{2}\Big(\frac{1}{\Psi_T(\boldsymbol{R})}\nabla_k^2 \Psi_T(\boldsymbol{R})\Big) + V\Big], \tag{10.3}$$

where we have $F = Nd$ degrees of freedom. As always, N is the number of particles and d is the number of dimensions. The first term, which is the kinetic energy term, is the only wave function-dependent one, and we will in this section split it up concerning the elements. The potential energy term, V, is not directly dependent on the wave function and will therefore not be further touched here.

From the definition of the derivative of a logarithm, we have that

$$\frac{1}{\Psi_T(\boldsymbol{R})}\nabla_k \Psi_T(\boldsymbol{R}) = \nabla_k \ln \Psi_T(\boldsymbol{R}), \tag{10.4}$$

which can be used to prove the following relation

$$\frac{1}{\Psi_T(\boldsymbol{R})}\nabla_k^2 \Psi_T(\boldsymbol{R}) = \nabla_k^2 \ln \Psi_T(\boldsymbol{R}) + (\nabla_k \ln \Psi_T(\boldsymbol{R}))^2. \tag{10.5}$$

Expressing the kinetic energy in terms of this relation is useful because most of the elements can be treated easily in the log-space. By using the fact that the trial wave function is a product of all the elements, the term above is calculated by

$$\frac{1}{\Psi_T(\boldsymbol{R})}\nabla_k^2 \Psi_T(\boldsymbol{R}) = \sum_{i=1}^{p} \nabla_k^2 \ln \Psi_i(\boldsymbol{R}) + \Big(\sum_{i=1}^{p} \nabla_k \ln \Psi_i(\boldsymbol{R})\Big)^2 \tag{10.6}$$

such that the total kinetic energy is given by

$$-\frac{1}{2}\frac{1}{\Psi_T(\boldsymbol{R})}\nabla^2 \Psi_T(\boldsymbol{R}) = -\frac{1}{2}\Big[\sum_{i=1}^{p} \nabla^2 \ln \Psi_i(\boldsymbol{R}) + \sum_{k=1}^{F}\Big(\sum_{i=1}^{p} \nabla_k \ln \Psi_i(\boldsymbol{R})\Big)^2\Big]. \tag{10.7}$$

This can be found when all local derivatives $\nabla^2 \ln \Psi_i(\boldsymbol{R})$ and $\nabla_k \ln \Psi_i(\boldsymbol{R})$ are given. By assuming that the former is returned by a function `computeLaplacian()` and the latter is returned by a function `computeGradient(k)`, we compute the kinetic energy using the following function

```
double System::getKineticEnergy()
{
    double kineticEnergy = 0;
    for (auto &i : m_waveFunctionElements) {
        kineticEnergy += i->computeLaplacian();
    }
    for (int k = 0; k < m_degreesOfFreedom; k++) {
        double nablaLnPsi = 0;
        for (auto &i : m_waveFunctionElements) {
            nablaLnPsi += i->computeGradient(k);
        }
        kineticEnergy += nablaLnPsi * nablaLnPsi;
    }
    return -0.5 * kineticEnergy;
}
```

Note that some of the variables are declared globally, here the vector m_waveFunctionElements and the integer m_degreesOfFreedom are denoted by an m_ to distinguish them from the variables declared locally.

10.2.2 Parameter gradients

In section 7.1.4, we presented how the parameters can be updated by minimizing the energy expectation value. We recall that the only closed-form expression needed in addition to the local energy is $\nabla_{\theta_j} \ln \Psi_T(\boldsymbol{r}_j)$, which needs to be found. By applying equation (10.2), we find that

$$\nabla_{\theta_j} \ln \Psi_T(\boldsymbol{R}) = \sum_{i=1}^{p} \nabla_{\theta_j} \ln \Psi_i(\boldsymbol{R}) = \nabla_{\theta_j} \ln \Psi_{\theta_j}(\boldsymbol{R}), \tag{10.8}$$

where $\Psi_{\theta_j}(\boldsymbol{R})$ is the only element which contains the parameter θ_j. With this in mind, we need to find closed-form expressions of $\nabla_{\theta_j} \ln \Psi_{\theta_j}(\boldsymbol{R})$ for all wave function elements $\Psi_{\theta_j}(\boldsymbol{R})$ that are associated with a variational parameter θ_j.

In the code, we store all the parameters in a *parameter matrix* where each element has its own row of parameters. Similarly, we create a *gradient matrix* of the same dimensions to store the gradients $\nabla_{\theta_j} \ln \Psi_{\theta_j}(\boldsymbol{R})$ for each variational parameter. The function which collect all the gradients is implemented straight-forwardly, and is given by

```
Eigen::MatrixXd System::getAllParameterGradients()
{
    for (int i = 0; i < m_numberOfElements; i++) {
        m_gradients.row(i) = m_waveFunctionElements[i]->computeParameterGradient();
    }
    return m_gradients;
}
```

where m_gradients has the same number of rows as the number of elements and the same number of columns as the maximum number of parameters in an element i. The function computeParameterGradient() returns a vector with all the gradients $\nabla_{\theta_j} \ln \Psi_i(\boldsymbol{R})$ of the respective element. Even though the gradients are used to update the parameters, we will postpone the discussion of the parameter update to section 10.6.

10.2.3 Probability ratio

In the two previous sections, we have seen how the derivatives of the wave function elements can be used in order to obtain the local energy and the parameter update. However, we also

need the evaluation of the wave function elements themselves to decide whether or not a move should be accepted. If we go back to equation (8.3), we see that what is actually needed is the ratio between the present and the previous probability, $P(\boldsymbol{R}_{\text{new}})/P(\boldsymbol{R}_{\text{old}})$. Further, we can write this as the product of the probability ratios of all the wave function elements,

$$\frac{P(\boldsymbol{R}_{\text{new}})}{P(\boldsymbol{R}_{\text{old}})} = \frac{|\Psi_T(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_T(\boldsymbol{R}_{\text{old}})|^2} = \prod_{i=1}^{p} \frac{|\Psi_i(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_i(\boldsymbol{R}_{\text{old}})|^2}, \tag{10.9}$$

again utilizing equation (10.2). Finding closed-form expressions for those ratios is not only beneficial because it is needed in the actual implementation, as we work in the log-space those ratios often take clean forms which are cheap to evaluate. Below, we will calculate those ratios for all the elements since we are going to use that directly in the sampling. We name the function returning the ratio for a particular element evaluateRatio(), and we obtain the total probability ratio in the following way

```
double System::evaluateProbabilityRatio()
{
    double ratio = 1;
    for (auto &i : m_waveFunctionElements) {
        ratio *= i->evaluateRatio();
    }
    return ratio;
}
```

With this, we have introduced the four central functions of the wave function elements: computeLaplacian(), computeGradient(k), computeParameterGradient() and evaluateRatio(). In the following, we will evaluate the Slater determinant and see how it can be split further in more elements.

10.3 Slater determinant

As we have seen above, the Slater determinant is the fundamental part of the trial wave function, together with the Jastrow factor. The main problem with the Slater determinant is that it is costly to deal with as the number of particles increases. To find the gradient of the Slater determinant, as requested by equation (10.7), we need to compute the inverse of the Slater matrix, which by standard LU decomposition scales as $\sim M^3$ for an $M \times M$ matrix [85]. Fortunately, there exist algorithms that let us obtain the inverse of the matrix by recursive relations, scaling as $\sim M^2$. This will be detailed in section 10.3.3.1.

Additionally, in section 7.1.1, we showed that the Slater determinant can be split in a spin-up part and a spin-down part, reducing its dimensionalities from $N \times N$ to two $N/2 \times N/2$ matrices, with N as the number of electrons. For an equal number of spin-up and spin-down electrons, this reduces in principle the cost of computing the Slater determinant with 87.5%! Also, factorizing out common factors from the Slater determinant will give some speed-up. In this section, we will mostly discuss the various methods to make the update of the Slater matrix more efficient. We will start from the general determinant containing spin-σ coordinates,

$$\Psi_{\text{sd}}(\boldsymbol{R}) = |\hat{D}_\sigma(\boldsymbol{R})| \propto \begin{vmatrix} \phi_1(\boldsymbol{r}_1) & \phi_2(\boldsymbol{r}_1) & \dots & \phi_N(\boldsymbol{r}_1) \\ \phi_1(\boldsymbol{r}_2) & \phi_2(\boldsymbol{r}_2) & \dots & \phi_N(\boldsymbol{r}_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(\boldsymbol{r}_N) & \phi_2(\boldsymbol{r}_N) & \dots & \phi_N(\boldsymbol{r}_N) \end{vmatrix}. \tag{10.10}$$

which we denote by Ψ_{sd} to emphasize that it is one of the wave function elements found in the product in equation (10.2). $\phi_j(\boldsymbol{r}_i)$ is the single-particle function occupying the matrix element D_{ij}.

10.3.1 Factorizing out elements

We have now seen how the Slater determinant can be split up in a spin-up part and a spin-down part. Before we evaluate these determinants, we should try to make the elements of the Slater matrices as simple as possible to save computational time. If all the elements have the same factor, the computations will get much cheaper if the factor is factorized out of the matrix. How this is possible can easiest be seen if we express the Slater determinant on a summation form,

$$\Psi_{\text{sd}}(\boldsymbol{R}) \propto \sum_q (-1)^q \hat{P} \phi_1(\boldsymbol{r}_1)\phi_2(\boldsymbol{r}_2)\dots\phi_N(\boldsymbol{r}_N), \tag{10.11}$$

where the sum runs over all the possible permutations and $\hat{P}$ is the permutation operator, permuting coordinates pairwise. If all the (spatial) single particle functions $\phi_j(\boldsymbol{r}_i)$ can be split in two functions $f_j(\boldsymbol{r}_i)$ and $g(\boldsymbol{r}_i)$ where the latter is common for all the single particle functions,

$$\phi_j(\boldsymbol{r}_i) = f_j(\boldsymbol{r}_i)g(\boldsymbol{r}_i) \tag{10.12}$$

the Slater determinant can be rewritten as

$$\begin{aligned}
\Psi_{\text{sd}}(\boldsymbol{R}) &\propto \sum_p (-1)^p \hat{P} f_1(\boldsymbol{r}_1)g(\boldsymbol{r}_1)f_2(\boldsymbol{r}_2)g(\boldsymbol{r}_2)\dots f_N(\boldsymbol{r}_N)g(\boldsymbol{r}_N) \\
&= g(\boldsymbol{r}_1)g(\boldsymbol{r}_2)\dots g(\boldsymbol{r}_N)\sum_p (-1)^p \hat{P} f_1(\boldsymbol{r}_1)f_2(\boldsymbol{r}_2)\dots f_N(\boldsymbol{r}_N) \\
&= \prod_{i=1}^N g(\boldsymbol{r}_i) \begin{vmatrix} f_1(\boldsymbol{r}_1) & f_2(\boldsymbol{r}_1) & \dots & f_N(\boldsymbol{r}_1) \\ f_1(\boldsymbol{r}_2) & f_2(\boldsymbol{r}_2) & \dots & f_N(\boldsymbol{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\boldsymbol{r}_N) & f_2(\boldsymbol{r}_N) & \dots & f_N(\boldsymbol{r}_N) \end{vmatrix}.
\end{aligned} \tag{10.13}$$

This is very useful when dealing with some common basises. For instance, the Hermite basis is given by

$$\phi_j(\boldsymbol{r}_i) \propto H_j(\boldsymbol{r}_i)\exp\left(-\frac{1}{2}\omega|\boldsymbol{r}_i|^2\right) \tag{10.14}$$

where $H_j(\boldsymbol{r}_i)$ are the Hermite polynomials and the Gaussian part fulfills the requirement of $g(\boldsymbol{r}_i)$. Therefore, we can construct a Slater determinant containing the Hermite polynomials only, treating the Gaussian as an independent element. This is not only preferable from an efficiency point of view, by doing this the variational parameter in the Gaussian is also removed from the determinant, which means that we can implement the determinant without worrying about the variational parameters. With this in mind, we will first treat the Gaussian element, obtaining its derivative and optimization schemes. Moreover, in section 10.3.3 we will discuss how the determinant can be treated efficiently.

10.3.2 Gaussian

When factorizing out the Gaussian part from the Slater determinant, we obtain the element

$$\Psi_{\text{sg}}(\boldsymbol{R};\alpha) = \prod_{j=1}^N g(\boldsymbol{r}_j) = \exp\Big(-\frac{1}{2}\omega\alpha\sum_{j=1}^N r_j^2\Big) = \exp\Big(-\frac{1}{2}\omega\alpha\sum_{j=1}^F x_j^2\Big), \tag{10.15}$$

with N as number of particles, d as the number of dimensions and $F = Nd$ as the degrees of freedom. ω is the oscillator strength and α is a variational parameter, which for non-interacting atoms is 1. Because of the presence of r_i^2 the function can easily be treated both in Cartesian

and spherical coordinates, but in this thesis we will focus on the former. We now use j as our summation index, and reserve i for the moved particle and k as our the differentiating index. When changing a coordinate x_i from x_i^{old} to x_i^{new}, the probability ratio can easily be found to be

$$\frac{|\Psi_{\text{sg}}(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_{\text{sg}}(\boldsymbol{R}_{\text{old}})|^2} = \exp\left(\omega\alpha\left((x_i^{\text{old}})^2 - (x_i^{\text{new}})^2\right)\right), \tag{10.16}$$

which is pretty cheap to evaluate. The gradient of $\ln\Psi_{\text{sg}}$ with respect to the coordinate x_k is

$$\nabla_k \ln\Psi_{\text{sg}} = -\omega\alpha x_k, \tag{10.17}$$

and the corresponding Laplacian is

$$\nabla^2 \ln\Psi_{\text{sg}} = -\omega\alpha F. \tag{10.18}$$

We observe that the factor $\omega\alpha$ is found in all the expressions above and only needs to be calculated again when the parameter α is updated, which is updated according to

$$\nabla_\alpha \ln\Psi_{\text{sg}} = -\frac{1}{2}\omega\sum_{j=1}^{F} x_j^2. \tag{10.19}$$

The implementation is very straight-forward, and looks like

```
double Gaussian::evaluateRatio()
{
    return m_probabilityRatio;
}

double Gaussian::computeGradient(const int k)
{
    return -m_omegalpha * m_positions(k);
}

double Gaussian::computeLaplacian()
{
    return -m_omegalpha * m_degreesOfFreedom;
}

Eigen::VectorXd Gaussian::computeParameterGradient()
{
    m_gradients(0) = -0.5 * m_omega * m_positions.cwiseAbs2().sum();
    return m_gradients;
}
```

where m_omegalpha is $\omega\alpha$. The probability ratio is calculated using

```
double void Gaussian::updateProbabilityRatio(int changedCoord)
{
    m_probabilityRatio = exp(m_omegalpha * (m_positionsOld(changedCoord) *
        ↪ m_positionsOld(changedCoord) - m_positions(changedCoord) *
        ↪ m_positions(changedCoord)));
}
```

We see that matrix-vector operations are used when it is possible, which makes the computations very fast.

10.3.3 The determinant

As discussed in section 7.1.1, the Slater determinant can be split in a spin-up part and a spin-down part, and further the common functions can be factorized out as shown in section 10.3.1. This means that the remaining determinant is not the full Slater determinant, and to distinguish it from the real Slater determinant, $\Psi_{\text{sd}}(\boldsymbol{R})$, we will denote the element by "det", $\Psi_{\text{det}}(\boldsymbol{R})$. This determinant can of course still be splitted like the Slater determinant,

$$\Psi_{\text{det}}(\boldsymbol{R}) = |\hat{D}_{\uparrow}(\boldsymbol{R}_{\uparrow})| \cdot |\hat{D}_{\downarrow}(\boldsymbol{R}_{\downarrow})|, \tag{10.20}$$

where $r_{\uparrow}$ are the coordinates of particles with spin up (defined as the first $N_{\uparrow}$ coordinates) and $r_{\downarrow}$ are the coordinates of particles with spin down (defined as the last $N_{\downarrow}$ coordinates).

We can now exploit the logarithmic scale, by using that the logarithm of a product corresponds to summarize the logarithm of each factor,

$$\ln \Psi_{\text{det}}(\boldsymbol{R}) = \ln |\hat{D}_{\uparrow}(\boldsymbol{R}_{\uparrow})| + \ln |\hat{D}_{\downarrow}(\boldsymbol{R}_{\downarrow})| \tag{10.21}$$

such that we only need to care about one of the determinants when differentiating, dependent on whether the coordinate we differentiate with respect to is among the spin-up or the spin-down coordinates,

$$\nabla_k \ln \Psi_{\text{det}}(\boldsymbol{R}) = \begin{cases} \nabla_k \ln |\hat{D}_{\uparrow}(\boldsymbol{R}_{\uparrow})| & \text{if} \quad k < N_{\uparrow} \\ \nabla_k \ln |\hat{D}_{\downarrow}(\boldsymbol{R}_{\downarrow})| & \text{if} \quad k \geq N_{\uparrow}. \end{cases} \tag{10.22}$$

Before we go further, we will introduce a more general notation which covers both the cases:

$$\hat{D}(\boldsymbol{R}) \equiv \hat{D}_{\sigma}(\boldsymbol{R}_{\sigma}) \tag{10.23}$$

where σ is the spin projection. When summarizing, the sum is always over all relevant coordinates. Furthermore, we have that

$$\nabla_k \ln |\hat{D}(\boldsymbol{R})| = \frac{\nabla_k |\hat{D}(\boldsymbol{R})|}{|\hat{D}(\boldsymbol{R})|} \tag{10.24}$$

and

$$\nabla_k^2 \ln |\hat{D}(\boldsymbol{R})| = \frac{\nabla_k^2 |\hat{D}(\boldsymbol{R})|}{|\hat{D}(\boldsymbol{R})|} - \left(\frac{\nabla_k |\hat{D}(\boldsymbol{R})|}{|\hat{D}(\boldsymbol{R})|} \right)^2 \tag{10.25}$$

which are consistent with the equations (10.4) and (10.5). At this point, there are (at least) two possible paths to the final expressions. We can keep on using matrix operations and find the expressions of $\nabla_k |\hat{D}(\boldsymbol{R})|/|\hat{D}(\boldsymbol{R})|$ and $\nabla_k^2 |\hat{D}(\boldsymbol{R})|/|\hat{D}(\boldsymbol{R})|$ using Jacobi's formula, or we can switch to element representations of the matrices. We choose the latter, because we believe that is the path of least resistance.

The determinant of an arbitrary matrix $\hat{A}$ can be expressed by its comatrix $\hat{C}$ in the following way,

$$|\hat{A}| = \sum_{ij} a_{ij} c_{ji} \tag{10.26}$$

where a_{ij} are the matrix elements of $\hat{A}$ and c_{ij} are the element of the comatrix. Going further, an element c_{ij} can be expressed in terms of an element from the inverse of $\hat{A}$, a_{ij}^{-1} [53],

$$c_{ij} = a_{ij}^{-1} |\hat{A}|. \tag{10.27}$$

Relating this to our particular problem, we can express

$$\begin{aligned}\frac{\nabla_k|\hat{D}(\boldsymbol{R})|}{|\hat{D}(\boldsymbol{R})|} &= \frac{\nabla_k \sum_{ij} d_{ij}c_{ji}}{\sum_{ij} d_{ij}c_{ji}} = \frac{\sum_j \nabla_k d_{kj}c_{jk}}{\sum_{ij} d_{ij}c_{ji}} \\ &= \frac{\sum_j \nabla_k d_{kj}d_{jk}^{-1}|\hat{D}|}{\sum_{ij} d_{ij}d_{ji}^{-1}|\hat{D}|} = \sum_j \nabla_k d_{kj}d_{jk}^{-1}\end{aligned} \tag{10.28}$$

where d_{ij} are elements of $\hat{D}$ and we have used the fact that the elements $\nabla_k d_{ij}$ contribute to the sum if and only if $i = k$, such that the sum over i collapses. Moreover, we use that multiplying a matrix with its inverse is identity, i.e, $\sum_{ij} d_{ij}d_{ji}^{-1} = 1$ and the determinants cancel. Similarly, we get

$$\frac{\nabla_k^2|\hat{D}(\boldsymbol{R})|}{|\hat{D}(\boldsymbol{R})|} = \sum_j \nabla_k^2 d_{kj}d_{jk}^{-1} \tag{10.29}$$

for the Laplacian. We are then ready to write up the final expressions for the gradient and Laplacian of the logarithm of the Slater determinant,

$$\begin{aligned}\nabla_k \ln|\hat{D}(\boldsymbol{R})| &= \sum_j d_{jk}^{-1}\nabla_k\phi_j(\boldsymbol{r}_k) \\ \nabla_k^2 \ln|\hat{D}(\boldsymbol{R})| &= \sum_j d_{jk}^{-1}(\boldsymbol{R})\nabla_k^2\phi_j(\boldsymbol{R}_k) - \Big(\sum_j d_{jk}^{-1}\nabla_k\phi_j(\boldsymbol{r}_k)\Big)^2\end{aligned} \tag{10.30}$$

where we have used that $d_{ij} = \phi_j(\boldsymbol{r}_i)$ with $\phi_j(\boldsymbol{r}_i)$ as the spatial functions found in the Slater determinant, see above.

10.3.3.1 Efficient calculation of the determinant

As aforementioned, dealing with the Slater determinant is very computational expensive, mainly because of the requirement of the inverse Slater matrix. However, by revealing that only one row in the Slater matrix is updated for each step, we can update the inverse recursively. We use the same element representation as above, and express the ratio between the new and the old determinant as

$$R \equiv \frac{|\hat{D}(\boldsymbol{R}_{\text{new}})|}{|\hat{D}(\boldsymbol{R}_{\text{old}})|} = \frac{\sum_j d_{ij}(\boldsymbol{R}_{\text{new}})c_{ij}(\boldsymbol{R}_{\text{new}})}{\sum_j d_{ij}(\boldsymbol{R}_{\text{old}})c_{ij}(\boldsymbol{R}_{\text{old}})} = \sum_j d_{ij}(\boldsymbol{R}_{\text{new}})d_{ji}^{-1}(\boldsymbol{R}_{\text{old}}) \tag{10.31}$$

which is very similar to the calculation given in equation (10.28). To calculate the inverse matrix $\hat{D}^{-1}$ efficiently, we need to calculate

$$S_j = \sum_{l=1}^{N} d_{il}(\boldsymbol{R}_{\text{new}})d_{lj}^{-1}(\boldsymbol{R}_{\text{old}}) \tag{10.32}$$

for all columns but the one associated with the moved particle, i. For all columns where $j \neq i$, we then find the new elements using

$$d_{kj}^{-1}(\boldsymbol{R}_{\text{new}}) = d_{kj}^{-1}(\boldsymbol{R}_{\text{old}}) - \frac{S_j}{R}d_{ki}^{-1}(\boldsymbol{R}_{\text{old}}) \tag{10.33}$$

while the remaining column, i, is updated using the simple formula [53]

$$d_{ki}^{-1}(\boldsymbol{R}_{\text{new}}) = \frac{1}{R}d_{ki}^{-1}(\boldsymbol{R}_{\text{old}}). \tag{10.34}$$

Those procedures makes the inverting scale as $\sim M^2$ instead of $\sim M^3$, which is largely beneficial for large systems.

We assume that we do not have any variational parameter in the determinant, and obtain three expressions of the case when a particle with spin up is moved and three of the case when a particle with spin down is moved,

$$\begin{aligned} &\text{if} \quad k < N_\uparrow : \\ &\frac{|\Psi_{\text{sd}}(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_{sd}(\boldsymbol{R}_{\text{old}})|^2} = \frac{|\hat{D}_\uparrow(\boldsymbol{R}_\uparrow^{\text{new}})|^2}{|\hat{D}_\uparrow(\boldsymbol{R}_\uparrow^{\text{old}})|^2} \\ &\nabla_k \ln |\hat{D}_\uparrow(\boldsymbol{R}_\uparrow)| = \sum_{j=1}^{N_\uparrow} \nabla_k d_{jk}(\boldsymbol{R}_\uparrow) d_{kj}^{-1}(\boldsymbol{R}_\uparrow) \end{aligned} \tag{10.35}$$

$$\begin{aligned} &\text{if} \quad k \geq N_\uparrow : \\ &\frac{|\Psi_{\text{sd}}(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_{sd}(\boldsymbol{R}_{\text{old}})|^2} = \frac{|\hat{D}_\downarrow(\boldsymbol{R}_\downarrow^{\text{new}})|^2}{|\hat{D}_\downarrow(\boldsymbol{R}_\downarrow^{\text{old}})|^2} \\ &\nabla_k \ln |\hat{D}_\downarrow(\boldsymbol{R}_\downarrow)| = \sum_{j=N_\uparrow}^{F} \nabla_k d_{jk}(\boldsymbol{R}_\downarrow) d_{kj}^{-1}(\boldsymbol{R}_\downarrow) \end{aligned} \tag{10.36}$$

$$\nabla^2 \ln |\hat{D}(\boldsymbol{R})| = \sum_{k=1}^{F} \left[\sum_{j=1}^{F} \nabla_k^2 d_{jk}(\boldsymbol{R}) d_{kj}^{-1}(\boldsymbol{R}) - \left(\sum_{j=1}^{F} \nabla_k d_{ik}(\boldsymbol{R}) d_{ki}^{-1}(\boldsymbol{R}) \right)^2 \right] \tag{10.37}$$

We have now presented the theory behind finding the ratio between the new and the old probability and the gradients of the determinant (equation (10.35-10.37)), and we have described how we efficiently can find the inverse of the Slater matrix (equation (10.31-10.34)). However, to make things more clear, we will outline some selected parts of the Slater determinant implementation.

In the code, we create a Slater matrix, m_slaterMatrix where all the elements are stored. This matrix contains both $\hat{D}_\uparrow$ and $\hat{D}_\downarrow$, and has therefore dimensions $N \times N/2$. Furthermore, we store the gradient of the elements concerning all the F elements in a matrix m_slaterMatrixDer, which naturally gets the dimensions $F \times N/2$. We also create a matrix m_slaterMatrixSecDer to store all the Laplacians of the elements, which also has the dimensions $F \times N/2$. In all of them, we only need to update a row when a particle is moved, which makes it quite efficient. The m_slaterMatrixDer is updated in the following way

```
void SlaterDeterminant::updateSlaterMatrixDerRow(const int row)
{
    int particle = int(row / m_numberOfDimensions);
    int dimension = row % m_numberOfDimensions;
    for (int col = 0; col < m_numberOfParticlesHalf; col++) {
        m_slaterMatrixDer(row, col) = m_basis->basisElementDer(col, dimension,
            ↪ m_positions.col(particle));
    }
}
```

where each element is taken from the basisElementDer function in the Basis class. This function returns just the derivative of the single particle function called for the chosen basis set. Note

also that only the coordinates of the moved particle, stored in a column of the m_positions matrix, is needed for the update. The update of m_slaterMatrix and m_slaterMatrixSecDer are very straightforward and similar to the example above, so we will not discuss them further.

Something that might be less intuitive is how to update the inverse of the Slater matrix. We also store this in a dedicated matrix m_slaterMatrixInverse, and we use LU decomposition only to initialize it. After that, we use the formulas above to update the inverse. We implement it as

```
void SlaterDeterminant::updateRatio()
{
    m_ratio = m_slaterMatrix.row(m_particle) * m_slaterMatrixInverse.col(m_particle);
}

void SlaterDeterminant::updateSlaterMatrixInverse(int start, int end)
{
    updateRatio();
    for (int j = start; j < m_particle; j++) {
        double S = m_slaterMatrix.row(m_particle) * m_slaterMatrixInverse.col(j);
        m_slaterMatrixInverse.col(j) -= S * m_slaterMatrixInverse.col(m_particle) /
            ↪ m_ratio;
    }
    for (int j = m_particle+1; j < end; j++) {
        double S = m_slaterMatrix.row(m_particle) * m_slaterMatrixInverse.col(j);
        m_slaterMatrixInverse.col(j) -= S * m_slaterMatrixInverse.col(m_particle) /
            ↪ m_ratio;
    }
    m_slaterMatrixInverse.col(m_particle) /= m_ratio;
}
```

where m_ratio is a global variable also returned by the function evaluateRatio (see section 10.2.3). Note that the loops never affect the i'th columns, where particle i is moved (in the code denoted by the global variable m_particle). The arguments to the function updateSlaterMatrixInverse specify which part of the matrix that should be updated, based on whether the moved particle has spin-up or spin-down. We will end our discussions of the Slater determinant by presenting the implementation of the gradient and the Laplacian of the logarithm of the determinant. These were decided to be stored in the vectors m_determinantDer and m_determinantSecDer for $\nabla_k|\hat{D}(\boldsymbol{R})|/|\hat{D}(\boldsymbol{R})|$ and $\nabla_k^2|\hat{D}(\boldsymbol{R})|/|\hat{D}(\boldsymbol{R})|$ respectively. These vectors are updated using vector operations in the following fashion

```
void SlaterDeterminant::updateSlaterDeterminantDerivatives(int start, int end)
{
    for (int i = start * m_numberOfDimensions; i < end * m_numberOfDimensions; i++) {
        int particle = int(i / m_numberOfDimensions);
        m_determinantDer(i) = m_slaterMatrixDer.row(i) *
            ↪ m_slaterMatrixInverse.col(particle);
        m_determinantSecDer(i) = m_slaterMatrixSecDer.row(i) *
            ↪ m_slaterMatrixInverse.col(particle);
    }
}
```

We avoid a double loop by taking a inner product instead. However, we are left with one loop which can also be avoided using smart matrix operations.

10.4 Jastrow factor

The second part of a standard VMC trial wave function is the Jastrow factor, which is meant to take care of the electron-electron correlations. The optimization scheme of this element is not as complex as the determinant, and this section will, therefore, be notably shorter than the previous. We will first discuss the two Jastrow factors given in section 7.1.2 and chapter **??**: the

simple Jastrow and the Padé-Jastrow factor, and then we look at how the distance matrix can be updated efficiently.

10.4.1 Simple Jastrow factor

Recall the simple Jastrow factor from (7.37),

$$\Psi_{\text{sj}}(\boldsymbol{R};\boldsymbol{\beta}) = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N}\beta_{ij}r_{ij}\right). \tag{10.38}$$

with N as the number of particles, r_{ij} as the distance between particle i and j and β_{ij} as variational parameters. This is a quite simple element, but one challenge is that we operate in Cartesian coordinates, while the expressed Jastrow factor obviously is easier to deal with in polar coordinates. Since we need to differentiate this with respect to all degrees of freedom, we need to be attentive not confusing the particle indices with the coordinate indices. Let us reserve j' as the coordinate index and j as the index of the corresponding particle. The relationship between j' and j is then *always* $j = j' \setminus D$, where the backslash means integer division. The other way around, we have $j' = j + d$ where d is the respective dimension of the coordinate j'. With that notation, the probability ratio is given by

$$\frac{|\Psi_{\text{sj}}(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_{\text{sj}}(\boldsymbol{R}_{\text{old}})|^2} = \exp\left(2\sum_{j=1}^{N}\beta_{ij}(r_{ij}^{\text{new}} - r_{ij}^{\text{old}})\right) \tag{10.39}$$

where i' again is the moved particle. The gradient is straight-forward to find, and reads

$$\nabla_{k'}\ln\Psi_{\text{sj}} = \sum_{j=1}^{N}\beta_{kj}\frac{x_{k'} - x_{j'}}{r_{kj}} \tag{10.40}$$

where j' is related to the same dimension as k'. Here we use $x_{j'}$ as a general coordinate, no matter if it is associated with the x-direction or not. This also applies to the Laplacian,

$$\nabla^2\ln\Psi_{\text{sj}} = \sum_{k'=1}^{F}\sum_{j=1}^{N}\frac{\beta_{kj}}{r_{kj}}\left[1 - \left(\frac{x_{k'} - x_{j'}}{r_{kj}}\right)^2\right]. \tag{10.41}$$

Finally, the parameter update is given by

$$\nabla_{\beta_{ml}}\ln\Psi_{\text{sj}} = r_{ml}. \tag{10.42}$$

For this element, the most important thing we can do to keep the computational cost as low as possible is to reveal that only a row and a column of the distance matrix is changed as we change a coordinate. Updating the entire distance matrix means updating N^2 elements, while updating a row and a column means updating $2N$ elements, which is an essential difference for large systems. This is detailed in section 10.4.3.

We also observe that the factor $(x_{k'} - x_{j'})/r_{kj}$ is found in both the gradient and the Laplacian, so by storing this matrix, we can speed-up the computations. Most naturally, the matrix has dimensions $F \times F$, but using that we only are interested in the elements where $x_{k'}$ and $x_{j'}$ have the same dimension and that the diagonal is zero, only F of the elements need to be found. Further,

the matrix is obviously anti-symmetric, so we only need to calculate $F/2$ of the elements. When one particle is moved, only the elements related to the moving particle need to be updated, which is $2(N-1)$ elements. Again, we utilize that the matrix is anti-symmetric and get the following efficient update scheme

```
void SimpleJastrow::updatePrincipalDistance(int i)
{
    int i_d = i % m_numberOfDimensions;
    for (int j_p = 0; j_p < i_p; j_p++) {
        int j = i_d + j_p * m_numberOfDimensions;
        m_principalDistance(i, j) = (m_positions(i) - m_positions(j)) /
            ↪ m_distanceMatrix(i_p, j_p);
        m_principalDistance(j, i) = -m_principalDistance(i, j);
    }
    for (int j_p = i_p + 1; j_p < m_numberOfParticles; j_p++) {
        int j = i_d + j_p * m_numberOfDimensions;
        m_principalDistance(i, j) = (m_positions(i) - m_positions(j)) /
            ↪ m_distanceMatrix(i_p, j_p);
        m_principalDistance(j, i) = -m_principalDistance(i, j);
    }
}
```

with `i_p` as the moved particle, `i_d` as the direction the particle is moved in and `i` as the coordinate index. Similarly, the loop goes over the particles `j_p` with the associated coordinate `j`. Note that we split the loop in two parts to avoid calculating the distance from a particle to itself. This trick is also done in many of the other functions in the simple Jastrow class, also in the Padé-Jastrow factor class.

10.4.2 The Padé-Jastrow factor

The Padé-Jastrow factor is a more complicated Jastrow factor, and was specified in equation (7.12),

$$\Psi_{\text{pj}}(\boldsymbol{R};\beta) = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{a_{ij}r_{ij}}{1+\beta r_{ij}}\right), \tag{10.43}$$

where β is a variational parameter and the a_{ij} is **not** a variational parameter, but rather constants dependent on the spin of particles i and j. Similarly to the simple Jastrow, we also here need to distinguish between particle indices and coordinate indices because of the radial distances r_{ij}. We do the same trick with denoting j' by the coordinate index and j as the particle index, and obtain the gradient

$$\nabla_{k'}\ln\Psi_{\text{pj}} = \sum_{j\neq k=1}^{N}\frac{a_{kj}}{(1+\beta r_{kj})^2}\frac{x_{k'}-x_{j'}}{r_{kj}} \tag{10.44}$$

with respect to the coordinate $x_{k'}$. By again differentiating this with respect to $x_{k'}$, we obtain the Laplacian

$$\nabla^2\ln\Psi_{\text{pj}} = \sum_{k'=1}^{F}\sum_{j\neq k=1}^{N}\frac{a_{kj}}{(1+\beta r_{kj})^2}\left[1-\left(1+2\frac{\beta r_{kj}}{1+\beta r_{kj}}\right)\left(\frac{x_{k'}-x_{j'}}{r_{kj}}\right)^2\right]\frac{1}{r_{kj}}. \tag{10.45}$$

Similar to the simple Jastrow factor, we again observe the factor $(x_{k'}-x_{j'})/r_{kj}$ in both the gradient and the Laplacian, which can be stored as a matrix and updated in the same way as described for the simple Jastrow factor. The last expression we need is the one used to update the variational parameter β, which is found to be

$$\nabla_{\beta}\ln\Psi_{\text{pj}} = -\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{a_{ij}r_{ij}^2}{(1+\beta r_{ij})^2}. \tag{10.46}$$

In addition to the factor $g_{i'j'} \equiv (x_{k'} - x_{j'})/r_{kj}$, there are multiple factors that we can store to make the computations cheaper. The factor $f_{ij} \equiv a_{ij}/(1+\beta r_{ij})^2$ is found both in the gradient, Laplacian and parameter gradient, and storing it will save a significant amount of computational time. Lastly, the factor $h_{ij} \equiv r_{ij}/(1+\beta r_{ij})$ is found in several places and will be stored as well. As a summary, we use

$$f_{ij} = \frac{a_{ij}}{(1+\beta r_{ij})^2} \qquad g_{i'j'} = \frac{x_{i'} - x_{j'}}{r_{ij}} \qquad h_{ij} = \frac{r_{ij}}{1+\beta r_{ij}}. \tag{10.47}$$

and obtain the simplified expressions

$$\begin{aligned}
\frac{|\Psi_{\text{pj}}(\boldsymbol{R}_{\text{new}})|^2}{|\Psi_{\text{pj}}(\boldsymbol{R}_{\text{old}})|^2} &= \exp\left(2\sum_{j=1}^{N} a_{ij}(h_{ij}^{\text{new}} - h_{ij}^{\text{old}})\right) \\
\nabla_{k'} \ln \Psi_{\text{pj}} &= \sum_{j\neq k=1}^{N} f_{kj} \cdot g_{k'j'} \\
\nabla^2 \ln \Psi_{\text{pj}} &= \sum_{k'=1}^{F} \sum_{j\neq k=1}^{N} \frac{f_{kj}}{r_{kj}} \left[1 - (1+2\beta h_{kj}) g_{k'j'}^2\right] \\
\nabla_\beta \ln \Psi_{\text{pj}} &= -\sum_{l=1}^{N}\sum_{j>l}^{N} a_{lj} h_{lj}^2 = -\sum_{l=1}^{N}\sum_{j>l}^{N} f_{lj} r_{lj}^2
\end{aligned} \tag{10.48}$$

with unmarked indices (j) as the particle related ones and the marked (j) as the coordinate related ones. i is the moved particle. We now proceed further to the update of the distance matrix, which is where we can find the remaining optimization possibilities.

10.4.3 Updating the distance matrix

The distance matrix, which is used in the Jastrow factors, gives an illustrating example on how we can avoid repeating calculations. The matrix, henceforth named M, contains the relative distances between all the particles, for three particles given by

$$M = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} = \begin{pmatrix} 0 & r_{12} & r_{13} \\ r_{12} & 0 & r_{23} \\ r_{13} & r_{23} & 0 \end{pmatrix} \tag{10.49}$$

where r_{ij} means the distance between particles i and j. Since $r_{ij} = r_{ji}$ and $r_{ii} = 0$, the matrix becomes symmetric with zeros on the diagonal, which means that we only need to calculate $N(N-1)/2$ elements instead of N^2. Further, we can utilize that only a particle is moved at a time, which means that only a row and a column are changed when a particle is moved. For instance, if particle 1 is moved, the upper row and the left-hand-side column in matrix M need to be updated. In our program, we have implemented this in the following way

```
double Metropolis::calculateDistanceMatrixElement(const int i, const int j)
{
    double dist = 0;
    int parti   = m_numberOfDimensions*i;
    int partj   = m_numberOfDimensions*j;
    for(int d=0; d<m_numberOfDimensions; d++) {
        double diff = m_positions(parti+d)-m_positions(partj+d);
        dist += diff*diff;
    }
    return sqrt(dist);
```

```
}

void Metropolis::calculateDistanceMatrixCross(const int particle) {
    for(int i=0; i<m_numberOfParticles; i++) {
        m_distanceMatrix(particle, i) = calculateDistanceMatrixElement(particle, i);
        m_distanceMatrix(i, particle) = m_distanceMatrix(particle, i);
    }
}
```

where the function calculateDistanceMatrixElement(i,j) returns element i, j of the matrix, which is called from the function calculateDistanceMatrixCross(particle). The latter takes the moved particle index as input, and updates the necessary row and column of the matrix.

For systems of non-interacting particles, the distance matrix is redundant, and should therefore not be calculated. We have solved this by giving all the wave function elements and the Hamiltonians a number which indicated whether they require the distance matrix or not, as mentioned above. If no part of the code needs the distance matrix, it is never calculated. We also calculate the radial position globally when any part of the code requires it. The components are stored in a vector named radialVector, applying the same optimization ideas as the distance matrix.

10.5 Sampling

Also, when it comes to the sampling itself, there exist optimization schemes to speed-up the process. Remember that the sampling algorithm often is repeated millions of times for each iteration, so even a small impact can give a massive speed-up. We will initially present the brute force sampling implementation in its entirety before we move on to the importance of sampling implementation. For the latter, we will discuss the optimization possibilities and connect them to the actual implementation.

10.5.1 Brute force sampling

The brute force sampling was introduced in section 8.0.1, and is the most basic sampling method implemented. The sampling function BruteForce::acceptMove, which returns true if the move is accepted, is implemented as

```
bool BruteForce::acceptMove()
{
    int i = m_RNG->nextInt(m_degreesOfFreedom);

    m_positionsOld = m_positions;
    m_radialVectorOld = m_radialVector;
    m_distanceMatrixOld = m_distanceMatrix;

    m_positions(i) += (m_RNG->nextDouble() - 0.5) * m_stepLength;
    if (m_calculateDistanceMatrix) {
        Metropolis::calculateDistanceMatrixCross(int(i / m_numberOfDimensions));
    }
    if (m_calculateRadialVector) {
        Metropolis::calculateRadialVectorElement(int(i / m_numberOfDimensions));
    }
    m_system->updateAllArrays(m_positions, m_radialVector, m_distanceMatrix, i);

    double p = m_system->evaluateProbabilityRatio();
    if (p < m_RNG->nextDouble()) {
        m_positions = m_positionsOld;
        m_distanceMatrix = m_distanceMatrixOld;
        m_radialVector = m_radialVectorOld;
        m_system->resetAllArrays();
        return false;
```

```
    }
    return true;
}
```

where i is the changed coordinate which is drawn from the random number generator m_RNG. Initially the old positions, radial vector and distance matrix are stored in case the move is rejected, and then a new move is proposed in positive or negative direction. If the radial vector or distance matrix (or both) are needed somewhere in the code, they are updated in this function, using the ideas and implementation presented in section 10.4.3. The they are distributed to the wave function elements using the function updateAllArrays.

In the end, the probability ratio is evaluated using the function evaluateProbabilityRatio presented in section 10.2.3. If this ratio is larger than a random number between 0 and 1, the move is accepted, and otherwise, we set all the arrays back to the old ones (also the ones in the wave function elements).

10.5.2 Importance sampling

The importance of sampling implementation is very similar to the brute force sampling implementation, and we will, therefore, not repeat it. However, we need to calculate the quantum force and the ratio between the new and the old Green's function, which can be calculated in clever ways to keep the code efficient.

We have already seen that the quantum force takes the same form as the gradient of the trial wave function, $\boldsymbol{F}(\boldsymbol{R}) = 2(\nabla\Psi_T(\boldsymbol{R}))/\Psi_T(\boldsymbol{R})$, and we can therefore simply reuse the function computeGradient, which is a part of the local energy computations from section 10.2.1. We call this from the function ImpotanceSampling::QuantumForce, which contains the few lines of code

```
double ImportanceSampling::QuantumForce(const int i)
{
    double QF = 0;
    for (auto &j : m_waveFunctionVector) {
        QF += j->computeGradient(i);
    }
    return 2 * QF;
}
```

where the force in dimension i is returned. The Green's function was first presented in section 8.0.2, and at first glance it might look computational expensive to evaluate. Fortunately, we only need the ratio between the old and the new function which can be found in a quite simple fashion. Actually, both the diffusion constant D and the time step Δt cancel in the exponent, and the ratio can be expressed in the elegant form

$$g(\boldsymbol{R}', \boldsymbol{R}, \Delta t) \equiv \frac{G(\boldsymbol{R}', \boldsymbol{R}, \Delta t)}{G(\boldsymbol{R}, \boldsymbol{R}', \Delta t)} = \exp\big((\boldsymbol{R}' - \boldsymbol{R}) \cdot (\boldsymbol{F}(\boldsymbol{R}) - \boldsymbol{F}(\boldsymbol{R}'))/2\big) \tag{10.50}$$

where $\boldsymbol{R}$ and $\boldsymbol{R}'$ differ by one element and so does $\boldsymbol{F}(\boldsymbol{R})$ and $\boldsymbol{F}(\boldsymbol{R}')$. It can therefore be evaluated in a very efficient scheme

```
double ImportanceSampling::GreenRatio(const int i)
{
    double dQF = m_quantumForceOld(i) - m_quantumForceNew(i);
    return exp(0.5 * dQF * m_dx) + 1;
}
```

where dQF is the difference between the new and the old force and m_dx is the distance particle i is moved. 1 appears from the term where m_dx is zero, such that we get zero in the exponent.

10.6 Update of parameters

The parameter update is a central part of a VMC implementation, and a good VMC implementation requires a good optimization algorithm. Since the optimization functions are called outside the sampling, they are just called a fraction of times, compared to the function called from the sampling. Therefore, we will not put too much effort into making them efficient, but they should still be thought-through. We will here discuss the gradient descent method with momentum and monotonic decaying step and the ADAM optimizer. The Optimizer class contains a pure virtual function updateParameters which is thus forced to be included in the optimizer subclasses. This function returns the update of the new parameters and is the function we will discuss in this section.

10.6.1 Gradient descent

Gradient descent is a simple optimization algorithm, and so is the implementation. Based on the theory presented in section 5.6.1, the implementation is really straight-forward and reads

```
Eigen::MatrixXd GradientDescent::updateParameters()
{
    m_step += 1;
    double monotonic = 1 / pow(m_step, m_monotonicExp);
    m_v = m_gamma * m_v + m_eta * Optimization::getEnergyGradient() * monotonic;
    return m_v;
}
```

where m_v is the momentum vector and m_monotonicExp describes how fast the rate should decrease. Further, m_gamma is the momentum parameter defining the relative size of the momentum. The function Optimization::getEnergyGradient returns a matrix with the gradients of the energy expectation value with respect to all the parameters, given in equation (7.21).

10.6.2 ADAM optimizer

The ADAM optimizer implementation also is very straight-forwardly based on the algorithm given in section 5.6.4. The momentum vectors were implemented as matrices to match the dimensions of the parameter matrix. By matrix operations, we could also have made the function efficient, but since that is not the aim here, we decided to keep the loops in order to make the code readable. The implementation looks like

```
Eigen::MatrixXd ADAM::updateParameters()
{
    m_step += 1;
    m_g = Optimization::getEnergyGradient();
    m_m = m_beta1 * m_m + (1 - m_beta1) * m_g;
    m_v = m_beta2 * m_v + (1 - m_beta2) * m_g.cwiseAbs2();
    m_mHat = m_m / (1 - pow(m_beta1, m_step));
    m_vHat = m_v / (1 - pow(m_beta2, m_step));
    for (int i = 0; i < m_numberOfElements; i++) {
        for (int j = 0; j < m_maxParameters; j++) {
            m_theta(i, j) = m_eta * m_mHat(i, j) / (sqrt(m_vHat(i, j) + m_epsilon));
        }
    }
    return m_theta;
}
```

where effort was made naming variables consistently with what we did in section 5.6.4. The parameter matrix, named m_parameters, can then easily be updated by the code

```
if (m_myRank == 0) {
    m_sampler->computeAverages();
    m_parameters -= m_optimization->updateParameters();
}
```

where m_optimization is the specified optimizer and m_myrank is the *rank* of the process. Parallel processing is not discussed yet, but we will describe it briefly in the following section.

10.7 Parallel processing

The code was parallelized using MPI to make studies of large systems possible. This means that the code can run multiple parallel treads and in that manner, utilize the processors. Most notably, this allows us to run on computer clusters which typically reduce the running time with a factor 10-100. We will not explain how MPI works in detail, nor will we detail the implementation of MPI since the commands are distributed over the entire code. The thing we present is a sketch of the idea behind the parallelization used for our particular code.

One of the things that makes VMC preferred over other many-body methods is that the algorithm quite easy can be split into independent parts, which encourages parallelization. The entire sampling can be split into as many parallel processes as needed, such that the code can be run on an arbitrary number of CPUs. We typically distinguish between wall clock time, t_{clock} and CPU time t_{cpu} where the former is the time measured by a clock, and the latter is the total computation time from all the CPUs. The speed-up will in general not be 100%, i.e., $t_{clock} \neq t_{cpu}/n$ with n as the number of processes, mainly because all of the samplings should have the same burn-in period as if we only run one process, but also because the code that is not part of the sampling cannot be parallelized and needs to be run on the same CPU. The process that takes care of this part is the primary process with rank 0.

In algorithm 3, we have sketched very roughly how the parallelization goes. We first run the entire sampling individually for all the n processes, and if something goes wrong, we call the MPI_Abort function. To align the processes before we collect all the cumulative values, we use the function MPI_Barrier and we use MPI_Reduce for the actual collection. After that, the average energies are calculated *by the main process only*, and in the end, the updated parameters are broadcast to all the other processes. Note that this is just a sketch where we avoid the arguments and the actual implementation of the MPI functions. This is of course found in the code.

10.8 Electron density

We presented the theory behind the electron density in section 3.3, where we saw that the P-body density is given by an integral over all probability density functions $|\Psi(\boldsymbol{R}_1, \ldots, \boldsymbol{R}_N)|^2$ but P of them. Usually, we look at the one-body density or the two-body density, leaving out one or two particles from the integration. Further in section 7.1.5, we gave a brief explanation of how the one-body density can be found using Monte Carlo integration in a VMC scheme. In this section, we will discuss the technique in more detail, and of course, give the actual implementation.

In our particular implementation, we have technically calculated the one-body density in two different ways; dividing the space into annuluses[2] to calculate the radial electron distribution and dividing the space into a grid to calculate the spatial electron distribution. The former is convenient when we want to present the density in a two-dimensional plot, making a comparison between multiple methods easy. Often, the one-body density is only dependent on the

[2] An annulus is a ring-shaped object with a region bounded by two concentric circles.

Algorithm 3: Sketch of the parallelization.

```
MPI_Init() (Initialize MPI);
while not converged do
    E_L = 0;
    gradient = 0;
    Egradient = 0;
    for i ← 1 to M do
        E_L += (Ĥ Ψ)/Ψ;
        gradient += ∇_θ ln Ψ;
        Egradient += (Ĥ Ψ)/Ψ * ∇_θ ln Ψ;
    end
    if something goes wrong then
        MPI_Abort() (Terminate all processes);
    end
    MPI_Barrier() (Align processes);
    MPI_Reduce(E_L, gradients, Egradients) (Collect cumulative values);
    if myrank == 0 then
        Ē_L = E_L/M;
        gradient‾ = gradient/M;
        Egradient‾ = Egradient/M;
        G = 2 * (Egradient‾ − Ē_L * Egradient‾);
        θ −= ηG;
    end
    MPI_Bcast(θ) (Broadcast parameters);
end
MPI_Finalize() (Finalize MPI);
```

Result: Optimal variational parameters θ.

radial distance from the center, and then it is sufficient to look at the radial density profile. On the other hand, the spatial density profile contains more information between the position of the particles, which is interesting when the density is also dependent on the angle.

For the radial density profile, we in practice deal with bins formed as annuluses of equal width Δr, the two-dimensional case is illustrated in figure (10.1). This means that the bins do not have an equal extent, and we need to compensate for this by dividing by the respective volume. In two dimensions, a bin i has the area

$$A_i = (2i+1)\pi\Delta r^2 \tag{10.51}$$

and in three dimensions the volume of bin i is

$$V_i = 4(i(i+1)+1/3)\pi\Delta r^3. \tag{10.52}$$

The most intuitive way of finding the correct bin of a particle, is to loop through all the bins and check if the particles belongs to the particular bin.however, this is a rather inefficient method of doing it, and it can be done much smarter revealing that a particle of radius r belong to the bin of index

$$i = r \setminus \Delta r + 1 \tag{10.53}$$

where $\setminus$ indicates integer division.

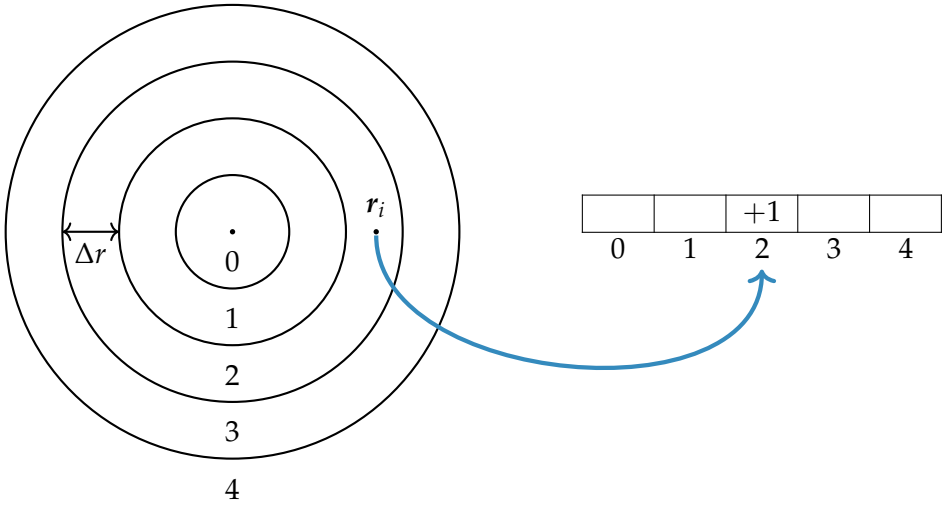


Figure 10.1: This figure is meant to illustrate how the one-body density is calculated using Monte-Carlo integration. One divides the space into n bins (here 5), and count the number of particles in each bin throughout the sampling. Afterward, the bins need to be normalized.

By defining a vector m_particlesPerBin with the length number of bins, we can find the number of particles in each bin by a simple loop over all particles,

```
void Sampler::computeOneBodyDensity(const Eigen::VectorXd radialVector)
{
    for (int i_p = 0; i_p < m_numberOfParticles; i_p++) {
        int bin = int(radialVector(i_p) / m_radialStep) + 1;
        m_particlesPerBin(bin)++;
    }
}
```

where m_radialStep is the width of each annulus, above denoted by d. In the end, m_particlesPerBin is printed to file, and we do the normalization when a script reads this file, see plot_ob_density.py. For the case where we look at the spatial density distribution, all the bins have equal size, and the approach is therefore straight-forward. Also, for the two-body density, we can choose to either calculate the radial or the spatial density distribution, but as the spatial distribution becomes many-dimensional, we stick to the radial distribution for this quantity. This distribution is calculated in a similar way to the one-body density, and it will not be further detailed.

10.9 Random number generator

In Monte Carlo integration, we are dependent on random numbers that are received from a random number generator (RNG). The RNG should have two main properties: It should give many independent, uncorrelated random numbers and it should be fast. The former depends on the *period* of the RNG, where a long period gives many independent numbers.

In this work, we have used the Mersenne Twister random number generator, as it has a period of $2^{19937} - 1$ which is known as the Mersenne prime. This is an incredibly large number and should be more than sufficient for our purpose. We use the built-in package in C++, std::mt19937, which is also quite fast.

CHAPTER 11

Implementation: Restricted Boltzmann Machines

In the previous chapter, we described common optimization procedures for a standard variational Monte Carlo (VMC) implementation, and we also presented implementation examples taken from the code. In this section, we will do the same, but for the restricted Boltzmann machines (RBM). As we have pointed out before, the same sampling methods and optimization algorithms can be used both for the VMC implementation and the RBM implementation, which means that much of the VMC framework can be reused. The already described parts of the code will naturally not be described again, and for that reason, this chapter will more or less exclusively concern the RBM wave function elements.

The main goal of this work is to reduce the physical intuition needed when doing quantum computations, and that is the task of the restricted Boltzmann machines. The idea is to use a flexible basis set based on RBMs, which needs to be the elements of the Slater matrix, as first seen in section 3.2.2. Further, we showed that the Slater determinant could be split in a spin-up part and a spin-down part in section 10.3.3, such that the spin can be factorized out and avoided. We therefore only need the spatial part of the wave functions, and we will henceforth assume that this spatial part is defined by the marginal distribution of the visible units, as suggested by Carleo & Troyer [15, 70]. Even though we want to reduce the need for physical intuition of the system, we still need to use some intuition to get reasonable results. For instance, for quantum dot systems, we add the Hermite polynomials to the marginal distribution such that each basis function becomes unique. The spatial part of the RBM single-particle functions for quantum dots then read

$$\phi_n(\boldsymbol{x}) = H_n(\boldsymbol{x})P(\boldsymbol{x}) \tag{11.1}$$

where $H_n(\boldsymbol{x})$ is the possibly multi-dimensional Hermite polynomial of degree n and $P(\boldsymbol{x})$ is the marginal distribution of the visible nodes. In section 10.3.1, we saw that a Slater determinant containing single-particle functions in the form of $\phi_j(\boldsymbol{r}_i) = f_j(\boldsymbol{r}_i)g(\boldsymbol{r}_i)$ can be simplified by factorizing out the function $g(\boldsymbol{r}_i)$. For that reason, we can treat the marginal distributions $P(\boldsymbol{r})$ as separate elements in combination with the determinant containing Hermite polynomials. In the following section, we will describe how these elements can be treated in the code.

11.1 Restricted Boltzmann machines

Back in chapter 6, we presented the marginal distribution of the visible units of a Gaussian-binary restricted Boltzmann machine. As this part can be factorized out of the Slater determinant, it can be treated as a separate wave function element,

$$\Psi_{\text{rbm}}(\boldsymbol{x};\boldsymbol{a},\boldsymbol{b},\boldsymbol{w}) = \exp\Big(-\sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma^2}\Big)\prod_{j=1}^{H}\Big(1+\exp\Big(b_j+\sum_{i=1}^{F}\frac{w_{ij}x_i}{\sigma^2}\Big)\Big), \tag{11.2}$$

where $\boldsymbol{x}$ contains all the coordinates, $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{w}$ are variational parameters (weights), σ is the width of the Gaussian distribution, F is the degrees of freedom and H is the number of hidden units. The wave function can naturally be split in a Gaussian part and a product, and we will henceforth work with them separately to simplify the calculations. They will also be implemented as separate wave function elements as this will reduce the complexity of the derivatives associated with each element. The first part will henceforth be denoted by RBM-Gaussian, while the last part will be denoted by RBM-Product.

11.1.1 RBM-Gaussian

The RBM-Gaussian is just the first part of equation (11.2) and reads

$$\Psi_{\text{rg}}(\boldsymbol{x};\boldsymbol{a}) = \exp\left(-\sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma^2}\right) \tag{11.3}$$

which is really similar to the simple Gaussian presented in section 10.3.2. Also the gradient, Laplacian and the gradient with respect to the variational parameters become similar, and we will for that reason just list them up,

$$\begin{aligned}
\frac{|\Psi_{\text{rg}}(\boldsymbol{x}_{\text{new}})|^2}{|\Psi_{\text{rg}}(\boldsymbol{x}_{\text{old}})|^2} &= \exp\left(\frac{(x_i^{\text{old}}-a_i)^2-(x_i^{\text{new}}-a_i)^2}{2\sigma^2}\right)\\
\nabla_k \ln\Psi_{\text{rg}} &= -\frac{x_k-a_k}{\sigma^2}\\
\nabla_k^2 \ln\Psi_{\text{rg}} &= -\frac{1}{\sigma^2}\\
\nabla_{a_l} \ln\Psi_{\text{rg}} &= \frac{x_l-a_l}{\sigma^2}.
\end{aligned} \tag{11.4}$$

Further, the frequency of the quantum dots should be inversely proportional to the Gaussian sampling width from the Gaussian-binary RBM, σ^2, such that we can set

$$\omega = \frac{1}{\sigma^2} \tag{11.5}$$

for the RBMs to account for the oscillator frequency. An obvious optimization concerning this element, is that we can introduce a vector $\boldsymbol{xa} \equiv \boldsymbol{x}-\boldsymbol{a}$, which we deal with instead of the position vector $\boldsymbol{x}$ and the parameter vector $\boldsymbol{a}$. We update the arrays using the pure virtual function `updateArrays`, which looks like

```
void RBMGaussian::updateArrays(const Eigen::VectorXd positions,
                               const Eigen::VectorXd radialVector,
                               const Eigen::MatrixXd distanceMatrix,
                               const int i)
{
    m_positions = positions;
    m_Xa = positions - m_a;
    double sqrdDiff = m_XaOld(i) * m_XaOld(i) - m_Xa(i) * m_Xa(i);
    m_probabilityRatio = exp(sqrdDiff / (2 * m_sigmaSqrd));
}
```

We see that the vector `m_Xa`, corresponding to $\boldsymbol{xa}$, is declared globally such that it can be used also in the gradients of the element. `i` is again the updated coordinate.

11.1.2 RBM-product

The RBM product is the last part of (6.19), and is thus given by

$$\Psi_{\text{rp}}(\boldsymbol{x};\boldsymbol{b},\boldsymbol{w}) = \prod_{j=1}^{H}\left[1+\exp\left(b_j+\sum_{i=1}^{F}\frac{w_{ij}x_i}{\sigma^2}\right)\right]. \tag{11.6}$$

In appendix C, section (C.3), a general Gaussian-binary RBM product in the form of

$$\Psi(\boldsymbol{x};\boldsymbol{\theta}) = \prod_{j=1}^{H}\left[1+\exp\left(f_j(\boldsymbol{x};\boldsymbol{\theta})\right)\right] \tag{11.7}$$

is differentiated, which for this element corresponds to setting $f_j = b_j + \boldsymbol{w}_j^T\boldsymbol{x}/\sigma^2$. As we further claim, the only expressions that need to be calculated are $\nabla_k(f_j)$, $\nabla_k^2(f_j)$ and $\partial_{\theta_i}(f_j)$ for all the coordinates k and all the parameters θ_i. They can easily be found to be

$$\begin{aligned}\nabla_k(f_j) &= \frac{w_{kj}}{\sigma^2}\\ \nabla_k^2(f_j) &= 0\\ \partial_{b_l}(f_j) &= \delta_{lj}\\ \partial_{w_{ml}}(f_j) &= \frac{x_m}{\sigma^2}\delta_{lj}\end{aligned} \tag{11.8}$$

for our specific function. δ_{lj} is the Kronecker delta. By reintroducing the sigmoid function and its counterpart

$$n_j(x) = \frac{1}{1+\exp(-x)} \quad \wedge \quad p_j(x) = n_j(-x) = \frac{1}{1+\exp(x)} \tag{11.9}$$

we can express the required derivatives in the following fashion

$$\begin{aligned}\frac{|\Psi_{\text{rp}}(\boldsymbol{x}_{\text{new}})|^2}{|\Psi_{\text{rp}}(\boldsymbol{x}_{\text{old}})|^2} &= \prod_{j=1}^{H}\frac{p_j(\boldsymbol{x}_{\text{old}})^2}{p_j(\boldsymbol{x}_{\text{new}})^2}\\ \nabla_k \ln\Psi_{\text{rp}} &= \sum_{j=1}^{H}\frac{w_{kj}}{\sigma^2}n_j\\ \nabla_k^2 \ln\Psi_{\text{rp}} &= \sum_{j=1}^{H}\frac{w_{kj}^2}{\sigma^4}p_j n_j\\ \nabla_{b_l} \ln\Psi_{\text{rp}} &= n_l\\ \nabla_{w_{ml}} \ln\Psi_{\text{rp}} &= \frac{x_m n_l}{\sigma^2}.\end{aligned} \tag{11.10}$$

This is the same result as obtained for the gradient of the log-likelihood function presented in chapter 5. In this element, there are plenty of optimization possibilities. For the RBM-Gaussian, we saw that the distribution width σ was set such that $\omega = 1/\sigma^2$, but for this product the value of σ has no impact on the outcome since it is always multiplied with the weights which are adjusted freely. By further revealing that some sums are vector products, we can get a significant speed-up. Firstly, we will define a vector

$$\boldsymbol{v} = \boldsymbol{b} + \boldsymbol{w}^T\boldsymbol{x} \tag{11.11}$$

which is what we above have called $f(\boldsymbol{x};\boldsymbol{\theta})$. Thereafter, we define the vectors $\boldsymbol{n}$ and $\boldsymbol{p}$ as described above. These vectors are declared as m_v, m_n and m_p respectively, and are initialized and updated using the function updateVectors in the following way

```
void RBMProduct::updateVectors()
{
    m_v = m_b + mW.transpose() * m_positions;
    Eigen::VectorXd e = m_v.array().exp();
    m_p = (e + Eigen::VectorXd::Ones(m_numberOfHiddenNodes)).cwiseInverse();
    m_n = e.cwiseProduct(m_p);
}
```

One can see that all the operations are vectorized, which makes the operations quite affordable.

11.2 Partly restricted Boltzmann machine

For the partly restricted Boltzmann machine given in equation (6.26), we observe that the only difference from a standard Boltzmann machine is the factor

$$\Psi_{\text{pr}} = \exp\Big(\sum_{i=1}^{F}\sum_{j=1}^{F} x_i c_{ij} x_j\Big) \tag{11.12}$$

which we can threat separately. To run a computation with the partly restricted Boltzmann machine, we thus need to add the elements RBMGaussian, RBMProduct and PartlyRestricted in a similar way as in the example 10.1. When differentiating, we end up with the expressions

$$\begin{aligned}
\frac{|\Psi_{\text{pr}}(\boldsymbol{x}_{\text{new}})|^2}{|\Psi_{\text{pr}}(\boldsymbol{x}_{\text{old}})|^2} &= \exp\Big(2\sum_{j=1}^{F} c_{ij}x_j(x_i^{\text{new}} - x_i^{\text{old}})\Big)\\
\nabla_k \ln\Psi_{\text{pr}} &= 2\sum_{j=1}^{F} c_{kj}x_j\\
\nabla_k^2 \ln\Psi_{\text{pr}} &= 2c_{kk}\\
\nabla_{c_{ml}} \ln\Psi_{\text{pr}} &= x_m x_l
\end{aligned} \tag{11.13}$$

where x_i is the changed coordinated. Also here can we use vectorization to speed-up the computations, most elegantly shown by the computeParameterGradient,

```
Eigen::VectorXd PartlyRestricted::computeParameterGradient()
{
    Eigen::MatrixXd out = m_positions * m_positions.transpose();
    m_gradients.head(out.size()) = WaveFunction::flatten(out);
    return m_gradients;
}
```

where we use that the parameter gradient $\nabla_{c_{ml}} \ln\Psi_{\text{pr}}$ is given by the outer product between the coordinate vectors, as already hinted in equation (6.27).

Part V

Results and Discussion

CHAPTER 12

Selected Results

Results! Why, man, I have gotten a lot of results. I know several thousand things that won't work.

Thomas A. Edison, [86]

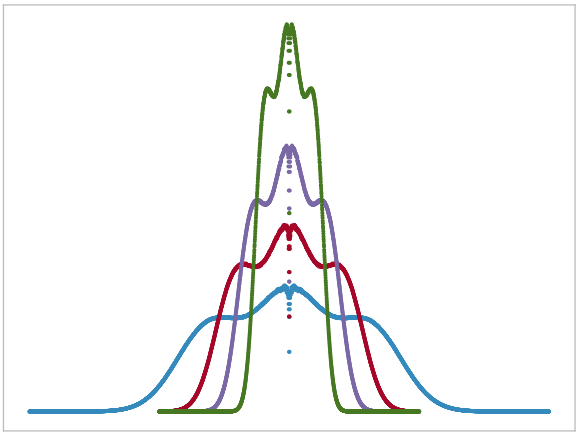


Figure 12.1: Radial one-body density profiles for two-dimensional quantum dots with $N = 12$ electrons, popularly titled an artificial Magnesium atom. The four graphs correspond to four different oscillator frequencies, where the weakest oscillator gives the broadest density distribution. It's quite artistic, isn't it?

We are finally ready to discuss the most exciting part of this thesis, namely the results. Since this work, after all, is about physics, the physical insight should be our focus and in this chapter we finally get the opportunity to discuss pure physics. However, we will also use this section for the comparison of the restricted Boltzmann machine (RBM) and the standard variational Monte Carlo method (VMC) to decide which one that is the better and in which situations they differ. We will look at three versions of the RBM; a Slater determinant with the Boltzmann machines as the single-particle functions (RBM), as described in chapter 11, an RBM with the simple Jastrow factor described in section 7.3.2 (RBM+SJ) and an RBM with the Padé-Jastrow factor described in section 7.1.2 (RBM+PJ). For VMC, we use the Slater-Jastrow ansatz, i.e., the trial wave function consists of a Padé-Jastrow factor and a Slater determinant containing standard single-particle functions, as detailed in chapter 10.

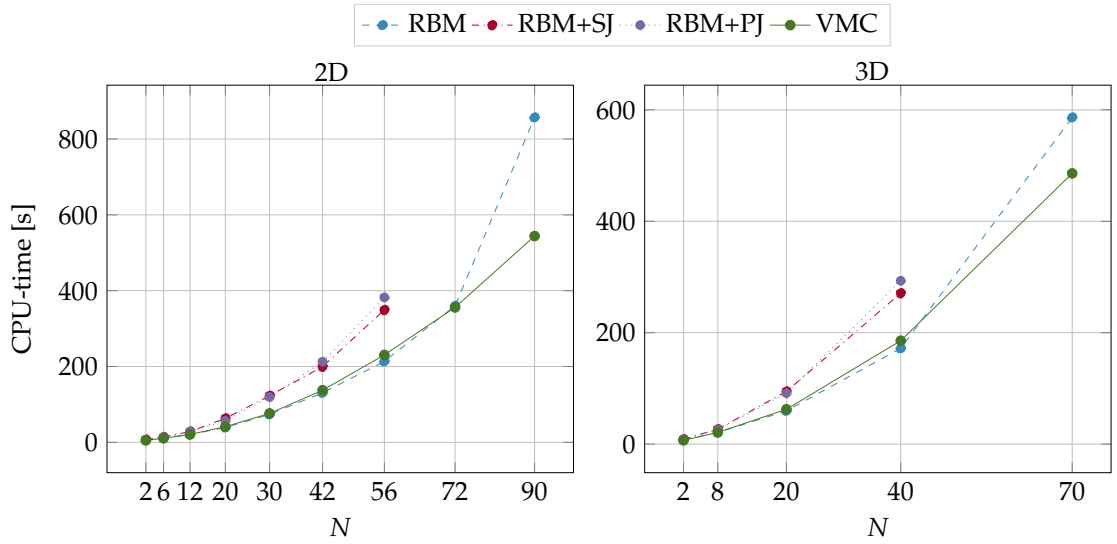


Figure 12.2: CPU-time per iteration for $M = 2^{20} = 1,048,576$ cycles as a function of the number of electrons, N, for two- dimensional (left) and three-dimensional (right) quantum dots. Table with numbers and more information about simulations can be found in appendix D, section D.1. For abbreviations see the text.

As we are capable of studying various systems, system sizes N, system strengths ω, and innumerous hyper-parameter settings by different methods, the number of results we are able to produce is uncountable. In total, we have looked at more than 150 different quantum dot systems using the four methods mentioned above, which means that we have generated a large set of results. Just a small subset of those results will be presented and discussed in this chapter, while a more extensive collection of results is presented in appendix D. The results presented below were selected since they describe physically interesting properties and compare the various methods in an informative way. Typically, we will choose systems and configurations that are comparable to references, such that we can benchmark the results. Our primary focus will be on ground state energy and electron density calculations, and we will stick to natural units as discussed in appendix B. A brief recap is that for quantum dots, the energy is given in units of Planck's reduced constant, $E' = E/\hbar$, length is scaled as $x' = x/\sqrt{\hbar/m}$ where m is the mass of a particle, which implies that the d-dimensional density, $\rho_d(\boldsymbol{r})$, is given in units of $(\hbar/m)^{-d/2}$. For atoms, we use Hartree atomic units, meaning that the length is given in units of the Bohr radius, a_0, the d-dimensional density is given in units of a_0^{-d} and the energy is scaled as $E' = E/(\hbar^2/m_e a_0)$.

Before we move on to the physical results, we will take a quick look at some more technical results, more precisely the computational cost of various wave function structures and the energy convergence using various optimization tools. For validation purposes, we will present a few selected results on the case without repulsive interaction and compare it to analytical results. After that, we study the case with repulsive interaction on a much larger scale, where we first look at quantum dots and then on double atoms.

12.1 Computational cost

Many-body quantum simulations are frequently found on lists of the most computationally expensive simulations for scientific purposes, which is caused by the large amount of information that is stored in the wave function. Albeit the VMC method is known to have a high

Table 12.1: The scaling of the computational cost of two-dimensional (2D) and three-dimensional (3D) quantum dots as a function of the number of electrons. The numbers presented in the table are the optimal b-value found fitting the power function $f(N) = 0.5N^b$ to the cost graph. For abbreviations see the text.

	RBM	RBM+SJ	RBM+PJ	VMC
2D	1.498	1.639	1.621	1.515
3D	1.584	1.710	1.729	1.605

performance-cost ratio, it is still not cheap. In this section, we will find the average cost of VMC and compare it to the cost of the RBM methods described above. In figure (12.2), the CPU-time is plotted as a function of the number of electrons for two-dimensional (left) and three-dimensional (right) quantum dots. To obtain accurate CPU-times, all the simulations were run on the Abel computer cluster with $M = 2^{20} = 1,048,576$ Monte Carlo cycles per iteration. The time presented is the average time over at least four independent runs with thousands of iterations each. To see the actual numbers that the plots are based on, go to appendix D, section D.1.

Our immediate observation is that the methods are pairwise quite similar, with RBM and VMC as the cheapest methods, and RBM+SJ and RBM+PJ as the most expensive ones. This is not surprising, as the RBM requires a neural network, VMC requires a Jastrow factor while RBM+SJ and RBM+SJ requires both a neural network and a Jastrow factor. For two-dimensional dots, the RBM is the cheapest among all the methods, but for larger systems ($N = 42, 56$ with N as the number of electrons), VMC gets cheaper due to an explosion in CPU-time for the RBM. This explosion can be explained by our choice of the number of hidden nodes, H, which consequently is set to the number of electrons, i.e., $H = N$, which by Gjestvang & Nordhagen [76] was found to give the lowest energy for small quantum dots. Since the RBM has $N \cdot d \cdot (1 + H) + H$ variational parameters with d as the number of dimensions, the number of variational parameters for a two-dimensional dot with $N = 90$ electrons is 16,470! On the other hand, the VMC method is equipped with two variational parameters for all system sizes, which obviously make the parameter update less costly. We also observe that the RBM+PJ is cheaper than the RBM+SJ for systems up to $N = 42$, but after that, the RBM+SJ gets slightly cheaper. This might be surprising, as the simple Jastrow contains N^2 variational parameters, but a possible explanation is that BLAS is optimized for large matrix-vector operations and is thus fully utilized first when the matrices get large. We observe the same behavior for the three-dimensional dots as for the two-dimensional dots, and the discussion above is representative for them as well.

The standard way of estimating the scaling of a VMC algorithm is to fit the power function $f(x) = ax^b$ to the cost graph. As all the simulations were performed with the same hyperparameters and with equal external factors, we fix the first parameter a and focus on the second parameter b only. It is this latter parameter that specifies the scaling, i.e.; we say that the method scales as N^b. For comparison reasons, we set $a = 0.5$ as this was found to be a good average value. In table (12.1), the optimal b from linear regression is presented for our four methods in two and three dimensions. We want to emphasize that we only did the regression for CPU-times up to $N = 56$ in two dimensions and $N = 40$ in three dimensions, partly because we only have data for all methods in this interval and partly because the CPU-time for the RBM explodes for large dots. We believe that this was the best way to do it in order to make the various methods comparable.

The numbers in the table match our impression from figure (12.2), where RBM and VMC pairwise were found to be more expensive than RBM+SJ and RBM+PJ. For all the methods, the

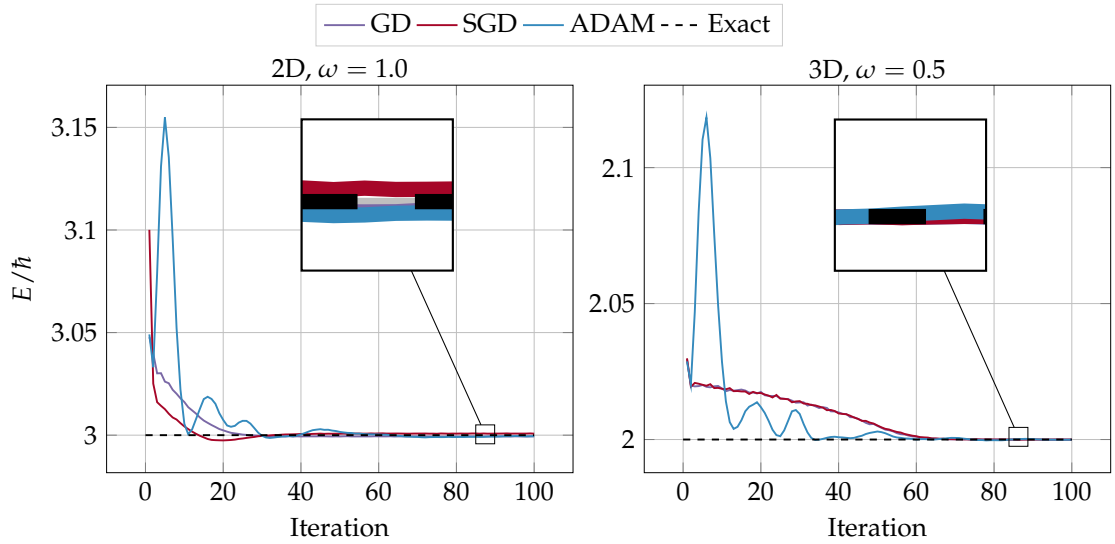


Figure 12.3: Convergence of estimated ground state energy, E, for quantum dots with $N = 2$ calculated using the VMC method, detailed in the text. We compare the optimization tools gradient descent (GD), stochastic gradient descent with 10 batches (SGD) and the ADAM optimizer. The left-hand side plot shows a two-dimensional quantum dot of frequency $\omega = 1.0$ with exact energy $E = 3.0$ [42]. The right-hand side plot shows a three-dimensional quantum dot of frequency $\omega = 0.5$ with exact energy $E = 2.0$ [43]. The learning rate was set to $\eta = 0.5$ and the number of Metropolis steps was $M = 2^{24} = 16,777,216$. All energies are given in units of $\hbar$ (natural units).

scaling was found to be between linear and quadratic as a function of the number of electrons, which is surprising as the update of the Slater matrix scales as $\sim N^2$.

12.2 Energy convergence

We want our calculations to converge fast and to be stable, and that is what the optimization tool is responsible for. In figure (12.3), we compare standard gradient descent (GD) to stochastic gradient descent (SGD) and ADAM for quantum dots with $N = 2$ interacting electrons in two and three dimensions. The optimization algorithms were detailed in section 5.6. The gradient descent methods are plain, i.e., without momentum and adaptive learning rate, while the ADAM optimizer has momentum and adaptivity by nature. The frequency $\omega = 1.0$ is used for the two-dimensional case since we know that the exact energy is $E = 3.0$ for this case [42]. Similarly, we use the frequency $\omega = 0.5$ for the three-dimensional case since the exact energy is $E = 2.0$ [43].

The first thing we observe is that all three optimization tools manage to converge to the exact energy (see spy window). The stochastic and non-stochastic gradient descent methods behave similarly, but we observe that the SGD goes below the exact energy before it stabilizes. The ADAM optimizer, on the other hand, fluctuates much more, which can be described by the momentum, as discussed in section 5.6.3. It is also important to remember that we use VMC, which is equipped with two variational parameters only, and thus is easier to control. The ADAM optimizer is known to be good at machine learning problems where we have many variational parameters, so we will stick to it even though gradient descent seems like a clever choice seen from the figure. Another point is that the circular quantum dots have neat potentials without local minima. When we move on to more complex systems, ADAM generally works better according to the literature.

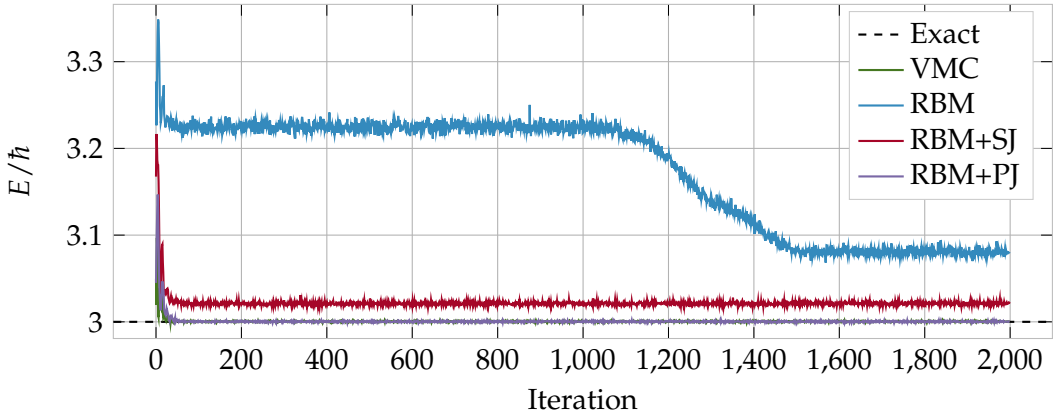


Figure 12.4: Energy convergence for quantum dots where we use the ADAM optimizer. We look at a two-dimensional quantum dot with $N = 2$ electrons and frequency $\omega = 1.0$, and compare VMC, RBM, RBM+SJ and RBM+PJ to the exact value $E = 3.0$ found by Taut [42]. The learning rate was set to $\eta = 0.5$ and the number of Metropolis steps used for each iteration was $M = 2^{20} = 1,048,576$. All energies are given in units of $\hbar$ (Hartree units), see text for abbreviations.

Further, we compare the energy convergence of the various methods in figure (12.4). VMC and RBM+PJ apparently give the lowest energy, followed by the RBM+SJ and RBM. Also the scale of the fluctuations (noise) seems to be in the same order, agreeing with the zero-variance property [70]. While VMC, RBM+SJ and RBM+PJ all immediately converge and stabilize, RBM first converges to a value before it after more than 1000 iterations converges further to a new value. This illustrates that it can be hard to decide whether or not the RBM has converged, and an intelligent system is needed in order to see if the simulation can be stopped or not. In practice, we often turn of the convergence check and let the simulation run the maximum number of iterations. It is apparent that the RBM is harder to train than the other methods, which is a result of the absence of a Jastrow factor.

12.3 No repulsive interaction

We start with the non-interacting case in order to validate the implemented code. For this case, we know the exact energy and the exact electron density for both quantum dots and atoms, which makes it a good test for the implementation. The physical significance is though limited as such systems do not appear in the real world. We will start with the quantum dot, and then move on to atoms.

12.3.1 Quantum dots

The quantum dot system is the system we will keep most of the attention on, and for that reason, we will also validate the implemented quantum dot code thoroughly. As we use the harmonic oscillator potential to describe the attraction in the quantum dots, the system has well-known analytical results for the most common observable. The quantum dots have analytical ground state energies given by equation (4.4) and the number of particles in a closed-shell given by the magic numbers in equation (4.5). For some selected number of electrons and frequencies $\omega = 0.5$ and $\omega = 1.0$, we compare our obtained energies to the analytical energies in the table (12.2). We observe that both the standard VMC, with $\alpha = 1$, and standard RBM, with all parameters set to zero, can reproduce the analytical expression. This is as expected since the exact wave

functions are found when the parameters have these particular values. RBM+SJ and RBM+PJ were omitted as we expect them to give similar results to RBM and VMC.

Table 12.2: The energy of quantum dots with frequencies $\omega = 0.5$, 1.0 and N non-interacting electrons. Exact values are obtained by $E = \omega(n + 1/2)$, and all values are given in units of $\hbar$. The standard error is zero down to machine precision, and for abbreviations see the text.

		$\omega = 0.5$			$\omega = 1.0$		
	N	RBM	VMC	Exact	RBM	VMC	Exact
2D	2	1.0	1.0	1	2.0	2.0	2
	12	14.0	14.0	14	28.0	28.0	28
	30	55.0	55.0	55	110.0	110.0	110
3D	2	1.5	1.5	1.5	3.0	3.0	3
	20	30.0	30.0	30	60.0	60.0	60
	70	157.5	157.5	157.5	315.0	315.0	315

We will also focus on the electron density throughout the results, and compare the obtained densities to the analytical ones is a good indicator of whether the implementation is correct or not. In figure (12.5) the radial one-body density is plotted for quantum dots with $N = 2$ non-interacting electrons and frequency $\omega = 1.0$ in two- (left) and three dimensions (right). We use VMC and RBM, and compare to the exact density profile found from the definition of an electron density in section 3.12. We observe that both VMC and RBM are more or less able to reproduce the exact distribution, which indicates that we have done the implementation correctly. However, the densities get noisy when they approach $r = 0$, most notably in three dimensions, which is caused by the way we calculate the radial one-body density. In section 10.8, we saw that the innermost bins are also the smallest, which means that particles are less likely to be observed there when the number of Monte Carlo cycles is finite. Apart from this noise, the density plot in two and three dimensions are identical, which can be explained mathematically by the fact that the non-interacting electron densities are trivially separable.

Furthermore, we present the spatial one-body distribution and radial two-body distribution and compare to analytical results in figure (12.6), in order to validate also these implementations. For the one-body density plots, we see that the both RBM, VMC and the exact plot are very similar. The radial two-body density plots are found in the same way as the radial one-body density plots, and we again observe some noise close to $r = 0$. We also get a cross on the simulated density profile, which occurs since we in practice only simulate one quadrant and add the three other quadrants to make the plot more intuitive and illustrative. Apart from that, the simulated two-body density plots for RBM and VMC seem to match the exact plot. As we have seen before, the electron densities are separable without interaction, which means that the radial two-body density presented here simply is the radial one-body density rotated in space.

As discussed before, the electron density should be normalized such that the integral over the density function ρ corresponds to the number of particles N. For the one-body density, this corresponds to $\int dr\rho(r) = N$ for the radial density profile and $\int dxdy\rho(x, y) = N$ for the spatial density distribution and for the two-body radial density profile the normalization condition is $\int dr_i dr_j \rho(r_i, r_j) = N$. However, there is no standard way of normalizing these densities when they are calculated using bins like us, as the normalization constant depends on the number of bins *et cetera*. This is not very important either; we are only interested in the relative densities and the shapes of the density plots. We have been consequent when normalizing the density

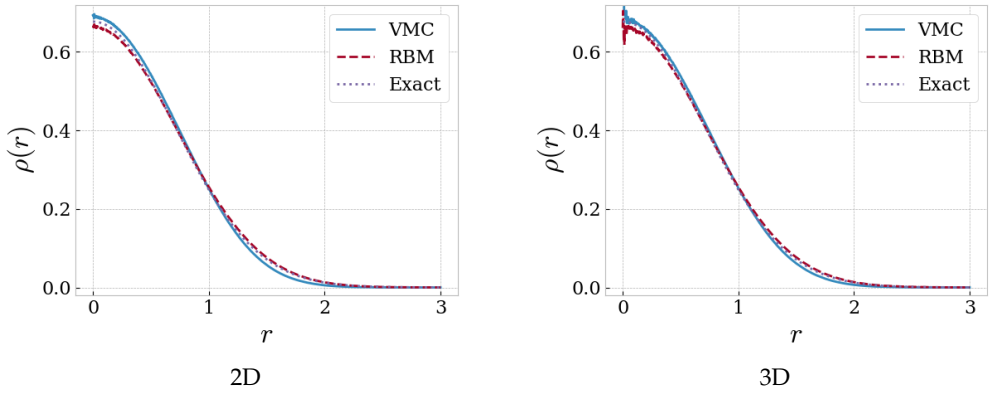


Figure 12.5: Radial one-body density plots for quantum dots with $N = 2$ non-interacting electrons and frequency $\omega = 1.0$. We look at both a two-dimensional dot (left) and a three-dimensional dot (right) with density profiles produced using VMC and RBM, and comparing to the analytical case $\rho(r) \propto \exp\left(-r^2\right)$. For abbreviations see the text.

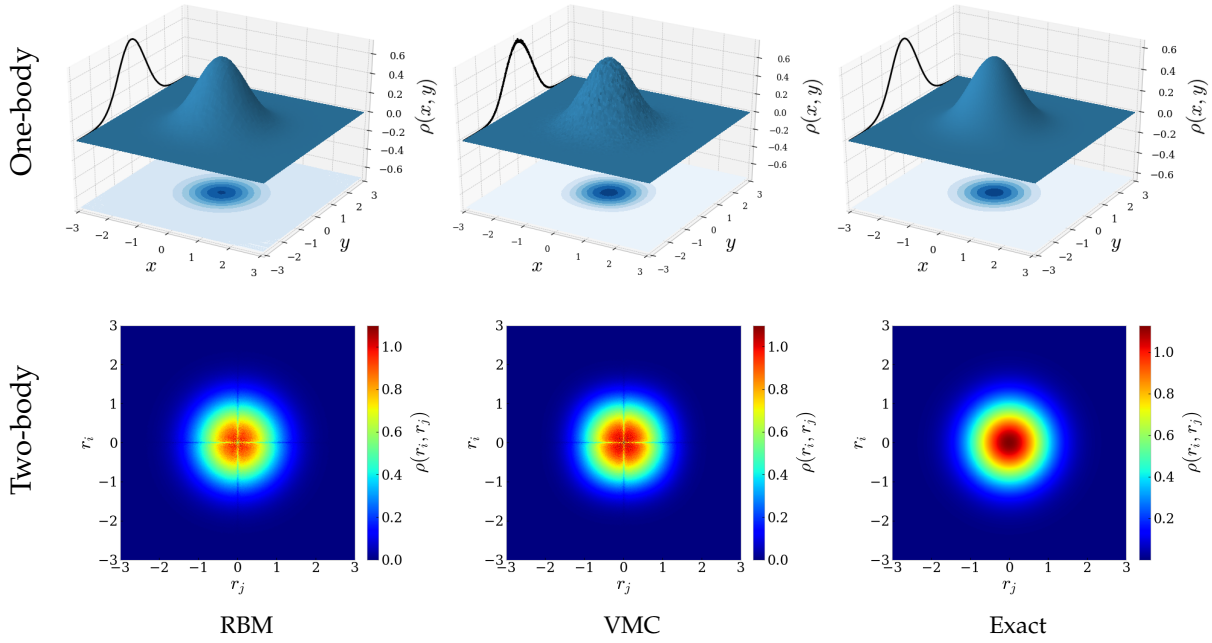


Figure 12.6: Electron density plots for a two-dimensional quantum dot with $N = 2$ non-interacting electrons and frequency $\omega = 1.0$ produces using RBM and VMC. The upper plots are the (spatial) one-body density distribution, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. The lower plots are the radial two-body density distributions. Besides, we compare the results to the exact densities, given by $\rho(x, y) \propto \exp\left(-x^2 - y^2\right)$ and $\rho(r_i, r_j) \propto \exp\left(-r_i^2 - r_j^2\right)$, respectively. For abbreviations see the text.

plots, such that the various density magnitudes can be compared to each other.

12.3.2 Atoms

Lastly, we will look at atoms with non-interacting electrons in order to validate our code. The analytical energy of those atoms is given by the Bohr formula presented in equation (4.17). In table (12.3), the lowest closed-shell atoms Helium, Beryllium and Neon are listed with their exact energy and the obtained energy using VMC with hydrogen-like orbitals. Besides, we add calculations of the Hydrogen ground state energy as another simple test. What distinguish the implementation of atoms from the implementation of quantum dots, is that we are forced to include variational parameters in the Slater determinant for the atoms with atomic number $Z > 2$. However, as all the energies are reproduced to machine-precision with the variational parameter $\alpha = 1.0$, we have a clear indication that also this part works as it should.

Table 12.3: Energy of neutral atoms with atomic number Z and non-interacting electrons. The energy is given in atomic units, see appendix B. The variance is zero to machine-precision for all listed results. For abbreviations see the text.

Atom	Z	VMC	Exact
H	1	-0.5	-0.5
He	2	-4.0	-4
Be	4	-20.0	-20
Ne	10	-200.0	-200

For completeness reasons, we also present the radial one-body density for Helium in figure (12.7). The density is multiplied with r^2 in order to reveal the structure, and later in section 12.5 to compare the obtained density to other references. We see that the produced density curve is more or less identical to the exact density curve, found from the definition in section 3.12.

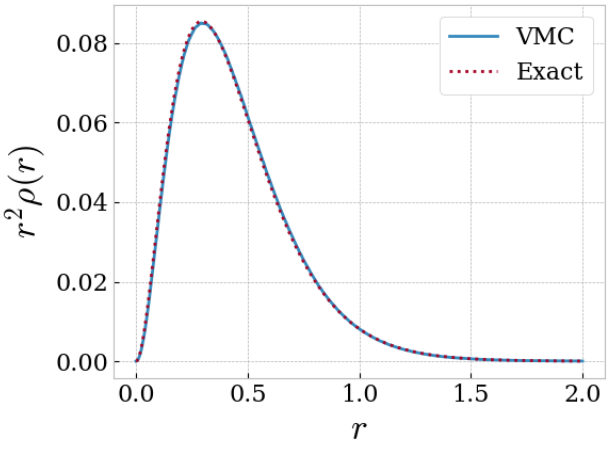


Figure 12.7: Radial one-body density plot for the non-interacting Helium atom produced using VMC. The density was multiplied with r^2 in order to reveal the structures. For abbreviations see the text.

12.4 Quantum dots

We now move on to the more interesting case with repulsive interaction between the electrons. Analytical results are in general unavailable for this case, apart from a few semi-analytical energies and wave functions for the two- and three-dimensional quantum dots. We will for this system calculate the ground state energy, the one-body density, and the two-body density.

The simulations are dependent on an array of parameters and factors, and in order to achieve good results, these hyper-parameters need to be reasonable. Firstly, the simulations are very sensitive about the learning rate, which needs to be small to avoid the quantum force to explode, but still large enough for the system to converge in an adequate number of iterations. Appropriate learning rates were found to be between $\eta = 0.5$ and 0.00001 dependent on the system, where large systems and/or systems with low frequency required the lowest learning rates. Also, the step length needs to be specified cleverly, as we want to sample over the space where the particles physically can be located in a reasonable number of cycles. For a narrow oscillator potential (high frequency), the step length needs to be smaller than for a wide oscillator (low frequency). The acceptance ratio is directly dependent on the step length, where we want a high acceptance ratio to keep the computational cost low. However, we experienced that a too high acceptance ratio is neither favorable, as it can cause a very slow convergence or even a divergence. Also, the statistical error is largely dependent on the step length, and keeping an acceptance ratio at around 0.995 was found to be optimal, with a step length spanning from 0.01 to 1.0. The initial particle configurations and initial parameters are crucial to make the simulations converge rapidly. Lastly, the statistical error naturally also depends on the number of Monte Carlo cycles, where a high number of cycles is preferred as discussed in section 7.1. We have applied an adaptive step number, which means that the number of cycles per iteration is increased for the last iterations. Firstly, this makes the final energy more accurate due to better statistics. Secondly, we get less noisy electron density plots by using this technique. All results below are produced using $2^{20} = 1,048,576$ number of steps per iteration until the energy has converged. Then we run 10 more iterations with the number of steps increased to $2^{24} = 16,777,216$ and for the very last iteration we run $2^{28} = 268,435,456$ or $2^{30} = 1,073,741,824$ steps.

Initially, we look at two-dimensional quantum dots with up to $N = 56$ electrons and three-dimensional quantum dots with up to $N = 40$ electrons and frequencies spanning from $\omega = 0.1$ to $\omega = 1.0$. For those systems, we will compute the ground state energy, the one-body density, and the two-body density. After that, we move on to some special cases where the dots have low frequency ($\omega = 0.01$) and large dots ($N > 56$) to test how far we can go. For those systems, we will typically focus on either the ground state energy or the one-body density dependent on what we want to investigate. For instance, we will focus on the one-body density for low-frequency dots because of the search for Wigner localization. We also have a thorough discussion of how the energy is distributed between kinetic and potential energy, and compare this to the virial theorem.

12.4.1 Ground state energy

The ground state energy is a natural starting point as our methods govern the minimization of the energy and it easily can be compared to benchmarks. By exploiting the symmetry of quantum dots with $n = 2$ electrons, Taut was able to obtain semi-analytical energies for some specific frequencies ω. More specifically, he found the energy to be $E = 3$ for the frequency $\omega = 1$ and $E = 2/3$ for the frequency $\omega = 1/6$ for the two-dimensional case [42], and $E = 2$ for the frequency $\omega = 1/2$ and $E = 1/2$ for the frequency $\omega = 1/10$ for the three-dimensional case [43]. For other references, we need to rely on what researchers have found before us. Since diffusion Monte Carlo (DMC) is known to give very accurate results, we will mainly compare our results

Table 12.4: The ground state energy of two-dimensional circular quantum dots with frequency ω containing N electrons. The HF results are taken from Mariadason [69], the DMC results are taken from Høgberget [19] and semi-analytical results (Exact) are taken from Taut [43]. For abbreviations see the text. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	RBM	RBM+SJ	RBM+PJ	HF (Ref.[69])	VMC	DMC (Ref.[19])	Exact (Ref.[43])
2	0.1	0.4728(1)	0.44856(1)	0.440975(8)	0.525635	0.44129(1)	0.44079(1)	
	1/6	0.7036(1)	0.67684(7)	0.66715(6)	0.768675	0.66710(1)	-	2/3
	0.28	1.07050(4)	1.03470(7)	1.021668(7)	1.14171	1.02192(1)	1.02164(1)	
	0.5	1.72293(7)	1.67739(9)	1.659637(6)	1.79974	1.65974(1)	1.65977(1)	
	1.0	3.0803(2)	3.02108(5)	2.999587(5)	3.16190	2.99936(1)	3.00000(1)	3.0
6	0.1	3.697(1)	3.63825(9)	3.5700(2)	3.85238	3.5695(1)	3.55385(5)	
	0.28	7.9273(9)	7.7313(2)	7.6203(2)	8.01957	7.6219(1)	7.60019(6)	
	0.5	12.241(1)	11.9659(5)	11.8074(2)	12.2713	11.8104(2)	11.78484(6)	
	1.0	20.716(1)	20.3393(8)	20.1832(2)	20.7192	20.1918(2)	20.15932(8)	
12	0.1	12.679(1)	12.5964(7)	12.3416(4)	12.9247	12.29962(9)	12.26984(8)	
	0.28	26.389(2)	26.051(1)	25.7331(5)	26.5500	25.7049(4)	25.63577(9)	
	0.5	40.440(3)	39.6340(7)	39.2743(6)	40.2161	39.2421(5)	39.1596(1)	
	1.0	67.632(3)	66.1898(8)	65.7911(7)	66.9113	65.7026(4)	65.7001(1)	

20	0.1	30.824(2)	30.567(3)	30.1553(9)	31.1902	30.0403(2)	29.9779(1)
	0.28	63.788(4)	62.786(3)	62.148(1)	63.5390	62.0755(7)	61.9268(1)
	0.5	96.410(1)	94.920(4)	94.104(1)	95.7328	94.0433(9)	93.8752(1)
	1.0	159.428(3)	156.816(4)	156.104(1)	158.004	155.8900(4)	155.8822(1)
30	0.1	61.829(5)	61.198(2)	60.774(2)		60.585(1)	60.4205(2)
	0.28	126.958(6)	125.413(2)	124.437(2)		124.195(2)	123.9683(2)
	0.5	191.495(7)	188.995(5)	187.488(2)		187.325(3)	187.0426(2)
	1.0	315.364(8)	309.997(6)	308.989(2)		308.576(1)	308.5627(2)
42	0.1	109.892(6)	109.48(2)	108.183(1)		107.928(2)	107.6389(2)
	0.28	224.462(8)	224.184(9)	222.200(5)		220.224(2)	219.8426(2)
	0.5	337.523(8)	333.582(9)	331.410(3)		331.276(3)	330.6306(2)
	1.0	553.40(1)	545.817(9)	543.746(3)		542.977(2)	542.9428(8)
56	0.1	179.789(6)	179.59(1)	178.501(5)		176.774(3)	175.9553(7)
	0.28	364.85(1)	364.165(9)	359.83(2)		359.63(1)	358.145(2)
	0.5	547.46(1)	545.74(1)	538.810(7)		538.686(9)	537.353(2)
	1.0	894.12(2)	882.93(1)	881.010(5)		879.514(3)	879.3986(6)

Table 12.5: The ground state energy of two-dimensional circular quantum dots with frequency ω containing N electrons. The HF results are taken from Mariadason [69], the DMC results are taken from Høgberget [19] and semi-analytical results (Exact) are taken from Taut [42]. For abbreviations see the text. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	RBM	RBM+SJ	RBM+PJ	HF (Ref.[69])	VMC	DMC (Ref.[19])	Exact (Ref.[42])
2	0.1	0.5177(1)	0.50214(3)	0.500080(6)	0.529065	0.500083(7)	0.499997(3)	0.5
	0.28	1.2261(1)	1.20475(4)	1.201710(6)	1.23722	1.201752(6)	1.201725(2)	
	0.5	2.0269(1)	2.00371(4)	1.999912(5)	2.03851	1.999977(5)	2.000000(2)	2.0
	1.0	3.7574(1)	3.73543(4)	3.729827(5)	3.77157	3.730030(5)	3.730123(3)	
8	0.1	5.8910(6)	5.7498(4)	5.718(4)	5.86255	5.7126(1)	5.7028(1)	
	0.28	12.650(1)	12.2492(4)	12.2056(2)	12.3987	12.2050(2)	12.1927(1)	
	0.5	19.487(2)	19.0241(4)	18.9747(2)	19.1916	18.9759(1)	18.9611(1)	
	1.0	33.302(1)	32.7159(6)	32.6820(2)	32.9246	32.6863(2)	32.6680(1)	
20	0.1	27.813(2)	27.470(1)	27.3382(8)		27.3144(5)	27.2717(2)	
	0.28	57.700(4)	56.600(1)	56.4477(6)		56.4297(5)	56.3868(2)	
	0.5	87.840(4)	85.893(1)	85.7153(6)		85.7161(5)	85.6555(2)	
	1.0	146.292(4)	143.209(2)	142.9409(6)		142.9560(7)	142.8875(2)	
40	0.1	89.45(8)	89.618(4)	88.596(4)		88.182(1)		
	0.28	182.714(6)	181.877(4)	179.630(1)		179.567(1)		
	0.5	275.262(7)	271.030(4)	269.782(2)		269.746(1)		
	1.0	452.732(8)	442.874(4)	442.630(1)		442.602(2)		

to the DMC computations of Høgberget [19], which exist for quantum dots with up to $N = 56$ electrons in two dimensions and with up to $N = 20$ electrons in three dimensions. Comparing the energy to the Hartree-Fock (HF) limit is also interesting as the HF method approximates the electron-electron interactions by a mean-field, while the RBM needs to model the interactions itself. Practically, the RBM should give a lower energy than the HF method as the latter traditionally is notably computationally cheaper. We use the HF computations of Mariadason [69] for two-dimensional quantum dots with up to $N = 20$ electrons and three-dimensional quantum dots with up to $N = 8$ electrons. Computations for larger dots were not included, as they seem to not have converged.

Ground state energy computations of two- and three dimensional quantum dots are found in tables (12.4) and (12.5) respectively. They are computed by an RBM, RBM+SJ, RBM+PJ, and VMC, in addition to the HF limit and DMC that are present for reference purposes. The exact values are found in the last column with the same name. We observe that the method where less physical intuition is used, RBM, is the one that gives the highest energies among our implemented methods. This is as expected as no Jastrow factor is added to account for the correlations. For small quantum dots ($N < 8$), the RBM energy is lower than the HF limit, and for low frequencies ($\omega < 0.5$) the RBM energy is generally also lower. However, for higher frequencies and larger dots, the HF limit is lower than the energy obtained with the RBMs. It is apparent that the mean-field approximation works better than the RBM when the interactions get less important. When we add more intuition in the form of a simple Jastrow factor, the energy drops significantly, especially for high frequencies. It is, therefore, lower than the HF limit for all simulations, and surprisingly close to the reference considering the simple form of the Jastrow factor. The statistical errors of the RBM+SJ is in general smaller than for RBM, which is a good indication that the method provides a better ground state estimate as the actual ground state obeys a zero-variance property.

Furthermore, we added a more complex Jastrow factor in the form of a Padé-Jastrow factor, and we observe that the energy drops further. The RBM+PJ provides energies entirely on par with VMC, in many cases even lower. Notably, RBM+PJ provides the lowest energies among our implemented methods for dots with two electrons, and the statistical errors are also the lowest. This hints that RBM+PJ can obtain a better ground state estimate than VMC for the smallest dots, which makes sense as the former method is more flexible than VMC. However, for large dots, RBM+PJ provides slightly higher energies than VMC, which we suspect is a result of the large number of variational parameters, and thus a too complex model for the problem. If we recall that the number of hidden units is set to the number of electrons, the number of variational parameters increases rapidly as the number of electrons increases. To see if the problem occurs because the model has not converged, we have decreased the number of hidden nodes for some selected systems, and it has given promising results.

We also see that all the methods can obtain energies closer to the reference energy in three dimensions compared to two dimensions. This is a well-known concept and occurs because we have more degrees of freedom. One can imagine that the electrons can move in the space instead of just on a plane, which makes it easier to form a configuration that minimizes the energy.

Efforts were made to reproduce the results using a partly restricted Boltzmann machine and a deep belief network, but due to convergence problems and energy fluctuations, the methods were not investigated further.

12.4.2 One-body density

Another quantity of particular interest is the one-body density. For all the systems presented in tables (12.4) and (12.5) we have also calculated the one-body density. However, since all of those plots would occupy tens of pages, we will not present all of them here. Instead, we select a

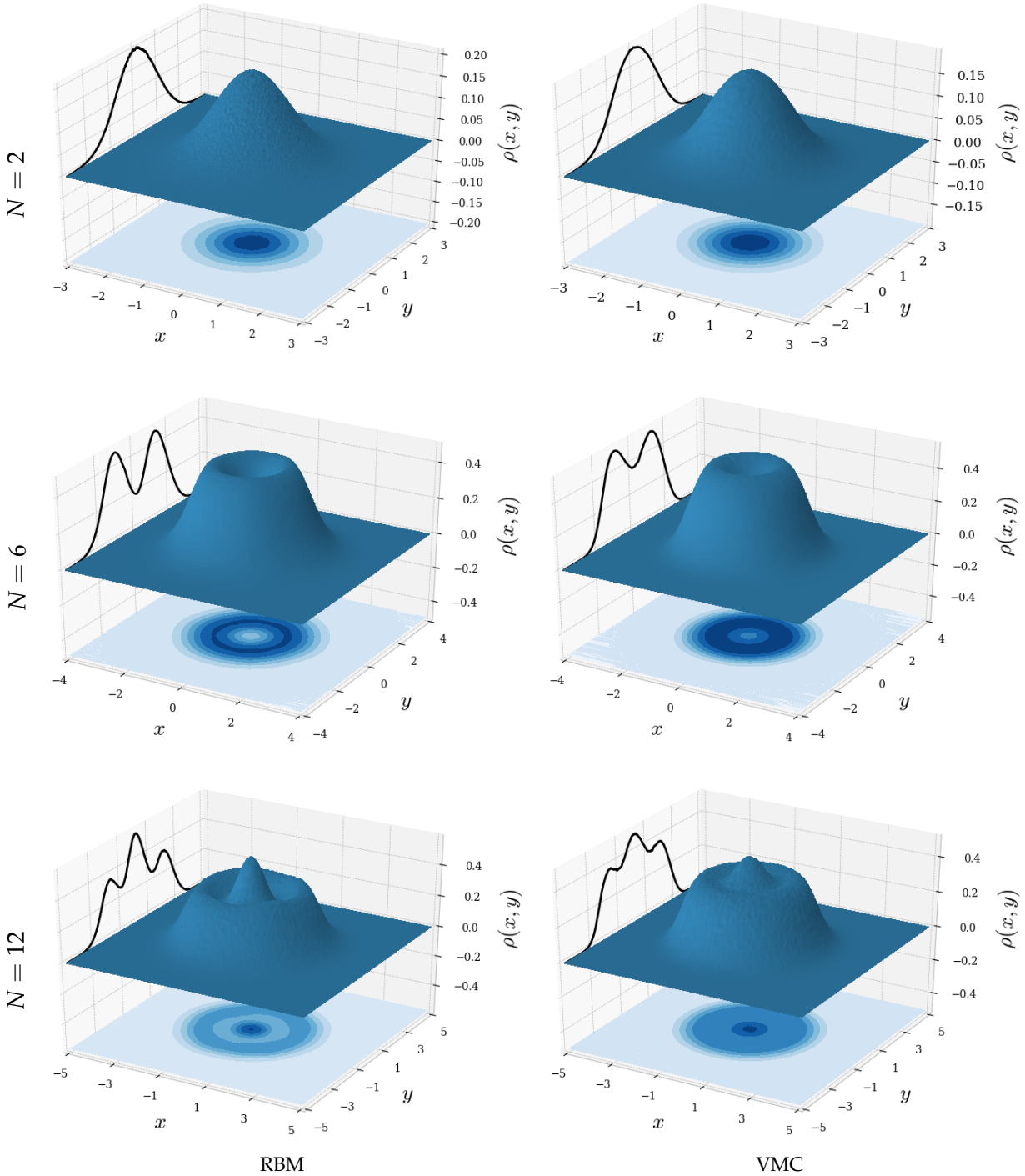


Figure 12.8: One-body density plots for two-dimensional quantum dots with frequency $\omega = 1.0$, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. From the top, the plots present quantum dots with $N = 2, 6$ and 12 electrons, and were all obtained using RBM (right column) and VMC (left column). The ADAM optimizer was used, and after convergence the number of Monte Carlo cycles was $M = 2^{30} = 1,073,741,824$. The plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

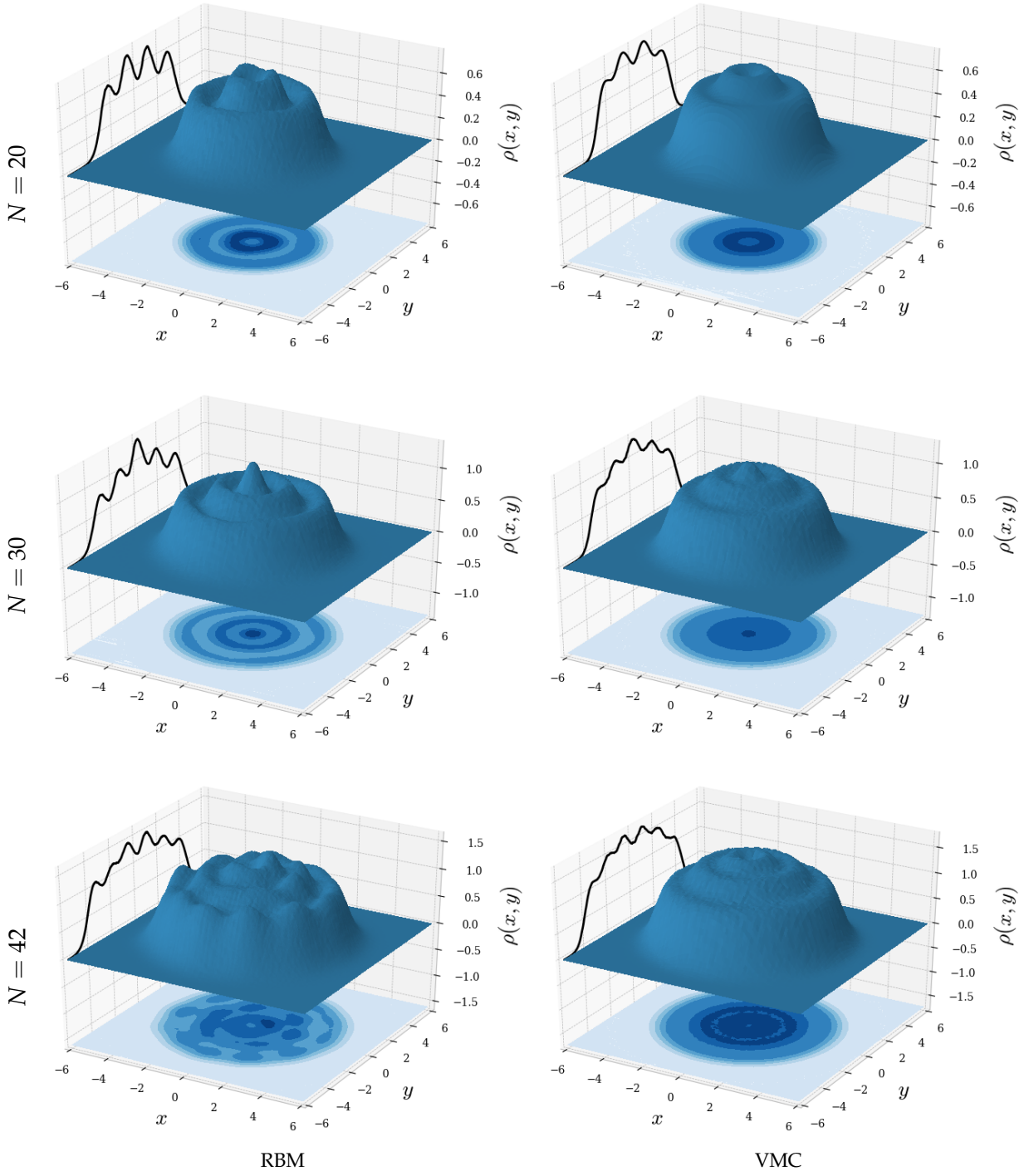


Figure 12.9: One-body density plots for two-dimensional quantum dots with frequency $\omega = 1.0$, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. From the top, the plots present quantum dots with $N = 20$, 30 and 42 electrons, and were all obtained using RBM (right column) and VMC (left column). The ADAM optimizer was used, and after convergence the number of Monte Carlo cycles was $M = 2^{30} = 1,073,741,824$. The plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

few sample plots that might be interesting for the reader and give a representative picture of the density. The total collection of one-body density plots can be found in section D.3 in appendix D.

We have computed the one-body density in two different ways: by dividing the space into annuluses and obtaining the radial density profile, as discussed in section 10.8, and by dividing the space into a grid of equally sized bins and obtaining the density profile throughout the space. The former is computationally beneficial as the density typically is stored in n bins compared to n^2 bins for the second method, but the latter is clearly able to store more information. The second method is thereby able to give more informative plots, but when we want to compare the various methods, a radial density profile is more convenient. In other words, there are pros and cons associated with both methods, and we will, therefore, present both to give an as comprehensive description of the results as possible.

Initially, we will present the spatial density profiles for two-dimensional quantum dots of frequency $\omega = 1.0$. In figure (12.8) and (12.9), the evolution of the one-body density for dots of sizes $N = 2$, 6, 12, 20, 30 and 42 are presented, produced with RBM and VMC. To maximize the amount of information in the plots, we have included a graph presenting the cross-section through $x = 0$ on the yz-plane and a contour plot of the density on the xy-plane, in addition to a 3D surface plot of the density. What we observe is that the density profile is smooth, with more peaks as the number of particles increases, representing high densities. For an odd number of shells ($N = 2$, 12 and 30) the density has its maximum at the center of the dot, resulting in equal shapes where for example the shape of $N = 12$ is seen as the top of $N = 30$. On the other hand, for an even number of shells ($N = 6$, 20 and 42) the highest density is found at a ridge that encircles the center, and also here the shape of for example $N = 20$ is seen as the top of $N = 42$. It is apparent that those are the configurations that minimize the energy, with a structure following directly from the Pauli principle, as the electrons are forced to distribute over multiple shells. Even when we remove the interaction between the electrons, we get the same characteristic wave shape, though narrower. The RBM provides sharper and more distinct peaks than the VMC, but at least the extrema seem to be located at the same radii. These differences are caused by the ways the methods model the correlations, where the VMC employs the Padé-Jastrow factor, and the RBM needs to find the best way to account for the correlations itself. To connect the observations to something everyone is familiar with, we can compare the density plots to water ripples if one sees it from the smallest to the largest dot. Høgberget [19] discusses this in his master thesis, and his one-body densities are consistent with what we have obtained. However, for $N = 42$ the RBM provides a density profile that has ripples in both radial and angular direction, indicating that the simulation has not fully converged to the minimum. This might be an effect of the large number of variational parameters. One-body density plots for the same systems using an RBM+SJ and RBM+PJ were generated, but in order to limit us to a few plots we decided to keep the extremes here, and placed the others in appendix D.

In figure (12.10), we stick to frequency $\omega = 1.0$ for small two- and three-dimensional quantum dots and compare the results obtained by VMC, RBM, RBM+SJ, and RBM+PJ. We investigate dots with $N = 2$, 6 and 12 in two dimensions and $N = 2$, 8 and 20 in three dimensions, which corresponds to one shell ($S = 1$), two shells ($S = 2$) and three shells ($S = 3$), respectively. For all the numbers of shells, we immediately see that the density plots are identical for two- and three-dimensional dots. The same phenomenon was observed for quantum dots with non-interacting electrons in figure (12.5), which can easily be explained as the wave function is trivially separable. However, this is not the situation when we look at the interacting case, and it was therefore not obvious to us that the same phenomenon would occur here. Physically, this means that the electrons configure in the same way in three dimensions as in two dimensions,

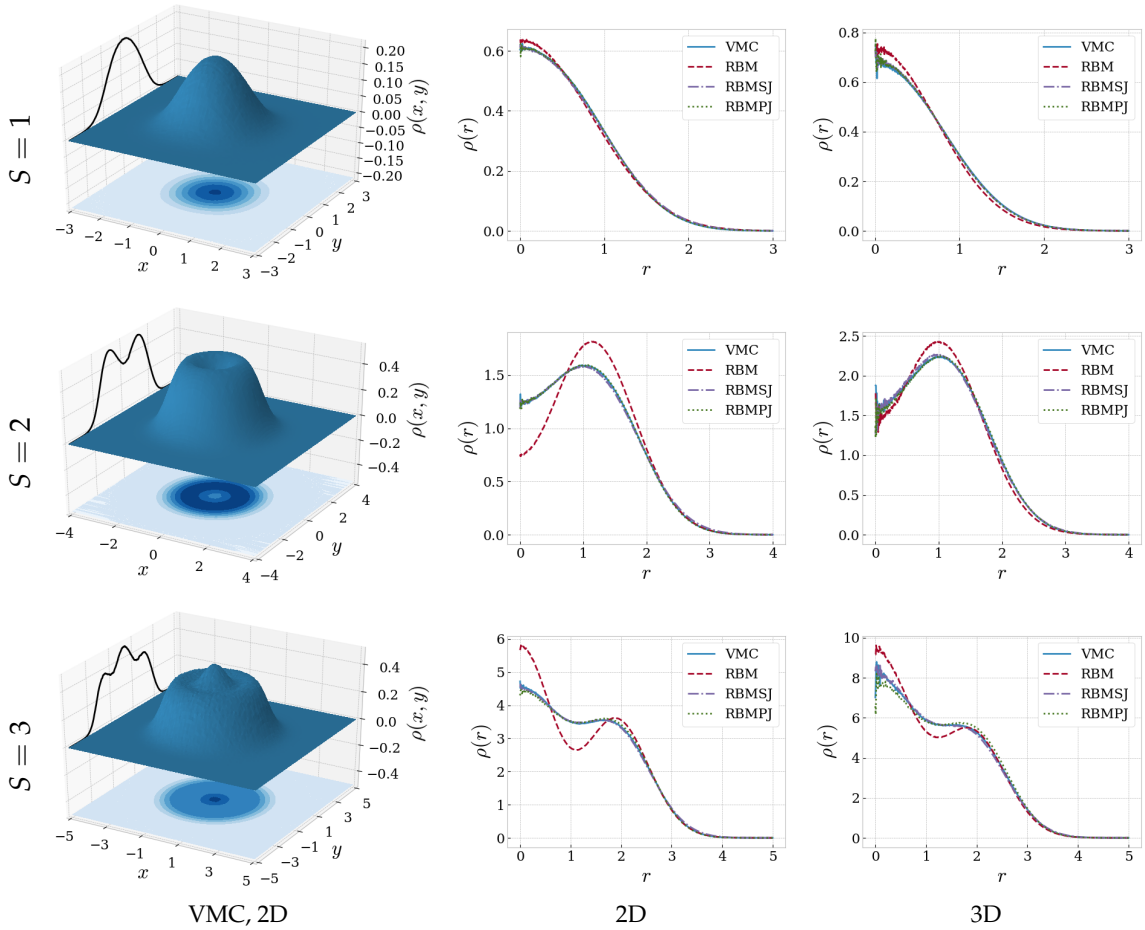


Figure 12.10: One-body density plot for small quantum dots with frequency $\omega = 1.0$. We look at the three lowest shells $S = 1, 2$ and 3 corresponding to $N = 2$, 6, and 12 electrons in two dimensions and $N = 2, 8$ and 20 electrons in three dimensions. In the first column, the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. In the middle column, the corresponding radial density profiles are obtained using RBM, RBM+SJ, RBM+PJ and VMC. On the right-hand side, the radial density profiles for the three-dimensional dots are given using the same methods. ADAM optimizer was used, and after convergence, the number of Monte-Carlo cycles was $M = 2^{30} = 1,073,741,824$. The surface plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

where the average distance from a "shell" to the center of the dot is dimensionless. Besides the physics, we are also interested in analyzing the performance of the various methods. We observe that all the methods give quite similar radial density profiles, but RBM is always the guy standing out. Apparently, the RBM tends to exaggerate the peaks discussed above and gives an even more wavy density plot, as also observed in figures (12.8) and (12.9). Further, the RBM+SJ is more similar to the VMC and RBM+PJ, which are almost identical. This is consistent with the energy analysis in section 12.4.1, where we found VMC and RBM+PJ to be the most accurate methods, followed by RBM+SJ and RBM. We also observe some noise close to $r = 0$, which is caused by the fact that the number of Monte Carlo cycles is finite and we in practice have bins of different sizes.

We now move on to larger two-dimensional dots and lower frequencies than $\omega = 1.0$ to see if the physics is conserved also when the interactions get stronger. In figure (12.11), we plot the

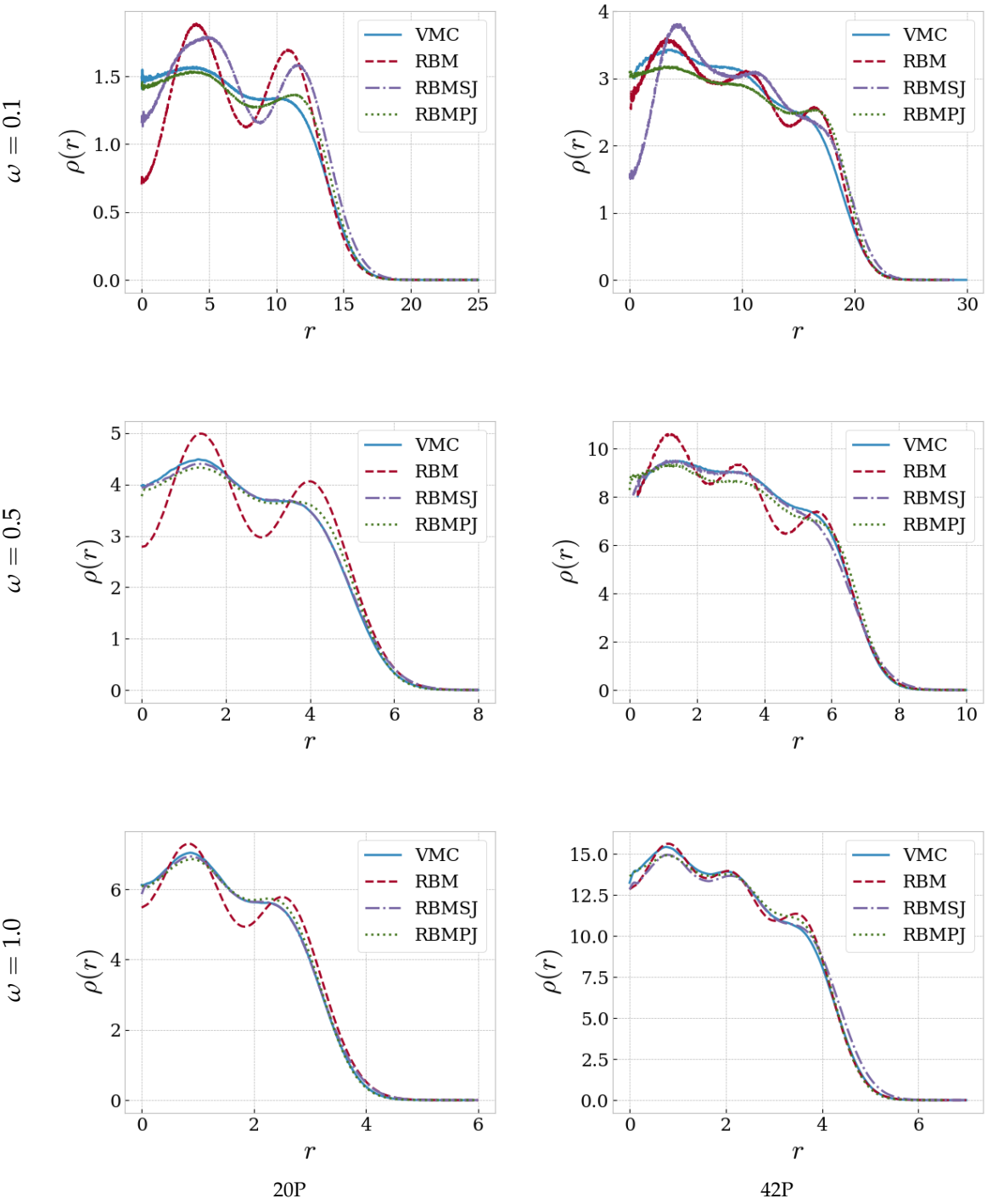


Figure 12.11: One-body density plot for two-dimensional quantum dots with oscillator frequencies $\omega = 0.1$, 0.5 and 1.0 and $N = 20$, 42 electrons. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

radial density profile for two-dimensional quantum dots with $N = 20, 42$ and frequencies $\omega = 0.1, 0.5$ and 1.0 to compare all the methods again. What we see is that the peaks discussed above are conserved for lower frequencies, and they change slightly as the frequency drops. Also, the density distribution is spatially expanded as the frequency drops, since the force pushing the electrons towards the center of the dot gets weaker. All the methods VMC, RBM+PJ, and RBM+SJ appear to give quite similar density plots, but for lower frequencies, the difference gets gradually more visible. In particular, the peaks for frequency $\omega = 0.1$ do no longer match for the various methods. Especially RBM and RBM+SJ differ from VMC and RBM+PJ, which is again a clear indication that the final results are heavily dependent on the way the methods account for the correlations. RBM+SJ tends to slightly exaggerate the peaks in the density plots, and RBM takes the exaggeration to the next level. This trend was also observed in the plots in figures (12.8, 12.9, 12.10), and technically speaking, this means that the method is capable of finding the most likely places to find the electrons, but fails to determine the actual density there. We have also seen that the one-body density is shape-invariant for the frequencies $\omega = 0.5$ and 1.0 except for their radial extent. For that reason, it is not very exciting to further study those high frequencies, and in section 12.4.5 we move on to low-frequency dots.

12.4.3 Two-body density

We have on multiple occasions in this thesis discussed the electron density, and we have also mentioned the two-body density. The two-body density gives the probability of finding an electron at a certain position, given the position of another electron. Unlike the one-body density, this density, therefore, gives the information of how the particles distribute pairwise, and it is therefore very informative when we want to study the electron-electron correlations. Similarly to the one-body density, we can both plot the radial two-body density profile and the actual two-body density distribution throughout the space. However, since the latter results in a higher-dimensional object, we stick to the radial density profile. Also for this quantity, it would be convenient to have references to benchmark our results, but for some reason, it is hard to find papers that discuss the two-body density for quantum dots. Nevertheless, someone needs to be the first, and we will use the plots to discuss physics and compare the various methods in the best possible way. Again, we will emphasize that a negative radial distance does not make sense, so all the quadrants in the two-body density plots are the same and presented in this way to make them more illustrative and intuitive. We will focus on the two-dimensional case, as the three-dimensional case becomes similar in the same way as for the one-body density.

In figure (12.12), we plot the two-body density for a two-dimensional quantum dot with $N = 2$ and frequencies $\omega = 0.1$, 0.5 and 1.0. For the frequency $\omega = 1.0$, the density plots for all the methods give quite similar results, but the RBM provides a higher two-body density in the middle of the quantum dot, compared to the other methods. This can be seen from the magnitude of the density at the center, but also the density itself is narrower than for its fellow methods, making it more compact. This can be explained by the absence of a Jastrow factor. Also, the RBM+SJ gives a slightly narrower distribution than the RBM+PJ and VMC, so it is apparent that the Padé-Jastrow factor results in a more repulsive factor than the simple Jastrow factor. The same indication can be seen at the density at the origin, which is low for RBM+PJ and VMC, but high for RBM and RBM+SJ. Furthermore, we know that the potential, and thus the interactions, get more dominating as the frequency drops. For VMC and RBM+PJ, this can be observed by a significant circle of higher density at a certain distance from the origin both for $\omega = 0.5$ and $\omega = 0.1$, which physically means that both particles are less likely to be found in the center of the dot at the same time. If one particle is close to the center, the other particle is probably far from the center and *vice versa*. For the RBM+SJ, this behavior can only be observed

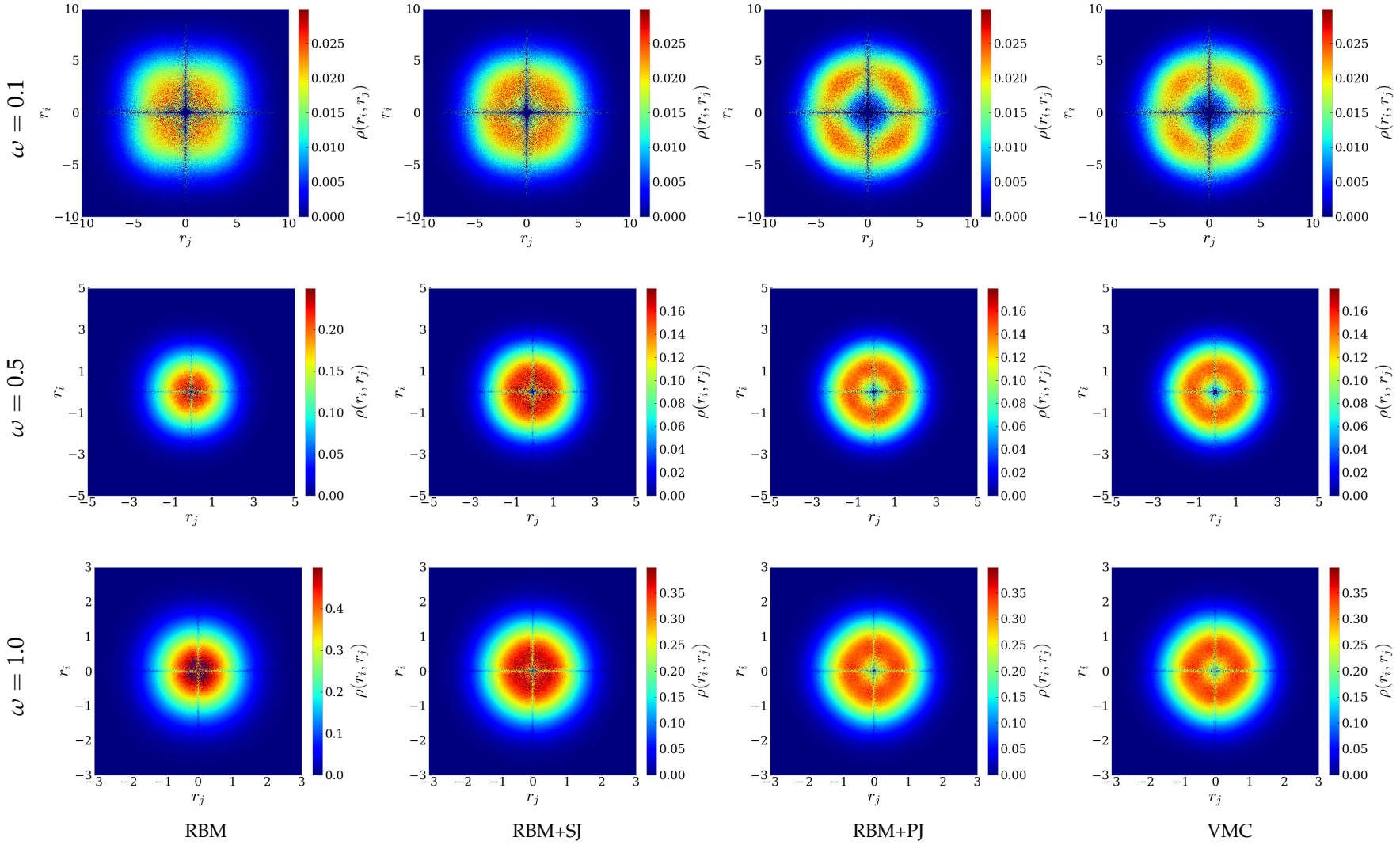


Figure 12.12: Two-body density plots for two-dimensional quantum dots with $N = 2$ electrons and oscillator frequencies $\omega = 0.1$, 0.5 and 1.0. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

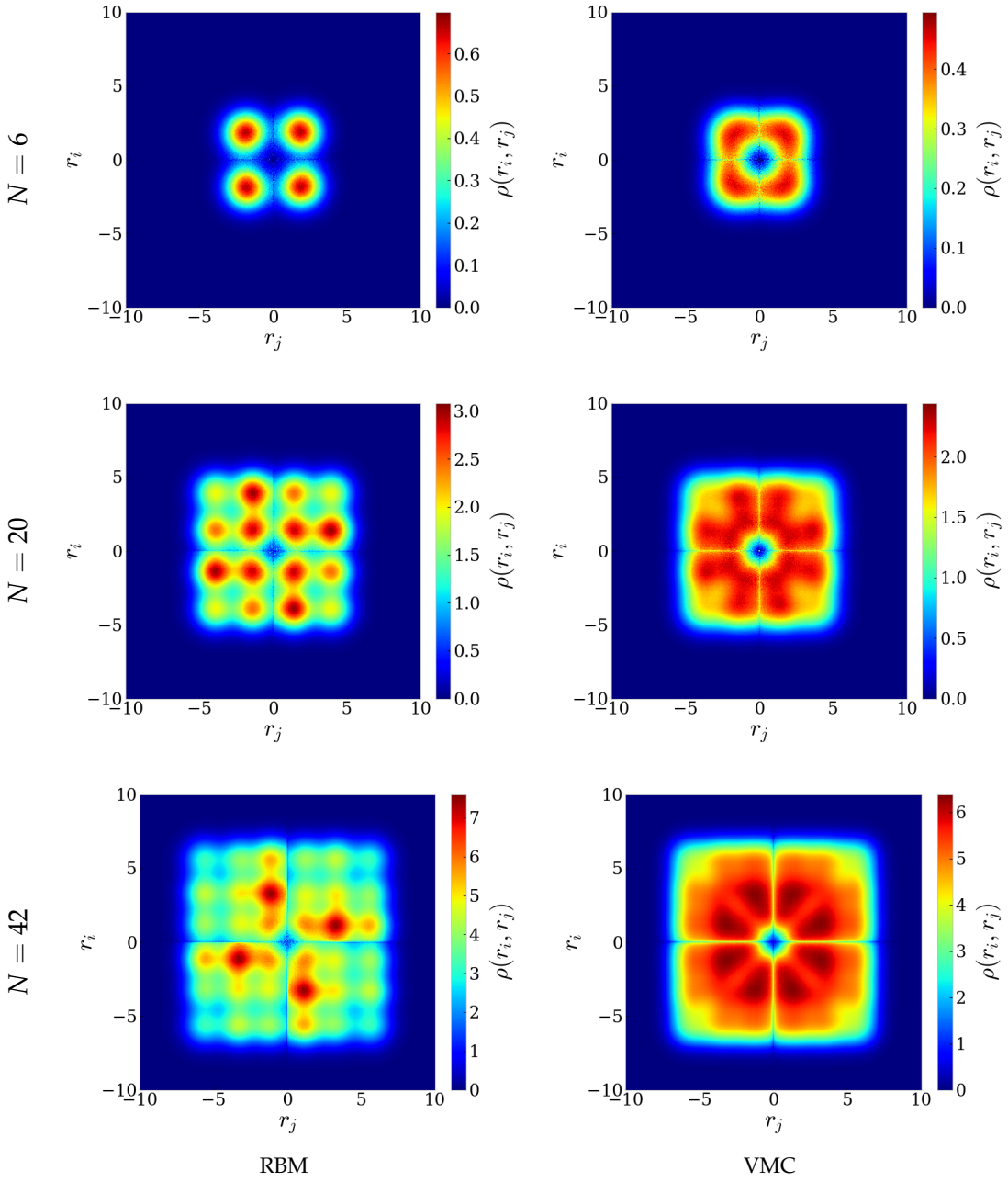


Figure 12.13: Two-body density plots for two-dimensional quantum dots with an even number of closed shells ($N = 6$, 20 and 42 electrons) and oscillator frequency $\omega = 0.5$. The ADAM optimizer was used, and after convergence, the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

for $\omega = 0.1$, while the RBM is not able to reproduce this phenomenon at all. This indicates that the RBM is not able to model the correlations correctly, and it needs a Jastrow factor to account for them. The resolutions of the plots get lower when we decrease the frequency, as the particles spread over a larger area and therefore more bins.

The calculations are repeated for two-dimensional quantum dots with frequency $\omega = 0.5$ and an even number of closed shells ($N = 6$, 20 and 42), with radial two-body density plots presented in figure (12.13). The first thing we observe is that the RBM manages to obtain the correct peaks, but all the peaks are circular unlike the density plots from the VMC, and they are also more

distinct and higher than for VMC. This is the same effect we saw in the one-body density plots, which we argued was caused by the different approaches to model the correlations. If we now keep our attention on the VMC plots, it is apparent that two particles will not be observed in the middle of the dot at the same time, as the density there is almost absent. For $N = 6$, a particle pair is most likely to be found at the same radius around $r = 2$, which matches the one-body density plot. When we move on to $N = 20$, the most probable location is not ambiguous anymore, but an electron pair is likely to be found at the same radius at around $r = 1$, which matches the highest peak in the one-body density plot. However, the density is also high along both axes, which means that if one of the electrons moves towards the center of the dot, the other will also move towards the center. This is probably a consequence of the repulsive interactions. For $N = 42$, finding two electrons at the same radius is not the most likely case anymore. This is because the electrons now spread over a large area. The density plot clearly has some distinct peaks around $r_i \sim 2$ and $r_j \sim 1$, matching the peaks found in the one-body density plots. For dots with an odd number of closed shells, the same tendency was found, but the peaks were found at the intersection between the quadrants, again matching the one-body density.

12.4.4 Energy distribution

If we now recall the general Hamiltonian presented in chapter 3, the total energy is just the sum of the kinetic, potential, and interaction energy. As our methods attempt to solve the Schrödinger equation directly, it is trivial to find the distribution between the various energy sources, which in general is interesting when we want to find the most important contributor to the energy. Additionally, we can use those results to verify the virial theorem presented in section 2.6, and it is also interesting to see if the different methods give different energy distributions. In figure (12.14), we plot the ratio between the kinetic and potential energy as a function of oscillator frequency for two-dimensional quantum dots with up to $N = 20$ electrons for all our methods. The plots are based on the numbers in the tables (D.6-D.5) in appendix D.

Firstly, the graphs are very similar for all the methods, which means that they all give the same distribution between kinetic and potential energy, although they do not provide the same energy. This is an interesting observation and means that the obtained kinetic and potential energy are both different for the various methods when the total energy is different. Physically, this indicates that the electron configurations are fundamentally different for the different methods, which is the only factor that can cause a change in potential energy. This is already observed in the one-body density plots. Further, we see a significant trend where the ratio drops as the frequency is decreased, implying that the potential energy dominates the kinetic energy at low frequencies, which is a known phenomenon already mentioned several times throughout this thesis. For all the methods, the ratio is also lower for larger dots, which is a result of gradually more interaction energy as the number of particles increases.

If we further recall the virial theorem presented in section 2.6, it states that the kinetic energy is related to the potential energies in a certain way. If we assume that the interaction potential goes as $V_{\text{int}} \propto r^{-1}$, we can write the virial theorem as

$$2\langle \hat{T} \rangle = 2\langle \hat{V}_{\text{ext}} \rangle - \langle \hat{V}_{\text{int}} \rangle, \tag{12.1}$$

according to equation (2.22). We could try to verify this formula from figure (12.14), but in order to do this accurately, we will list up the energy distributions for some selected systems. For a two-dimensional quantum dot with $N = 20$ electrons the energy distribution is given in table (12.6) for some selected frequencies. By doing the math, we see that the virial theorem is not satisfied for any of the methods or frequencies! For the lowest frequency, the theorem is far from

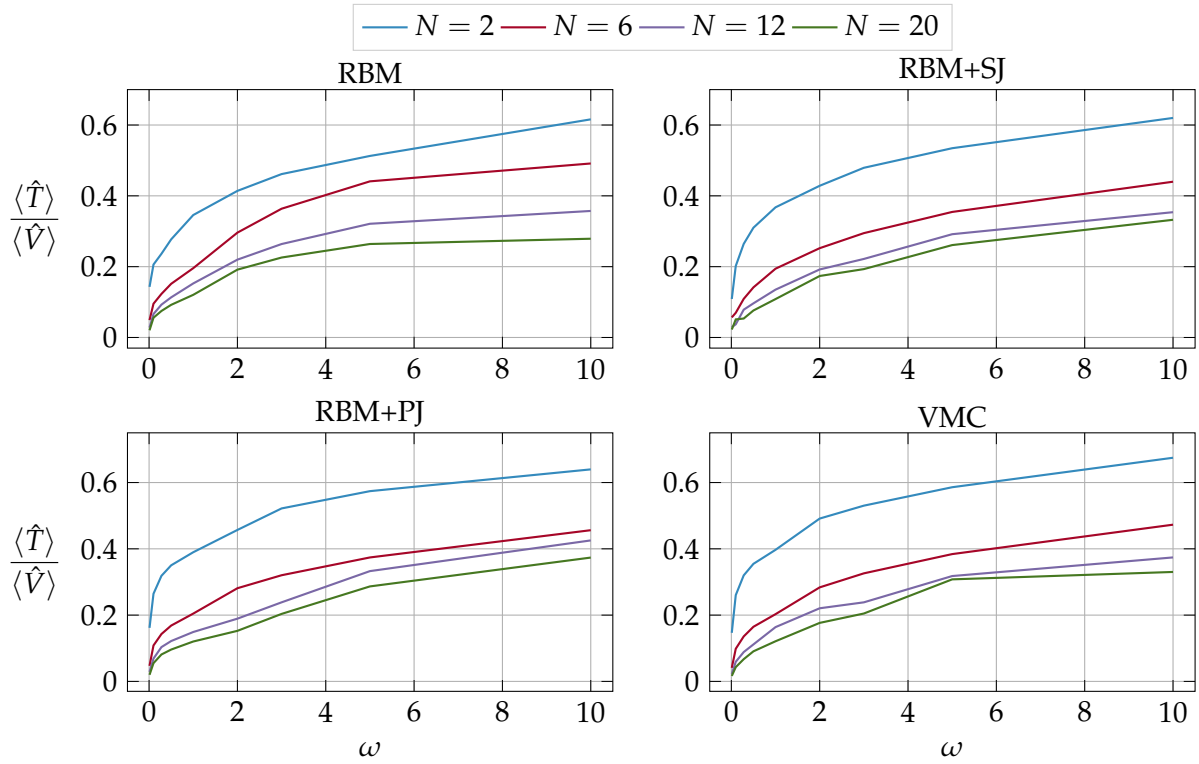


Figure 12.14: The kinetic-potential energy ratio $\langle \hat{T} \rangle / \langle \hat{V} \rangle$ plotted as a function of the oscillator frequency for two-dimensional circular quantum dots with $N = 2$, 6, 12 and 20 electrons. The frequencies $\omega = 0.01$, 0.1, 0.28, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0 were run, see appendix D for exact energies. For abbreviations see the text.

being correct, but as the frequency increases the error decreases. To estimate the accuracy of the theorem, we introduce a ratio

$$R = \frac{2\langle \hat{T} \rangle}{2\langle \hat{V}_{\text{ext}} \rangle - \langle \hat{V}_{\text{int}} \rangle} \tag{12.2}$$

which naturally is $R_{\text{exact}} = 1$ if our version of the virial theorem is satisfied. To estimate the error, we calculate the absolute error $E_\omega = ||R_\omega - 1||$. Take, for example VMC, we get $E_{\omega=0.01} = 0.713$, $E_{\omega=2} = 0.221$ and $E_{\omega=10} = 0.021$, which shows that equation (12.1) gradually gets more accurate as the potential energy gets more important. To see why this is the case, we need to go back to the original virial theorem given in equation (2.21). By inserting the interaction energy, we see that our modified virial theorem is inaccurate, and breaks down when the interaction energy dominates. However, it should be quite accurate when the interaction energy is not so important, which is consistent with our results.

Another interesting thing is how the energy is distributed for the various methods. Most notably, we see that VMC and RBM+PJ provide different distributions between the different methods, albeit the total energy is more or less identical. Physically, this means that the RBM+PJ finds another configuration than VMC to minimize the energy, which is exciting as the former method is supposed to be more flexible than the latter. For more frequencies and system sizes, please look at appendix D.

Table 12.6: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) for two-dimensional quantum dots with $N = 12$ electrons and at a wide range of frequencies ω. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit. For abbreviations see the text.

	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
RBM	0.01	6.217(2)	0.1236(4)	2.244(2)	3.849(2)
	2.0	269.086(8)	43.262(8)	95.17(2)	130.65(1)
	10.0	961.03(4)	260.2(1)	364.8(1)	336.06(7)
RBM+SJ	0.01	6.239(2)	0.1372(6)	2.184(2)	3.919(3)
	2.0	265.66(9)	39.31(8)	95.78(1)	130.57(2)
	10.0	952.71(2)	237.65(4)	392.06(7)	323.00(3)
RBM+PJ	0.01	6.210(1)	0.1208(5)	2.189(2)	3.900(2)
	2.0	262.598(1)	34.758(6)	108.546(9)	119.293(7)
	10.0	947.33(2)	257.67(5)	348.35(6)	341.31(3)
VMC	0.01	6.2097(8)	0.1005(4)	2.270(3)	3.839(3)
	2.0	262.5339(9)	38.402(3)	95.681(7)	128.451(5)
	10.0	945.596(8)	231.56(4)	389.26(7)	324.77(3)

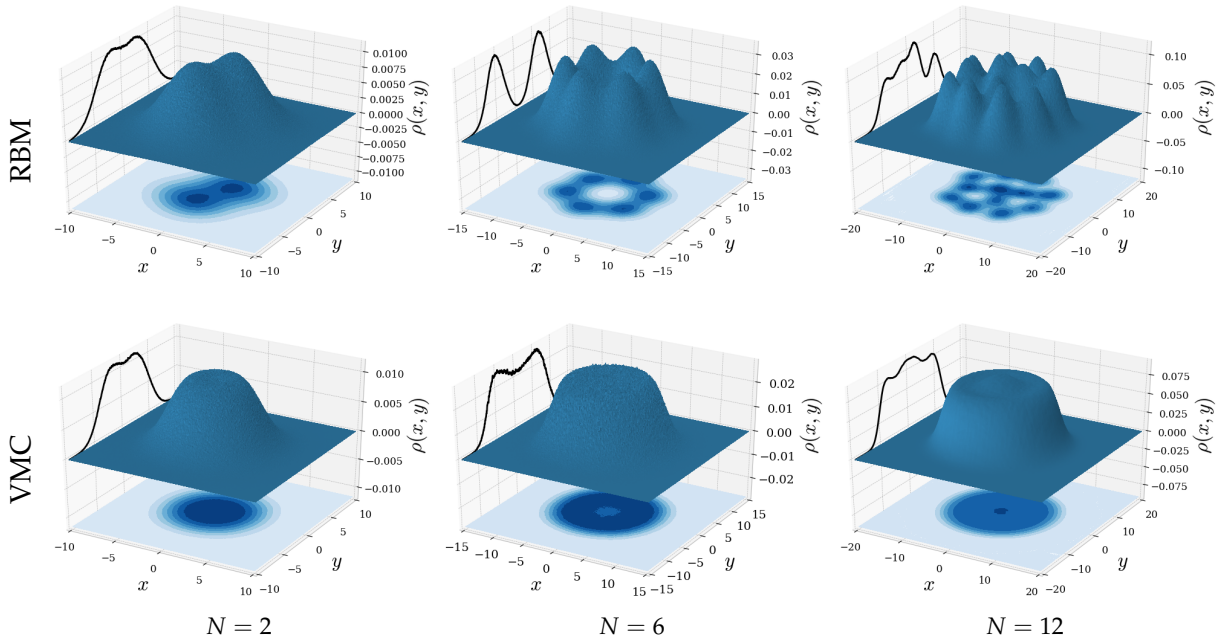


Figure 12.15: One-body density of two-dimensional circular quantum dots with frequency $\omega = 0.1$ with $N = 2$, 6 and 12 electrons, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. The surface plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

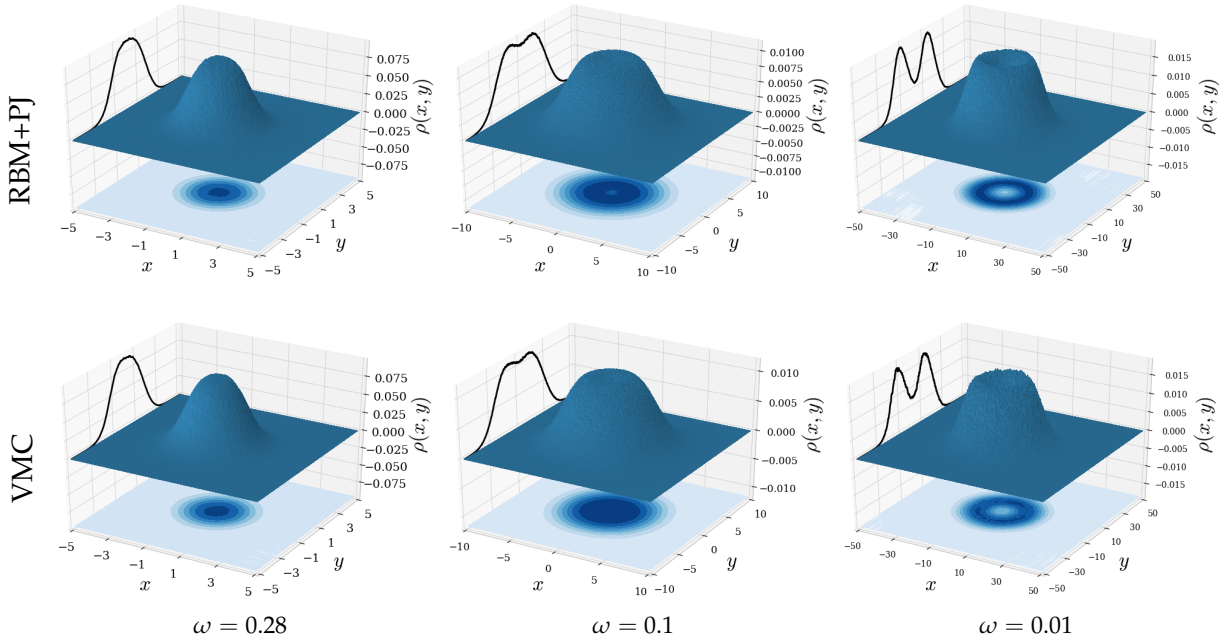


Figure 12.16: One-body density of two-dimensional circular quantum dots with $N = 2$ electrons and frequencies $\omega = 0.28$, 0.1 and 0.01, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. The surface plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

12.4.5 Low frequency dots

In section 12.4.2, we found the one-body density to be shape-invariant for high-frequency dot with $\omega \geq 0.28$. However, when we further decreased the frequency down to $\omega = 0.1$, the density profiles changed significantly, and we got other extrema and so on. This section aims to investigate the transitions between the various shapes with frequencies down to $\omega = 0.01$. We start by looking at quantum dots with frequency $\omega = 0.1$, and in figure (12.15) we compare the radial density profiles for $N = 2$, 6 and 12 produced by RBM to the results produced by VMC. What we observe is that the density plots are completely different. While VMC provides a single peak with some bumps on the top, RBM gives several peaks for all the numbers of electrons. The numbers of peaks apparently is equal to the number of electrons, which indicates that the RBM finds the electrons to be localized at some fixed spots. The two methods model the correlations in completely different ways, and as the correlations become more important for low frequencies, it was expected that the results would also be different. However, the obtained ground state energies by the two methods are not so different for the same systems, indicating that the electron density plots are better at reveal differences between the methods than the energy itself.

Further, we fix the number of electrons to be $N = 2$, and vary the frequency from $\omega = 0.28$ down to $\omega = 0.01$ with density profiles produced by VMC and RBM+PJ found in figure (12.16). Those methods were selected as they hitherto have shown the most promising results and we want to see if the RBM+PJ can reveal effects that the VMC is not able to capture. We see that the two methods obtain very similar density plots, where they agree that a ridge around the center of the dot should be more distinct as the frequency drops. If we go back to figure (12.14), we saw that the kinetic energy was negligible for the lowest energies, and the effect is, therefore, an

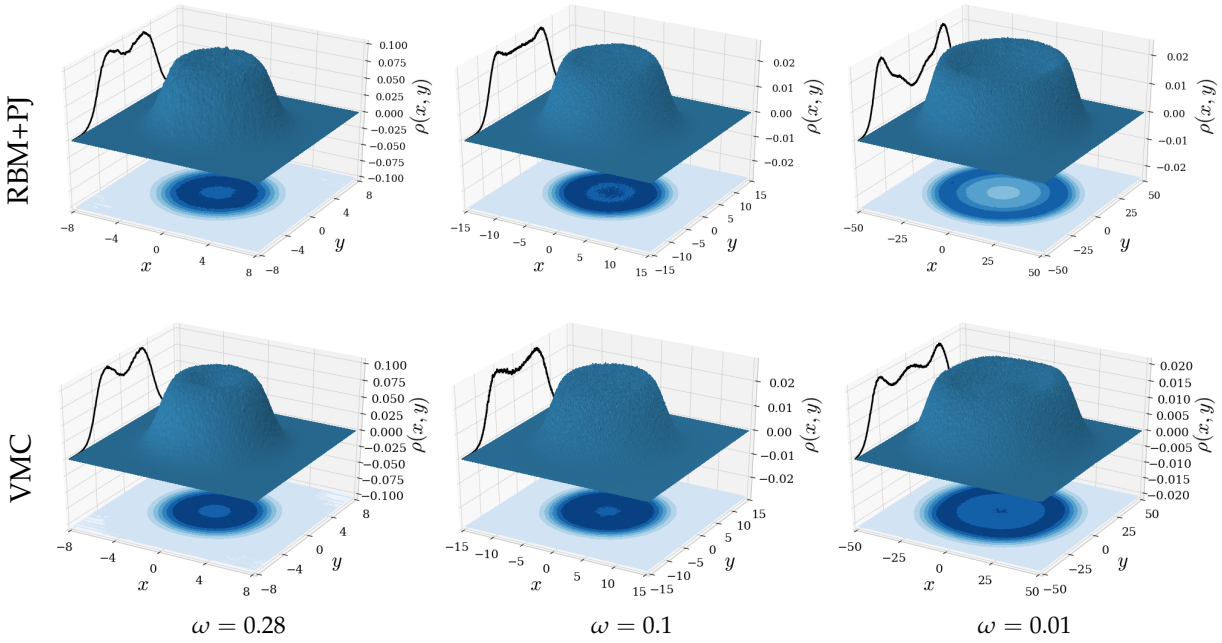


Figure 12.17: One-body density of two-dimensional circular quantum dots with $N = 6$ electrons and frequencies $\omega = 0.28$, 0.1 and 0.01, where the surface plot and the contour plot on the xy-plane illustrate the density and the graph on the yz-plane represents the cross-section through $x = 0$. The surface plots are noise-reduced using a Savitzky-Golay filter, and for abbreviations see the text.

indication of the Wigner localization effect discussed in section 3.3.1. As in the classical limit, the two electrons will repel each other and seldom be located at the same place when their kinetic energy is low. RBM+PJ possibly provides a sharper ridge for $\omega = 0.01$, which is closer to the DMC results found from Høgberget [19] and thus perhaps more correct. However, the energy provided by RBM+PJ and VMC are more or less identical (0.074107 and 0.074070 respectively), which means that the potential difference does not give effect to the energy.

We repeat the exercise for quantum dots with $N = 6$ electrons, and obtain the plots in figure (12.17). When using VMC, we observe the same tendency as for the dots with $N = 2$ electrons, where we get an additional peak in the density plot as the frequency decreases. However, for RBM+PJ, we do not get this peak in the center, but rather a significant density drop. This is contrary to the DMC one-body density plots obtained by Høgberget [19], where it is a sharp peak in the center. The density plot for quantum dots is also known to approach the classical limit as the frequency is decreased [18], where the potential energy is minimized when we have one electron in the center and five electrons surrounding it. In other words, RBM+PJ appears to be wrong for this case.

In order to give a more qualitative comparison of the various methods, we also present the radial one-body density profile of the quantum dots of frequency $\omega = 0.01$ with $N = 2$ and $N = 6$ electrons, see figure (12.18). We see that all the methods agree for $N = 2$, where RBM+PJ gives the most distinct peak, followed by VMC, RBM+SJ and RBM. For $N = 6$, however, the various methods give completely different density profiles. As discussed for the spatial density profile, we believe that the VMC is the most correct because of the peak in the center.

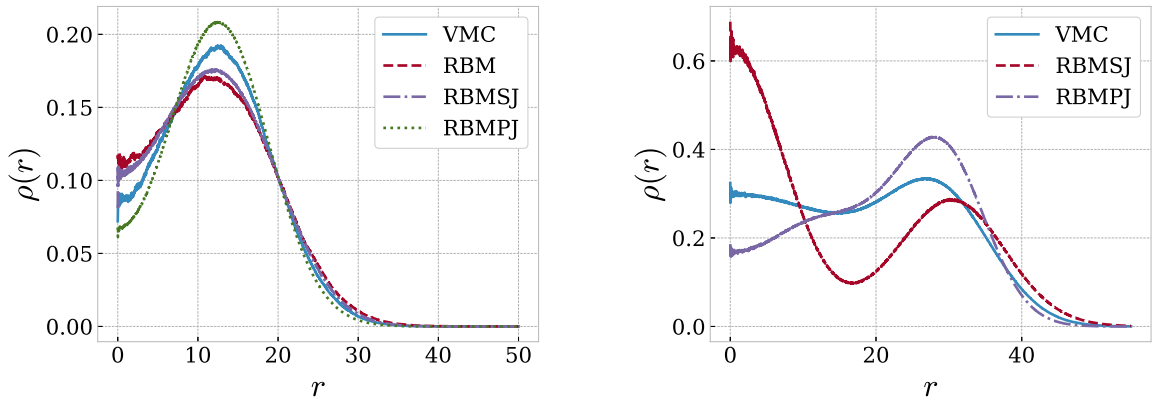


Figure 12.18: One-body density of two-dimensional quantum dots of frequency $\omega = 0.01$ and $N = 72, 90$. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

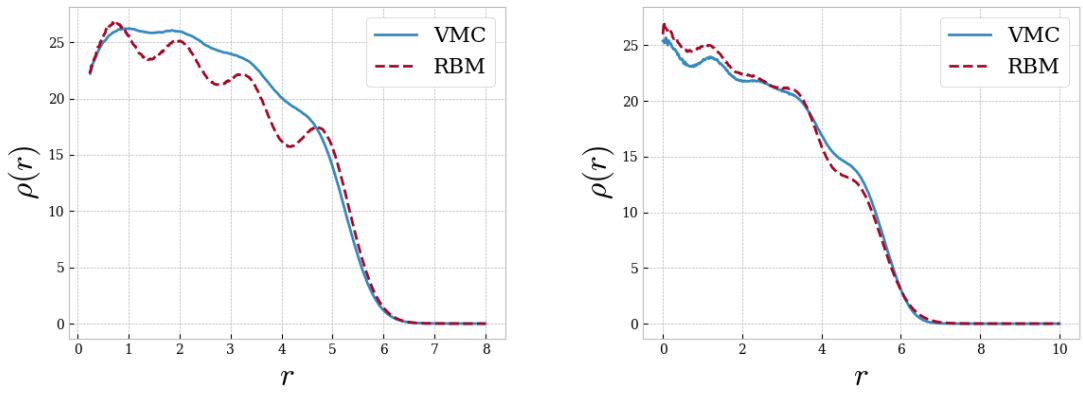
Table 12.7: Energy of large quantum dots with $N = 72$ and 90, and $\omega = 1.0$. All energies are given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit. For abbreviations see the text.

	N	RBM	VMC
2D	72	1355.37(2)	1340.520(7)
	90	2194.12(9)	1990.89(2)
3D	70	1129.40(2)	1108.950(4)

12.4.6 Large dots

In order to test the code, we also decided to run for systems with $N > 56$. This has no scientific significance, other than testing how far a VMC code can go. One thing is that the computations get extremely expensive as the number of particles increases, but we have also seen that the statistical error increases as the system size increases. This means that we cannot just crack up the wall clock time and wait when studying large systems; at some point, the standard error gets insufficiently large, and we need to increase the number of Monte Carlo cycles further. The learning rate also needs to be decreased as the system sizes increase, which requires more iterations. We look at weakly interacting particles, so we will not get any relativistic effects even when adding many particles. In table (12.7), the ground state energy of quantum dots with frequency $\omega = 1.0$ and $N = 72, 90$ electrons in two dimensions and $N = 70$ particles in three dimensions is listed. We observe that the difference between VMC and RBM is quite significant, especially for $N = 90$ electrons in two dimensions. We suspect that this simulation simply has not converged.

In figure (12.19), the radial one-body density profile is plotted for two-dimensional quantum dots with frequency $\omega = 1.0$ and $N = 72, 90$ electrons. Again, we observe the same peaks as we observed in section 12.4.2, and the number of peaks substantiates that we get an additional peak every time we add a closed-shell. However, for $N = 90$, the peaks are not as significant as before, most notably for the RBM, which is another evidence that the simulation has not converged.



$N = 72$ $N = 90$

Figure 12.19: Radial one-body density profile of large quantum dots with $N = 72$ and 90, and frequency $\omega = 1.0$ produced with RBM and VMC. The methods were detailed in the introductory words to this chapter, and for abbreviations see the text.

12.5 Atoms

The next systems we will address are the real atomic systems, which have been investigated by physicists since the childhood of quantum mechanics. This section is added to show the flexibility of the implemented code, which can easily be expanded to new systems. We use the simple hydrogen-like orbitals in our calculations, detailed in section 4.3. In table (12.8) we present the ground state energy of the four smallest closed-shell atoms Helium, Beryllium, Neon, and Magnesium, including an overview of how the energy is distributed and the ratio between kinetic and potential energy. We also compare the obtained results to experimental values. What we see is that the total energy is similar but slightly higher than our reference, which is as expected as we use a simple basis.

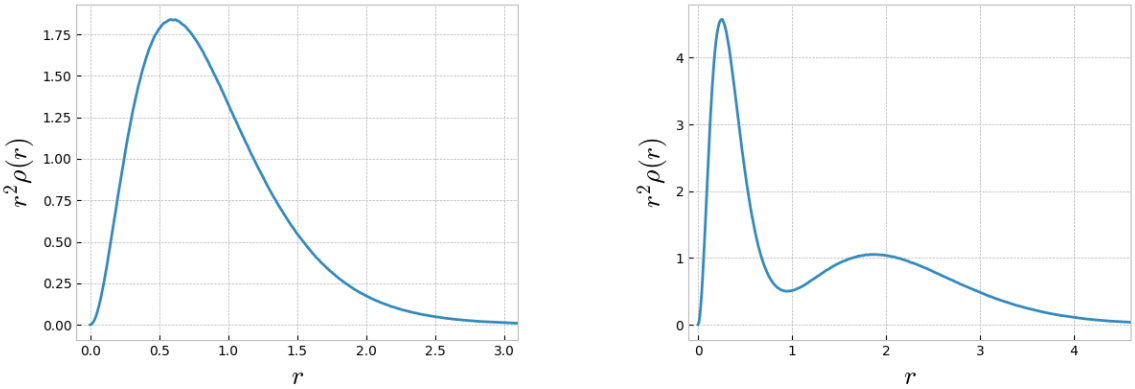
Table 12.8: Ground state energy of neutral atoms with atomic number Z produced by VMC. We present the total energy ($\langle \hat{H} \rangle$), the external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$), the interaction energy ($\langle \hat{V}_{\text{int}} \rangle$), the kinetic-potential energy ratio ($\langle \hat{T} \rangle / \langle \hat{V} \rangle$) and experimental values (Expr.). The latter were taken from Degroote [87], table 4.4. The energy is given in atomic units, and the numbers in parenthesis is the statistical error. For abbreviations see the text.

Atom	Z	Expr. (Ref.[87])	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$	$\langle \hat{T} \rangle / \langle \hat{V} \rangle$
He	2	-2.9037	-2.8719(3)	2.813(3)	-6.696(4)	1.010(7)	-0.495
Be	4	-14.6674	-14.4992(5)	15.465(6)	-34.987(7)	5.023(1)	-0.516
Ne	10	-128.9383	-128.09(1)	133.4(2)	-318.4(2)	56.94(5)	-0.510
Mg	12	-200.054	-196.81(4)	251.9(2)	-557.0(2)	108.23(6)	-0.379

If we once again recall the virial theorem first introduced in section 2.6, it reads

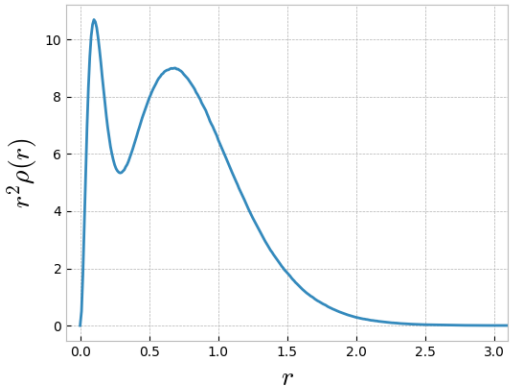
$$2\langle \hat{T} \rangle = -(\langle \hat{V}_{\text{ext}} \rangle + \langle \hat{V}_{\text{int}} \rangle) \tag{12.3}$$

for atoms and molecules under the assumption that $V_{\text{int}} \propto r^{-1}$. For quantum dots, we found this assumption to be inaccurate, and our modified virial theorem broke down for strongly interacting systems. To see if the same happens fro atoms, we again introduce the ratio between the kinetic and potential energy, $R = \langle \hat{T} \rangle / \langle \hat{V} \rangle$, which for this system should be $R = -0.5$ in order to fulfill equation (12.3). In the last column of table 12.8, this ratio is presented for the various atoms, and for Helium, Beryllium and Neon we see that the ratio actually is close to -0.5. However, for Magnesium, the value is off, which could either be a result of too strong attraction force from the nucleus or some errors in the calculations. We believe that it is the latter, as we get an accurate result with Neon, and perhaps the model has just not converged. Further, we look at the radial one-body density profiles, which are presented in figure (12.20) for our four atoms. To reveal details, they are multiplied with r^2 as for example Høgberget [19] has done before us. We observe that the densities mostly are similar to his DMC results, which again substantiates that our framework and electron density computations work as they should. However, for Magnesium, the density plot is slightly different, which probably is related to the potential bug discussed above.

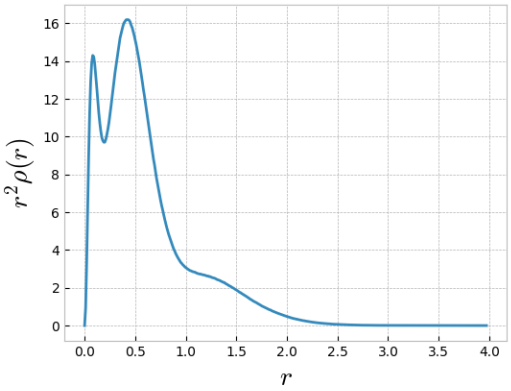


Helium

Beryllium



Neon



Magnesium

Figure 12.20: Radial one-body density profiles, $\rho(r)$, as a function of the radial distance from the nucleus, multiplied with r^2 in order to reveal fluctuations. We look at the Helium atom (upper left), the Beryllium atom (upper right), the Neon atom (lower left) and the Magnesium atom (lower right). VMC was used, which is detailed in the text. The number of Monte Carlo cycles used was $M = 2^{28} = 268,435,456$ and the ADAM optimizer was used. Natural units were used, see appendix B.

Part VI

Conclusion

CHAPTER 13

Retrospect and Future Work

In this chapter, we will make an attempt to compress the relatively comprehensive discussion in the previous chapter down to a more tangible conclusion. Thereafter, we address some possible extensions of our work, how our contributions can be used to solve the many-body quantum puzzle and why it is important.

We have seen that the plain restricted Boltzmann machine (RBM) is capable of producing reasonable ground state energy estimates, and when we add more intuition in the form of Jastrow factors of different complexities the energy drops further towards the diffusion Monte Carlo (DMC) energy. Most notably, an RBM with the Padé-Jastrow factor (RBM+PJ) provides ground state energies and statistical errors lower than the VMC energy for the smallest dots. This indicates that the method can provide a wave function closer to the exact one than standard VMC. However, for larger quantum dots, the RBM+PJ gives a slightly higher energy than the VMC, but we suspect this is a consequence of a large number of variational parameters as we consequently set the number of hidden units, H, equal to the number of electrons in the dot, N. In machine learning terms, we use a too complex model for our problem. We decided to do this because Gjestvang & Nordhagen [76] found $H = N$ to be optimal for small quantum dots, but it could be different for larger dots. Carleo & Troyer [15] operates with a hidden variable density $\alpha = H/F$ with F as the degrees of freedom (number of visible units), which they set to an integer number and thus end up with more variational parameters than we do. A conclusion is that the number of hidden nodes might not be optimal, and thus some more investigation is needed. We also observe that all the methods more or less give the same ratio between kinetic and potential energy for all system sizes and all frequencies, which means the electron configuration is fundamentally different for the different methods.

Throughout the results, we had a thorough discussion of the electron density provided by the various methods, which revealed some significant differences between the methods that cannot be seen just from the ground state energy. The most notable difference is found for the one-body density produced using VMC and RBM, where RBM tends to exaggerate the fluctuations compared to VMC. As discussed, this difference is probably caused by how the two methods model the electron correlations. The same effect was found in the two-body density plots, where the difference between the various correlation models is even more significant. In general, the RBM+PJ and VMC give a more significant electron-electron repulsion than the fellow methods RBM and RBM with a simple Jastrow factor (RBM+SJ).

This announces that energy estimates are not necessarily the best way to compare an RBM to standard VMC, other observable are potentially more crucial. The RBM+SJ is an excellent example of this, as it provides energy estimates similar to the VMC, but the two-body density plots exploit that the correlations were somewhat weaker. In general, we believe that the Padé-Jastrow factor works better than the simple Jastrow factor as it provides a lower energy and is constructed to model the electron-electron cusp correctly. As the simple Jastrow factor is more or less as computationally expensive as the Padé-Jastrow factor, we see no reason to select RBM+SJ

instead of RBM+PJ.

Based on the discussions above, the RBM provides exciting results, but at its current version it is not able to compete with the existing many-body methods neither when it comes to performance, nor computational cost. However, we see the outcome of this work as a step in the right direction, and with some more investigation we believe that the RBM can be an alternative to traditional methods. More precisely, the plain RBM has some properties that makes it able to estimate the ground state energy at a lower cost than the VMC, and other RBM structures, for instance based on spherical coordinates, might enhance the performance at the same cost. We also see a bright future for the RBM+PJ, which for some systems give a lower energy than VMC, and our thought is that it can outperform the VMC if the cost is reduced.

If we now recall the goals presented in the introduction, we see that the first goal was to develop a VMC framework for the studies of large fermionic systems. This framework has been validated in the results, and give consistent results with references. The next goal was to extend the code to include RBMs. As far as we know, no one has done anything comparable, and it is therefore hard to validate this specific implementation. However, when comparing to the results obtained using other methods, we are confident that also this implementation is correct. Thus, it implies that our 7000 lines of code is implemented correctly. The third goal was to use our framework to study ground state properties of atoms and quantum dots. The quantum dots were studied thoroughly, both using VMC and RBM with various correlation factors. Further, we studied the atoms using VMC, but did not have time or manage to study them using the RBMs. We put some effort in trying to model the atoms using the same RBMs as for the quantum dots, but even with a large number of hidden nodes, those Gaussian-binary unit RBMs were not flexible enough to capture the properties of the atoms. Other possible attempts include expanding a Hartree-Fock basis in a set of RBMs, or simply choose another RBM structure which is not based on the Gaussian mapping. This is something that can and should be tried, and is one among many things that could be done in the future work.

Future work

As the use of machine learning for solving the many-body problem is just in the starting block, there are millions of things one could try. With the same approach as we did, there are plenty of structures, hyper-parameter setting and initial conditions that we did not have time to check out. An example is to investigate RBMs with a smaller number of hidden nodes than we did. Also, writing an RBM-code in spherical coordinates, instead of Cartesian coordinates, could be interesting as it might be easier to model the correlations in that coordinate system. Additionally, implementing a VMC code in spherical coordinates is less hassle because of the Jastrow factors.

Moreover, investigating more complicated systems with an unknown wave function would be interesting, as we believe those systems are the primary applications of the RBMs. In the first place, multi-quantum dots (multiple quantum dots with intern connections) is a good candidate as there exist comparable experiments [22, 23]. To simulate quantum-dots in-medium, one can use a screening of Coulomb interactions. As the RBM is able to model the two-body correlations, it is also imaginable that it can model the three-body correlations and therefore be used to simulate nuclear systems.

In section 7.2, we discussed the sign-problem plaguing DMC, and that shadow wave functions can be used to bypass this problem. There are many similarities between the shadow wave functions and wave functions based on restricted Boltzmann machines, and it would be interesting to investigate this link further.

As the quantum simulations are costly, one should always try to find the bottleneck and optimize that part of the code. For RBMs, the bottleneck might be the neural networks, which

can be evaluated extremely fast on a GPU. However, the remaining framework is probably faster to evaluate on CPUs, so a combination of GPU and CPU would might be optimal.

APPENDIX A

Dirac Formalism

The Dirac formalism, also called bracket notation, was suggested by Dirac [88] in a 1939 paper to improve the reading ease. The notation unites the integral representation and the matrix representation in an elegant fashion by representing all elements in the *Hilbert-space*.

The Hilbert-space is a complete linear vector space, which allows length, inner products and angles between vectors to be measured. A column vector in the space is denoted by $|\psi\rangle$, which is called the *ket*, and by taking the Hermitian conjugate of it we obtain the corresponding row vector, $(|\psi\rangle)^+ = \langle\psi|$ called the *bra*. As the Hilbert-space is characterized by linearity, it requires that the sum of two element in the space is also an element in the space,

$$|\psi_1\rangle + |\psi_2\rangle = |\psi_1 + \psi_2\rangle\,, \tag{A.1}$$

and that two elements always *commute*,

$$|\psi_1\rangle + |\psi_2\rangle = |\psi_2\rangle + |\psi_1\rangle\,. \tag{A.2}$$

Another important property is that an element of the space multiplied with a complex number is also an element of the space,

$$c\,|\psi\rangle = |c\psi\rangle \qquad \forall \quad c \in \mathbb{C}. \tag{A.3}$$

To fully utilize the notation, orthogonality properties need to be taken into account. Assume that we have a orthogonal basis set

$$\{\psi_1, \psi_2, \ldots, \psi_n\}. \tag{A.4}$$

The inner product between two basis elements is then given by

$$\langle\psi_i|\psi_j\rangle = \begin{cases} \neq 0 & \text{if} \quad i = j \\ = 0 & \text{otherwise} \end{cases} \equiv \delta_{ij} \tag{A.5}$$

where we have introduced the Kronecker delta δ_{ij}. If we further require that our basis is *orthonormal*, the inner product is 1, and we can prove the *completeness relation*. Assume we want to expand a vector $|\chi\rangle$ in our orthonormal basis set,

$$|\chi\rangle = \sum_{i=1}^{n} c_i\,|\psi_i\rangle\,. \tag{A.6}$$

By multiplying with one of the basis vectors on the left hand side, the sum collapses, and we are just left with one of the coefficients c_j,

$$\langle\psi_j|\chi\rangle = \sum_{i=1}^{n} c_i\,\langle\psi_j|\psi_i\rangle = c_j. \tag{A.7}$$

If we now again insert this into the expansion, we obtain

$$|\chi\rangle = \sum_{i=1}^{n} \underbrace{\langle \psi_i | \chi \rangle}_{c_i} |\psi_i\rangle = \left[\sum_{i=1}^{n} |\psi_i\rangle \langle \psi_i| \right] |\chi\rangle \tag{A.8}$$

which implies that the outer product is equal to 1 (vectorized equal to the identity matrix),

$$\sum_{i=1}^{n} |\psi_i\rangle \langle \psi_i| = 1. \tag{A.9}$$

We can use these properties to demonstrate how the normalization constant can be obtained without explicitly solving any integrals. Consider two spin-1/2 particles, for example the electron and the proton in the ground state of Hydrogen. There are then four possible states: the *triplet* with $s = 1$ and the *singlet* with $s = 0$. The latter reads [40]

$$|00\rangle = A\big(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle \big) \tag{A.10}$$

which is associated with the bra

$$\langle 00| = \big(|00\rangle \big)^{+} = A\big(\langle\uparrow\downarrow| - \langle\downarrow\uparrow| \big). \tag{A.11}$$

The inner product is then given by

$$\begin{aligned} \langle 00|00\rangle &= A\big(\langle\uparrow\downarrow| - \langle\downarrow\uparrow| \big) A\big(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle \big) \\ &= A^2\big(\langle\uparrow\downarrow|\uparrow\downarrow\rangle - \langle\uparrow\downarrow|\downarrow\uparrow\rangle - \langle\downarrow\uparrow|\uparrow\downarrow\rangle + \langle\downarrow\uparrow|\downarrow\uparrow\rangle \big) \\ &= A^2\big(1 - 0 - 0 + 1\big) \\ &= 2A^2 = 1 \end{aligned} \tag{A.12}$$

where we have assumed that also the states $|\uparrow\downarrow\rangle$ and $|\downarrow\uparrow\rangle$ are orthonormal. From this, we can see that the normalization constant A must be equal to $1/\sqrt{2}$.

Further, we also use the Dirac notation as a short-hand notation of the Slater determinant, discussed in section 3.2.2. Instead of writing out the entire determinant, it is common to write it as

$$|\psi_1\psi_2,\ldots,\psi_N\rangle \equiv \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\boldsymbol{r}_1,\sigma_1) & \psi_2(\boldsymbol{r}_1,\sigma_1) & \ldots & \psi_n(\boldsymbol{r}_1,\sigma_1) \\ \psi_1(\boldsymbol{r}_2,\sigma_2) & \psi_2(\boldsymbol{r}_2,\sigma_2) & \ldots & \psi_N(\boldsymbol{r}_2,\sigma_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\boldsymbol{r}_N,\sigma_N) & \psi_2(\boldsymbol{r}_N,\sigma_N) & \ldots & \psi_N(\boldsymbol{r}_N,\sigma_N) \end{vmatrix} \tag{A.13}$$

which is extensively used in for instance second quantization. This should not be confused with the Hartree product.

APPENDIX B

Natural Units

In everyday life, we usually stick to the standard SI units when measuring or expressing distances, energies, weights, time *et cetera*. A standard unit system is important because it is common for people around the world so that people can communicate with each other conveniently across borders with a reduced risk of misconceptions. Another point is that people gradually develop an intuition about a set of units when they are used frequently, such that they immediately observe if a number makes sense or not when it is expressed in the preferred units. As a European, I always measure length in meters, so when people say they are 1.80m tall, I can easily imagine their height. On the other hand, when Americans tell their height, I do not have an intuition for how tall 5 feet 3 inches is.

However, in science, the SI units are often not the preferred ones, especially when things get very large or very small. For instance, measuring cosmological distances in meters is very unpractical, as the distance to the Sun is $\sim 1.5 \cdot 10^{11}$m and the distance to our closest neighbor galaxy Andromeda is $\sim 2.4 \cdot 10^{22}$m! Instead, we use units like the astronomical unit [a.u.] (should not be confused with atomic units) and light-years.

For small scales, the situation is similar. For instance, the most probable distance between the nucleus and the electron in the Hydrogen atom is the Bohr radius, which is $a_0 \approx 5.3 \cdot 10^{-11}$m, which again is very unpractical to work with. Instead, we define so-called natural units where $a_0 = 1$, and scale the other quantities after that. We will in this chapter present how the quantum dot and atomic Hamiltonian can be scaled in natural units. We will stick to Hartree atomic units, as it gives elegant expressions of the Hamiltonians.

B.1 Quantum dots

For circular quantum dots, the one-dimensional Hamiltonian in SI units reads

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}m\omega^2 x^2 \tag{B.1}$$

with $\hbar$ as the reduced Planck's constant, m as the electron mass and ω as the oscillator frequency. The corresponding wave functions read

$$\phi_n(x) = \frac{1}{\sqrt{2^n n!}} \cdot \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \exp\Big(-\frac{m\omega}{2\hbar}x^2\Big) H_n\Big(\sqrt{\frac{m\omega}{\hbar}}x\Big), \tag{B.2}$$

where $H_n(x)$ is the Hermite polynomials. We want to get rid of $\hbar$ and m in equation (B.1) to make it dimensionless, which can be accomplished by scaling $\hat{\mathcal{H}}' = \hat{\mathcal{H}}/\hbar$, such that the Hamiltonian reduces to

$$\hat{\mathcal{H}}' = -\frac{\hbar}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}\frac{m\omega^2}{\hbar}x^2. \tag{B.3}$$

Moreover, we observe that the fraction $\hbar/m$ appears in both terms, which can be avoided by introducing a characteristic length $x' = x/\sqrt{\hbar/m}$. The final Hamiltonian is

$$\hat{\mathcal{H}} = \frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}\omega^2 x^2 \tag{B.4}$$

which corresponds to setting $\hbar = m = 1$. In natural units, one often sets $\omega = 1$ as well by scaling $\hat{\mathcal{H}}' = \hat{\mathcal{H}}/\hbar\omega$, but since we want to keep the ω-dependency, we do it slightly different. This means that the exact wave functions for the one-particle one-dimensional case is given by

$$\phi_n(x) = \frac{1}{\sqrt{2^n n!}} \cdot \left(\frac{\omega}{\pi}\right)^{1/4} \exp\Big(-\frac{\omega}{2}x^2\Big) H_n(\sqrt{\omega}x). \tag{B.5}$$

B.2 Atoms

The atomic Hamiltonian for an electron in subshell l affected by a nucleus with atomic number Z reads

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m_e}\nabla^2 - \frac{1}{4\pi\epsilon_0}\frac{Ze^2}{r} + \frac{\hbar^2 l(l+1)}{2m_e r^2}. \tag{B.6}$$

in SI units where e is the elementary charge and $k_e = 1/4\pi\epsilon_0$ is Coulomb's constant. Again we want to get rid of the reduced Planck's constant $\hbar$ and the electron mass m_e in order to make the Hamiltonian dimensionless, which we do by multiplying all terms by $(4\pi\epsilon_0)^2\hbar^2/m_e e^4 Z^2$,

$$\hat{\mathcal{H}} \cdot \frac{(4\pi\epsilon_0)^2\hbar^2}{m_e e^4 Z^2} = -\frac{(4\pi\epsilon_0)^2\hbar^4}{2m_e^2 e^4 Z^2}\nabla^2 + \frac{4\pi\epsilon_0\hbar^2}{m_e e^2 Z r} - \frac{(4\pi\epsilon_0)^2\hbar^4}{m_e^2 e^4 Z^2}\frac{l(l+1)}{r^2}. \tag{B.7}$$

This might look very chaotic, but by exploiting that the Bohr radius,

$$a_0 \equiv \frac{4\pi\epsilon_0\hbar^2}{m_e e^2 Z}, \tag{B.8}$$

is found in all the right hand side terms, we can simplify the Hamiltonian to be

$$\hat{\mathcal{H}} \cdot \frac{(4\pi\epsilon_0)^2\hbar^2}{m_e e^4 Z^2} = -\frac{a_0^2}{2}\nabla^2 + a_0\frac{Z}{r} - a_0^2\frac{l(l+1)}{2r^2}. \tag{B.9}$$

We obtain the dimensionless Hamiltonian in Atomic units by scaling $r' = r/a_0$ and $\hat{\mathcal{H}}' = \hat{\mathcal{H}}/(m_e e^4 Z^2/(4\pi\epsilon_0)^2\hbar^2)$, getting

$$\hat{\mathcal{H}} = -\frac{1}{2}\nabla^2 - \frac{Z}{r} + \frac{l(l+1)}{2r^2}, \tag{B.10}$$

which again corresponds to setting $\hbar = m_e = k_e = e = 1$.

APPENDIX C

Evaluation of a general Gaussian-binary RBM wave function

In this appendix, we start from a general Gaussian-binary restricted Boltzmann machine (RBM) and set up the system energy and joint probability distribution. Thereafter, we derive the marginal distribution which will later be used as the wave function. Closed-form expressions for the logarithmic gradient and the Laplacian of the wave function to be used in the local energy calculations, as we as the parameter gradient used in the parameter update will be given. We start from the most basic Gaussian-binary restricted Boltzmann machine in the form of

$$\begin{aligned} E(\boldsymbol{x},\boldsymbol{h}) &= \sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} b_j h_j - \sum_{i=1}^{F}\sum_{j=1}^{H}\frac{x_i w_{ij} h_j}{\sigma_i^2} \\ &= \sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} h_j\Big(b_j + \sum_{i=1}^{F}\frac{x_i w_{ij}}{\sigma_i^2}\Big) \end{aligned} \tag{C.1}$$

as discussed in chapter 5. If we now denote the expression in the last parenthesis by $f_j(\boldsymbol{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ includes all the parameters, we end up with the expression of the general Gaussian-binary restricted Boltzmann machine,

$$E(\boldsymbol{x},\boldsymbol{h}) = \sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{H} h_j f_j(\boldsymbol{x};\boldsymbol{\theta}) \tag{C.2}$$

where $f_j(\boldsymbol{x};\boldsymbol{\theta})$ in principle can be any function of $\boldsymbol{x}$ and $\boldsymbol{\theta}$. From this expression, we obtain the joint probability distribution

$$\begin{aligned} P(\boldsymbol{x},\boldsymbol{h}) &= \frac{1}{Z}\exp(-\beta E(\boldsymbol{x},\boldsymbol{h})) \\ &\propto \exp\left(\sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma^2}\right)\exp\left(\sum_{j=1}^{H}\Big(h_j f_j(\boldsymbol{x};\boldsymbol{\theta})\Big)\right) \\ &\propto \exp\left(\sum_{i=1}^{F}\frac{(x_i-a_i)^2}{2\sigma^2}\right)\prod_{j=1}^{H}\exp\Big(h_j f_j(\boldsymbol{x};\boldsymbol{\theta})\Big) \end{aligned} \tag{C.3}$$

where we fix $\beta = 1/k_B T = 1$ and ignore the partition function Z. Our main interest is the marginal distribution, which we will derive in detail.

C.1 Marginal distribution

In chapter 5, we presented the marginal distribution of the visible nodes as

$$P(\boldsymbol{x}) = \sum_{\{\boldsymbol{h}\}} P(\boldsymbol{x}, \boldsymbol{h}) \tag{C.4}$$

which for our function is

$$\begin{aligned}
P(\boldsymbol{x}) &\propto \sum_{\{\boldsymbol{h}\}} \exp\Big(\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\Big) \prod_{j=1}^{H} \exp\Big(h_j f_j(\boldsymbol{x}, \boldsymbol{\theta})\Big) \\
&= \sum_{h_1=0}^{1} \sum_{h_2=0}^{1} \dots \sum_{h_H=0}^{1} \exp\Big(\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\Big) \exp\Big(h_1 f_1\Big) \exp\Big(h_2 f_2\Big) \dots \exp\Big(h_H f_H\Big) \\
&= \exp\Big(\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\Big) \sum_{h_1=0}^{1} \exp\Big(h_1 f_1\Big) \sum_{h_2=0}^{1} \exp\Big(h_2 f_2\Big) \dots \sum_{h_H=0}^{1} \exp\Big(h_H f_H\Big) \\
&= \exp\Big(\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\Big) \prod_{j=1}^{H} \sum_{h_j=0}^{1} \exp\Big(h_j f_j\Big) \\
&= \exp\Big(\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\Big) \prod_{j=1}^{H} \Big[1 + \exp\Big(f_j(\boldsymbol{x}; \boldsymbol{\theta})\Big)\Big]
\end{aligned} \tag{C.5}$$

where we exploit that only an element in the product contains a certain hidden node h_j.

C.2 Conditional distribution

WRITE

C.3 Closed-form expressions of gradients

By defining the single-particle function as the marginal probability, we have seen that the wave function of a general Gaussian-binary restricted Boltzmann machine takes the form

$$\Psi(\boldsymbol{x}; \boldsymbol{a}, \boldsymbol{\theta}) = \exp\left(-\sum_{i=1}^{F} \frac{(x_i - a_i)^2}{2\sigma^2}\right) \prod_{j=1}^{H} \Big[1 + \exp\big(f_j(\boldsymbol{x}; \boldsymbol{\theta})\big)\Big] \tag{C.6}$$

where $f_j(\boldsymbol{x}; \boldsymbol{\theta})$ is an arbitrary function of the coordinates $\boldsymbol{x}$ and the weights $\boldsymbol{\theta}$. The Gaussian part is straight-forward to differentiate, so we will keep our attention on the product,

$$\Psi_{\mathrm{rp}}(\boldsymbol{x}; \boldsymbol{\theta}) = \prod_{j=1}^{H} \Big[1 + \exp\big(f_j(\boldsymbol{x}; \boldsymbol{\theta})\big)\Big]. \tag{C.7}$$

We will henceforth no longer specify the arguments $\boldsymbol{x}$ and $\boldsymbol{\theta}$ of the functions, as all the functions take the same arguments. By introducing the functions

$$p_j \equiv \frac{1}{1 + \exp\big(+f_j\big)} \quad \wedge \quad n_j \equiv \frac{1}{1 + \exp\big(-f_j\big)}, \tag{C.8}$$

where the last one is the sigmoid function, we find the gradient and Laplacian of $\ln \Psi_{rp}$ to be

$$\nabla_k \ln \Psi_{\text{rp}} = \sum_{j=1}^{H} n_j \nabla_k(f_j) \tag{C.9}$$

and

$$\nabla_k^2 \ln \Psi_{\text{rp}} = \sum_{j=1}^{H} n_j \left[\nabla_k^2(f_j) + p_j \left(\nabla_k(f_j)\right)^2\right] \tag{C.10}$$

respectively. Those expressions can be used to find the kinetic energy directly, as the kinetic energy contribution from this specific element is just the sum over the gradients and Laplacians, $T = \sum_{k=1}^{F} [\nabla_k^2 \ln \Psi_{\text{rp}} + (\nabla_k \ln \Psi_{\text{rp}})^2]$. An arbitrary parameter θ_i can be updated according to the log-likelihood function which turns out to be just the derivative of the log-likelihood function

$$\frac{\partial}{\partial \theta_i} \ln \Psi_{\text{rp}} = \sum_{j=1}^{H} n_j \frac{\partial}{\partial \theta_i}(f_j) \tag{C.11}$$

and the ratio between the new and the old wave function elements can be found by the product

$$\frac{\Psi_{\text{rp}}^{\text{new}}}{\Psi_{\text{rp}}^{\text{old}}} = \prod_{j=1}^{H} \frac{p_j^{\text{old}}}{p_j^{\text{new}}}. \tag{C.12}$$

As a conclusion, what we actually need to calculate to find respective expressions for each wave function element is $\nabla_k(f_j)$, $\nabla_k^2(f_j)$ and $\partial_{\theta_i}(f_j)$, which is naturally simpler than differentiating the entire wave function element. This applies to all elements on the form presented in equation (C.6), including general Gaussian-binary restricted Boltzmann machines and deep Boltzmann machines as long as all the units are Gaussian-binary.

APPENDIX D

Collection of Results

In this appendix, we present a more or less complete collection of the results obtained through our work, including energies, one-body density profiles and two-body density results. It is meant as a complementary part to substitute the selected results in chapter 12, where the (subjective) most important results are presented and discussed. For that reason, the discussion above covers most of the results also in this appendix, and they will not be discussed further. The raw files, containing direct energies and standard errors from the Abel computer cluster, can be found in Ref. [33]. A complete collection of the plots in this appendix can be found on https://github.com/evenmn/Master-thesis/plots/.

First, the computational time for two- and three-dimensional quantum dots will be listed, which is just the information plotted in figure (12.2). After that, we list all the obtained energies for quantum dots with up to $N = 20$ electrons in two and three dimensions, included the distribution between potential and kinetic energy. Lastly, we present a complete set of the radial and spatial one-body densities and the radial two-body densities.

D.1 CPU time

The CPU-time per iteration was calculated for all the systems we have been looking at, i.e., two-dimensional quantum dots containing up to $N = 90$ electrons and three-dimensional quantum dots containing up to $N = 70$ electrons. As we only consider closed-shell dots, the N's in the tables do only include the magic numbers. Also, we assume that the CPU-time per iteration is independent of the frequency, such that the times are obtained using a variety of different oscillator frequencies. To make precise estimations of the CPU-time per iteration, all simulations were run with $M = 2^{20} = 1,048,576$ cycles per iteration on the computer cluster Abel. As not the entire code can be parallelized, and as the processes need to communicate, the parallelism is not 100% efficient and we also need the same number of cores for all the simulations in order to get comparable times. We decided to use 8 nodes á 16 cores. To get good statistics, we performed at least four independent runs for each system, where the average time over thousands of iterations was calculated automatically by the program. The results can be found in the tables (D.1) and (D.2) for two- and three dimensional quantum dots respectively.

Table D.1: The CPU time (in seconds) for each iteration when simulating two-dimensional circular quantum dots with $N = 2 - 90$ electrons. The time was clocked for $M = 2^{20} = 1,048,576$ Monte Carlo cycles, and to get accurate times we took the average over at least four independent runs with thousands of iterations.

$N \rightarrow$	2	6	12	20	30	42	56	72	90
RBM	6.05	11.25	20.53	38.99	73.72	130.49	213.47	360.22	856.84
RBM+SJ	7.12	14.07	28.42	63.27	122.93	199.60	349.22	-	-
RBM+PJ	7.26	13.50	27.68	57.09	119.17	212.53	382.13	-	-
VMC	5.11	10.51	20.85	41.20	76.26	137.39	230.63	355.81	544.03

Table D.2: The CPU time (in seconds) for each iteration when simulating three-dimensional circular quantum dots with $N = 2 - 70$ electrons. The time was clocked for $M = 2^{20} = 1,048,576$ Monte Carlo cycles, and to get accurate times we took the average over at least four independent runs with thousands of iterations.

$N \rightarrow$	2	8	20	40	70
RBM	7.69	20.92	59.67	171.84	586.39
RBM+SJ	8.95	26.86	94.64	270.92	-
RBM+PJ	8.87	26.36	91.40	293.25	-
VMC	6.70	20.99	62.54	185.65	486.02

D.2 Ground state energy

In this section, we present the ground state energy for two- and three-dimensional quantum dots with up to $N = 20$ electrons with frequencies spanning from $\omega = 0.01$ to $\omega = 10$. We also list the distribution between kinetic energy, external potential energy and internal potential energy, like we did in table (12.6). This serves as another dimension of comparison for the various methods, at the same time as we can verify the virial theorem, described in section 2.6. In addition, one can also compare the energy of dots with the same number of electrons, but in different dimensions (for $N = 2$ and $N = 20$).

We have created respective tables for each of our four methods variational Monte Carlo with a Slater-Jastrow trial wave function (VMC), a Slater determinant with single-particle functions specified by a restricted Boltzmann machine (RBM), an RBM with a simple Jastrow factor (RBM+SJ), and an RBM with the Padé-Jastrow factor(RBM+PJ) in two and three dimensions, resulting in eight tables in total. We first present the energy of the two-dimensional dots, starting from the RBM and then RBM+SJ, RBM+PJ and VMC (D.3-D.6), and then the three-dimensional dots in the same order (D.7-D.10).

D.2.1 Two dimensions

Table D.3: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of two-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.078643(5)	0.009835(3)	0.031930(8)	0.03688(1)
	0.1	0.4743(1)	0.08102(8)	0.2082(2)	0.1851(2)
	0.28	1.07050(4)	0.20869(2)	0.47103(7)	0.39078(7)
	0.5	1.72293(7)	0.38006(6)	0.75598(1)	0.5869(1)
	1.0	3.0803(2)	0.7919(2)	1.3657(4)	0.9227(3)
	2.0	5.5936(3)	1.6377(4)	2.5507(5)	1.4051(4)
	3.0	7.9859(2)	2.5215(3)	3.6733(4)	1.7910(2)
	5.0	12.6141(2)	4.2752(4)	5.9493(6)	2.3896(3)
	10.0	23.7748(7)	9.063(3)	11.132(4)	3.580(1)
6	0.01	0.7072(5)	0.033(2)	0.2660(4)	0.4080(6)
	0.1	3.7337(5)	0.3251(3)	1.4070(9)	2.002(1)
	0.28	7.9273(9)	0.8684(6)	3.009(1)	4.050(2)
	0.5	12.241(1)	1.611(1)	4.709(2)	5.921(2)
	1.0	20.716(1)	3.391(1)	7.914(3)	9.411(2)
	2.0	36.383(5)	8.311(7)	13.705(8)	14.367(6)
	3.0	49.415(1)	10.309(3)	21.456(4)	17.649(2)
	5.0	76.801(6)	23.50(1)	27.33(1)	25.967(7)
	10.0	137.338(4)	45.25(1)	55.75(1)	36.336(6)
12	0.01	2.5106(8)	0.0682(2)	0.893(1)	1.549(1)
	0.1	12.679(2)	0.8141(7)	4.692(1)	7.173(2)
	0.28	26.564(3)	2.254(2)	9.635(3)	14.675(4)
	0.5	40.442(3)	4.116(2)	14.868(4)	21.458(4)
	1.0	67.614(3)	8.953(3)	25.207(6)	33.455(5)
	2.0	115.214(5)	20.760(6)	43.69(1)	50.764(7)
	3.0	158.145(6)	33.020(8)	59.72(1)	65.407(9)
	5.0	239.527(8)	58.22(1)	93.92(2)	87.39(1)
	10.0	435.36(2)	114.61(2)	200.13(4)	120.62(1)
20	0.01	6.217(2)	0.1236(4)	2.244(2)	3.849(2)
	0.1	32.308(5)	1.708(2)	7.680(4)	22.919(7)
	0.28	63.788(4)	4.443(3)	22.707(6)	36.638(7)
	0.5	96.491(4)	8.144(3)	34.953(8)	53.394(8)
	1.0	159.645(5)	17.12(5)	58.74(5)	83.397(9)
	2.0	269.086(8)	43.262(8)	95.17(2)	130.65(1)
	3.0	362.52(1)	57.005(9)	148.08(2)	157.43(1)
	5.0	551.21(2)	115.12(2)	219.12(4)	216.97(2)
	10.0	961.03(4)	260.2(1)	364.8(1)	336.06(7)

Table D.4: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of two-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM+SJ. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.074940(4)	0.007688(2)	0.029945(7)	0.037307(8)
	0.1	0.44856(1)	0.07620(2)	0.19318(4)	0.17918(4)
	0.28	1.03470(7)	0.2163(1)	0.4547(2)	0.3637(2)
	0.5	1.67636(8)	0.3967(2)	0.7366(3)	0.5430(2)
	1.0	3.02108(5)	0.8139(1)	1.3548(2)	0.8524(1)
	2.0	5.5254(1)	1.6572(3)	2.5447(5)	1.3234(3)
	3.0	7.9103(2)	2.5627(7)	3.664(1)	1.6840(5)
	5.0	12.5322(2)	4.369(1)	5.890(2)	2.2838(7)
	10.0	23.6773(3)	9.062(3)	11.208(3)	3.408(1)
6	0.01	0.7006(3)	0.0376(2)	0.2467(4)	0.4163(4)
	0.1	3.63825(9)	0.23798(6)	1.3992(2)	1.3992(2)
	0.28	7.7313(2)	0.7703(1)	2.9608(4)	4.0002(4)
	0.5	11.9392(5)	1.4801(5)	4.678(1)	5.781(1)
	1.0	20.3393(8)	3.308(1)	7.983(2)	9.048(2)
	2.0	35.2446(8)	7.098(2)	14.587(3)	13.560(2)
	3.0	49.050(1)	11.164(3)	20.6469(5)	17.240(3)
	5.0	75.116(1)	19.661(5)	32.283(7)	23.172(3)
	10.0	136.331(2)	41.65(1)	60.53(1)	34.152(5)
12	0.01	2.4950(5)	0.07(2)	0.845(4)	1.58(2)
	0.1	12.5964(7)	0.4520(4)	4.644(1)	7.500(1)
	0.28	26.051(1)	1.7307(6)	9.668(2)	14.652(2)
	0.5	39.6340(7)	3.4852(5)	14.948(2)	21.201(2)
	1.0	66.1898(8)	7.8777(8)	25.822(2)	32.490(2)
	2.0	112.502(2)	18.118(4)	44.118(4)	50.201(6)
	3.0	154.521(3)	28.050(5)	63.84(1)	62.634(6)
	5.0	234.110(6)	52.86(1)	94.50(2)	86.75(1)
	10.0	415.384(7)	108.57(2)	181.90(3)	124.92(1)
20	0.01	6.239(2)	0.1372(6)	2.184(2)	3.919(3)
	0.1	30.624(3)	1.487(2)	10.893(5)	18.243(5)
	0.28	62.786(3)	3.190(2)	22.782(7)	36.814(6)
	0.5	94.755(3)	6.709(2)	34.845(7)	53.200(6)
	1.0	156.816(4)	15.340(3)	59.931(9)	81.545(7)
	2.0	265.66(9)	39.31(8)	95.78(1)	130.57(2)
	3.0	360.630(6)	58.36(1)	141.54(2)	160.72(2)
	5.0	543.06(1)	112.38(2)	210.52(4)	220.15(2)
	10.0	952.71(2)	237.65(4)	392.06(7)	323.00(3)

Table D.5: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of two-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM+PJ. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.074107(8)	0.01031(3)	0.02703(4)	0.03677(3)
	0.1	0.440975(8)	0.09223(9)	0.1757(1)	0.17304(9)
	0.28	1.021668(7)	0.2468(1)	0.4258(2)	0.3490(1)
	0.5	1.659637(6)	0.4305(2)	0.7112(2)	0.5179(2)
	1.0	2.999587(5)	0.8440(3)	1.3418(3)	0.8238(2)
	2.0	5.49475(1)	1.7234(4)	2.4657(4)	1.3057(3)
	3.0	7.87961(1)	2.3144(5)	3.9349(6)	1.6413(3)
	5.0	12.49832(1)	3.9569(7)	6.3068(8)	2.2347(4)
	10.0	23.65275(7)	9.228(3)	11.059(3)	3.366(1)
6	0.01	0.6932(5)	0.031(2)	0.260(2)	0.401(1)
	0.1	3.5700(2)	0.3494(3)	1.2805(9)	1.9401(8)
	0.28	7.6203(2)	0.9519(6)	2.82(1)	3.84(1)
	0.5	11.8074(2)	1.7018(7)	4.513(1)	5.5927(9)
	1.0	20.1832(1)	3.428(1)	8.068(1)	8.687(1)
	2.0	35.0872(3)	7.670(2)	14.139(3)	13.279(2)
	3.0	48.9157(8)	10.789(5)	20.383(5)	17.743(2)
	5.0	74.9545(5)	20.402(5)	31.744(7)	22.809(3)
	10.0	136.1738(8)	42.66(1)	59.71(1)	33.799(5)
12	0.01	2.5019(4)	0.0699(2)	0.893(1)	1.539(1)
	0.1	12.361(1)	0.797(1)	4.394(3)	7.169(3)
	0.28	25.7461(6)	2.415(1)	9.050(2)	14.281(2)
	0.5	39.2661(6)	4.262(2)	14.277(2)	20.728(2)
	1.0	65.7911(5)	8.537(3)	25.197(4)	32.067(3)
	2.0	111.9426(5)	17.817(3)	46.532(4)	47.593(3)
	3.0	154.206(1)	29.701(6)	60.74(1)	63.763(7)
	5.0	233.633(4)	58.33(1)	84.76(1)	90.537(9)
	10.0	415.943(9)	124.13(2)	157.92(3)	133.89(1)
20	0.01	6.210(1)	0.1208(5)	2.189(2)	3.900(2)
	0.1	30.156(1)	1.574(1)	10.473(3)	18.109(3)
	0.28	62.210(1)	4.657(2)	21.227(4)	36.106(4)
	0.5	94.127(1)	8.249(3)	33.543(5)	52.335(4)
	1.0	156.099(1)	16.768(6)	58.513(8)	80.818(6)
	2.0	262.598(1)	34.758(6)	108.546(9)	119.293(7)
	3.0	359.072(4)	60.75(2)	140.51(4)	157.82(2)
	5.0	539.13(1)	120.09(2)	188.45(2)	230.58(1)
	10.0	947.33(2)	257.67(5)	348.35(6)	341.31(3)

Table D.6: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of two-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using VMC. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.074070(8)	0.00947(3)	0.02732(5)	0.03728(4)
	0.1	0.44129(1)	0.09117(9)	0.1789(1)	0.17119(9)
	0.28	1.02192(1)	0.2477(1)	0.4256(2)	0.3487(1)
	0.5	1.65974(1)	0.4346(2)	0.7057(2)	0.5195(2)
	1.0	2.99936(1)	0.8523(3)	1.3149(3)	0.8321(2)
	2.0	5.49689(4)	1.811(2)	2.403(2)	1.283(1)
	3.0	7.88401(4)	2.732(2)	3.500(3)	1.652(1)
	5.0	12.50405(5)	4.619(4)	5.629(4)	2.255(2)
	10.0	23.65035(1)	9.529(1)	10.538(1)	3.583(1)
6	0.01	0.69647(2)	0.02886(1)	0.23363(5)	0.43398(5)
	0.1	3.5695(1)	0.3201(3)	1.2934(6)	1.9560(5)
	0.28	7.6219(1)	0.9105(4)	2.8821(9)	3.8292(7)
	0.5	11.8104(2)	1.6710(7)	4.535(1)	5.6045(9)
	1.0	20.1918(2)	3.405(1)	8.046(1)	8.741(1)
	2.0	35.0734(3)	7.751(2)	13.846(3)	13.476(2)
	3.0	48.8728(4)	12.016(2)	19.682(4)	17.175(2)
	5.0	74.9356(5)	20.796(4)	31.043(6)	23.097(3)
	10.0	136.1522(7)	43.712(9)	58.20(1)	34.240(5)
12	0.01	2.4972(3)	0.05506(2)	0.858(1)	1.584(1)
	0.1	12.29962(9)	0.7524(2)	4.2159(4)	7.3312(4)
	0.28	25.7049(4)	2.090(1)	9.355(2)	14.260(2)
	0.5	39.2421(5)	3.939(2)	14.564(3)	20.739(3)
	1.0	65.7026(4)	9.246(2)	23.079(3)	33.378(3)
	2.0	111.8377(3)	19.678(2)	41.349(3)	50.811(2)
	3.0	154.206(1)	29.701(6)	60.74(1)	63.763(7)
	5.0	232.818(2)	56.157(9)	88.18(1)	88.478(9)
	10.0	415.056(4)	112.99(2)	173.91(3)	128.15(1)
20	0.01	6.2097(8)	0.1005(4)	2.270(3)	3.839(3)
	0.1	30.0403(2)	1.3743(3)	10.206(1)	18.4604(9)
	0.28	62.0755(7)	3.902(2)	22.228(5)	35.946(4)
	0.5	94.0433(9)	7.823(3)	33.938(6)	52.282(5)
	1.0	155.8900(4)	17.921(2)	54.076(3)	83.893(3)
	2.0	262.5339(9)	38.402(3)	95.681(7)	128.451(5)
	3.0	358.927(1)	61.017(5)	133.99(1)	163.924(7)
	5.0	542.680(7)	127.77(2)	177.89(2)	237.12(2)
	10.0	945.596(8)	231.56(4)	389.26(7)	324.77(3)

D.2.2 Three dimensions

Table D.7: This table shows how the total energy ($\langle\hat{H}\rangle$) is distributed between kinetic energy ($\langle\hat{T}\rangle$), external potential energy ($\langle\hat{V}_{\text{ext}}\rangle$) and interaction energy ($\langle\hat{V}_{\text{int}}\rangle$) of three-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle\hat{H}\rangle$	$\langle\hat{T}\rangle$	$\langle\hat{V}_{\text{ext}}\rangle$	$\langle\hat{V}_{\text{int}}\rangle$
2	0.01	0.85193(5)	0.014853(4)	0.03141(9)	0.03893(1)
	0.1	0.5177(1)	0.1249(1)	0.2065(2)	0.1863(2)
	0.28	1.22565(3)	0.36111(4)	0.51675(7)	0.34779(4)
	0.5	2.0269(1)	0.6595(3)	0.8778(4)	0.4896(2)
	1.0	3.7574(1)	1.3224(5)	1.7215(5)	0.7136(2)
	2.0	7.0870(1)	2.7338(6)	3.3183(7)	1.0350(2)
	3.0	10.2981(2)	3.7507(8)	5.2896(9)	1.2578(2)
	5.0	16.7018(1)	6.211(1)	8.890(1)	1.6012(2)
	10.0	32.2186(2)	7.879(3)	22.306(3)	2.0343(2)
8	0.01	1.1350(1)	0.0626(2)	0.3951(7)	0.6774(7)
	0.1	5.8910(6)	0.6480(6)	2.075(2)	3.168(2)
	0.28	12.650(1)	1.931(1)	4.641(2)	6.078(2)
	0.5	19.680(2)	3.601(2)	7.289(4)	8.786(3)
	1.0	33.305(1)	7.032(2)	12.267(3)	14.006(2)
	2.0	58.889(3)	16.976(4)	19.717(5)	22.195(3)
	3.0	81.648(3)	23.987(6)	31.157(8)	26.504(4)
	5.0	126.03(9)	42.68(6)	47.58(5)	35.77(2)
	10.0	231.410(6)	85.62(2)	95.00(2)	50.789(7)
20	0.01	5.6448(4)	0.1624(4)	1.955(2)	3.527(2)
	0.1	27.9277(5)	1.7925(5)	9.676(2)	16.459(2)
	0.28	57.822(1)	5.240(1)	20.384(4)	32.198(3)
	0.5	87.798(5)	9.635(5)	32.12(1)	46.047(9)
	1.0	147.407(3)	22.085(5)	52.32(1)	73.003(8)
	2.0	250.159(7)	49.76(1)	83.69(2)	116.71(1)
	3.0	335.440(5)	64.59(1)	130.27(2)	140.578(9)
	5.0	524.94(2)	121.39(3)	225.61(5)	177.94(2)
	10.0	1005.24(6)	225.76(5)	552.3(1)	227.20(2)

Table D.8: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of three-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM+SJ. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.07994(2)	0.01069(3)	0.03190(8)	0.03735(8)
	0.1	0.50214(3)	0.1178(1)	0.2177(2)	0.1666(1)
	0.28	1.20475(4)	0.3497(2)	0.5326(3)	0.3225(1)
	0.5	2.00371(4)	0.6340(3)	0.9201(4)	0.4496(2)
	1.0	3.73543(4)	1.2801(4)	1.7871(5)	0.6683(2)
	2.0	7.06343(7)	2.7117(7)	3.3574(9)	0.9944(2)
	3.0	10.32289(5)	3.5281(8)	5.6147(9)	1.1801(2)
	5.0	16.7155(1)	7.035(2)	8.035(2)	1.6462(4)
	10.0	32.6045(9)	14.568(4)	15.613(5)	2.4238(6)
8	0.01	1.1371(5)	0.02(1)	0.388(6)	0.73(2)
	0.1	5.7498(4)	0.4107(3)	2.113(1)	3.226(1)
	0.28	12.2492(4)	1.3909(6)	4.756(2)	6.101(1)
	0.5	19.0241(4)	2.7417(9)	7.579(2)	8.704(2)
	1.0	32.7159(6)	6.137(1)	13.440(3)	13.139(2)
	2.0	57.4473(8)	13.451(3)	24.361(5)	19.636(2)
	3.0	80.6370(9)	21.039(5)	34.888(8)	24.710(3)
	5.0	124.955(1)	37.126(9)	54.81(1)	33.020(5)
	10.0	230.149(2)	76.75(2)	105.83(2)	47.560(6)
20	0.01	5.6448(4)	0.1624(4)	1.955(2)	3.527(2)
	0.1	27.470(1)	0.9593(9)	9.711(6)	16.800(5)
	0.28	56.600(1)	3.515(1)	20.616(7)	32.469(6)
	0.5	85.893(1)	7.212(2)	31.722(8)	46.958(7)
	1.0	143.209(2)	16.531(7)	54.86(1)	71.819(7)
	2.0	242.195(2)	37.591(8)	96.36(2)	108.24(1)
	3.0	333.07(6)	53.2(5)	138.0(5)	141.835(9)
	5.0	507.35(1)	119.91(2)	196.89(4)	190.55(2)
	10.0	903.79(2)	253.59(5)	372.83(7)	277.37(3)

Table D.9: This table shows how the total energy ($\langle\hat{H}\rangle$) is distributed between kinetic energy ($\langle\hat{T}\rangle$), external potential energy ($\langle\hat{V}_{\text{ext}}\rangle$) and interaction energy ($\langle\hat{V}_{\text{int}}\rangle$) of three-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using RBM+PJ. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle\hat{H}\rangle$	$\langle\hat{T}\rangle$	$\langle\hat{V}_{\text{ext}}\rangle$	$\langle\hat{V}_{\text{int}}\rangle$
2	0.01	0.079312(6)	0.01283(4)	0.02987(6)	0.03661(4)
	0.1	0.500080(6)	0.1271(1)	0.2085(2)	0.1644(1)
	0.28	1.201710(6)	0.3624(2)	0.5253(3)	0.3140(1)
	0.5	1.999912(5)	0.6515(3)	0.9040(3)	0.4444(1)
	1.0	3.729827(5)	1.2995(4)	1.7688(5)	0.6615(2)
	2.0	7.05785(1)	2.7017(6)	3.3705(6)	0.9856(1)
	3.0	10.31271(4)	3.5410(8)	5.5989(9)	1.1728(2)
	5.0	16.7170(1)	7.012(2)	8.072(2)	1.632(4)
	10.0	32.44255(9)	8.054(3)	22.384(3)	2.0042(2)
8	0.01	1.1346(1)	0.0624(2)	0.3910(7)	0.6812(7)
	0.1	5.8562(9)	0.6134(7)	2.088(2)	3.155(2)
	0.28	12.2056(2)	1.5665(7)	4.605(1)	6.034(1)
	0.5	18.9747(2)	2.972(1)	7.344(2)	8.659(1)
	1.0	32.6820(2)	6.266(2)	13.390(3)	13.026(1)
	2.0	57.4148(3)	13.744(3)	24.205(5)	19.466(2)
	3.0	80.6280(3)	18.35(2)	38.64(2)	23.627(2)
	5.0	124.915(1)	37.61(2)	54.44(2)	32.87(8)
	10.0	230.186(1)	78.64(4)	103.59(5)	47.95(1)
20	0.01	5.6328(3)	0.1621(4)	1.923(2)	3.558(2)
	0.1	27.3382(8)	1.336(3)	9.408(4)	16.595(3)
	0.28	56.4477(6)	4.157(2)	20.124(4)	32.167(4)
	0.5	85.7153(6)	8.028(2)	31.333(6)	46.354(4)
	1.0	142.9409(6)	17.603(3)	54.592(7)	70.746(5)
	2.0	242.1168(8)	38.487(5)	96.23(1)	107.403(7)
	3.0	333.027(1)	49.3(3)	152.1(3)	131.618(7)
	5.0	506.58(1)	130.79(3)	172.35(3)	203.45(2)
	10.0	897.68(2)	273.53(5)	328.64(6)	295.52(3)

Table D.10: This table shows how the total energy ($\langle \hat{H} \rangle$) is distributed between kinetic energy ($\langle \hat{T} \rangle$), external potential energy ($\langle \hat{V}_{\text{ext}} \rangle$) and interaction energy ($\langle \hat{V}_{\text{int}} \rangle$) of three-dimensional circular quantum dots for a wide range of frequencies ω and electron numbers N calculated using VMC. The energy is given in units of $\hbar$, and the numbers in parenthesis are the statistical uncertainties in the last digit.

N	ω	$\langle \hat{H} \rangle$	$\langle \hat{T} \rangle$	$\langle \hat{V}_{\text{ext}} \rangle$	$\langle \hat{V}_{\text{int}} \rangle$
2	0.01	0.079284(6)	0.01221(4)	0.039757(6)	0.036319(4)
	0.1	0.500083(7)	0.1263(1)	0.2082(2)	0.1656(1)
	0.28	1.201752(6)	0.3606(2)	0.5272(3)	0.3140(1)
	0.5	1.999977(5)	0.6517(3)	0.9032(3)	0.4451(1)
	1.0	3.730030(5)	1.3105(4)	1.7551(5)	0.6644(2)
	2.0	7.065911(7)	3.2766(4)	2.6932(5)	1.0961(2)
	3.0	10.31717(1)	3.8365(7)	5.2770(8)	1.2037(2)
	5.0	16.713925(4)	8.1523(8)	6.7797(9)	1.7819(2)
	10.0	32.449053(8)	14.586(2)	15.470(2)	2.3933(2)
8	0.01	1.12283(7)	0.04384(7)	0.3832(2)	0.6958(2)
	0.1	5.7126(1)	0.4930(4)	2.085(1)	3.1342(9)
	0.28	12.2050(2)	1.5332(7)	4.630(2)	6.041(1)
	0.5	18.96747(8)	3.2098(4)	6.7892(8)	8.9647(6)
	1.0	32.6863(2)	6.244(2)	13.378(3)	13.064(1)
	2.0	57.4197(5)	14.344(6)	23.14(1)	19.932(5)
	3.0	80.6193(3)	22.281(5)	33.286(7)	25.052(3)
	5.0	124.9024(4)	38.713(8)	52.89(1)	33.300(4)
	10.0	230.1668(7)	81.03(2)	100.56(2)	48.573(6)
20	0.01	5.6428(3)	0.1621(4)	1.923(2)	3.558(2)
	0.1	27.3152(5)	1.247(1)	9.392(3)	16.676(3)
	0.28	56.4386(5)	3.991(2)	20.125(5)	32.322(4)
	0.5	85.7197(6)	7.868(2)	31.383(6)	46.469(5)
	1.0	142.9561(7)	17.29(2)	54.45(3)	71.218(6)
	2.0	242.0320(6)	42.246(3)	86.317(9)	113.469(6)
	3.0	332.6976(6)	67.976(5)	119.95(1)	144.772(7)
	5.0	509.45(1)	137.93(2)	163.28(3)	208.24(2)
	10.0	902.58(2)	288.99(4)	310.87(5)	302.72(3)

D.3 One-body density plots

The one-body density gives the probability of finding a particle at a certain position in the space. We have both calculated the radial one-body density profile and the actual one-body density profile throughout the space. The former contains all the information about the density as long as the distribution is symmetric around the origin, and is a compact and informative way of comparing the various methods. However, sometimes the densities are not symmetric around the origin, and then only the spatial profile contains all the information about the density.

The density profile becomes identical for two- and three-dimensional quantum dots, and we will for that reason focus on the two-dimensional ones. In figure (D.1), we present the radial profile for quantum dots with $N = 2 - 42$ electrons and frequencies $\omega = 0.1$, 0.28, 0.5 and 1.0 produced with RBM, RBM+SJ, RBM+PJ and VMC. Further, in figures (D.2-D.5), we present the corresponding spatial density profiles, but the frequency $\omega = 0.28$ is omitted because of layout challenges. Lastly, the spatial one-body density profile of large quantum dots with $N = 30$, 42 and 56, and frequency $\omega = 1.0$ is presented in figure (D.6).

$N = 2$
$N = 6$
$N = 12$
$\omega = 0.1$
$\omega = 0.28$
$\omega = 0.5$
$\omega = 1.0$
VMC
RBM
RBMSJ
RBMPJ
$\rho(r)$
r

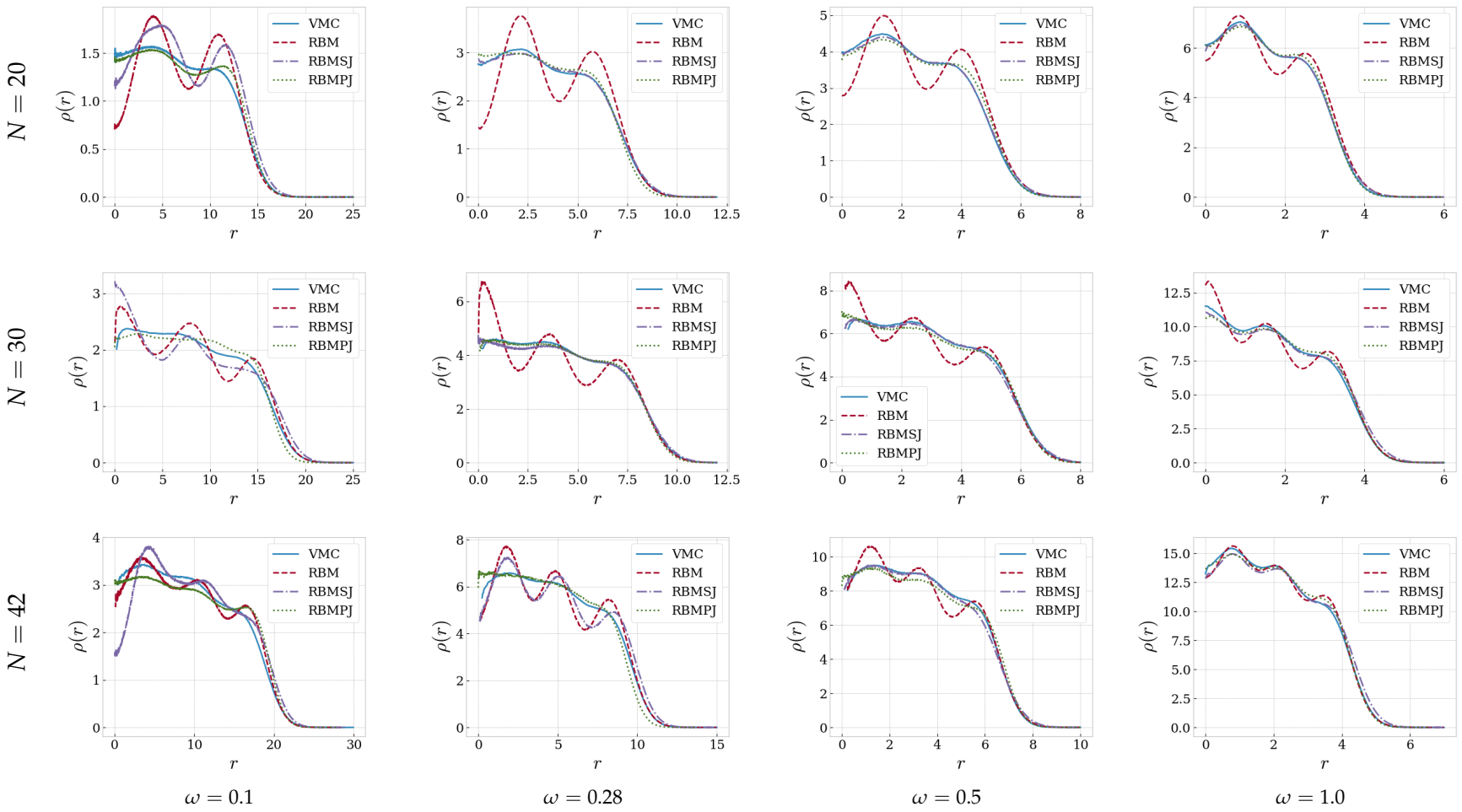


Figure D.1: Radial one-body density plots for two-dimensional circular quantum dots with $N = 2, 6, 12, 20, 30$ and 42 interacting electrons for the oscillator frequencies $\omega = 0.1, 0.28, 0.5, 1.0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

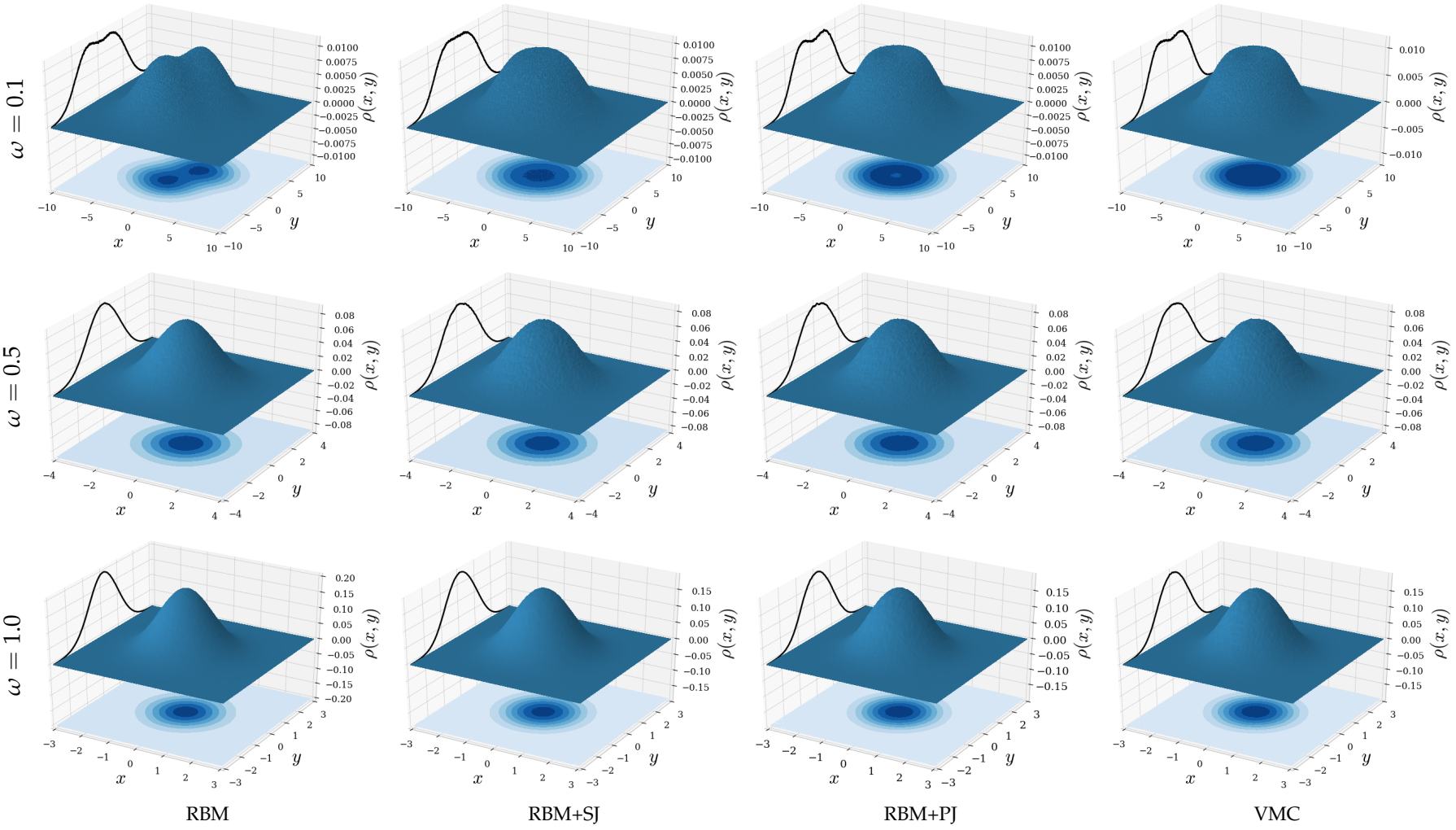


Figure D.2: Spatial one-body density plots for two-dimensional circular quantum dots with $N = 2$ interacting electrons for the oscillator frequencies $\omega = 0.1, 0.5, 1.0$. The graph on the yz-plane represent the cross section through $x = 0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

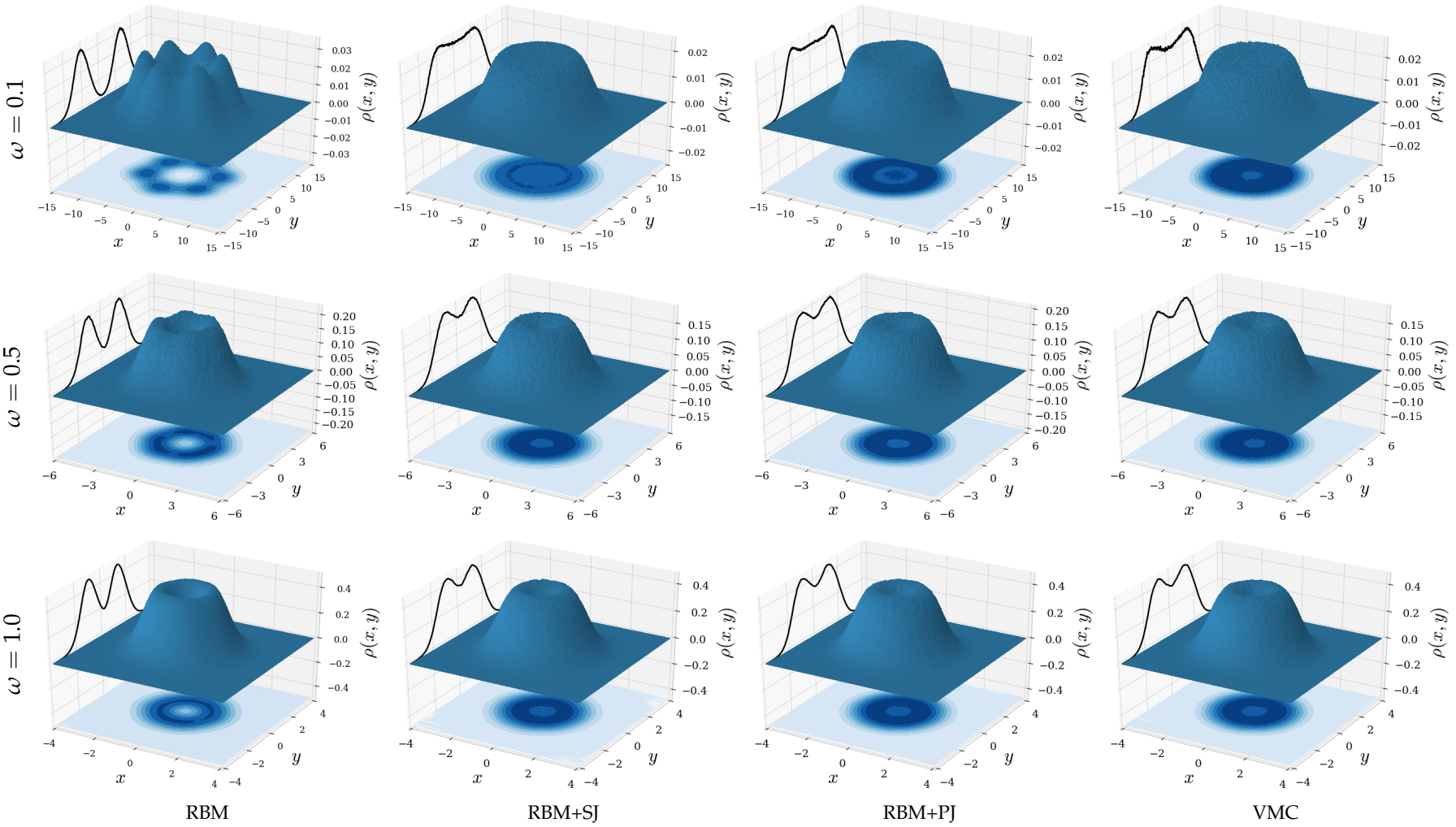


Figure D.3: Spatial one-body density plots for two-dimensional circular quantum dots with $N = 6$ interacting electrons for the oscillator frequencies $\omega = 0.1, 0.5, 1.0$. The graph on the yz-plane represent the cross section through $x = 0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

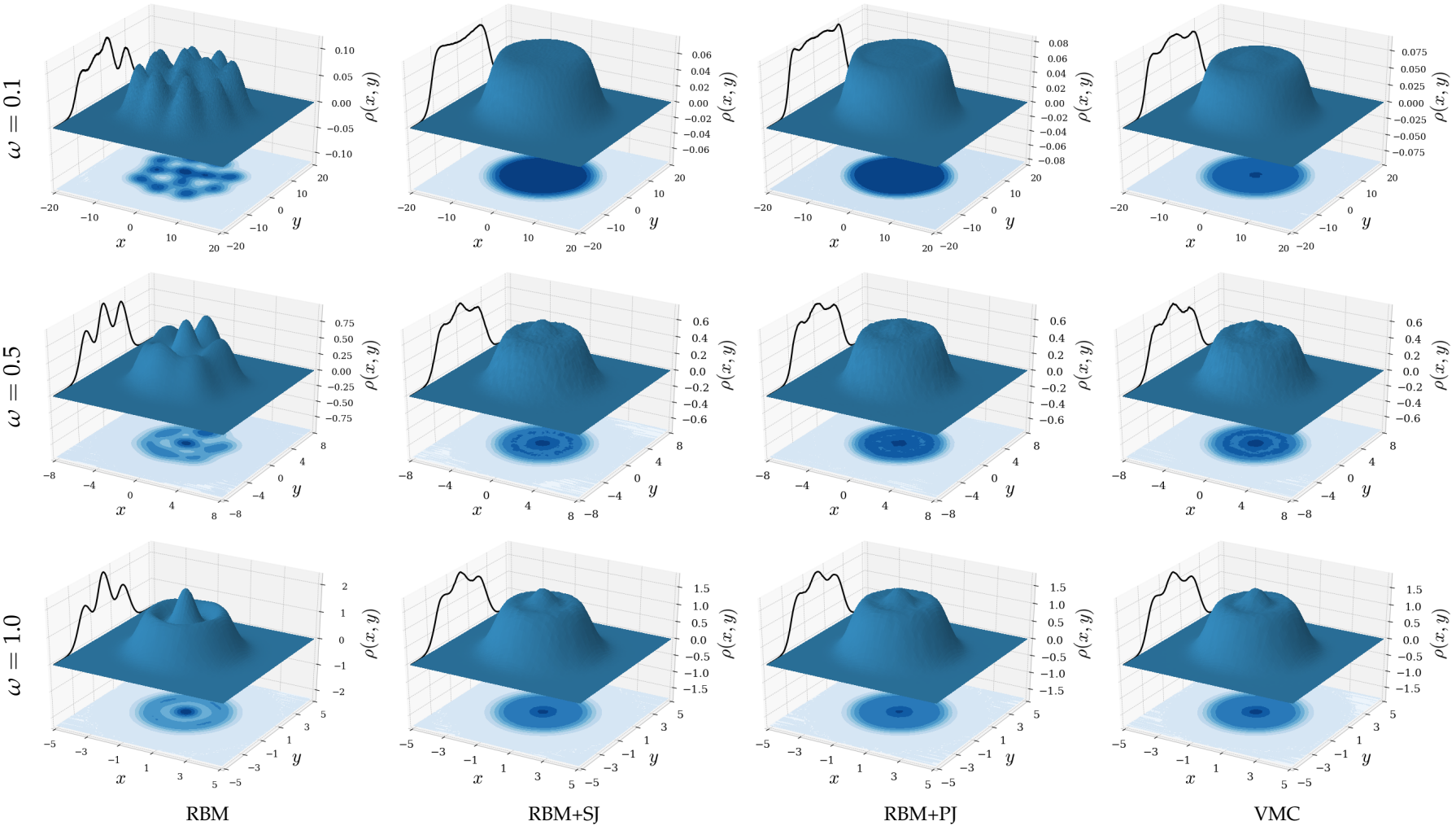


Figure D.4: Spatial one-body density plots for two-dimensional circular quantum dots with $N = 12$ interacting electrons for the oscillator frequencies $\omega = 0.1, 0.5, 1.0$. The graph on the yz-plane represent the cross section through $x = 0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

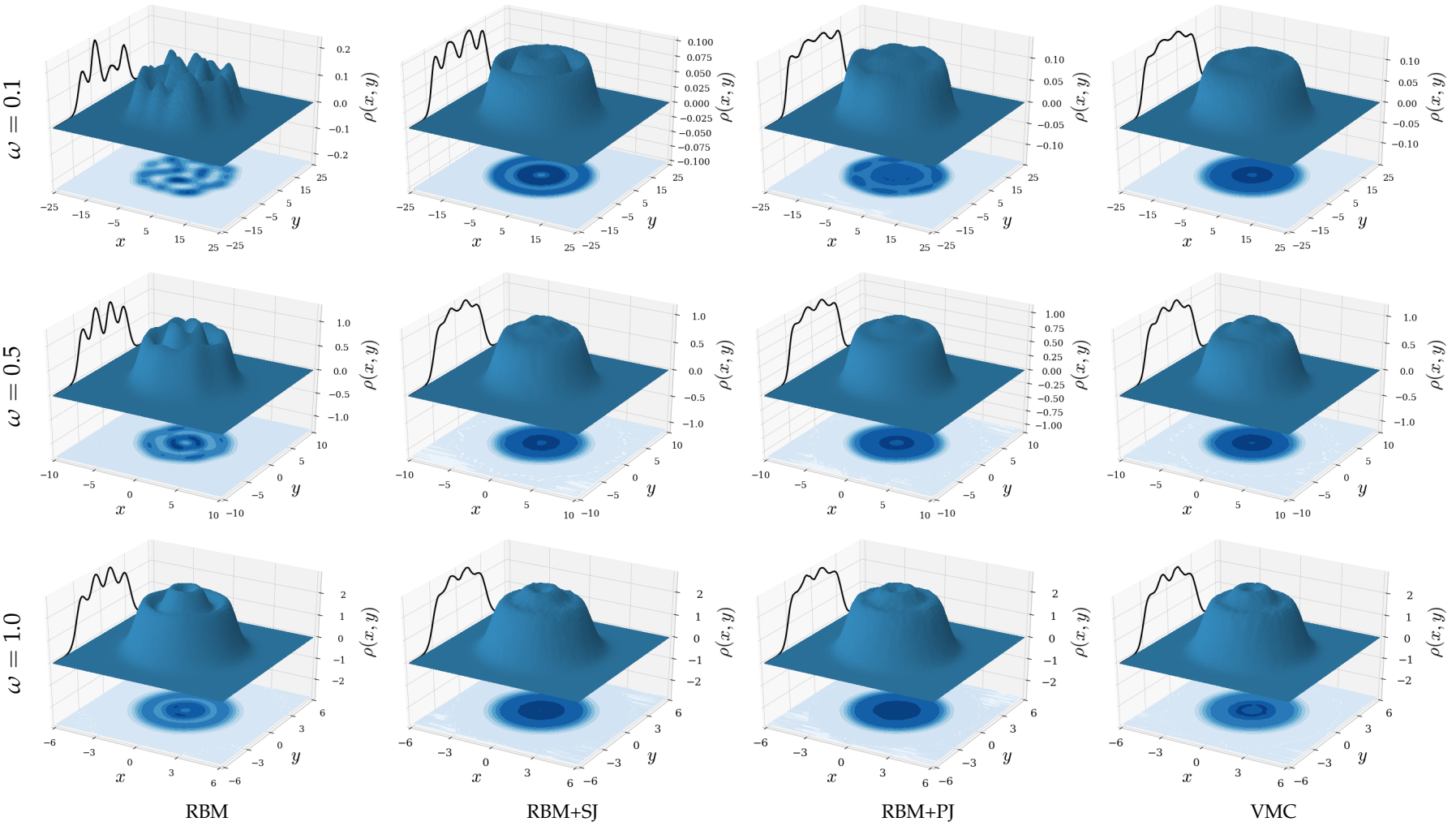


Figure D.5: Spatial one-body density plots for two-dimensional circular quantum dots with $N = 20$ interacting electrons for the oscillator frequencies $\omega = 0.1, 0.5, 1.0$. The graph on the yz-plane represent the cross section through $x = 0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

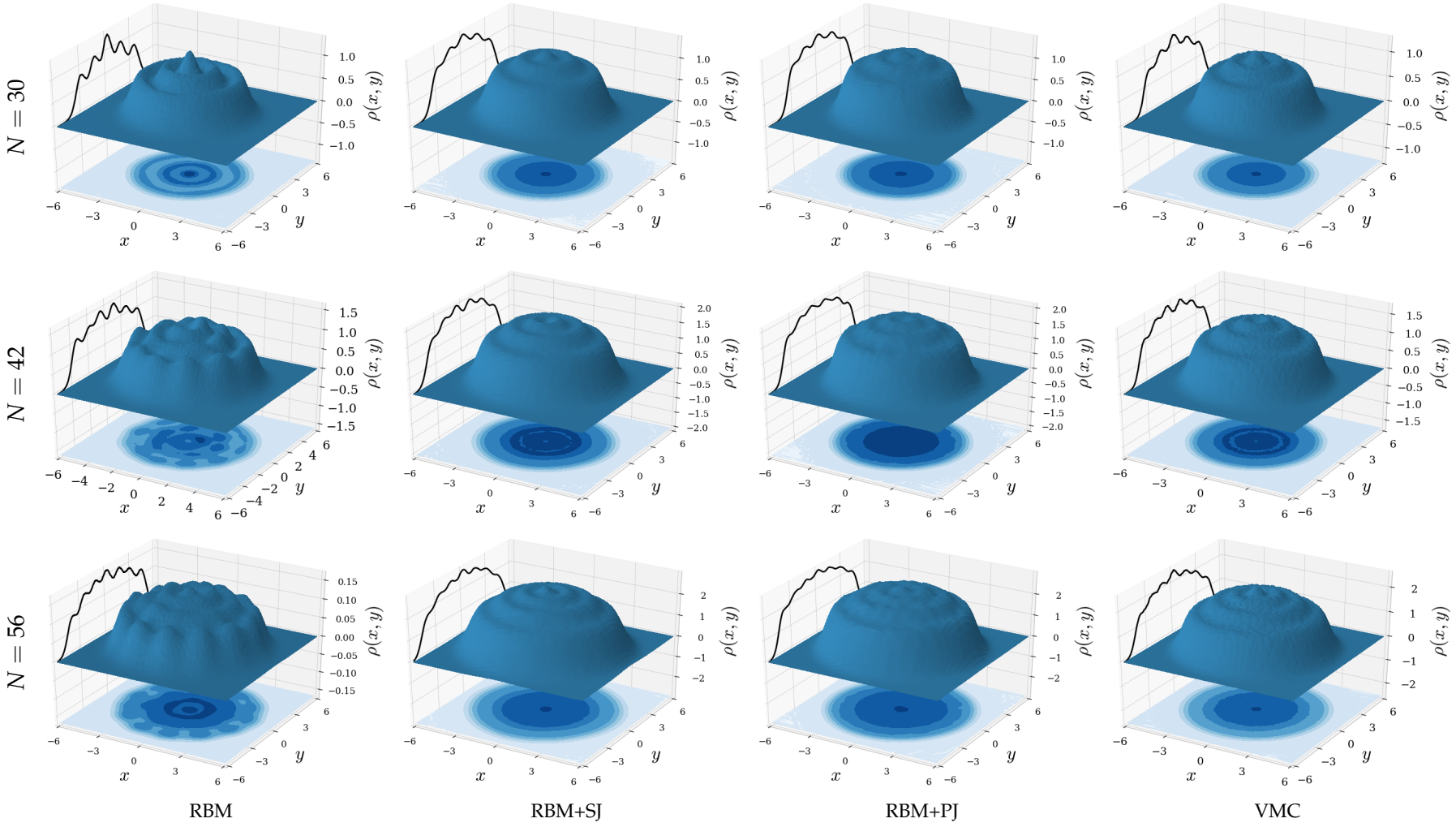


Figure D.6: Spatial one-body density plots for two-dimensional circular quantum dots with $N = 30$, 42 and 56 interacting electrons for the oscillator frequency $\omega = 1.0$. The graph on the yz-plane represent the cross section through $x = 0$. ADAM optimizer was used and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

D.4 Two-body density plots

The two-body density gives the probability of finding a particle at a certain position in the space, given the position of another particle. Similarly to the one-body density, it is possible to calculate both the radial two-body density profile and the actual two-body density profile throughout the space. However, since it is hard to visualize the actual density throughout the space, as it turns out to be a multi-dimensional monster, we keep our focus on the radial profile. The density becomes identical in two- and three-dimensional quantum dots, and we therefore focus on the two-dimensional case.

In figure (D.7-D.9), we present the two-body density for quantum dots with $N = 2$, 6, 12, 20, 30 and 42 electrons and the frequencies $\omega = 0.1$, 0.5 and 1.0 respectively.

$N = 2$
$N = 6$
$N = 12$
RBM
RBM+SJ
RBM+PJ
VMC
r_i
r_j
$\rho(r_i, r_j)$

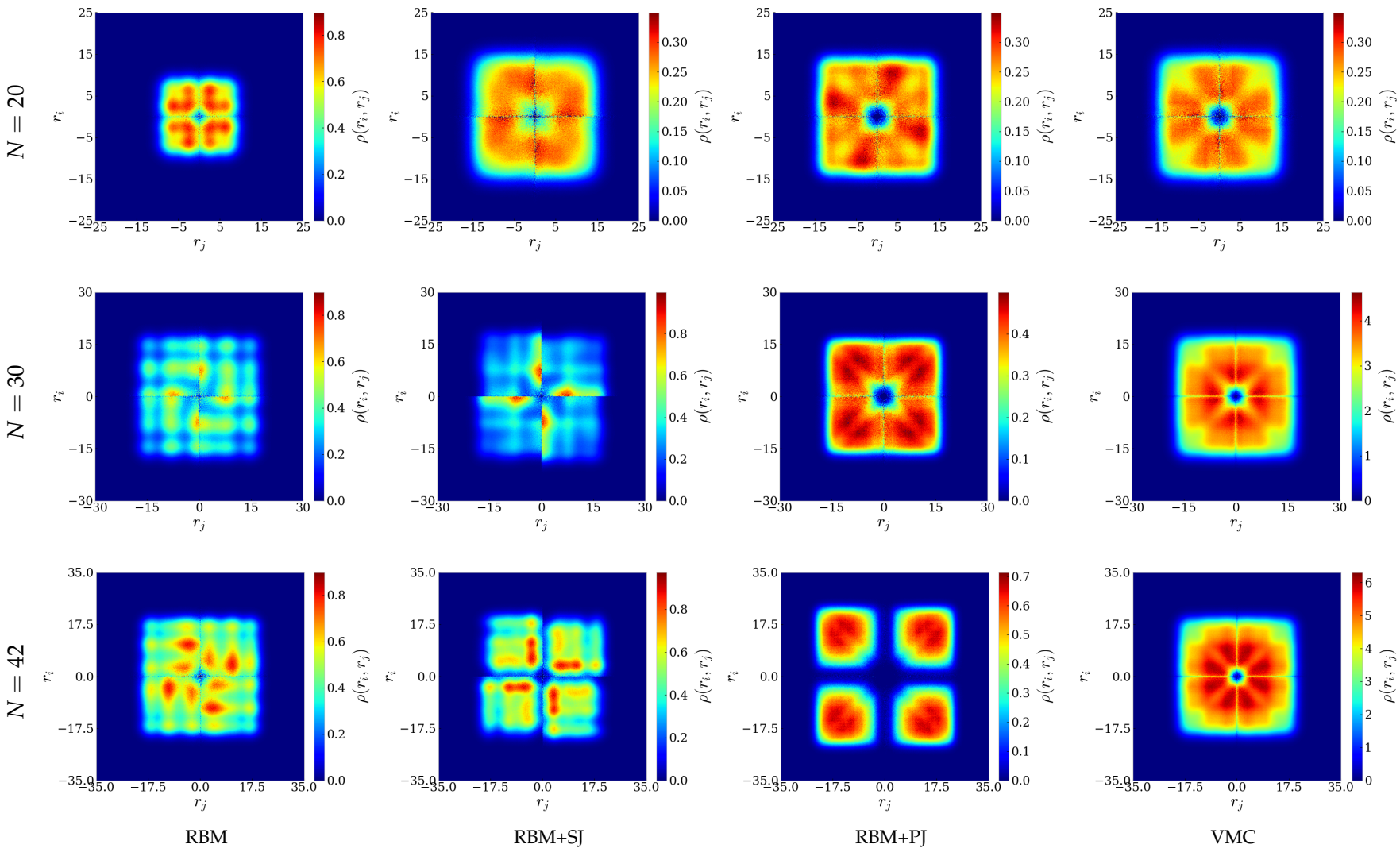


Figure D.7: Two-body densities for two-dimensional circular quantum dots containing up to $N = 42$ electrons with oscillator frequency $\omega = 0.1$. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

$N = 2$
$N = 6$
$N = 12$
RBM
RBM+SJ
RBM+PJ
VMC
r_i
r_j
$\rho(r_i, r_j)$

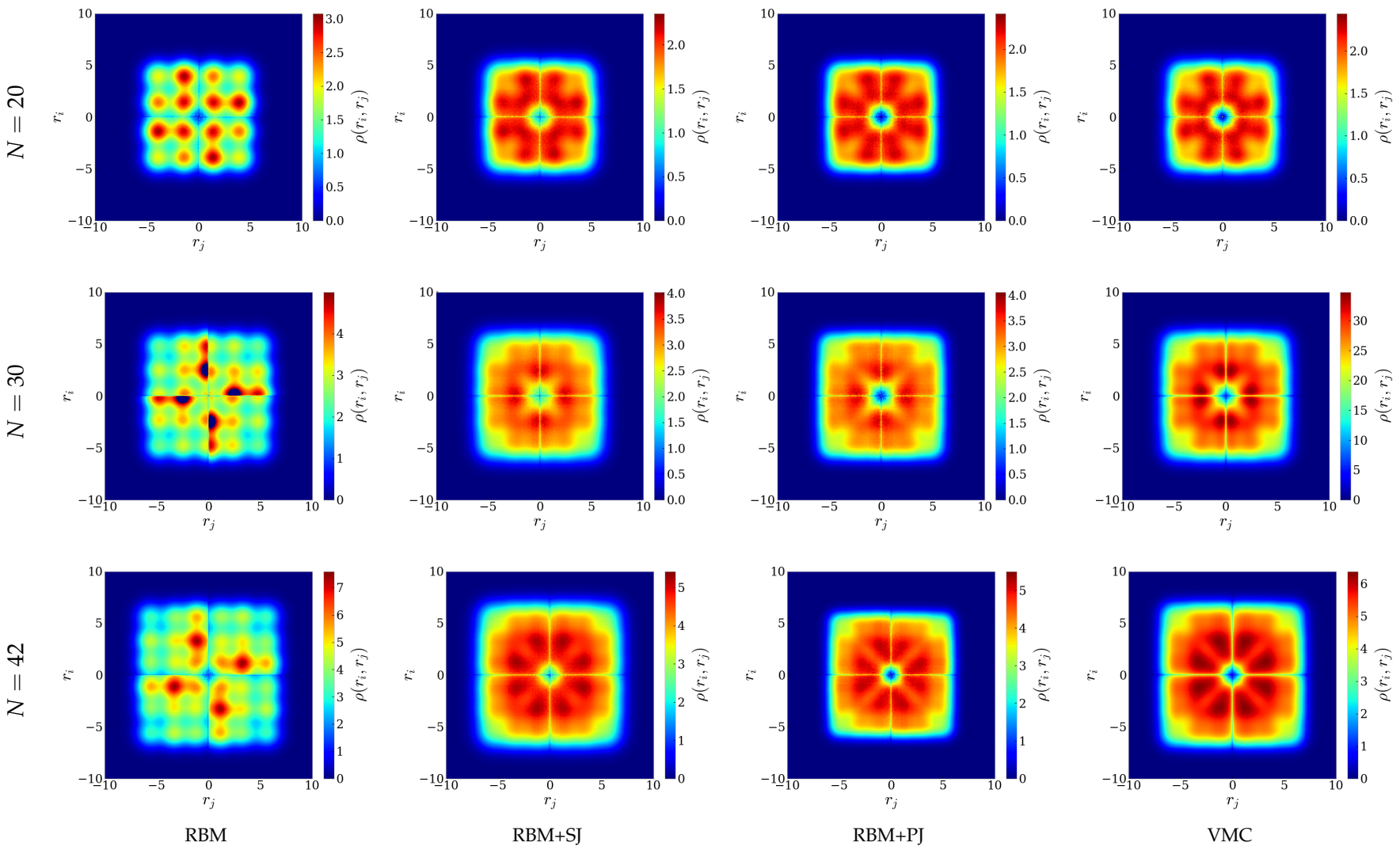


Figure D.8: Two-body densities for two-dimensional circular quantum dots containing up to $N = 42$ electrons with oscillator frequency $\omega = 0.5$. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

$N = 2$
$N = 6$
$N = 12$
RBM
RBM+SJ
RBM+PJ
VMC
r_i
r_j
$\rho(r_i, r_j)$

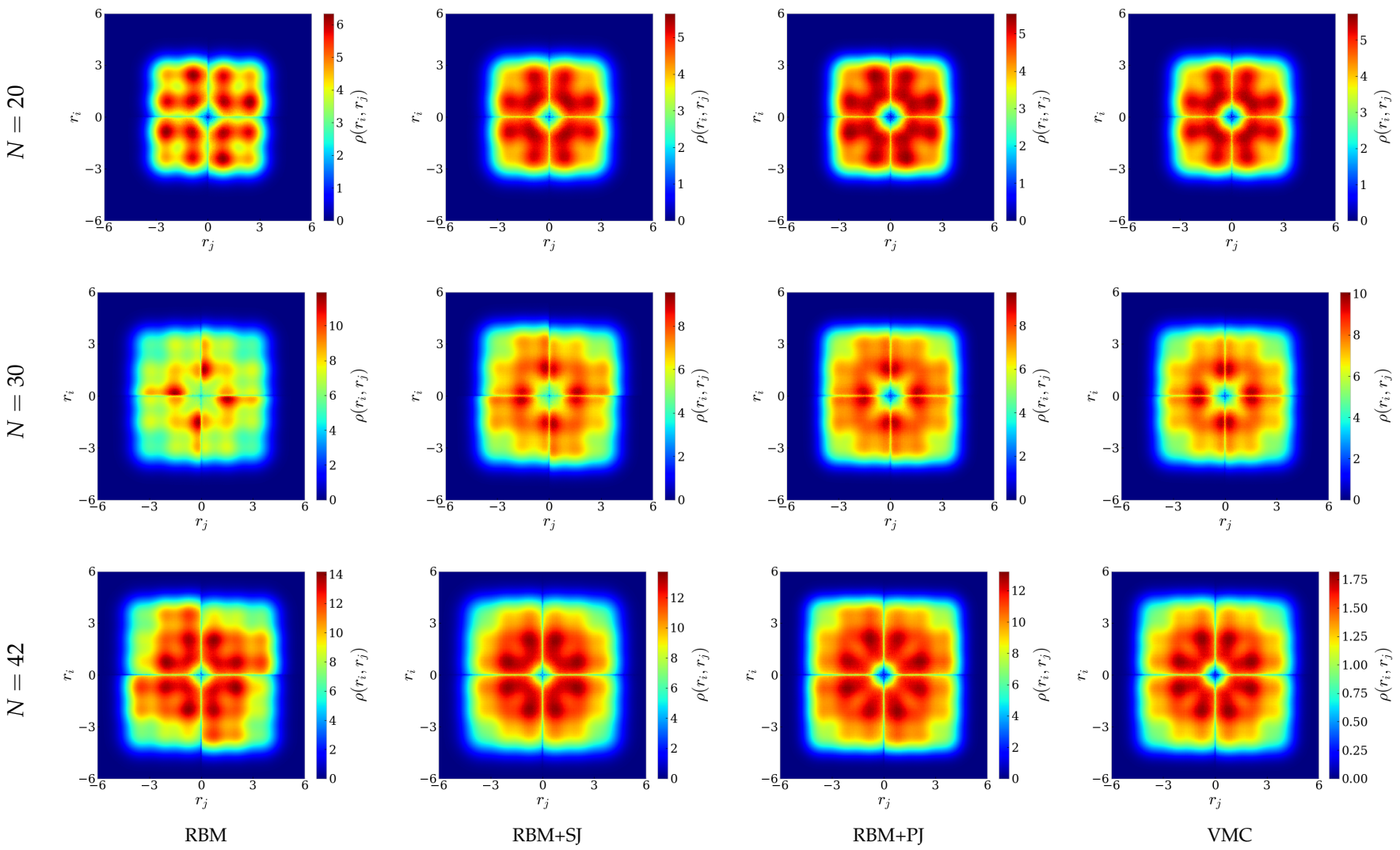


Figure D.9: Two-body densities for two-dimensional circular quantum dots containing up to $N = 42$ electrons with oscillator frequency $\omega = 1.0$. The ADAM optimizer was used, and after convergence the number of Monte-Carlo cycles was $M = 2^{28} = 268,435,456$. For abbreviations see the text.

Bibliography: Articles

2. Hunter, J. D. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering* **9,** 90 (2007).
5. Troyer, M. & Wiese, U.-J. Computational Complexity and Fundamental Limitations to Fermionic Quantum Monte Carlo Simulations. *Physics Review Letters* **94,** 170201 (2005).
6. Odom, B., Hanneke, D., D'Urso, B. & Gabrielse, G. New Measurement of the Electron Magnetic Moment Using a One-Electron Quantum Cyclotron. *Physical Review Letters* **97,** 030801 (2006).
7. Dirac, P. A. M. & Fowler, R. H. Quantum mechanics of many-electron systems. *Proceedings of the Royal Society A* **123,** 714 (1929).
8. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* **25,** 1097 (2012).
9. Alom, Z. *et al.* The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. arXiv: 1803.01164 (2018).
10. Wu, Y. *et al.* Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv: 1609.08144 (2016).
13. Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484 (2016).
14. Silver, D. *et al.* Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv: 1712.01815 (2017).
15. Carleo, G. & Troyer, M. Solving the Quantum Many-Body Problem with Artificial Neural Networks. *Science* **355,** 602 (2017).
16. Pfau, D., Spencer, J. S., Matthews, A. G. G. & Foulkes, W. M. C. Ab-Initio Solution of the Many-Electron Schrödinger Equation with Deep Neural Networks. arXiv: 1909.02487 (2019).
18. Ghosal, A., Güçlü, A. D., Umrigar, C. J., Ullmo, D. & Baranger, H. U. Incipient Wigner localization in circular quantum dots. *Physical Review B* **76,** 085341 (2007).
21. Manders, J. R. *et al.* 8.3: Distinguished Paper: Next-Generation Display Technology: Quantum-Dot LEDs. *SID Symposium Digest of Technical Papers* **46,** 73 (2015).
22. Marzin, J. Y., Gérard, J. M., Izraël, A., Barrier, D. & Bastard, G. Photoluminescence of Single InAs Quantum Dots Obtained by Self-Organized Growth on GaAs. *Physical Review Letters* **73,** 716 (1994).
23. Brunner, K., Abstreiter, G., Böhm, G., Tränkle, G. & Weimann, G. Sharp-Line Photoluminescence and Two-Photon Absorption of Zero-Dimensional Biexcitons in a GaAs/AlGaAs Structure. *Physical Review Letters* **73,** 1138 (1994).

24. Bajdich, M. & Mitas, L. Electronic structure quantum Monte Carlo. *Acta Physica Slovaca* **59,** 81 (2010).

25. Metropolis, N. & Ulam, S. The Monte Carlo Method. *Journal of the American Statistical Association* **44,** 335 (1949).

26. Kalos, M. H. Monte Carlo Calculations of the Ground State of Three- and Four-Body Nuclei. *Physical Review* **128,** 1791 (1962).

27. Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. en. *Biometrika* **57,** 97 (1970).

28. Ceperley, D. & Alder, B. Quantum Monte Carlo. *Science* **231,** 555 (1986).

35. Moreau, P. A. *et al.* Imaging Bell-type nonlocal behavior. *Science Advances* **5,** aaw2563 (2019).

36. Schrödinger, E. An Undulatory Theory of the Mechanics of Atoms and Molecules. *Physical Review* **28,** 1049 (1926).

37. Born, M. Zur Quantenmechanik der Stoßvorgänge. *Zeitschrift für Physik* **37,** 863 (1926).

38. Heisenberg, W. Über quantentheoretische Umdeutung kinematischer und mechanischer Beziehungen. *Zeitschrift für Physik* **33,** 879 (1925).

39. Einstein, A., Podolsky, B. & Rosen, N. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Physical Review* **47,** 777 (1935).

42. Taut, M. Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem. *Physical Review A* **48,** 3561 (1993).

43. Taut, M. Two electrons in a homogeneous magnetic field: particular analytical solutions. *Journal of Physics A* **27,** 1045 (1994).

44. Clausius, R. XVI. On a mechanical theorem applicable to heat. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **40,** 122 (1870).

45. Fock, V. Bemerkung zum Virialsatz. *Zeitschrift für Physik* **63,** 855 (1930).

46. Flyvbjerg, H. & Petersen, H. G. Error estimates on averages of correlated data. *The Journal of Chemical Physics* **91,** 461 (1989).

48. Stodolna, A. S. *et al.* Hydrogen Atoms under Magnification: Direct Observation of the Nodal Structure of Stark States. *Physical Review Letters* **110,** 213001 (2013).

49. Leinaas, J. M. & Myrheim, J. On the Theory of Identical Particles. *IL NUOVO CIMENTO* **37,** 1 (1977).

52. Jelic, V. & Marsiglio, F. The double well potential in quantum mechanics: a simple, numerically exact formulation. *European Journal of Physics* **33,** 1651 (2012).

54. Hehre, W. J., Stewart, R. F. & Pople, J. A. Self-Consistent Molecular-Orbital Methods. I. Use of Gaussian Expansions of Slater-Type Atomic Orbitals. *The Journal of Chemical Physics* **51,** 2657 (1969).

56. Legendre, A. M. Nouvelles méthodes pour la détermination des orbites des comètes (1805).

57. Gauss, C. F. Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum (1809).

59. Mehta, P. *et al.* A high-bias, low-variance introduction to Machine Learning for physicists. *Physics Reports* **810,** 1 (2019).

60. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Networks* **2,** 359 (1989).

61. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323,** 533 (1986).

62. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations.* arXiv: 1412.6980 (2014).

63. Rajasekar, S. & Athavan, N. Ludwig Edward Boltzmann. arXiv: physics/0609047 (2006).

64. Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. A Learning Algorithm for Boltzmann Machines. *Cognitive Science* **9,** 147 (1985).

65. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv: 1207.0580 (2012).

68. Fischer, A. & Igel, C. Training restricted Boltzmann machines: An introduction. *Pattern Recognition* **47,** 25 (2014).

71. Deb, S. Variational Monte Carlo technique. *Resonance* **19,** 713 (2014).

72. Calcavecchia, F., Pederiva, F., Kalos, M. H. & Kühne, T. D. Sign Problem of the Fermionic Shadow Wave Function. *Physical Review E* **90,** 053304 (2014).

73. Kosztin, I., Faber, B. & Schulten, K. Introduction to the Diffusion Monte Carlo Method. *American Journal of Physics* **64,** 633 (1996).

74. Jonsson, M. Standard error estimation by an automated blocking method. *Physical Review E* **98,** 043304 (2018).

75. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics* **21,** 1087 (1953).

78. Helgaker, T. & Klopper, W. Perspective on "Neue Berechnung der Energie des Heliums im Grundzustande, sowie des tiefsten Terms von Ortho-Helium". *Theoretical Chemistry Accounts* **103,** 180 (2000).

80. Allen, F. E. The History of Language Processor Technology in IBM. *IBM Journal of Research and Development* **25,** 535 (1981).

81. Holmevik, J. R. Compiling SIMULA: a historical study of technological genesis. *IEEE Annals of the History of Computing* **16,** 25 (1994).

87. Degroote, M. Faddeev random phase approximation applied to molecules. *The European Physical Journal Special Topics* **218,** 1 (2013).

88. Dirac, P. A. M. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society* **35,** 416 (1939).

93. Pederiva, F., Umrigar, C. J. & Lipparini, E. Diffusion Monte Carlo study of circular quantum dots. *Physical Review B* **62,** 8120 (2000).

96. Assaraf, R. & Caffarel, M. Zero-Variance Zero-Bias Principle for Observables in quantum Monte Carlo: Application to Forces. *The Journal of Chemical Physics* **119** (2003).

97. Hartree, D. R. The Wave Mechanics of an Atom with a Non-Coulomb Central Field. Part II. Some Results and Discussion. *Mathematical Proceedings of the Cambridge Philosophical Society* **24,** 111 (1928).

98. Fock, V. „Selfconsistent field" mit Austausch für Natrium. *Zeitschrift für Physik* **62,** 795 (1930).

Bibliography: Books

17. Flugsrud, V. M. *Solving Quantum Mechanical Problems with Machine Learning* MA thesis (2018).

19. Høgberget, J. *Quantum Monte-Carlo Studies of Generalized Many-body Systems* MA thesis (2013).

30. Vance, A. *Elon Musk: Tesla, SpaceX, and the Quest for a Fantastic Future* ISBN: 978-0-062301-25-3 (Ecco, 2015).

40. Griffiths, D. J. *Introduction to Quantum Mechanics* 2nd Edition. ISBN: 0-13-191175-9 (Pearson PH, 2005).

47. Heisenberg, W. *Across the frontiers* ISBN: 978-0-918024-80-0 (Ox Bow Press, 1990).

50. Heisenberg, W. *Physics and Beyond: Encounters and Conversations* ISBN: 978-0-049250-08-6 (Allen and Unwin, 1971).

55. Domingos, P. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World* ISBN: 978-0-241-00455-5 (Penguin UK, 2015).

69. Mariadason, A. A. *Quantum Many-Body Simulations of Double Dot System* MA thesis (2018).

79. Gustafsson, B. *Scientific Computing: A Historical Perspective* ISBN: 978-3-319-69846-5 (Springer, 2018).

85. Trahan, J., Kaw, A. & Martin, K. *Computational time for finding the inverse of a matrix: LU decomposition vs. naive gaussian elimination* (2006).

89. Paldus, J. *The Beginnings of Coupled-Cluster Theory: An Eyewitness Account* ISBN: 978-0-444517-19-7 (Elsevier, 2005).

90. Crawford, D. & Schaefer III, H. *An Introduction to Coupled Cluster Theory for Computational Chemists* 33. ISBN: 978-0-470125-91-5 (Wiley-VCH, 2007).

91. Szabo, A. & Ostlund, N. S. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory* ISBN: 978-0-486-69186-2 (Dover Publications, 1996).

94. Nissenbaum, D. A. *The Stochastic Gradient Approximation: An Application to Li Nanoclusters* PhD dissertation (2008).

95. Jonsson, M. *Standard error estimation by an automated blocking method* MA thesis (2018).

Bibliography: Online

3. ISO/TC 12. *ISO 80000-2:2009* http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/18/31887.html.

4. ISO/TC 12. *ISO 80000-9:2009* http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/18/31894.html.

11. Smith, C. *iOS 10: Siri now works in third-party apps, comes with extra AI features* 2016. https://bgr.com/2016/06/13/ios-10-siri-third-party-apps/.

12. *Bringing the Magic of Amazon AI and Alexa to Apps on AWS. - All Things Distributed* https://www.allthingsdistributed.com/2016/11/amazon-ai-and-alexa-for-all-aws-apps.html.

20. *Samsung QLED TV | The Frame | Premium UHD TV* //www.samsung.com/global/tv/blog/why-are-quantum-dot-displays-so-good/.

29. Cellan-Jones, R. *Hawking: AI could end human race* 2014. https://www.bbc.com/news/technology-30290540 (2019).

31. Ledum, M. *Simple Variational Monte Carlo solve for FYS4411* 2016. https://github.com/mortele/variational-monte-carlo-fys4411.

32. *MIT License* https://choosealicense.com/licenses/mit/ (2019).

33. Nordhagen, E. M. *Ground state properties of quantum dots provided by VMC and RBM* 2019. doi:10.5281/zenodo.3477946.

34. *A Quantum Sampler* 2005. https://www.nytimes.com/2005/12/26/science/a-quantum-sampler.html.

41. Sherrill, D. *Postulates of Quantum Mechanics* 2003. http://vergil.chemistry.gatech.edu/notes/intro_estruc/node4.html.

51. *What are the applications of quantum harmonic oscillator in different fields? - Quora* https://www.quora.com/What-are-the-applications-of-quantum-harmonic-oscillator-in-different-fields.

53. Hjorth-Jensen, M. *Computational Physics 2: Variational Monte Carlo methods* 2019. http://compphysics.github.io/ComputationalPhysics2/doc/pub/vmc/html/vmc-reveal.html.

58. *Machine Learning - Stanford University* http://mlclass.stanford.edu/.

70. Carleo, G. *Neural-Network Quantum States* 2017. https://gitlab.com/nqs/ucas_workshop/blob/master/lecture_notes.pdf.

76. Gjestvang, D. & Nordhagen, E. M. *Computational physics II: Quantum mechanical systems: evenmn/FYS4411* 2018. https://github.com/evenmn/FYS4411.

77. Woods, J. F. *Usage of comma operator - Google Groups* https://groups.google.com/forum/#!msg/comp.lang.c++/rYCO5yn4lXw/oITtSkZOtoUJ.

82. *C++ Data Types* 2017. https://www.geeksforgeeks.org/c-data-types/.

83. Fowler, M. *Two Hard Things* https://martinfowler.com/bliki/TwoHardThings.html.

84. AlDanial. *cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.: AlDanial/cloc* original-date: 2015-09-07T03:30:43Z. 2019. https://github.com/AlDanial/cloc.

86. *The Edisonian - Volume 8 The Thomas Edison Papers at Rutgers University* http://edison.rutgers.edu/newsletter9.html.

92. Nordhagen, E. M. *General variational Monte-Carlo solver written in C++: evenmn/VMC* 2019. https://github.com/evenmn/VMC.

99. *Training Restricted Boltzmann Machines: An Introduction** https://webcache.googleusercontent.com/search?q=cache:JHzm9N4wEaoJ:https://christian-igel.github.io/paper/TRBMAI.pdf+&cd=1&hl=en&ct=clnk&gl=no&client=ubuntu.