Master’s thesis

VMC simulation of quantum particles in harmonic trap

Yevhenii Volkov

Computational Science: Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

30th May 2025

Abstract

In this research we study quantum systems of fermionic and bosonic quantum particles trapped in harmonic potential and develop Variational Monte Carlo (VMC) machinery to estimate Ground State (GS) energy of such systems. As bosonic systems we consider neutral alkali atoms cooled to Bose-Einstein condensate state. GS energy for such system is found analytically to be equal to $E_0 = Nd/2$. As fermionic system we consider first three closed electronic shells. We study fermionic systems with and without interaction in Coulomb's form. We pay most attention to development of VMC algorithm for GS energy estimation. Special attention is paid to development of flexible object oriented VMC framework. We implement Metroplis and Metropolis-Hastings as solving algorithms. Additionally, we develop parallel VMC and VMC that uses automatic differentiation. To study interacting fermionic systems trial wave function is modified with Jastrow and Pade-Jastrow factors. For systems that have analytical solution VMC recreates exact energies. Energy of first three interacting closed shells are found to be $E_0^{(1)} = 3$, $E_0^{(2)} = 20$ and $E_0^{(3)} = 66$ in atomic units.

Chapter 1

Introduction

When I was sixteen years old, my teacher gifted me a book about quantum physics. It was a time I met quantum physics for the first time in my life. I was surprised, because in school I never heard about this "quantum" physics. Teacher said that there exist particles that have "charm" and "color", and I started to think that this is a joke, not science. And here I am now, person that devoted last seven years of my life to physics writing a 130 pages long thesis.

This research has two related purposes – solve problem of quantum particles trapped in Harmonic Oscillator (HO) potential and develop VMC machinery in order to solve the problem.

In quantum physics only a few problems have analytical solution. When we talk about many body physics (quantum physics of many particles), there is almost no problem with analytical solution. Therefore, numerical methods are used to solve quantum problems numerically. Monte Carlo is a statistical method to estimate values of integrals. Variational Monte Carlo is a combination of Variational Principle and Monte Carlo algorithm. Variational Principle states that mean values of energy of the system is always bigger or equal to its GS energy. It may seem that this principle is a joke, because it seems to be obvious, but it turns out to be a very powerful statement. In quantum physics wave functions are used to calculate mean values of observables analytically. Variational principle states that mean value of energy of the system is always bigger than GS energy for any wave function used to compute GS energy. This principle provides a simple recipe to find GS energy: choose a wave function, compute GS energy, minimize GS energy. On practice it means that wave function can be "guessed" and modified with variational parameters. The goal is to find optimal variational parameters that minimize GS energy. Since energy is never smaller than GS energy, once energy has converged to some value, most likely it is GS energy.

But how Variational Principle and Monte Carlo related? Very simple! Mean energy in quantum physics in basically a very complicated integral. If we are able to solve this integral - problem is solved. The problem is that in quantum physics we need to integrate over the whole phase space. For every particle... For every dimension... Not an easy task. The most numerical algorithms has a "curse of dimensionality". It means that the world would end and begin once again faster than computer will calculate GS energy using standard algorithms. And here Metropolis algorithm comes for help. This algorithm is based on simulation that use random numbers. Idea is very simple. We set initial position of all particles in system to random uniform numbers. Then we compute value, local energy, which is much simpler to compute and move particles in random positions in random directions. And compute local energy once again. We repeat this process one million (or even more) times, and sample mean value of local energy turns out to be estimation of that huge and scary GS energy integral.

As a model system we choose a system of particles trapped in HO potential. Why? Because this is a problem of interest in resent thirty years. Scientist found a way to trap quantum particles in potentials that can be approximated by HO potential. Therefore quantum VMC simulation in harmonic potential is popular and choose this system. Okay, we have a system, but what particles should we trap? There is two fundamental types of particles - bosons and fermions. So why not choose both types? If we are being serious, both fermionic and bosoinc systems are of interest. Bosonos and fermion are integer spin and spin half particles. These particles have a very different properties. On of the main difference between bosons and fermions is that fermions follow Pauli principle and bosons do not follow it. Pauli principle states that bosons are symmetric under exchange of coordinates, while fermions are anti-symmetric. Consequence of Pauli Principle is two fermions can not occupy the same quantum state and two bosons can. If bosonic system is cooled to critical temperature, bosons turn into Bose-Einstein Condensate (BEC). A lot of nobel prizes were given for the study of BEC. From analytical point of view, it is easier to study BEC, than not cooled bosons. When bosonic system is cooled, all bosons are in the lowest state - ground state of the system. For such state, wave function has a much simpler form that for not cooled sysytem. As for fermions, wave function need to be anti-symmetric under exchange of coordinates, and therefore wave fermionic wave function is given by determinant. It is hard to derive analytical expressions for determinants, and it is very computationally expensive to compute it numerically when dimension are high. Therefore fermionic system is more complicated.

If we include interaction between fermions situation becomes even more complex. Colulomb's potential has a singularity when two particles are arbitrary close. And while performing a simulation that is dependent of random numbers we can not guarantee that two particles will not be close. Therefore, to nullify this singularity, wave function must satisfy Kato's cusp condition. On practice it means that wave function needs to be modified with so-called Jastrow factor.

The main purpose of this research is to study described system and develop flexible object-oriented VMC framework. In quantum physics system is fully described by Hamiltonian and wave function. Hamiltonian is a quantum analogue of classical energy. Energy of the system is a sum of kinetic energy of all particles and potential energy. If we develop a program in such a way that Hamiltonians and wave functions can be easily changed – it is a great achievement. We also parallelize VMC algorithm, which speeds computations up decently. Additionally we develop VMC that computes derivatives numerically. VMC with automatic differentiation can be relatively easily programmed. It has a downside – it is computationally expensive. Therefore we will use this method and compare method that uses analytical derivatives. If both algorithms generate the same results, it is likely that program is implemented correctly.

This research requires a lot of quantum physics background for understanding, and therefore in the beginning we focus on introduction of key quantum physical concepts needed for understanding, and only than start discussing relative topics.

Chapter 2

Theory

2.1 Quantum-Mechanical Foundations

2.1.1 Model System, Part 1

This subsection is designed to give the reader a brief overview of the model system and motivate the following theory, which is crucial for understanding physical and computational aspects of this research.

This research is dedicated to studying a system of quantum particles trapped in the Harmonic Oscillator (HO) potential.

"Trapped particles" means that the number of particles in the system is conserved - particles inside the trap can not exit the trap and no particle from outside the trap can enter.

Throughout this thesis, “we” denotes the author and the reader collectively. As we will discuss in Section 2.1.8, quantum systems—unlike their classical counterparts—are fundamentally discrete and can occupy only states with predetermined energies. The lowest-energy state is called the ground state (GS). The aim of this research is to compute numerically the GS energy for a specified number of particles in a given dimensionality, and to study how the GS energy depends on the additional features described in Section 2.1.21.

Research will focus on two different types of particles: bosons and fermions. Bosons and fermions are very different on the fundamental level, and their theoretic description differs. Both bosons and fermions are quantum particles, described by the Quantum Physics (QP). Bosonic subject of research is a Bose-Einstein Condensate (BEC) in alkali atoms ^{87}Rb, ^{23}Na and ^{7}Li confined in magnetic traps, [1] (page 1). While fermionic subject of study is just a system of electrons trapped in Penning trap.

Quantum states of the system is fully described by a wave function, while kinetic and potential energies of the system are hidden within the Hamiltonian of the system. So, Hamiltonian of the system defines the problem to solve, while wave function is a solution to the main equation of non-relativistic QP - Schrödinger’s Equation (SE) and it fully determines the quantum state and is key concept for quantum calculation.

In QP only a few problems have an analytical solution. The problem in this research, unfortunately, does not have a general analytical solution. Analytical solution can be found only for a very simple cases, like 2 bosons in HO potential. Therefore, the problem will be solved numerically, using Variational Monte Carlo (VMC) method.

GS energy, as we will discuss in the section 2.1.21, can be computed by using straightforward numerical integration algorithms. However, these integrals can have a huge dimensionality they are supposed to be integrated over the whole space $(-\infty, +\infty)$, meaning that numerical integration is very computationally expensive. VMC is an approximate method that allows compute the integrals. It is less computationally expensive than straightforward numerical integration algorithms when dimensionality is big and integration goes though the whole space.

The theory section is designed to provide the reader with all background necessary for understanding the research. The theory will be presented in a pedagogical way, so it is easier for the reader to understand it.

Before jumping straight in the word of QP, let us begin with the classical HO, since it will refresh the knowledge and will help to understand better quantum HO.

2.1.2 Classical Harmonic Oscillator

This section is based on the Philip L. Bowers' book [2], chapter 1.

Consider a one dimensional (1D) particle moving in the quadratic potential

$$V(x) = \frac{kx^2}{2}.$$

Here, $k > 0$ is a *spring constant* and x is a position of the particle on the line (since we are in 1D).

In classical mechanics, one of the main differential equation is the Newton's equation. In general form it reads as

$$m\ddot{\mathbf{r}} = \sum F(\mathbf{r}), \tag{2.1}$$

where m represents mass of the particle, t represents time, $\mathbf{r}$ represents radius vector[1], which describes position of the particle and $\sum \mathbf{F}(\mathbf{r})$ represents all forces acting on the particle. Dot notation is used to denote the time derivative, $\ddot{\mathbf{r}} = d^2\mathbf{r}/dt^2$.

If the particles moves in the potential, force acting on the particle can we found by using

$$\mathbf{F}(\mathbf{r}) = -\nabla V(\mathbf{r}).$$

Here, $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ is a gradient vector. In 1D, vectors simplify to scalars, and the force acting on the particle becomes

$$F(x) = -\frac{d}{dx}\left(\frac{kx^2}{2}\right) = -kx.$$

This force is trying to return particle to the equilibrium state $x = 0$. If the particle is not in the equilibrium state, the force acts of the particle and diminishes as particle come closer to $x = 0$. But the particle overshoots the equilibrium position. When particles reaches equilibrium, it moves with a certain speed. At this point, no force is acting on the particle, so it just keeps moving and starts feeling the restorative force again. It reaches the position of the maximal displacement, and starts to move back to equilibrium again. This motion of particle back and forth does not stop. Such movement is known as an *oscillation* and the system is known as a *Harmonic Oscillator*.

Movement of the particle is fully described by Newton's equation, which simplifies to

$$m\ddot{x} + kx = 0.$$

[1]Bold symbols will be used to denote vectors form now on.

Solution to this equation is

$$x(t) = b\cos(\omega t) + c\sin(\omega t),$$

where $\omega = \sqrt{k/m}$ is the natural frequency of the oscillator, b and c are constants determined by the initial conditions, $x(0) = b$ and $\dot{x}(0) = c\omega$.

Let us study energy of such system and set $x(0) = X$ and $\dot{x}(0) = 0$ for simplicity. Specific solution that satisfies these initial condition can obtained from the general one and has a form

$$x(t) = X\cos(\omega t).$$

Note, that X is a magnitude of maximal displacement. Kinetic energy of the particle is given by

$$T = \frac{m\dot{x}^2(t)}{2} = \frac{m\omega^2 X^2}{2}\sin^2(\omega t),$$

and potential energy is given by

$$V = \frac{kx(t)^2}{2} = \frac{kX^2}{2}\cos^2(\omega t).$$

Using $k = m\omega^2$, total energy of the system is equal to

$$E = T + V = \frac{m\omega^2 X^2}{2}\sin^2(\omega t) + \frac{m\omega^2 X^2}{2}\cos^2(\omega t) = \frac{1}{2}m\omega^2 X^2 = \frac{1}{2}kX^2.$$

Energy of the system is not a function of time, it is constant. It means that energy of the system is conserved. It is an important feature of HO and it comes from the fact that $V(x)$ is time-independent.

I would like to say that we are ready to perform the similar calculation for the quantum HO, but we are not yet ready. Before that, we need to introduce the basic concept of quantum mechanics. Allow me to begin this journey with a small historic digression.

2.1.3 The Dawn of Quantum Physics

Quantum Physics as a concept began in the beginning of twentieth century when Max Planck solved ultraviolet catastrophe – problem of the black body radiation [3], chapter 1. Classical physics seemed to be able to solve every problem. However, there were a couple of problems that classical physics could not solve. There were three critical failures: problem of the black body radiation, photoelectrical effect and the lifetime of the Bohr atom, and black body radiation led to the quantum revolution. Classical physics could not solve a problem means that experimental result misaligned with classical predictions. And if the measurement is done correctly, then obviously the problem is in the theory that can not explain such a phenomenon. But classical physics was correct in almost every other place, so how can it be wrong? The answer is pretty straightforward – classical physics perfectly explains classical phenomena (large systems), but fails when trying to explain quantum phenomena (very small systems, like atoms). Planck was able to solve ultraviolet catastrophe by assuming that the light is radiated discretely, in discrete packets (quanta) of light. Quantum of light propagates with the speed of light with energy equal to $\epsilon = h\nu$, where ν is a frequency, and h is the Planck's constant, which we will discuss later in the section 2.1.4. By applying such a trick, Planck's theory explained experimental data perfectly [4], chapter 1. But assumption that the light is emitted discretely was unheard of from the classical point of view, where everything were continuous. Physicists of that time divided into two groups: those who supported quantum theory, and those who did not. The most of debates happened because quantum theory was probabilistic, as if nature is fundamentally probabilistic and random, not deterministic. Even great physicists such as Albert Einstein were against quantum theory, especially its probabilistic interpretation. He even coined the famous phase: "God does not play dice". Jim Baggott has a great article with discussion on that phrase and possible explanation of what Einstein meant by saying this [5]. Although Einstein disagreed with some quantum postulates, he advanced the field greatly by solving the photoelectrical effect.

Today, quantum physics not only explains these early puzzles, but underprints technologies from transistors to lasers. We now turn to its key concepts, starting with Schrödinger's Equation.

2.1.4 Schrödinger Equation & Planck's Constant

In what follows, presentation closely follows David J. Griffiths' classic text *Introduction to Quantum Mechanics* [6], starting with §1.1. Griffiths' treatment is both clear and concise, making it an excellent companion as we develop the key equations and fundamental concepts of non-relativistic quantum theory.

Consider a 1D particle of mass m moving in the potential V. In classical mechanics we would solve Newton's equation (2.1) to obtain the position of the particle $x(t)$ as a function of time. Once we have the position, we can restore all the quantities that we need - velocity, momentum, energy etc. We already did that while looking at the classical HO problem. Quantum mechanics replaces the particle's trajectory with a wave function $\Psi(x, t)$. Wave function is obtained by solving one of the main QP equations - Schrödinger's equation:

$$i\hbar \frac{\partial \Psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \Psi(x,t)}{\partial x^2} + V\Psi(x,t). \tag{2.2}$$

This equation describes a quantum particle moving in the potential V, $i = \sqrt{-1}$ is imaginary unit, $\hbar$ is reduced Plank's constant, m is the mass of the particle and Ψ is a wave function - function that describes quantum state of the particle, which we aim to find by solving this equation.

Planck's constant is one of the universal constants, like Gravitational constant G, speed of light in vacuum c, as discussed in §1 of D. Chang's article [4]. He states that "The Planck's constant was found to play a major role in many aspects of quantum physics". First of all, it is a part of SE, and it appears in many other equation in quantum physics. For example, it is a part of de Broglie momentum relation, Dirac's fundamental quantum condition and Heisenberg's uncertainty principle. de Broglie relation, or de Broglie wavelength is of special interest, because it can be used to gauge whether quantum mechanics is necessary to describe the system.

De Broglie Wavelength

De Broglie wavelength is defined by $\lambda_{dB} = h/p$, where p is a momentum of the particle. "This concept, proposed by Louis de Broglie, suggests that all matter, including particles like electrons and atoms, exhibits wave-like behavior. The de Broglie wavelength is inversely proportional to the momentum of a particle. So, if the momentum (or velocity) of a particle is high enough that its de Broglie wavelength becomes comparable to the size of the system or the distance it's interacting over, then we need to consider quantum mechanics [7].".

According to CODATA's archive of physical constant by Physical Measurement Laboratory of NIST,

Planck's Constant and Reduced Planck's Constant

Planck's constant $h \approx 6.626070 \times 10^{-34}$ J $\cdot$ s and
Reduce Planck's constant $\hbar = \frac{h}{2\pi} \approx 1.054573 \times 10^{-34}$ J $\cdot$ s [8].

Equation (2.2) is a second-order partial differential equation. The solution to this equation is the wave function, which we will discuss in the following section 2.1.5. Just by looking at the equation (2.2) we can see that it is different from its classical analogue. Firstly, this equation contains i - imaginary unit, a beast that is not a frequent guest in the classical mechanics. Secondly, unknown wave function is a function of both x and t. And finally, what is the physical meaning of the wave function? How can we measure it?

With Eq. (2.2) established, we now turn to the interpretation and properties of the wave function in § 2.1.5.

2.1.5 Wave function & Statistical Concepts

Solving the SE yields the wave function $\Psi(x,t)$. But what physical meaning does Ψ carry? In this section, we follow Griffiths (§1.2–1.3 and 1.6) to develop its statistical interpretation.

The SE contains the imaginary unit. This means that, in general, wave function is a complex-valued function. Mathematically - wave function is a member of the space of complex numbers $\Psi(x,t) \in \mathbb{C}$. The fact that wave function is a complex-valued function means that it can not be measured on the experiment, only $|\Psi(x,t)|^2$ carries measurable meaning.

As we saw, SE replaces Newton's equation. Its solution Ψ defines the permissible 'states' rather than definite trajectories. Quantum behavior departs from classical mechanics at atomic scales. When considering such systems, position and momentum of the particle can not be determined precisely.

Heisenberg's Uncertainty Principle

A spread in momentum corresponds to a spread in position, and general observation says that the more precisely the particle's position is determined, the less precisely its momentum is determined,

$$\sigma_x \sigma_p \geq \frac{\hbar}{2}.$$

Here, σ_x is the standard deviation in x and σ_p is the standard deviation in p (will be discussed later in this section)

It is a feature that make quantum description different from the classical.

Returning to the wave function meaning. Born's statistical interpretation (alternatively, Born's rule) states what the wave function is, or rather what the square of its norm is:

Born's statistical interpretation of the Wave Function

$\Psi(x,t)\cdot\Psi^*(x,t) = |\Psi(x,t)|^2$ gives the probability of finding particle at the point x at the time t.
More precisely: $P_{ab} = \int_a^b |\Psi(x,t)|^2 dx$ is a probability of finding particle between a and b at the time t

Statistical Physics has a specific name for $|\Psi(x,t)|^2$, and it is probability density ρ, defined as

$$P_{ab} = \int_a^b \rho(x)dx.$$

So the wave function itself does not have a physical meaning, but the norm of the wave function squared $|\Psi(x,t)|^2$ represents probability density ρ. This introduces indeterminism in the quantum physics - even if we know all about the system, we can not predict the outcome precisely. We can only say with which probability which outcome is likely to happen. Quantum Physics is fundamentally probabilistic. The outcome of the experiment, in general, can not be predicted with 100 % accuracy.

Three key statistical relations in quantum mechanics are:

1. Probability of finding particle in the whole space is equal to 1 (different form of the Born's rule):
$$\int_{-\infty}^{\infty} |\Psi(x,t)|^2 dx = 1. \tag{2.3}$$
2. Expectation value of any observable $f(x)$:
$$\langle f(x)\rangle = \int_{-\infty}^{\infty} f(x)|\Psi(x,t)|^2 dx \tag{2.4}$$
3. A definition of the standard deviation, which we met earlier while discussing uncertainty principle.
$$\sigma^2 = \left\langle (\Delta x)^2 \right\rangle = \left\langle x^2 \right\rangle - \langle x\rangle^2 \tag{2.5}$$

Now, when we know what physical meaning wave function has, we are ready to discuss its important feature - normalization.

2.1.6 Wave Function Normalization

Following Griffiths (§1.4), we require that the wave function must satisfy Born's rule

$$\int_{-\infty}^{\infty} |\Psi(x,t)|^2 dx = 1.$$

Without a statistical interpretation, the wave function has no physical meaning. Thus any solution of the SE must also satisfy Born's rule (2.3) to be admissible. In general, solution of the SE does not satisfy Born's rule - wave function is not **normalized**. The general solution is only determined up to a multiplicative constant $A \in \mathbb{C}$ - $\Phi(x,t) = A\Psi(x,t)$. $\Phi(x,t)$ can be normalized, so that it satisfies equation (2.3):

Normalization Of The Wave Function

Having wave function that satisfies SE (2.2), the process of finding **normalization constant** A by solving

$$A = \frac{1}{\sqrt{\int_{-\infty}^{\infty} |\Psi(x,t)|^2 dx}}.$$

is called **normalization of the wave function**.

Once the wave function is normalized, it stays normalized all the time. Let us show that this statement is true and perform a short derivation, since we will need part of it in the next section, starting with

$$\frac{d}{dt}\int_{-\infty}^{\infty} |\Psi(x,t)|^2 dx = \int_{-\infty}^{\infty} \frac{\partial}{\partial t}|\Psi(x,t)|^2 dx.$$

Notice, that full derivative with respect to t changes to partial derivative, because under the integral $|\Psi(x,t)|^2$ is a function of both x and t. Derivative can be brought under the integral, because the integral does not depend on t. Using chain rule, partial derivative can be expanded like

$$\frac{\partial}{\partial t}|\Psi(x,t)|^2 = \frac{\partial}{\partial t}\left(\Psi\Psi^*\right) = \Psi\frac{\partial \Psi^*}{\partial t} + \Psi^*\frac{\partial \Psi}{\partial t}.$$

$\frac{\partial \Psi}{\partial t}$ can be taken from the SE (2.2):

$$\frac{\partial \Psi(x,t)}{\partial t} = -\frac{\hbar}{2mi}\frac{\partial^2 \Psi(x,t)}{\partial x^2} + \frac{V}{i\hbar}\Psi(x,t) = \frac{i\hbar}{2m}\frac{\partial^2 \Psi(x,t)}{\partial x^2} - \frac{iV}{\hbar}\Psi(x,t)$$

and $\frac{\partial \Psi^*}{\partial t}$ can be obtained by simply taking complex conjugate of the expression above:

$$\frac{\partial \Psi^*(x,t)}{\partial t} = -\frac{i\hbar}{2m}\frac{\partial^2 \Psi^*(x,t)}{\partial x^2} + \frac{iV}{\hbar}\Psi^*(x,t).$$

Collecting everything together,

$$\frac{\partial}{\partial t}|\Psi|^2 = \frac{i\hbar}{2m}\left(\Psi^*\frac{\partial^2 \Psi}{\partial x^2} - \frac{\partial^2 \Psi^*}{\partial x^2}\Psi\right) = \frac{\partial}{\partial x}\left[\frac{i\hbar}{2m}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)\right]. \tag{2.6}$$

Substituting into the integral,

$$\int_{-\infty}^{\infty} \frac{\partial}{\partial t}|\Psi(x,t)|^2 dx = \frac{i\hbar}{2m}\int_{-\infty}^{\infty} \frac{\partial}{\partial x}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right) dx = \frac{i\hbar}{2m}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)\Bigg|_{-\infty}^{\infty}.$$

Since Ψ and Ψ^* are continuous functions, they both should go to zero when x goes to infinity, meaning that

$$\frac{d}{dt}\int_{-\infty}^{\infty} |\Psi(x,t)|^2 dx = 0.$$

Thus, once normalized at one time Ψ remains normalized for all t, as required for a consistent probability density.

We now turn to the concept of observables in quantum mechanics, beginning with position and momentum operators.

2.1.7 Position, Momentum and Kinetic Energy in Quantum Mechanics

Following Griffiths (§1.5), recall that Heisenberg's uncertainty principle states that the position and momentum can not be measured precisely at the same time. This means, that we can no longer use an ordinary position x, but rather deal with its expectational value given by (2.4):

$$\langle x\rangle = \int_{-\infty}^{\infty} x \cdot |\Psi(x,t)|^2 dx.$$

But what does exactly mean "expectational value of x"? If we perform an experiment on a system in the state Ψ to measure position of the particle x, system will no longer be in the state Ψ. It is said that measurement breaks the state. So, if we perform a series of consequent measurements - we will not obtain expectational value of x. To obtain expectational value of x we need to have a way to return the system back to the state Ψ and then perform a measurement. Alternatively, we can prepare a lot of systems in the state Ψ and make a measurement on all of them one time. $\langle x\rangle$ is the average of the measurement of the system in the state Ψ. "*The expectation value is the average of measurements on an ensemble of identically-prepared systems, not the average of repeated measurements on one and the same system*".

Now that we figured out what the $\langle x\rangle$ means, it is worth studying $\frac{d\langle x\rangle}{dt}$, which reminds classical velocity. By definition, it is given by

$$\frac{d\langle x\rangle}{dt} = \int_{-\infty}^{\infty} x\frac{\partial}{\partial t}|\Psi(x,t)|^2 dx.$$

We can use the expression (2.6) for $\frac{\partial}{\partial t}|\Psi(x,t)|^2$ obtained in the previous section:

$$\frac{d\langle x\rangle}{dt} = \frac{i\hbar}{2m}\int_{-\infty}^{\infty} x\frac{\partial}{\partial x}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)dx.$$

This expression can be simplified using integration by parts (to recall what how to integrate by parts, see the lecture [9]):

$$\int_a^b u dv = uv|_a^b - \int_a^b v du,$$

where $u = x$ and $dv = d\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)$, meaning that

$$\begin{aligned}\frac{d\langle x\rangle}{dt} &= \frac{i\hbar}{2m}\left[x\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)\Bigg|_{-\infty}^{\infty} - \int_{-\infty}^{\infty}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)dx\right] \\ &= -\frac{i\hbar}{2m}\int_{-\infty}^{\infty}\left(\Psi^*\frac{\partial \Psi}{\partial x} - \Psi\frac{\partial \Psi^*}{\partial x}\right)dx.\end{aligned}$$

Since Φ vanishes as $|x| \to \infty$, the fist term drops out. This expression can be simplified by performing integration by parts once again to the second term with $u = \Psi$ and $dv = \Psi^*$:

$$-\int_{-\infty}^{\infty}\Psi\frac{\partial \Psi^*}{\partial x}dx = -\Psi\Psi^*\Big|_{-\infty}^{\infty} + \int_{-\infty}^{\infty}\Psi^*\frac{\partial \Psi}{\partial x}dx = \int_{-\infty}^{\infty}\Psi^*\frac{\partial \Psi}{\partial x}dx,$$

meaning that

$$\frac{d\langle x\rangle}{dt} = -\frac{i\hbar}{m}\int_{-\infty}^{\infty}\Psi^*\frac{\partial \Psi}{\partial x}dx.$$

This expression cannot be simplified further, meaning that it is the final result. $\frac{d\langle x\rangle}{dt}$ represents velocity of the expectation value of x. This is not same as the velocity of the particle. So this quantity represents expectational value of velocity of the particles,

$$\langle v\rangle = \frac{d\langle x\rangle}{dt} = -\frac{i\hbar}{m}\int_{-\infty}^{\infty}\Psi^*\frac{\partial\Psi}{\partial x}dx.$$

It is more convenient to work with momentum instead of velocity, which can be defined as

$$\langle p\rangle = m\frac{d\langle x\rangle}{dt} = -i\hbar\int_{-\infty}^{\infty}\Psi^*\frac{\partial\Psi}{\partial x}dx$$

or equivalently,

$$\langle p\rangle = \int_{-\infty}^{\infty}\Psi^*\left(-i\hbar\frac{\partial}{\partial x}\right)\Psi dx. \tag{2.7}$$

Expectation value of position can be written in the similar form,

$$\langle x\rangle = \int_{-\infty}^{\infty}\Psi^*\cdot x\cdot\Psi\ dx \tag{2.8}$$

Values between Ψ^* and Ψ in equations (2.7) and (2.8) represent operators

$$\hat{x} = x,\ \hat{p} = -i\hbar\frac{\partial}{\partial x}. \tag{2.9}$$

Hats over x and p signifies that these are operator, not an ordinary position or momentum. Operators are very important concept of the quantum mechanics and we will discuss them in detail in the respective Section 2.1.11. The only thing we need to know for now is that the operators act on the function that stays to the right side of it and their order can not be changed, meaning that

$$\psi_1(x,t)(\hat{p}\psi_2(x,t)) \neq \psi_2(x,t)(\hat{p}\psi_1(x,t)).$$

If operator acts on the constant, their order can be changed. It is related to the fact that operators usually contain derivatives, and if derivative acts on the function to the right, the result is a completely different function.

Now that we have coordinate and momentum operators, we can construct kinetic energy operator. In quantum mechanics, to obtain a quantum mechanical analogue for classical expression, it is needed to replace classical quantity with its respective operator: observables are represented by hermitian operators. Position, momentum, kinetic energy are observables, they can be measured on the experiment. Therefore, it is easy to obtain operator for kinetic energy:

$$\hat{T} = \frac{\hat{p}^2}{2m} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}.$$

Having derived quantum expression for kinetic energy $\hat{T}$, we can rewrite SE in operator form and tackle its solutions.

2.1.8 Time Independent Schrödinger Equation & Stationary States

The next step is to finally study SE and its general solutions following Griffiths, §2.1.

Potential V in equation (2.2) represent external forces acting on the system. For example, it can be electric potential, magnetic potential or HO potential. If V is not a function of time, then V is called time-independent and SE can be simplified. It can be done by separating of variables in the form

$$\Psi(x,t) = \psi(x)\phi(t).$$

Inserting it into SE (2.2) leads to

$$\begin{aligned} i\hbar\frac{\partial\left(\psi(x)\phi(t)\right)}{\partial t} &= -\frac{\hbar^2}{2m}\frac{\partial^2\left(\psi(x)\phi(t)\right)}{\partial x^2} + V(x)\psi(x)\phi(t); \\ i\hbar\psi(x)\frac{\partial\phi(t)}{\partial t} &= \frac{\hbar^2}{2m}\phi(t)\frac{\partial^2\psi(x)}{\partial x^2} + V(x)\psi(x)\phi(t)\Bigg| \times 1/(\psi(x)\phi(t)); \\ i\hbar\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t} &= \frac{\hbar^2}{2m}\frac{1}{\psi(x)}\frac{\partial^2\psi(x)}{\partial x^2} + V(x). \end{aligned}$$

Left hand side is a function of t, while right hand side is function of x. They can be equal only if both left and right hand sides are equal to constant:

$$i\hbar\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t} = E; \; \frac{\hbar^2}{2m}\frac{1}{\psi(x)}\frac{\partial^2\psi(x)}{\partial x^2} + V(x) = E.$$

Firstly consider left equation,

$$i\hbar\frac{1}{\phi(t)}\frac{\partial\phi(t)}{\partial t} = E.$$

Multiplying by $\phi(t)$ from both side yields

$$\frac{\partial\phi(t)}{\partial t} + \frac{iE}{\hbar}\phi(t) = 0. \tag{2.10}$$

Right hand side of SE with separated variables now reads as

$$\frac{\hbar^2}{2m}\frac{1}{\psi(x)}\frac{\partial^2\psi(x)}{\partial x^2} + V(x) = E,$$

which can be rewritten as

$$\frac{\hbar^2}{2m}\frac{\partial^2\psi(x)}{\partial x^2} + V(x)\psi(x) = E\psi(x). \tag{2.11}$$

This is a time-independent SE. It is an ordinary differential equation of second order. Though deceptively simple, Eq. (2.11) encodes the spectrum of allowed energies for the chosen potential. To proceed forward and actually solve the equation, we need to know the potential $V(x)$. The main focus of this research is HO potential.

Equation (2.10) can be easily solved by multiplying with dt from both sides and integrating, which yields

$$\phi(t) = e^{-iEt/\hbar}.$$

Note, that there is no integration constant. In general, there would be integration constant C after integration, but ψ can be redefined as $\psi' = C\psi$ and prime notation changed back to not primed. In general, any constant from ϕ can be absorbed into ψ.

Three important features of time-independent SE:

1. Time-independent SE describes stationary states. Although full wave function is time dependent $\Psi(x,t) = \psi(x)e^{-iEt/\hbar}$, probability density

   $$\rho = \Psi(x,t)\Psi^*(x,t) = \psi(x)e^{-iEt/\hbar} \cdot \psi^*(x)e^{iEt/\hbar} = |\psi(x)|^2$$

   is time-independent. Moreover, every expectational value of observable variable is equal to

   $$\langle Q(x,p)\rangle = \int_{-\infty}^{\infty} \psi^* Q(x,p)\psi dx \tag{2.12}$$

   and it is time-independent.

2. These stationary states are states of definite total energy. In classical physics total (kinetic + potential) energy of the system is referred to as Hamiltonian

   $$H = \frac{p^2}{2m} + V.$$

   Its quantum analogue is obtained by changing observables to respective operator[2]. By doing that, we obtain Hamiltonian operator, which describes total energy of the system

   $$\hat{H} = \frac{\hat{p}^2}{2m} + V = \frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V. \tag{2.13}$$

   Thus, SE can be rewritten as

   $$\hat{H}\psi = E\psi. \tag{2.14}$$

   In this form, SE can be viewed as an eigenvalue problem. Operator $\hat{H}$ acts on the state ψ and returns constant energy E and the same state ψ. Exactly this equation is the point of interest in this research.

   An important point is that expectational value of $\hat{H}$ is equal to E:

   $$\langle\hat{H}\rangle = \int_{-\infty}^{\infty} \psi^* \left(\hat{H}\psi\right) dx = \int_{-\infty}^{\infty} \psi^* \left(E\psi\right) dx = E\int_{-\infty}^{\infty} \psi^*\psi dx = E. \tag{2.15}$$

   Moreover, expectational value of $\hat{H}^2$ is equal to E^2:

   $$\langle\hat{H}^2\rangle = \int_{-\infty}^{\infty} \psi^* \left(\hat{H}^2\psi\right) dx = \int_{-\infty}^{\infty} \psi^* \left(E^2\psi\right) dx = E^2\int_{-\infty}^{\infty} \psi^*\psi dx = E^2.$$

   It means that the standard deviation of energy is equal to zero,

   $$\sigma_H = \langle\hat{H}\rangle^2 - \langle\hat{H}^2\rangle = E^2 - E^2 = 0.$$

   If we perform a measurement of energy of the system in the state ψ, we will always get value E.

3. Normally, SE 2.25 has more then one solution, and in some cases even infinite amount of solutions. A solution to this equation is an eigenvalue E and eigenfunction ψ. They come as a pair. So, if the equation has infinite solutions, it means that it has infinite amount of eigenvalues $\{E_1, E_2, ...,\}$ and eigenfunctions $\{\psi_1, \psi_2, ...,\}$. The solutions to time-dependent SE are

   $$\Psi_1 = \psi_1(x)e^{-\frac{iE_1}{\hbar}},\ \Psi_2 = \psi_2(x)e^{-\frac{iE_2}{\hbar}},\ ...$$

[2]This is referred to as "quantization".

and there is a different wave function for each allowed energy.

SE has a property, that if Ψ_1 and Ψ_2 are two different solutions, then their linear combination is also a solution to SE. If all the solution are found, then the general solution as a linear combination of separate solutions can be constructed,

$$\Psi(x,t) = \sum_{n=1}^{\infty} c_n \psi_n e^{-\frac{iE_n}{\hbar}}.$$

Expansion coefficients have a physical meaning:

Meaning of Expansion Coefficients

$|c_n|^2$ is a probability that a measurement of the energy would give the energy E_n, and the expectation value of the energy is $\langle \hat{H} \rangle = \sum_n |c_n|^2 E_n$.

And the last important thing to note

Energy Conservation in Stationary States

expansion coefficients c_n are independent of time, and so is the probability of finding system to be in the state with energy E_n, meaning that **energy is conserved**.

Non-relativistic quantum physics is described by linear algebra. Therefore, in the next basic definitions and notation will be introduced.

2.1.9 Linear Algebra

"Quantum theory is based on two constructs: wave functions and operators. The state of a system is represented by its wave function, observables are represented by operators. Mathematically, wave functions satisfy the defining conditions for abstract vectors, and operators act on them as linear transformations. So the natural language of quantum mechanics is linear algebra", Griffiths, §3.1.

However, it is not the same linear algebra familiar from high school or first years in university. In general, linear algebra studies vectors in vector spaces and how they transform. The difference between normal linear algebra and linear algebra in quantum mechanics is the subject of study is the definition of vectors. To understand the idea of linear algebra in quantum mechanics, the simplest concepts of it will be introduced and then "translated into quantum language".

In an N-dimensional space, simplest mathematical representation of a vector is just a tuple of its components $\{a_n\}$ with N entries,

$$|\alpha\rangle \to a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}.$$

Here $|\alpha\rangle$ represents a vector α. Notation $|\rangle$ is referred to as a "ket vector", it represents a column vector. Bra vector is obtained via hermitian conjugation (complex conjugation and transposition) of ket vector,

$$\langle\alpha| = |\alpha\rangle^\dagger \to a = \begin{bmatrix} a_1^* & a_2^* & \cdots & a_N^* \end{bmatrix}.$$

"Bra" and "ket" notation are referred to as Dirac formalism, it is just a way to denote a vector and it is used a lot in quantum mechanics. J. Binney explanation of Dirac formalism in his book [10], §1.3.2 is used. Basic linear algebra is introduced in this formalism so that it is easier to compare it to the quantum analogue.

From mathematical point of view, entries of bra and ket vectors are complex numbers, $\{a_n\} \in \mathbb{C}$ and the whole vector exists in N-dimensional complex space $|\alpha\rangle \in \mathbb{C}^N$.

The inner product (or scalar product) $\langle\alpha|\beta\rangle$ of two N-dimensional vectors is a complex number,

$$\langle\alpha|\beta\rangle = a_1^* b_1 + a_2^* b_2 + ... + a_N^* b_N \in \mathbb{C}.$$

Inner product can be viewed as operation that takes two vectors as input and outputs a complex number. It says that inner product maps two $\mathbb{C}^N$ object (N-dimensional vectors) into $\mathbb{C}$ object (complex scalar).

The inner product satisfies linearity, positivity, and conjugate symmetry (Binney, §1.3.3):

1. Linearity:
$$\langle f|(a\psi + b\phi)\rangle = a\,\langle f|\psi\rangle + b\,\langle f|\phi\rangle\,;$$

2. Positive definiteness:
$$\langle\psi|\psi\rangle \geq 0;$$

3. Conjugate symmetry:
$$\langle\psi|\phi\rangle = (\langle\phi|\psi\rangle)^*.$$

In linear algebra, matrices represent linear transformation of vectors,

$$|\beta\rangle = \hat{T}\,|\alpha\rangle \to b = Ta = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1N} \\ t_{21} & t_{22} & \cdots & t_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N1} & t_{N2} & \cdots & t_{NN} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}.$$

A matrix act of a vector and transforms it into a completely different vector, in general.

After reviewing the foundational principles of linear algebra, we turn to their application in quantum mechanics.

2.1.10 Hilbert space

Wave function in quantum mechanics can be viewed as a vector. The reason the same approach as in the previous section can not be used in quantum mechanics is that wave functions, in general, exist in infinite-dimensional spaces. This means that, for example, inner product would be defined as an infinite sum, and matrices that act on the vector would be infinite-dimensional. Another approach needs to be implemented in quantum physics - Griffiths §3.1.

Recall that in order to represent a state, wave function must satisfy Born's rule (2.3):

$$\int_{-\infty}^{\infty} |\Psi|^2 dx = 1.$$

Hilbert Space

The set of all square-integrable functions f on a specific interval (a,b)

$$f(x) \;:\; \int_a^b |f(x)|^2 dx < \infty$$

forms a vector space. Physicists call this vector space **Hilbert Space**.

Just from the glance on the equation (2.3) and definition of Hilbert space, it is obvious that **wave function exists in a Hilbert Space**.

Inner product in Hilbert space can be defined as

$$\langle f(x)|g(x)\rangle = \int_a^b f(x)^* g(x) dx. \tag{2.16}$$

If both $f(x)$ and $g(x)$ are members of the Hilbert Space (they both are square integrable), then inner product is guaranteed to exist. This definition of inner product is guaranteed to preserve 3 properties stated in the previous section[3].

Normalization, Orthogonality & Orthonormality

If inner product of the function with itself is equal to one,

$$\langle \psi(x)|\psi(x)\rangle = \int_a^b |\psi(x)|^2 dx = 1,$$

such a function is called **normalized**;
If inner product of two functions is equal to zero,

$$\langle \psi(x)|\phi(x)\rangle = \int_a^b \psi^*(x)\phi(x) dx = 0.$$

such functions are called **orthogonal**;
If a set of functions f_n is normalized and they are mutually orthogonal,

$$\langle f_n|f_m\rangle = \delta_{nm},$$

such set is said to be **orthonormal**. Here, $\delta_{mn} = 1$ if $m = n$ and $\delta_{mn} = 0$ if $m \neq n$ is a Kronecker's delta function.

It is very common, that the set of wave functions is orthonormal, and it simplifies calculation a lot. Instead of calculation of a complicated integral, it can be seen immediately what the integral is equal to.

Having defined the quantum version of vectors, we can now introduce operators—the transformations in quantum mechanics—and explain their connection to physical observables.

2.1.11 Observables & Operators

We keep following Griffiths, §3.2. Recall that expression for computing expectation value of observable is given by equation (2.12):

$$\langle Q(x,p)\rangle = \int_{-\infty}^{\infty} \psi^* Q(x,p)\psi dx.$$

[3] In math, in order for an operation be called "inner product", it should satisfy those 3 properties.

Observable Q is "sandwiched" between wave functions under the integral. This expression can be rewritten as a scalar product in Hibert space:

$$\langle Q(x,p)\rangle = \langle\psi|Q(x,p)\psi\rangle = \int_{-\infty}^{\infty} \psi^* Q(x,p)\psi dx.$$

Consider a complex conjugate of the expression above:

$$\langle Q(x,p)\rangle^* = (\langle\psi|Q(x,p)\psi\rangle)^* = \langle Q(x,p)\psi|\psi\rangle = \int_{-\infty}^{\infty} Q^*(x,p)\psi^*\psi dx.$$

The outcome of the measurement has to be real, meaning that

$$\langle Q(x,p)\rangle = \langle Q(x,p)\rangle^* \implies \int_{-\infty}^{\infty} \psi^* Q(x,p)\psi dx = \int_{-\infty}^{\infty} Q^*(x,p)\psi^*\psi dx.$$

This statement is true when $Q(x,p)$ is an operator, not just an ordinary function, meaning that $Q(x,p) \to \hat{Q}(x,p)$. This means that observables in quantum mechanics represented by operators. Moreover, operators that represent observables need to satisfy

$$\left\langle f \middle| \hat{Q} f \right\rangle = \left\langle \hat{Q} f \middle| f \right\rangle \text{ for all } f$$

Operators that satisfy this condition called **Hermitian** and **observables are represented by hermitian operators**.

A **Hermitiac conjugate** of an operator $\hat{Q}$ is the operator $\hat{Q}^\dagger$ such that

$$\left\langle f \middle| \hat{Q} g \right\rangle = \left\langle \hat{Q}^\dagger f \middle| g \right\rangle.$$

A hermitian operator has a property that $\hat{Q} = \hat{Q}^\dagger$.

Actually, when the concept of operators is introduced, it is customary to write its expectation value as

$$\langle Q(x,p)\rangle = \langle\psi|\, \hat{Q}(x,p)\, |\psi\rangle = \int_{-\infty}^{\infty} \psi^* Q(x,p)\psi dx.$$

When written like this, it is considered that operator acts of the vector to the right of it. This definition is equivalent to the previous one, but it signifies that operator Q acts on the state $|\psi\rangle$. Taking the adjoint of the ket-action gives

$$(\hat{Q}\,|\psi\rangle)^\dagger = \langle\psi|\,\hat{Q}^\dagger.$$

In other words, $\hat{Q}$ naturally acts on kets from the right, while its adjoint $\hat{Q}^\dagger$ acts on bras from the left. If the operator is hermitian, the same operator may be viewed as acting either on the ket or on the bra, but in any single expectation bracket it occupies a single position.

Operators are quantum analogs of matrices in linear algebra - they act on the quantum vector - wave function and transform it.

With the minimal quantum framework in place, we can proceed to the quantum harmonic oscillator problem in the following section.

2.1.12 Quantum Harmonic Oscillator

This section draws on Philip L. Bowers's lecture (§2), Griffiths's treatment in §2.3, and Binney & Skinner's The Physics of Quantum Mechanics (§3.1) [10].

Recall, that in section 2.1.2 we discussed a classical 1D particle moving in HO potential

$$V(x) = \frac{kx^2}{2} = \frac{m\omega^2}{2}x^2. \tag{2.17}$$

In this section we will consider a quantum particle moving in HO potential and generalize this concept for 2 and 3 dimensions.

When considering a quantum particle moving in HO potential - we need to solve SE. You can notice that this potential is time-independent, and therefore we need to solve time-independent SE (2.11). We can just use this equation with potential (2.17), because quantum HO potential has exactly the same form:

$$\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \frac{m\omega^2}{2}x^2\psi(x) = E\psi(x). \tag{2.18}$$

For better understanding, let us derive this equation from the general Schrödinger eigenvalue equation (2.14):

$$\hat{H}\psi(x) = E\psi(x).$$

First step in solving the equation is always to write down the Hamiltonian operator for the problem. Recall, that Hamiltonian in classical physics represents sum of kinetic and potential energies,

$$H = \frac{mv^2}{2} + V(x) = \frac{p^2}{2m} + V(x) = \frac{p^2}{2m} + \frac{m\omega^2}{2}x^2.$$

Substituting $x \to \hat{x} = x;\ p \to \hat{p} = -i\hbar\frac{d}{dx}$ immediately yields

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{m\omega^2}{2}x^2.$$

Inserting this Hamiltonian into equation (2.14),

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{m\omega^2}{2}x^2\right)\psi(x) = E\psi(x)$$

leads us to the same equation as (2.18)

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \frac{m\omega^2}{2}x^2\psi(x) = E\psi(x).$$

This equation may seem straightforward, yet finding its solutions proves to be surprisingly difficult. However, there is an analytical solution. It can be obtained by using second quantization or by making an analytical substitution. Both methods described very well in Griffiths' (§2.3). We will not derive the solution, but rather use the results from Griffiths (§2.3):

$$\psi_n(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}}\frac{1}{\sqrt{2^n n!}}He_n(\xi)e^{-\xi^2/2},\ E_n = \left(n + \frac{1}{2}\right)\hbar\omega. \tag{2.19}$$

The answer to this equation is a pair of eigenvalue E_n and eigenfunction $\psi_n(x)$. Here, $\xi = \sqrt{\frac{m\omega}{\hbar}}x$ is dimensionless coordinate, and

$$He_n(\xi) = n! \sum_{m=0}^{[n/2]} (-1)^m \frac{1}{m!2^m(n-2m)!}\xi^{n-2m}$$

is n-th probabilist's Hermite polynomial, according to Abramowitz's book [11], §22.3.

In the research we will need Hermite polynomial up to second order. There is two definition of Hermite polynomials: probabilist's and physicist's polynomials. It is a complicated topic, but we will stick with probabilist's polynomials. First three probabilist's Hermite polynomials stated in the table 2.1.

$He_0(x)$	1
$He_1(x)$	x
$He_2(x)$	x^2-1

Table 2.1: First three probabilist's Hermite polynomials.

Once the solutions for the quantum HO are known, it is possible to perform all different sorts of analysis, and there are huge books that study just HO and its properties. The harmonic oscillator has infinitely many eigenstates. We focus on the ground state by virtue of the variational principle (see Section 2.1.19). Note that even the ground state has $E_0 = \hbar\omega/2 \neq 0$. It is a feature of quantum physics, that even in the lowest possible state, energy is not equal to zero.

It is not hard to generalize HO problem to 2D and 3D. First, we will work through the full three-dimensional case, and then specialize the result to two dimensions. For generalization of 1D concepts to 3D see Griffiths §4.1

Once again, we start with the classical Hamiltonian, which now reads as

$$H = \frac{\mathbf{p}^2}{2m} + \frac{m\omega^2}{2}\mathbf{r}^2.$$

In 3D, momentum operator becomes

$$\mathbf{p} \to \hat{\mathbf{p}} = -i\hbar\nabla,$$

where ∇ is the same gradient vector we met before. Operator ∇^2 is a Laplacian operator Δ and it is equal to

$$\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

Similarly, position operator becomes

$$\mathbf{r} \to \hat{\mathbf{r}} = \mathbf{r}.$$

So that the Hamiltonian operator reads as

$$\hat{H} = -\frac{\hbar^2}{2m}\Delta + \frac{m\omega^2}{2}\mathbf{r}^2. \tag{2.20}$$

Now wave function is a function of three coordinates, $\psi = \psi(x,y,z)$ and SE reads as

$$-\frac{\hbar^2}{2m}\Delta\psi + \frac{m\omega^2}{2}\mathbf{r}^2\psi = E\psi,$$

which expands as

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2}\right)+\frac{m\omega^2}{2}\left(x^2+y^2+z^2\right)\psi=E\psi.$$

This equation can be solved by changing coordinate system to polar or by separation of variables. Great way to solve this problem is presented in the chapter 7.3 of J. Binney's and D. Skinner's book. The answer is

$$\psi_{n_x,n_y,n_z}(x,y,z)=\psi_{n_x}(x)\psi_{n_y}(y)\psi_{n_z}(z),\ E_{n_x,n_y,n_z}=\left(n_x+n_y+n_z+\frac{3}{2}\right)\hbar\omega, \qquad (2.21)$$

where $\psi_{n_x}(x)$, $\psi_{n_y}(y)$ and $\psi_{n_z}(z)$ are the same 1D HO wave function as in (2.19).

Having 1D and 3D solutions, it is easy to write down the 2D HO solution,

$$\psi_{n_x,n_y}(x,y)=\psi_{n_x}(x)\psi_{n_y}(y),\ E_{n_x,n_y}=(n_x+n_y+1)\,\hbar\omega. \qquad (2.22)$$

Subscripts n, n_x, n_y and n_z are called Primal quantum numbers and they denote in which state in which dimension particle is. Quantum state of the system can be obtained just by setting these quantum numbers. Indeed, just by setting n_x, n_y and n_z we can write down energy and wave function of the system.

Notice, that in 2D and 3D cases energy becomes degenerate. It means that the same values of energy correspond to specific different pairs of n_x, n_y and n_z. For example, in 2D $E_{n_x=1,\ n_y=0}=2\hbar\omega$ and $E_{n_x=0,\ n_y=1}=2\hbar\omega$.

In the research we will mostly study 2D HO, but we will look at some particular examples in 1D and 3D as well.

In the next chapter, we turn to charged-particle interactions in quantum mechanics.

2.1.13 Coulomb's Potential

To describe the electrostatic interaction between charged quantum particles, we introduce the Coulomb potential. For two electrons – each with charge $-e$ the repulsive potential energy as a function of their separation r is (Griffiths, §4.2)

$$V(r)=\frac{e^2}{4\pi\varepsilon_0 r}. \qquad (2.23)$$

Here $e\approx 1.602\cdot 10^{-19}$ C is the elementary charge, and $\epsilon_0\approx 8.854\cdot 10^{-12}$ F $\cdot$ m is the vacuum permittivity (CODATA).

As we discussed earlier, to obtain quantum expression we just need to change all variables to respective operators. In this case:

$$r\to\hat{r}=r.$$

So the potential remains unchanged.

For an N-electron system, each electron experiences the sum of Coulomb potentials from all others. We return to this many-body interaction in the Section 2.1.21.

Having introduced the electrostatic potential, the only remaining one-particle quantum concept for our study is the electron's spin.

2.1.14 Spin

Spin is not an entirely quantum concept, it appears in classical physics. In classical physics, a rigid object has two kinds of angular momentum: **orbital** ($\mathbf{L} = [\mathbf{r} \times \mathbf{p}]$), associated with motion of its center of mass, and **spin** ($\mathbf{S} = I\omega$), associated with rotation about its own axis. Simply said, if object rotates - it has spin. Earth, for example, has an orbital momentum due to its rotation around the Sun, and spin angular momentum due to its rotation around its north-south axis. In classical physics, spin is just a sum of all orbital momenta of all the things that make up the Earth, Griffiths, §4.4.

The concept of spin in quantum physics is somewhat similar, but at the same time is fundamentally different. In hydrogen atom, electron carries orbital momentum due to its motion around the nucleolus, and has an intrinsic spin, which has nothing to do with the motion in space, but is somewhat similar to classical spin. Quantum spin is analogous in that it contributes to total angular momentum, but it is intrinsic – electrons have no spatial extent or internal parts to "spin" around. Thus spin is a purely quantum degree of freedom.

We have seen that for quantum particle in HO potential, state of the system can be described by the primal quantum number n. Spin introduces two new quantum numbers - s spin value of the particle and m_s - projection of spin of z-axis. m_z has $2s+1$ possible projections, i.e. $m_s \in \{-s, -s+1, ..., 0, ..., s-1, s\}$. All quantum particles have exactly one fixed value of spin s, while spin projections can take different values in the range $\{-s, -s+1, ..., 0, ..., s-1, s\}$.

Elector has a spin $s = 1/2\hbar$ and it can be parallel to z-axis, $m_s = 1/2\hbar$ or anti-parallel, $m_s = -1/2\hbar$. On the other hand, photon has spin $s = \hbar$ and m_s can be equal to $\hbar$, 0 and $-\hbar$. Electron and photon are two fundamentally different particles, because of their spin values.

Fermions & Bosons

Particle with half integer spin is called **fermion**;
Particle with integer spin is called **boson**. Griffiths, § 5.1.1.

Following Szabo and Ostlund many-body spin introduction [12] (§2.2.1), to describe quantum system completely, we need to take spin into account. To keep it simple, consider an electron. Electron can be in the "spin-up" state, when its spin is parallel to z-axis, and in "spin-down" state, when its spin is anti-parallel to z-axis. Define two orthogonal functions the represent spin-up $\alpha(\xi)$ and spin-down $\beta(\xi)$ states of unidentified spin ξ, so that

$$\langle\alpha|\alpha\rangle = \int_{-\infty}^{\infty} \alpha^*(\xi)\alpha(\xi) = 1d\xi \ , \ \langle\beta|\beta\rangle = \int_{-\infty}^{\infty} \beta^*(\xi)\beta(\xi)d\xi = 1$$

and

$$\langle\alpha|\beta\rangle = \int_{-\infty}^{\infty} \alpha^*(\xi)\beta(\xi)d\xi = 0 \ , \ \langle\beta|\alpha\rangle = \int_{-\infty}^{\infty} \beta^*(\xi)\alpha(\xi)d\xi = 0.$$

In this formalism, electron is described by four coordinates - three spatial coordinates and spin,

$$\mathbf{x} = \{\mathbf{r}, \xi\} = \{x, y, z, \xi\}.$$

To completely describe an electron, its spin need to be specified. Electron can in spin-up or a spin-down state, and therefore wave function that describe both its spatial and spin distributions has a form

$$\phi(\mathbf{x}) = \psi(\mathbf{r})\alpha(\xi) \text{ or } \phi(\mathbf{x}) = \psi(\mathbf{r})\beta(\xi).$$

If wave function set is orthonormal, so is it its spin part,

$$\langle \phi_m | \phi_n \rangle = \int_{-\infty}^{\infty} \phi_m^*(\xi) \phi_n(\xi) d\xi = \delta_{mn}.$$

Most quantum problems does not have a spin dependency in Hamiltonian, and wave function is a product of spatial and spin wave function. Therefore spin degrees of freedom are frequently left out with the remark that it is easy to restore them.

Since we will study Bose–Einstein condensates (bosons) and electron gases (fermions) in harmonic traps, understanding of the boson and fermion concepts are crucial for the work. The same goes for the concept of spin, it will also play a huge role when defining a wave function for the fermionic system.

With spin and statistics introduced, we now turn to the Pauli exclusion principle and its role in many-particle quantum states.

2.1.15 Pauli Principle

To model N electrons interacting both with M fixed nuclei and with each other, we next construct the many-electron Hamiltonian. The Hamiltonian includes N kinetic-energy operators„ $N \times M$ electron-nucleus Coulomb terms, and $1/2N(N-1)$ electron-electron Coulomb terms:

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N} \nabla_i^2 + \sum_{i=1}^{M}\sum_{j=1}^{N} \frac{Z_j}{r_{ij}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}}. \tag{2.24}$$

Here $r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ is a relative distance between particle i and particle j and Z_j is a charge number of j-th particle, if its charge is not elementary.

Notice, that this Hamiltonian is written using different units. These units are called **atomic units**. Atomic units are frequently used in quantum physics, because we do not need to take into account a huge amount of constants[4]. In atomic units, setting $\hbar = e = 4\pi\varepsilon_0 = m_e = 1$ makes all prefactors unity according to Szabo and Ostlund (§2.1.1). Step by step derivation of atomic units is introduced in Appendix A (Section **??**).

Hamiltonian (2.39) depends only on spatial coordinates of the electron. Wavefunction represent a wave function of N electrons and it is a function of N $\mathbf{x}$ coordinates (Szabo and Ostlund, §2.1.2),

$$\Psi = \Psi(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}).$$

Definition of new coordinates does not change the fact that the Hamiltonian is spin-independent. The way to include spin is to add an additional requirement of a wave function:

Antisymmetry Pauli Principle

A many-electron wave function must be antisymmetric with respect to interchange of the coordinates $\mathbf{x}_i$ and $\mathbf{x}_j$ (Szabo, §2.1.3)

$$\Psi(\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x}_i, ..., \mathbf{x}_j, ..., \mathbf{x_N}) = -\Psi(\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x}_j, ..., \mathbf{x}_i, ..., \mathbf{x_N}).$$

This is a postulate of many body physics. It implies that fermionic wave function need to be antisymmetric with respect two interchange of two particles.

In the next section we introduce the Slater determinant construction, which automatically enforces this antisymmetry for fermions.

[4]However, restoring SI units later requires careful bookkeeping.

2.1.16 Fermionic Many Body Wave Function

Following Griffiths (§5.1.1), to understand how to construct a many body wave function, consider a system of only two fermion and then generalize the concept to N fermions. Suppose we have two non interacting particles: particle 1 in the state $\phi_a(\mathbf{x_1})$ and particle 2 in the state $\phi_b(\mathbf{x_2})$. Let Ψ_- denote a fermionic wave function. Then total wave function can be written as

$$\Psi_-(\mathbf{x_1}, \mathbf{x_2}) = \phi_a(\mathbf{x_1})\phi_b(\mathbf{x_2}).$$

This expression makes sense only if we can distinguish these particles. If there is no way to distinguish these particles, expression above is incorrect, because there is no way to know which particle is in which state. In this case, all we can say in that one of the particles is in the state ϕ_a and another one is in the state ϕ_b. This is exactly the case in quantum world. All electrons in the world are identical. We can not paint 1 and 2 on electrons and tell them apart this way. The only way to tell apart two electrons is to perform a measurement, but this measurement breaks the state in which the electrons were. So, when constructing a wave function that describes two electrons, we need to take into account that both electrons can be in the state ϕ_a, as well as in the state ϕ_b. There is two ways to construct a wave function that takes into account that particles are indistinguishable:

$$\Psi_\pm(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\sqrt{2}}\left(\phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) \pm \phi_b(\mathbf{x}_1)\phi_a(\mathbf{x}_2)\right).$$

$\Psi_+(\mathbf{x}_1, \mathbf{x}_2)$ is a bosoinc wave function, while $\Psi_-(\mathbf{x}_1, \mathbf{x}_2)$ is a fermionic wave function. Factor $1/\sqrt{2} = 1/\sqrt{2!}$ comes from wave function normalization.

Bosons are symmetric under exchange of two particles,

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2) = \Psi_+(\mathbf{x}_2, \mathbf{x}_1),$$

while fermions are antysymmetric

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2) = -\Psi_-(\mathbf{x}_2, \mathbf{x}_1).$$

This leads to an important fact – wavefunction of two fermions in the same state is equal to zero:

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\sqrt{2}}(\phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2) - \phi_a(\mathbf{x}_2)\phi_a(\mathbf{x}_1)) = 0.$$

If wave function is equal to zero – it means that probability of finding two electrons in all space is equal to zero. But we had two electrons to begin with and they have to be somewhere. It implies that two fermions can not be in the same state. This is called **Pauli exclusion principle**. Two fermions cannot share the same spin-orbital. Recall, that a quantum state of the particle can be described by quantum numbers. Thus, all fermions in the system can not have the same quantum numbers. At least one quantum number must be different.

To generalize this to N particles, total wave function needs to satisfy antisymmetry Pauli principle. Also, it needs to be constructed from single particle wave function. A mathematical object that just perfectly satisfies this condition is a determinant, so that fermionic wave function is given by

$$\Psi_- = \frac{1}{\sqrt{N!}}\begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{vmatrix} \tag{2.25}$$

Here, $\phi_i(\mathbf{x}_j)$ is a spinorbital, a function of the space and spin coordinates of j-th electron. Factor $\frac{1}{\sqrt{N!}}$ ensures then the wave function is properly normalized, meaning that $\langle \Psi_- | \Psi_- \rangle = 1$. This determinant wave function is called **Slater Determinant** (Shavitt & Barlett, §1.3).

Even when $N = 3$ it is hard to deal with such wave function. And keep in mind that this wave function is needed to perform any type of calculation. This is why in many body physics there is almost no problem with analytical solution, and numerical and approximate methods replace analytical ones.

Determinant is an object, that is very computationally expensive to compute. According to the article of Sariyanidi [**?**], it takes more than 10^{141} years to compute 100×100 determinant using modern CPU. Therefore, we need to keep the dimensionality low, or use some methods to approximate the determinant. For the system of interest, Slater determinant of size N can be approximated by the product of two Slater determinants of size $N/2$, see Section **??**.

Next, we construct the totally symmetric many-boson wave function, which contrasts sharply with the fermionic determinant.

2.1.17 Bosonic Many Body Wavefunction

Like fermions, indistinguishable bosons require symmetrized many-body states. This means that, in general it is needed to construct a similar to fermions wave function. In previous section it was shown that for 2 bosons wave function is

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\sqrt{2}}(\phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) + \phi_b(\mathbf{x}_1)\phi_a(\mathbf{x}_2))$$

and it needs to be symmetric under exchange of two particles,

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2) = \Psi_+(\mathbf{x}_2, \mathbf{x}_1).$$

Let us try to extend this concept up to 3 bosons. There is $6 = 3!$ permutations of 3 bosons:

$$\begin{aligned}\Psi_+(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) &= \frac{1}{\sqrt{6}}(\phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2)\phi_c(\mathbf{x}_3) + \phi_b(\mathbf{x}_1)\phi_c(\mathbf{x}_2)\phi_a(\mathbf{x}_3) + \phi_c(\mathbf{x}_1)\phi_a(\mathbf{x}_2)\phi_b(\mathbf{x}_3) \\ &\quad + \phi_a(\mathbf{x}_1)\phi_c(\mathbf{x}_2)\phi_b(\mathbf{x}_3) + \phi_b(\mathbf{x}_1)\phi_a(\mathbf{x}_2)\phi_c(\mathbf{x}_3) + \phi_c(\mathbf{x}_1)\phi_b(\mathbf{x}_2)\phi_a(\mathbf{x}_3)) \\ &= \frac{1}{\sqrt{3!}} \sum_{\text{perm}} \phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2)\phi_c(\mathbf{x}_3),\end{aligned}$$

where sum contains all possible permutations.

It is easy to generalize the bosonic wave function to N bosons,

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) = C_N \sum_{\text{perm}} \phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) \cdots \phi_z(\mathbf{x}_N).$$

Here

$$C_N = \left(\tfrac{N_1!\,N_2! \cdots N_n!}{N!}\right)^{1/2}$$

is the normalization constant, where N_i is the number of bosons in orbital i (so $\sum_i N_i = N$). In the special cases this becomes

$$C_N = \begin{cases} 1/\sqrt{N!}, & \text{if each boson occupies a distinct orbital,} \\ 1, & \text{if all } N \text{ bosons occupy the same orbital.} \end{cases}$$

The labels $a, b, \ldots, z$ denote the chosen single-particle states. This symmetrized-product (Hartree-Fock approximation) Ansatz is the standard mean-field form for bosons [13], §8.3.1.

Since bosons do not follow antysymmetry Pauli principle, two or more bosons can be in the same quantum state. If the system is cooled down to the critical temperatures, all the bosons occupy the lowest possible state - GS. This type of cooled bosonic system is referred to as Bose-Einstein condensate. BEC is a purely quantum phenomenon and it has very interesting properties. A lot of Nobel prizes in the resent years were given for its study. We will study exactly this type of system – BEC.

For BEC, total wave function can be simplified further. Once again, consider 2 bosons system, where the wave function is

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2) = C_2(\phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2) + \phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2)) = \phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2).$$

This is possible, because we know for sure that all bosons are in the lowest energy state a. Expanding this to N bosons:

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) = \phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2)\cdots\phi_a(\mathbf{x}_N). \tag{2.26}$$

Note, that Ψ_+ does not have normaliztion constant, since of all wavefunctions are normalized themselves. Lecture notes on The Gross-Pitaevskii equation [14] utilizes exactly the same wave function for BEC.

Next, we'll look at issues the electrostatic potential introduces and discuss how to address them.

2.1.18 Kato's Cusp Condition

Certain potentials can have singularity points. Consider a two electron problem. Hamiltonian (in atomic units) for a such system is

$$\hat{H} = -\frac{1}{2}\Delta_1 - \frac{1}{2}\Delta_2 + \frac{1}{r_{12}}.$$

Here Coulomb's potential, which is the part of the Hamiltonian, has a singularity point. If the electrons are arbitrary close, meaning $r_{12} \to 0$, $V(x)$ blows up and energy of the system becomes infinite. However, this is not the end of the story. Hamiltonian has two more terms, that represent kinetic energies of two electrons, which contains second derivatives. If wave function can be modified in such a way, that second derivative of the modified wave function cancels out the divergent term – the problem is solved.

Tosio Kato was the first one to study such divergences in his work [15]. He found that wavefunction must satisfy Kato's theorem,

$$\lim_{r_{ij}\to 0}\left(\frac{\partial\psi}{\partial r_{ij}}\right) = \mu Z_i Z_j \psi(r_{ij} = 0).$$

If wave function satisfies Kato's theorem, then divergence is canceled out. Here $\mu = m_i m_j/(m_i + m_j)$ is the reduced mass of two electrons, and Z is the charge number.

We now introduce the Jastrow cusp factor that enforces Kato's condition and renders our local energy finite.

2.1.19 Jastrow factor & Pade-Jastrow factor

To satisfy Kato's cusp condition, wave function ansatz should contain additional term, which is called Jastrow factor. It is easy to obtain it from the Kato's theorem explicitly. Let us walk through this brief calculation, starting with Kato's theorem in a slightly different form:

$$\begin{aligned}\frac{d\psi}{dr_{ij}} &= C\psi;\\ \frac{d\psi}{\psi} &= Cdr_{ij};\\ \ln(\psi) &= Cr_{ij};\\ \psi &= e^{Cr_{ij}}.\end{aligned}$$

The constant for two electrons is equal to $C = \mu Z_i Z_j = 1/2$, but it can take another values for different particles. Jastrow factor is frequently used in VMC simulations, and therefore this constant C becomes a variational parameter.

A simple Jastrow factor for two electrons is equal to $J = \exp\left(Cr_{ij}\right)$. More generally, it can be written as $J = \exp\left(\sum_{i>j} u(r_{ij})\right)$. Here, the sum takes into account correlation of electron with all other particles. For the system of N particles, Jastrow factor is equal to

$$J = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N} C_{ij} r_{ij}\right). \tag{2.27}$$

Pade-Jastrow factor has a slightly different form [16],

$$P = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N} \frac{ar_{ij}}{(1+\beta r_{ij})}\right), \tag{2.28}$$

where $a = 1$ if spins of two particles are parallel and $1/3$ if spin of particles are anti-parallel.

Jastrow factors of various forms are used in variational Monte Carlo calculations of quantum problem. These factors make sure that if two particles get close, energy will not diverge. Obviously, it introduces additional term to the wave function ansatz and calculation with different Jastrow factors can have slightly different results. Only comparison with experiment can tell which factor is better for the particular problem.

Now, when all puzzle pieces are set in places, we are ready to move on to variational principle theorem.

2.1.20 Variational Principle

The variational principle provides a straightforward upper bound to the ground-state energy. It is easy to derive it, so let us follow Griffiths' approach in §7 to do it.

Consider a system with a Hamiltonian $\hat{H}$. Unknown wave function satisfies time-independent SE (2.14)

$$\hat{H}\left|\psi\right\rangle = E\left|\psi\right\rangle.$$

There exist a set of eigenfunction $\left|\psi_n\right\rangle$ and eigenvalues E_n – solutions of this equation, which are unknown. Then, general solution $\left|\psi\right\rangle$ is a linear combination of $\left|\psi_n\right\rangle$,

$$\left|\psi\right\rangle = \sum_n c_n \left|\psi_n\right\rangle.$$

Here, c_n are the same expansion coefficients discussed in Section 2.1.8. State $|\psi_n\rangle$ is an eigenstate of the Hamiltonian, meaning

$$\hat{H}\,|\psi_n\rangle = E_n\,|\psi_n\rangle\,.$$

Assuming that basis wave functions $|\psi_n\rangle$ form an orthonormal basis ($\langle\psi_n|\psi_m\rangle = \delta_{mn}$), wave function $|\psi\rangle$ is normalized:

$$\langle\psi|\psi\rangle = \sum_n\sum_m c_n^* c_m \langle\psi_n|\psi_m\rangle = \sum_n\sum_m c_n^* c_m \delta_{mn} = \sum_n |c_n|^2 = 1,$$

In this basis, expectational value of energy is given by

$$\langle\hat{H}\rangle = \langle\psi|\,\hat{H}\,|\psi\rangle = \sum_n\sum_m c_n^* c_m \langle\psi_n|\,\hat{H}\,|\psi_m\rangle = \sum_n\sum_m c_n^* c_m E_m \langle\psi_n|\psi_m\rangle = \sum_n E_n|c_n|^2.$$

By definition, GS energy is the lowest energy, meaning $E_{GS} = E_0 < E_1 < E_2 < \cdots$. If in the sum $\sum_n E_n|c_n|^2$ all energies E_n will be replaced with the ground state energy $E_0 = E_{GS}$, this sum will always be smaller than $\sum_n E_n|c_n|^2$:

$$\sum_n E_{GS}|c_n|^2 \leq \sum_n E_n|c_n|^2$$

and the sum over GS energies is equal to

$$\sum_n E_{GS}|c_n|^2 = E_{GS}\sum_n |c_n|^2 = E_{GS}.$$

These two expression above form a variation principle:

Variational Principle

for any normalized $|\psi\rangle$, GS energy is always smaller or equal to the expectational value of Hamiltonian,

$$E_{GS} \leq \langle\hat{H}\rangle = \langle\psi|\,\hat{H}\,|\psi\rangle\,.$$

This principle may seem a bit strange, because the only thing it states is that GS energy is always bigger than expectational value of Hamiltonian. But at the same time, it gives a lot of freedom, wave functioncs can be picked arbitrary. Value $\Delta E = \langle\hat{H}\rangle - E_{GS}$ measures the quality of the ansatz wave function. If wave function ansatz is an exact wave function - then equality hods – $\langle\hat{H}\rangle = E_{GS}$. And if the ansatz is not close to the true wave function, then ΔE can be huge. Minimizing $\langle\hat{H}\rangle$ over variational parameters drives $\Delta E \to 0$. Variational principle is the reason GS energy is studied, because it provides a recipe to compute it.

In quantum chemistry, scientists compute GS energies of very complicated molecules with a very high precision using variational principle. This principle plays a key role in almost all of computational physics and chemistry.

The recipe is very simple - choose a wave function ansatz (generally with variational parameters), compute expectational value of Hamiltonian using this wave function, tune the parameters to get the smallest possible energy.

In this research, VMC method is used to compute GS energy of bosonic and fermionic systems, and the variational principle plays a key role in it.

Armed with the variational principle, we now introduce the treatment of the general time-independent SE.

2.1.21 Local energy & Variational Monte Carlo

Following Morten Hjorth-Jensen's lecture notes [17]. Consider a general time-independent quantum problem, without specifying Hamiltonian and wave function. This will give the understanding of the general problem and motivate Variational Monte Carlo Machinery.

Consider a general system of N particles and a given Hamiltonial $\hat{H}$. According to the variational principle, to compute GS energy it is needed to choose a trial wave function anstaz $\psi_T(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n; \alpha) = \psi_T(\mathbf{R}, \alpha)$. The ansatz contains m variational parameters $\alpha = \{\alpha_1, ..., \alpha_m\}$. GS energy can be obtained from time-independet SE equation (2.14)

$$\hat{H}\psi_T(\mathbf{R}, \alpha) = E\psi_T(\mathbf{R}, \alpha).$$

Multiply this equation from the left with $\psi_T^*(\mathbf{R}, \alpha)$ and integrate over all spatial coordinates $dV = d\mathbf{x}_1 \cdots \mathbf{x}_N$:

$$\int \psi_T^*(\mathbf{R}, \alpha)\hat{H}\psi_T(\mathbf{R}, \alpha)dV = E\int |\psi_T(\mathbf{R}, \alpha)|^2 dV.$$

We make sure to approximate GS energy by choosing the trial wave function of specific form. The problem is solved if we are able to solve this equation:

$$E_{GS} = \frac{\int \psi_T^*(\mathbf{R}, \alpha)\hat{H}\psi_T(\mathbf{R}, \alpha)dV}{\int |\psi_T(\mathbf{R}, \alpha)|^2 dV}. \tag{2.29}$$

It does not look terrifying, but let us take a closer look at the integrals. For simplicity, consider BEC system, where the ansatz is given by (2.26)

$$\psi_T(\mathbf{R}, \alpha) = \phi(\mathbf{x}_1; \alpha)\phi(\mathbf{x}_2; \alpha)\cdots\phi(\mathbf{x}_N; \alpha).$$

The integral than simplifies to

$$\int |\psi_T(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n; \alpha)|^2 dV = \int_{-\infty}^{\infty} \phi(\mathbf{x}_1; \alpha)d\mathbf{x}_1 \cdots \int_{-\infty}^{\infty} \phi(\mathbf{x}_N; \alpha)d\mathbf{x}_N.$$

This expression contains N d-dimensional integrals, that need to be computed on the interval $(-\infty, \infty)$. The same can be said about the integral in the numerator, but it contains even more complicated expression under every integral. "Given 10 particles and 10 mesh points for each degree of freedom and an ideal 1 Tflops machine (all operations take the same time), how long will it take to compute the integral (2.29)? The lifetime of the universe is of the order of 10^{17} s. Morten Hjorth-Jensen's lecture notes for FYS4411, page 5."

Therefore, another approach need to be implemented, namely Variational Monte Carlo. For a trial wave function, probability distribution function P is equal to

$$P = \frac{|\psi_T(\mathbf{R}, \alpha)|^2}{\int |\psi_T(\mathbf{R}, \alpha)|^2 dV}. \tag{2.30}$$

Note, that this is exactly the same probability density ρ discussed in the Section 2.1.5 for not normalized wave function. We can define a new quantity – **local energy**

$$E_L = \frac{1}{\psi_T(\mathbf{R}, \alpha)}\hat{H}\psi_T(\mathbf{R}, \alpha). \tag{2.31}$$

From now on, the dependencies will be dropped for simplicity. Expectation value of the local energy is equal to

$$\langle E_L\rangle = \int PE_L dV = \int \left(\frac{|\psi_T|^2}{\int |\psi_T|^2 dV} E_L\right) dV = \frac{\int |\psi_T|^2 E_L dV}{\int |\psi_T^*|^2 dV} = \frac{\int \psi_T^* \psi_T E_L dV}{\int |\psi_T|^2 dV}$$
$$= \frac{\int \psi_T^* \psi_T \psi_T^{-1} \hat{H} \psi_T dV}{\int |\psi_T|^2 dV} = \frac{\int \psi_T^* \hat{H} \psi_T dV}{\int |\psi_T|^2 dV}.$$

Thus, expectational value of the local energy is equal to the GS energy, given by equation (2.29). This is a truly remarkable result, because it enables computation of GS energy using VMC:

$$E_0 = \langle E_L\rangle = \int PE_L dV \approx \frac{1}{N}\sum_{i=1}^{N} E_L(\mathbf{R}_i, \alpha). \tag{2.32}$$

Here, N is the number of Monte Carlo samples. Note, that we avoid computing complicated integral $\int PE_L dV$ and approximate it with the mean value of sampled local energies. This approach is much less computationally expensive, while still very precise and the variational principle ensures that $E_{GS}^{\text{VMC}} \geq E_{GS}^{\text{True}}$.

We are ready to return to the model system and define it in the next section.

2.1.22 Model System, Part 2

We now turn to the specific model systems, each consisting of N particles confined in a harmonic-oscillator potential. These are not purely theoretical constructs – there are well-established experimental techniques for trapping both charged fermions and neutral bosons. For a comprehensive survey of these methods, see Knoop, Madsen and Thompson [18]. In practice, electrons (fermions) are held in HO-like wells using Penning traps, while alkali atoms (composite bosons with integer total spin) are captured in magneto-optical or magnetic traps and then evaporatively cooled to form a BEC.

As we discussed earlier, quantum system are defined by the Hamiltonian. Variational principle states that the trial wave function ansatz can be chosen. General form of the Hamiltonian for the model system is

$$\hat{H} = \hat{H}_0 + \hat{H}_I + \hat{V}.$$

Here, $\hat{H}_0$ represents non interacting part of the Hamiltonian - kinetic energies of all particles,

$$\hat{H}_0 = -\frac{\hbar^2}{2m}\sum_{i=1}^{N} \Delta_i.$$

$\hat{H}_I$ represents interacting part of the Hamiltonian - electrostatic Coulomb's Potential,

$$\hat{H}_I = \frac{1}{4\pi\varepsilon_0}\sum_{i=1}^{N}\sum_{j>i}^{N} \frac{q_i q_j}{r_{ij}}, \tag{2.33}$$

where q is the charge of the particle. $\hat{V}$ represents harmonic potential, that every particle feel,

$$\hat{V} = \frac{m\omega^2}{2}\sum_{i=1}^{N} r_i^2. \tag{2.34}$$

We will study interacting and non-interacting fermionic systems and non-interacting bosonic systems.

In atomic units (see Appedix 1), Hamiltonian parts have a form:

$$\hat{H}_0 = \sum_{i=1}^{N} -\frac{1}{2}\Delta_i, \tag{2.35}$$

$$\hat{H}_I = \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}} \tag{2.36}$$

and

$$\hat{V} = \frac{1}{2}\sum_{i=1}^{N} r_i^2 \tag{2.37}$$

Bosonic trial wave function ansatz has a form (2.26)

$$\Psi_{\mathrm{B}} = \prod_{i=1}^{N} \psi_a(\mathbf{r}_i;\alpha)$$

with one variational parameter α. In this case, we consider the system to be cooled down to critical temperatures, so that ψ_a represent singe body GS HO wave function. Its form will be specified in the Methods and Implementations section.

Fermionic trial wave function ansatz is given by Slater determinant (2.25)

$$\Psi_{\mathrm{F}} = \frac{1}{\sqrt{N!}}\begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \cdots & \psi_N(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_N(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_N) & \psi_2(\mathbf{r}_N) & \cdots & \psi_N(\mathbf{r}_N) \end{vmatrix} \tag{2.38}$$

Once again form of determinant components and approximation to its alternative form will be discussed in the Methods and Implementations section.

When considering interacting fermions, trial wave function ansatz should be multiplied with Jastrow factor J or Pade-Jastrow factor P:

$$J = \exp\left(\sum_{i=1}^{N}\sum_{j>i}^{N} \beta_{ij} r_{ij}\right), \tag{2.39}$$

$$P = \exp\left(\sum_{i>j} \frac{a r_{ij}}{(1+\beta r_{ij})}\right), \tag{2.40}$$

Jastrow factor J contains $1/2N(N-1)$ variational parameters β_{ij}, while Pade-Jastrow factor has a single variational parameter β.

Our aim is to compute and compare the ground-state energies of bosonic and fermionic HO-trapped systems across varying particle numbers and trap strengths, while also assessing the effects of Jastrow and Padé–Jastrow correlation factors, and other factors that will be discussed further.

Let us summarize all the different things that define a quantum system:

1. A set of parameters (number of particles N, number of dimensions M, trap stregth ω, etc.);
2. Hamiltonian of the system $\hat{H}$, that is used to compute and sample local energy;

3. Trial wave function ansatz ψ_T and variational parameters α and β, that are a part of the ansatz.

Once the setup is chosen, GS energy is computed using VMC by implementing equation (**??**). Let us continue with introducing Markov Chain Monte Carlo method and related concepts in the next section.

2.2 Computational Foundations

2.2.1 Bayesian Statistics

Following "Markov Chain Monte Carlo in Practice" [19] by W.R. Gilks, S. Richardson and D.J. Spiegelhalter (§1.2), MCMC methods are designed primarily for Bayesian inference, in which both data and model parameters are treated as random variables. Let D denote the observed data and θ denote the missing data and model parameters. Inference than requires a construction the joint distribution $P(D, \theta)$ over all random quantities

$$P(D, \theta) = P(D|\theta)P(\theta).$$

This probability distribution gives a full probability model. $P(\theta)$ is a prior distribution and $P(D|\theta)$ is a likelihood.

Having collected observed data D, Bayesian theorem is used to determine the distribution of θ conditional on D:

$$P(\theta|D) = \frac{P(\theta)P(D|\theta)}{\int P(\theta)P(D|\theta)d\theta}.$$

This is a posterior distribution of θ - and object of Bayesian statistics. The posterior expectation value of a function $f(\theta)$ is

$$\mathbb{E}\left[f(\theta)|D\right] = \langle f(\theta|D)\rangle = \frac{\int f(\theta)P(\theta)P(D|\theta)d\theta}{\int P(\theta)P(D|\theta)d\theta}.$$

This expectation value $\langle f(\theta)\rangle$ is impossible to find analytically in high dimensions, and numerical methods are used to compute it. Markov Chain Monte Carlo is the most used and stable.

In general terms, X denotes a vector of k variables with distribution $\pi(x) = P(D|\theta)P(\theta)$. In these terms, expectational value is

$$\mathbb{E}\left[f(X)\right] = \langle f(X)\rangle = \frac{\int f(x)\pi(x)dx}{\int \pi(x)dx}.$$

In these terms, the distribution of X is known only up to a constant factor, meaning that $\int \pi(x)dx$ is unknown.

2.2.2 Monte Carlo Integration

According to Gilks (§1.3.1), numerical Monte Carlo method evaluates $\mathbb{E}\left[f(X)\right]$ by drawing samples $\{X_t,\ t = 1, ..., n\}$ from $\pi(x)$ and then approximating

$$\mathbb{E}\left[f(X)\right] \approx \frac{1}{n}\sum_{t=1}^{n} f(X_t).$$

In other words, the population mean of $f(X)$ is replaced by a sample mean. Laws of large numbers ensure that estimation can be made as accurate as possible by increasing number of Monte Carlo samples n.

$\pi(x)$ can be a complicated distribution, meaning that drawing samples $\{X_t\}$ independently from $\pi(x)$ is not possible. However, samples need not be independent draws from $\pi(x)$. $\{X_t\}$ can be generated by drawing samples with support of $\pi(x)$ in the correct proportions. The most frequently used way of generating $\{X_t\}$ is called Markov Chain, and it draws samples with $\pi(x)$ as its stationary distribution.

2.2.3 Markov Chains

We continue by covering Gilks, §1.3.2. Suppose that we generate a sequence of $t+1$ random variables $\{X_0, X_1, ..., X_t\}$ in such a way that at each time $t > 0$, the next state X_{t+1} is sampled from a distribution $P(X_{t+1}|X_t)$. It means that the state X_{t+1} is only dependent of the previous state X_t. The state X_{t+1} is not dependent of the history of the chain $\{X_0, ..., X_{t-1}\}$. This sequence is called Markov Chain and $P(X_{t+1}|X_t)$ is called a transition kernel of the chain. Generally, the kernel $P(X_{t+1}|X_t)$ can be time dependent, but we will assume it is time-independent.

Natural question may arise: how does the first sampled state X_0 affect X_t? In this scenario, we know nothing about intermediate states $\{X_1, ..., X_{t-1}\}$ and the question concerns distribution of X_t with given X_0: $P^{(t)}(X_t|X_0)$. Due to the regularity conditions, the chain will eventually "forget" its initial state and $P^{(t)}(X_t|X_0)$ will converge to a unique stationary distribution $\phi(x)$. After some number of samples, as time t increases, the samples $\{X_t\}$ will increasingly look like dependent samples from $\phi(x)$.

This means, that we need to discard with m samples, to be sure that $\{X_{m+1}, ...\}$ are samples drawn from $\phi(x)$ distribution. These first m samples are referred to as burn-in or equilibrium samples. After burn-in stage, the next samples $\{X_{m+1}, ...\}$ can be used in Monte Carlo computation of value $\mathbb{E}[f(X)]$, and the estimator is

$$\mathbb{E}[f(X)] = \langle f(X) \rangle = \frac{1}{n-m} \sum_{t=m+1}^{n} f(X_t). \tag{2.41}$$

2.2.4 Monte Carlo Error Estimate

According to the article by U. S. Fjordholm [20], the measure of Monte Carlo error is the root mean square error

$$\varepsilon_M = \sqrt{\mathbb{E}\left[(\mathbb{E}[X] - I_M)^2\right]},$$

where $I_M = \frac{1}{n}\sum_{t=1}^{n} f(X_t)$ is the Monte Carlo estimation. It can be show that

$$\varepsilon_M^2 = \frac{\mathbb{E}[X^2] - \mathbb{E}[X]^2}{M}$$

and

$$\varepsilon_M = \frac{\sigma[X]}{\sqrt{M}}.$$

Here $\sigma[X] = \sqrt{\mathrm{Var[X]}}$ is the standard deviation. Since the standard deviation is a constant, the root mean square error of Monte Carlo estimator scales as $M^{-1/2}$ - error decreases with increase of number of Monte Carlo samples.

2.2.5 Metropolis-Hastings Algorithm

Equation (2.41) provides a recipe to compute the Monte Carlo estimator $\langle f \rangle$, drawing samples from the unitary distribution $\phi(x)$. Metropolis-Hastings algorithm makes sure that the stationary distribution $\phi(x)$ is precisely the distribution of interest $\pi(x)$.

"For the Metropolis-Hastings algorithm, at each time step t, the next state X_{t+1} is chosen by first sampling a candidate point Y from a proposal distribution $q(\cdot|X_t)$, Gilks (§1.3.3).". Proposal distribution q can depend on the current state X_t and it is frequently chosen as a normal distribution $N(0,1)$, but it can have any form. The candidate state Y is accepted with probability $\alpha(X_t, Y)$:

$$\alpha(X_t, Y) = \min\left(1, \frac{\pi(Y)q(X|Y)}{\pi(X)q(Y|X)}\right).$$

If the step is accepted, the next state is $X_{t+1} = Y$. If the step is rejected, the chain does not move, meaning $X_{t+1} = X_t$.

Thus, the Metropolis-Hastings algorithm is presented in the Algorithm 1.

Algorithm 1 Metropolis–Hastings step

```
 1: procedure MHSTEP($X_t, \pi, q$)
 2:     Sample point $Y$ from $q$;
 3:     Sample a uniform random variable $u \sim U(0,1)$;
 4:     Compute acceptance ratio
            $\alpha(X_t, Y) \leftarrow \min\left(1, \frac{\pi(Y)q(X|Y)}{\pi(X)q(Y|X)}\right);$
 5:     if $u \leq \alpha$ then
 6:         $X_{t+1} \leftarrow Y$;
 7:     else
 8:         $X_{t+1} \leftarrow X_t$;
 9:     end if
10:     return $X_{t+1}$.
11: end procedure
```

2.2.6 Generation of Random Numbers

In order to run a Metropolis-Hastings algorithm, we need to have a way of generating random numbers. If a person is asked to pick a number between 1 and 10 - this answer would represent a random number. However, we can not ask a computer to pick a random number. Computer is a machine, that performs arithmetic operations, it can not pick a purely random number. Therefore, scientists developed pseudo random number generation algorithms, that generates pseudo random numbers.

"Random Number Generation and Monte Carlo Methods" [21] by James E. Gentle has a great description of pseudo random number generation algorithms. We fill discuss the very basics of random number generation according to Gentle's §1.1 and §1.2.

The standard methods of random numbers generations are based on the modular arithmetic. The basic operation in modular arithmetic is equivalence modulo m, where m is an integer. Two numbers are said to be equal to modulo m, if their difference is divisible by m and it is and integer:

$$a = b \bmod m.$$

For example, 7 and 18 are modulo 11, and 7.08 with 6.08 are modulo 1. Note, that a and b can be real numbers, while m has to be an integer.

One of the simplest random number generation algorithms is a called Simple Linear Congruential Generator. It has a form

$$x_i = (ax_{i-1} + c) \bmod m$$

with $0 \leq x_i < m$. a and c are constant factors, that define a Simple Linear Congruential Generator. Although this simple way of producing random numbers is not very accurate for a long sequences of numbers, it is used in the more advanced algorithms.

One feature of pseudo random number generation, is that the sequences is generally defined by some number of constants. The sequences with the same constants are, obviously, the same. These parameters that defines a sequence are called a seed. It allows to generate the same sequences of pseudo random numbers, which is very helpful while developing a code, since the computation that include random numbers can be recreated.

2.2.7 Direct Sampling & Importance Sampling

While discussing Metropolis-Hastings algorithm, it was stated that the random variable is sampled from uniform distribution. In this section, we will follow F. Becca's [22] approach in §3.3 and §3.4 to discuss concept of direct sampling and importance sampling.

Consider that we want to estimate value of π stochastically. In this case, we would draw a uniform circle (with a radius $r = 1$) and square that contains it, see Figure 2.1. Then we can randomly "shoot bullets" inside the square, counting every trial. By

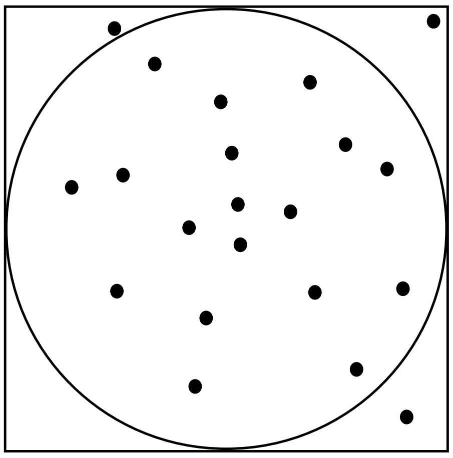


Figure 2.1: Simulation of 20 random shots inside the circle. The scheme is inspired by Firuge 3.1 in F. Becca's book.

keeping track of trials and hits, we can perform a Monte Carlo integration to estimate π:

$$\frac{\int_{\text{circle}} dxdy}{\int_{\text{square}} dxdy} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} = \int_{\text{square}} f(x,y)\mathcal{P}(x,y)dxdy,$$

where

$$f(x) = \begin{cases} 1, & \text{if } (x,y) \text{ is inside the circle;} \\ 0 & \text{otherwise.} \end{cases}$$

and $\mathcal{P}(x,y) = \left(\int_{\text{square}} dxdy\right)^{-1}$ represents probability function. In this analogy, "shooting bullets" randomly means drawing uniform random numbers x and y. If $x^2 + y^2 < 1$, then it is counted as a hit, and doe not count as a hit otherwise. Thus, π can be estimated as

$$\frac{\pi}{4} = \frac{1}{N} \sum f(x_i, y_i).$$

In this case, direct uniform sampling works perfectly fine. However, if $f(x)$ is a sharply peaked function, the uniform drawing of random numbers is not efficient. The reason is that we would compute many points x_i that almost do not give contribution to the integral. In this case a technique known as importance sampling needs to be implemented.

Consider a stochastic computation of the integral

$$I = \int_a^b F(x)dx,$$

where function $F(x)$ is peaked at the point c, so that $a < c < b$. Whenever we approximately know the location of the region where $F(x)$ is sizable, we can define probability density $\mathcal{P}$ on the interval $[a, b]$ which is also seizable in the same region and small everywhere else. Then, integral can be rewritten as

$$I = \int_a^b \frac{F(x)}{\mathcal{P}(x)} \mathcal{P}(x)dx.$$

And this integral can be estimated as

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{F(x_i)}{\mathcal{P}(x_i)}.$$

If the function $\mathcal{P}(x_i)$ can be chosen to be close enough to $F(x)$, then statistical errors are drastically reduced.

2.2.8 Variational Monte Carlo Revisited

As comes from its name, Variational Monte Carlo is a combination of two techniques – variational principle, that provides a recipe to estimate GS energy and Monte Carlo, which is used to compute the integral estimation of GS energy. Let us revisit Section 2.1.21, but this time in the different manner.

Variational principle states that given wave function ansatz ψ_T, GS energy estimation is

$$E_T = \frac{\langle \psi_T | \hat{H} | \psi_T \rangle}{\langle \psi_T | \psi_T \rangle} \geq E_{GS}$$

According to J. Gubernatis' book about quantum Monte Carlo methods [23] (§9.1), the goal of VMC method is to make E_T as close as possible to E_{GS}.

Suppose that the state $|\psi_T\rangle$ is a good estimate of some eigenstate of a Hamiltonian. This eigenstate can be or not be the ground state. In general, the trial state ψ_T is not an eigenstate of a Hamiltonian, meaning we can not use the standard $\hat{H} |\psi_T\rangle = E_T |\psi_T\rangle$ relationship. However, we can define the residual state

$$|\phi\rangle = (\hat{H} - E_T) |\psi_T\rangle,$$

which measures how far $|\psi_T\rangle$ is from the true eigenstate associated with the energy E_T. Obviously, residual state is equal to zero if the trial state $|\psi_T\rangle$ is one of the eigensatates of $\hat{H}$. If $|\psi_T\rangle$ is not an eigenstate, we want to find a value E_T that minimizes

$$\langle \phi | \phi \rangle = \left\langle (\hat{H} - E_T)\psi_T \middle| (\hat{H} - E_T)\psi_T \right\rangle.$$

Expanding the equation above yields

$$\langle\phi|\phi\rangle = \left\langle \hat{H}\psi_T \middle| \hat{H}\psi_T \right\rangle - E_t \left\langle \hat{H}\psi_T \middle| \psi_T \right\rangle - E_T \left\langle \psi_T \middle| \hat{H}\psi_T \right\rangle + E_T^2.$$

Define $\lambda = \left\langle \psi_T \middle| \hat{H}\psi_T \right\rangle / \langle\psi_T|\psi_T\rangle$, so that equation takes a form

$$\langle\phi|\phi\rangle = \left\langle \hat{H}\psi_T \middle| \hat{H}\psi_T \right\rangle + \langle\psi_T|\psi_T\rangle \left[(E_T - \lambda)^2 - \lambda^2\right].$$

We want to minimize this equation, and since both inner products are positive, it can be achieved by setting

$$\lambda = E_T = \frac{\langle\psi_T| \hat{H} |\psi_T\rangle}{\langle\psi_T|\psi_T\rangle}.$$

The ration on the right hand side is called Rayleigh's quotient. We have shown, that lower bound of the Rayleigh's quotient is the GS energy. Upper bound is for it is the highest energy eigenstate E_N.

If the trial state is accurate to $\mathcal{O}(\varepsilon)$, then the trial energy is accurate up to $\mathcal{O}(\varepsilon^2)$.

The most general way to compute E_T is to define a configurational basis $\{|C\rangle\}$. This basis is orthonormal, meaning $\sum_C |C\rangle\langle C| = I$ and $\langle C|C'\rangle = \delta_{CC'}$. Using this basis, trial energy becomes

$$E_T = \frac{\sum_C \langle\psi_T|C\rangle \langle C| \hat{H} |\psi_T\rangle}{\sum_C |\langle C|\psi_T\rangle|^2}.$$

This can be rewritten as

$$E_T = \sum_C E(C)P(C),$$

where

$$E(C) = \frac{\langle C| \hat{H} |\psi_T\rangle}{\langle C|\psi_T\rangle} \text{ and } P(C) = \frac{|\langle C|\psi_T\rangle|^2}{\sum_C |\langle C|\psi_T\rangle|^2}$$

Notice, that this is a discrete case. This case becomes the one we discussed in Section 2.1.21 in the continuous limit.

Trial energy can be approximated using Monte Carlo integration:

$$E_T \approx \frac{1}{N}\sum_{i=1}^{N} E(C_i).$$

Finally, we can use Metropolis-Hasting algorithm to sample C_i from $P(C)$. This algorithm requires proposal probability $T(C'|C)$, and the acceptance probability becomes

$$\alpha(C, C') = \min\left(1, \frac{P(C')T(C|C')}{P(C')T(C|C')}\right) = \min\left(1, \frac{|\langle C'|\psi_T\rangle|^2 T(C|C')}{|\langle C|\psi_T\rangle|^2 T(C|C')}\right)$$

2.2.9 Gradient Descent

In the research we will need to optimize variational parameters of a trial wave function ansatz, i.e. find parameters that minimize equation (2.29). It can be shown that E_T is a convex function, and different methods can be used to minimize it. One of the standard methods is a Gradient Descent.

Following [24] (§9.1, 9.2, 9.3), Gradient Descent is a method to solve the minimization problem

$$\text{minimize } f(x).$$

Here, $f(x) : \mathbb{R}^N \to \mathbb{R}$ is a convex function. We suppose that there exists a solution x^* to this problem, so that $p^* = f(x^*)$ is the optimal value.

A necessary and sufficient condition for point x^* to be optimal is

$$\nabla f(x^*) = 0.$$

For general f, equation can not be solved analytically, and iterative algorithm needs to implemented. By iterative algorithm we mean sequence $x^{(0)}$, $x^{(1)}$, ... with $f(x^{(k)}) \to p^*$ as $k \to \infty$. The algorithm requires a stopping criterion η and it is terminated when $f(x^{(k)}) - p^* < \eta$.

Descent family of methods produce a minimizing sequence

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)}.$$

Here, $t^{(k)}$ is a step, $\Delta x^{(k)}$ is a step size and $k = 0, 1, ...$ is an iteration number. For this family of methods $f(x^{(k+1)}) < f(x^{(k)})$.

Gradient Descent is a member of Descent family with $\Delta x = -\nabla f(x)$. Minimizing sequence is then

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)}).$$

Gradient Descent procedure is presented in the Algorithm 2. $\|x\|_2$ represent L2 norm of a vector x defined by $\|x\|_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}$. Function GD has only one argument – starting position x_0 and returns x when $\nabla \|f(x)\|_2 < \eta$.

Algorithm 2 Gradient Descent

```
1: procedure GD($x_0$)
2:     Choose time step $t$;
3:     $x \leftarrow x_0$;
4:     while $||\nabla f(x)||_2 > \eta$ do
5:         $\Delta x \leftarrow -\nabla f(x)$;
6:         $x \leftarrow x + t\Delta x$;
7:     end while
8:     return $x$.
9: end procedure
```

Chapter 3

Methods and implementations

3.1 Single-Particle Orbitals

Wave function that describes n-th single-particle orbital is given by equation (2.19),

$$\psi_n(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} \frac{1}{\sqrt{2^n n!}} He_n(\xi) e^{-\xi^2/2},$$

where

$$\xi = \sqrt{\frac{m\omega}{\hbar}} x.$$

Energy of n-th single-particle orbital is

$$E_n = \left(n + \frac{1}{2}\right)\hbar\omega.$$

As we will see in the following sections, VMC only uses wave-function ratios and any constant prefactors in ψ_n cancel out and can be dropped. Therefore single-particle orbital wave function simplifies to

$$\psi_n(x) = He_n(\xi) e^{-\xi^2/2}.$$

As was discussed earlier, we introduce a variational exponent. Starting from the unvaried form $e^{-\xi^2/2}$, we promote the exponent to $e^{-\alpha\xi^2}$ to introduce our variational freedom,

$$\psi_n(x) = He_n(\xi) e^{-\alpha\xi^2}$$

We will work in HO units for non-interacting cases and switch to atomic units when we include interactions. In those unit systems, the single-particle orbitals and energies take the following simple forms:

- HO units set $\hbar = m = \omega = 1$, so that single-particle orbital wave function becomes

  $$\psi_n(x) = He_n(x) e^{-\alpha x^2}. \tag{3.1}$$

  Energy E_n, which is an eigenfunction of non-interacting harmonic problem in these units becomes

  $$E_n = n + \frac{1}{2}. \tag{3.2}$$

- In atomic units $\hbar = m = 1$, but ω does not vanish. Single-particle orbital wave function takes the form

  $$\psi_n(x) = He_n\left(\sqrt{\omega}x\right) e^{-\alpha\omega x^2} \tag{3.3}$$

  and energy is equal to

  $$E_n = \left(n + \frac{1}{2}\right)\omega. \tag{3.4}$$

In Appendix 6.2 we define single-particle orbital wave functions up $n = 2$ and derive their gradients, Laplacians and derivatives with respect to α.

3.2 Bosonic Trial Wave Function Ansatz

In Section 2.1.16 and 2.1.17 we discussed a general way of constructing many-body wave functions from singe-body wave function for fermions and bosons. Two factors impact the form of wave function: variational principle and harmonic potential. Variational principle implies that trial GS energy, obtained by using trial wave function needs to be minimized. Therefore, trial wave function needs to contain some number of variational parameters. By optimizing these parameters, trial GS energy is minimized. As we discussed earlier, trial wave function can be chosen arbitrary, but the closer it is to the true wave function – the better. Since particles are trapped in Harmonic potential, we can assume that singe-body wave functions are solutions to the quantum HO problem - single-particle orbitals.

Consider bosonic trial wave function ansatz first. Recall that for BEC general wave function is given by equation (2.26),

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) = \phi_a(\mathbf{x}_1)\phi_a(\mathbf{x}_2)\cdots\phi_a(\mathbf{x}_N).$$

Since bosons does not follow Pauli Principle, they can occupy the same state. For BEC, all particles are in the ground state. Single-body wave function that describe one particles in the harmonic potential is given by equations (2.19) and (2.21). GS has a Primal quantum number $n = 0$, and therefore variational wave function that describes this state is given by (6.7)

$$\psi_1(\mathbf{r}) = \psi_{n_x=0}(x)\psi_{n_y=0}(y)\psi_{n_z=0}(z) = e^{-\alpha r^2}.$$

Many-body wave function is a function of spin as well. The spin dictated a form of the wave function. Since spin is not present in the Hamiltonian, for bosons it can be just left out, meaning $\phi_a(\mathbf{x}) \to \psi_1(\mathbf{r})$. Therefore, trial wave function ansatz for bosons is

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_N) = \prod_{i=1}^{N} \psi_1(\mathbf{r}_i) = e^{-\alpha \sum_{i=1}^{N} r_i^2}. \tag{3.5}$$

Expression in this form applies in any spatial dimension. Here $\mathbf{r} = (x, y, z)$ in 3D so that $r^2 = x^2 + y^2 + z^2$; in 2D $\mathbf{r} = (x, y)$ and $r^2 = x^2 + y^2$; and in 1D $\mathbf{r} = (x)$ so $r^2 = x^2$.

3.3 Fermionic Trial Wave Function Ansatz

Unlike bosons, fermions follow Pauli principle. We will focus on the 2D particles trapped in harmonic potential, because in 2D Slater determinant can be simplified. 2D quantum state in harmonic trap is defined by two quantum numbers – primal quantum numbers n_x and n_y. Particles, trapped in such potential have an intrinsic quantum number – projection of the spin on z-axis m_z. Therefore, fermion trapped in quantum potential is described by quantum numbers n_x, n_y and m_z. Since we are interested in GS energy, we need to construct a trial wave function that describes many-body state with the lowest energy.

We will consider only "closed-shell" states. Closed shell state means that all vacant places up to $n = n_x + n_y$ are occupied. First shell has $n = 0$. This is possible only when $n_x = n_y = 0$. First closed shell has $n = 0$ and two fermions with opposite spin projections occupy all vacant places. No more fermions can be added to this state. This case is schematically presented on Figure 3.1.

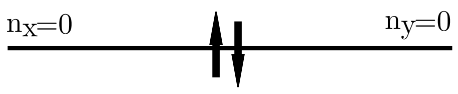


Figure 3.1: Schematic representation of first closed shell. Here $\uparrow$ represent a fermion with $m_z = \hbar/2$ and $\downarrow$ represents a fermion with $m_z = -\hbar/2$.

Next shell has $n = 1$. This shell has two fermions in the state $n = 0$, two fermions in the state $(n_x = 1, n_y = 0)$ and two fermions in the state $(n_x = 0, n_y = 1)$. In total, second closed shell has 6 fermions. Second shell is schematically depicted on Figure 3.2.

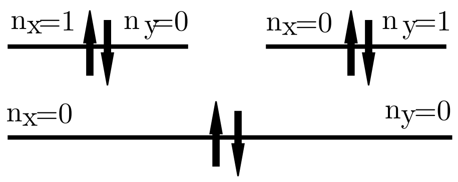


Figure 3.2: Schematic representation of second closed shell. Here $\uparrow$ represent a fermion with $m_z = \hbar/2$ and $\downarrow$ represents a fermion with $m_z = -\hbar/2$.

Third shell has $n = 2$. There are six fermions that occupy states up to $n = 1$. It has additional two fermions in the state $(n_x = 2, n_y = 0)$, two more fermions in the state $(n_x = 0, n_y = 2)$ and last two fermions in the state $(n_x = 1, n_y = 1)$. Overall, third closed shell has twelve fermions. Schematic representation of third shell is on Figure 3.3

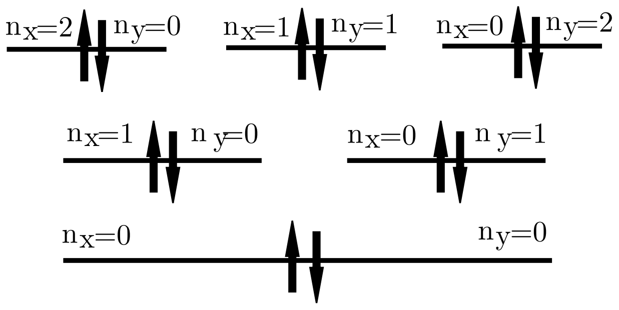


Figure 3.3: Schematic representation of third closed shell. Here $\uparrow$ represent a fermion with $m_z = \hbar/2$ and $\downarrow$ represents a fermion with $m_z = -\hbar/2$.

We will limit ourselves with first three closed shells, meaning that we will consider system with two, six and twelve fermions.

Generally, wave function that represents such states is a Slater determinant of the form (2.25). However, for systems like ours it can be simplified. J. W. Moskowitz and M.H Kalos were one of the first scientists that proposed the idea that Slater determinant can be split into parts in their article [25] (Gubernatis follows this approach in §9.2.1),

$$\Psi_T = D_{\uparrow}(\mathbf{r}_1, ..., \mathbf{r}_{N/2}) \cdot D_{\downarrow}(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N). \tag{3.6}$$

Here,

$$D_{\uparrow}(\mathbf{r}_1, ..., \mathbf{r}_{N/2}) = \begin{vmatrix} \phi_1(\mathbf{r}_1, \uparrow) & \phi_2(\mathbf{r}_1, \uparrow) & \cdots & \phi_{N/2}(\mathbf{r}_1, \uparrow) \\ \phi_1(\mathbf{r}_2, \uparrow) & \phi_2(\mathbf{r}_2, \uparrow) & \cdots & \phi_{N/2}(\mathbf{r}_2, \uparrow) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{r}_{N/2}, \uparrow) & \phi_2(\mathbf{r}_{N/2}, \uparrow) & \cdots & \phi_{N/2}(\mathbf{r}_{N/2}, \uparrow) \end{vmatrix}$$

and

$$D_\downarrow(\mathbf{r}_{N/2+1},...,\mathbf{r}_N) = \begin{vmatrix} \phi_1(\mathbf{r}_{N/2+1},\downarrow) & \phi_2(\mathbf{r}_{N/2+1},\downarrow) & \cdots & \phi_{N/2}(\mathbf{r}_{N/2+1},\downarrow) \\ \phi_1(\mathbf{r}_{N/2+2},\downarrow) & \phi_2(\mathbf{r}_{N/2+2},\downarrow) & \cdots & \phi_{N/2}(\mathbf{r}_{N/2+2},\downarrow) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{r}_N,\downarrow) & \phi_2(\mathbf{r}_N,\downarrow) & \cdots & \phi_{N/2}(\mathbf{r}_N,\downarrow) \end{vmatrix}.$$

In general, normalization constant is needed. But VMC machinery includes only ratios of wave functions, and therefore we did not include the constant.

In definitions of $D_\uparrow$ and $D_\downarrow$ we labeled all particles in spin-up state as $\{1, 2, ..., N/2\}$ and all particles in spin down state as $\{N/2+1, N/2+2, ..., N\}$. It is just a matter of definition. We could have chosen particles $1 - N/2$ as spin-down particles.

The benefit of this approach is that we need to compute two $N/2 \times N/2$ determinants instead of one $N \times N$ determinant, which is less computationally expensive.

When the split of original determinant is done, we can replace wave functions that include spin with the ones that do not include spin, meaning $\phi_i(\mathbf{r}_j, \xi) \to \psi_i(\mathbf{r}_j)$ and determinants now have a form

$$D_\uparrow(\mathbf{r}_1,...,\mathbf{r}_{N/2}) = \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \cdots & \psi_{N/2}(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_{N/2}(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_{N/2}) & \psi_2(\mathbf{r}_{N/2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2}) \end{vmatrix} \tag{3.7}$$

and

$$D_\downarrow(\mathbf{r}_{N/2+1},...,\mathbf{r}_N) = \begin{vmatrix} \psi_1(\mathbf{r}_{N/2+1}) & \psi_2(\mathbf{r}_{N/2+1}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+1}) \\ \psi_1(\mathbf{r}_{N/2+2}) & \psi_2(\mathbf{r}_{N/2+2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+2}) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_N) & \psi_2(\mathbf{r}_N) & \cdots & \psi_{N/2}(\mathbf{r}_N) \end{vmatrix}. \tag{3.8}$$

Let us now consider first three closed shell systems. First system has only two particles. In this case spin-up and spin down Slater determinants simplify to

$$D_\uparrow(\mathbf{r}_1) = |\psi_1(\mathbf{r}_1)| \quad \text{and} \quad D_\downarrow(\mathbf{r}_2) = |\psi_1(\mathbf{r}_2)|.$$

Here, $\psi_1(\mathbf{r})$ is given by formula (6.13):

$$\psi_1(\mathbf{r}) = \psi_{n_x=0}(x)\psi_{n_y=0}(y) = e^{-\alpha r^2}.$$

Trial wave function takes a form

$$\Psi_T(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1^2+r_2^2)}. \tag{3.9}$$

Notice, that this wave function is equivalent to bosonic trial wave function, given by equation (3.5) for $N = 2$. These wave functions describe fermions and bosons that occupy lowest shell with $n = 0$, and it is not a surprise that trial wave functions have exactly the same form. Note, that this wave function is symmetrical under coordinate exchange of two particles. Only wave function that contains spin degrees of freedom need to anti-symmetric. In this case everything is alright, because spin wave function is antisymmetric.

For second closed shell trial wave function is constructed from spin-up and spin-down Slater determinant. Wave function $\psi_1(x, y)$ is the same as for two particles system. Wave functions $\psi_2(x, y)$ and $\psi_3(x, y)$ are given by formulae (6.12) and (6.7):

$$\psi_2(x, y) = \psi_{n_x=1}(x)\psi_{n_y=0}(y) = xe^{-\alpha r^2};$$

$$\psi_3(x,y) = \psi_{n_x=0}(x)\psi_{n_y=1}(y) = ye^{-\alpha r^2}.$$

Trial wave function for the third closed shell is also a product of spin-up and spin-down Slater determinants. Wave function $1-3$ are as before, and wave functions $4-6$ are given by (6.18), (6.19) and (6.20):

$$\psi_4(x) = \psi_{n_x=2}(x)\psi_{n_y=0}(y) = (x^2-1)e^{-\alpha r^2};$$

$$\psi_5(x) = \psi_{n_x=0}(x)\psi_{n_y=2}(y) = (y^2-1)e^{-\alpha r^2}$$

and

$$\psi_6(x) = \psi_{n_x=1}(x)\psi_{n_y=1}(y) = xye^{-\alpha r^2}.$$

Ansatzes that we have construed describe two, six and twelve non-interacting fermions. In order to take into account that fermions interact, these ansatzes need to be modified with Jastrow factor (2.27) or Pade-Jastrow factor (2.28),

$$\Psi_J = D_\uparrow D_\downarrow J \quad \text{and} \quad \Psi_P = D_\uparrow D_\downarrow P. \tag{3.10}$$

With our fermionic and bosonic trial wave functions in hand, we can now apply the VMC method to estimate GS energies. Th next step is to discuss VMC algorithm implementation.

3.4 VMC Algorithm

Recall that VMC machinery is designed to estimate trial GS energy (2.31)

$$E_T \approx \frac{1}{N}\sum_{i=1}^{N} E_L(\mathbf{R}_i, \alpha).$$

Let us go though the VMC algorithm step by step. First step is to choose a system we will study. On practice it means to choose parameters of the system: number of particles, number of dimension, Hamiltonian of the system and a trial wave function with variational parameters. Variational parameters can be initialized with some values in range $(0,1]$. Quality of initial variational parameters determine how much VMC will be needed to find optimal variational parameters and optimal GS energy. For now we will not vary oscillator frequency. With all the parameters set, we can start performing VMC algorithm.

In order to start the algorithm, we need to set position for all particles. Recall that VMC has a burn-in time, meaning that first m VMC iteration will be discarded, so the initial position of the particles does not really matter. Popular choice is to sample particles position form uniform distribution $U(0,1)$. This means that every particles will have a random position in the interval $[0,1)$ in every dimension.

When the particles have a starting position, VMC procedure can be initialized. The general idea is pretty simple: compute local energy with current position, accept or reject the state, update positions if the step is accepted, repeat. In theory, nothing complicated. On practice, computing local energy is a very complicated task due to complexity of trial wave functions and Hamiltonian.

New particles position is also achieved by using random numbers $\mathbf{u} = (u_x, u_y, u_z)$ drawn from uniform distribution $U(0,1)$. Let $\mathbf{r}^{(t)} = (x^{(t)}, y^{(t)}, z^{(t)})$ denote position of one of the particles on iteration t. New position of the particle $\mathbf{r}^{(t+1)} = (x^{(t+1)}, y^{(t+1)}, z^{(t+1)})$ can be computed as

$$x^{(t+1)} = x^{(t)} + (u_x - 0.5h); \quad y^{(t+1)} = y^{(t)} + (u_y - 0.5h); \quad z^{(t+1)} = z^{(t)} + (u_z - 0.5h).$$

Here h is a step size, which determines how far particles are moved every iteration. Since $u \in [0, 1)$, $u - 0.5h$ can be positive or negative number, so that position can increase or decrease. On the long run, each particle is just moving around the origin $(0, 0, 0)$. The same procedure is applied for all particles on every VMC iteration. New random number is generated for each dimension of each particle. Every iteration, either a set of all random numbers or a set of all positions needs to be saved, so that if the step is rejected, old positions can be restored.

Let Ψ_{t+1} denote proposed wave function with new coordinates and Ψ_t denote previous step wave function. The step is accepted if a random number u sampled from the uniform distribution $U(0, 1)$ is smaller that ratio of new state probability density and old state probability density,

$$u < \min\left(1, \quad \frac{|\Psi^{(t+1)}|^2}{|\Psi^{(t)}|^2}\right).$$

The idea of this criterion is that we want the system to evolve not only towards state with the highest probability, but sometimes to the state towards states with lower probability. If the system would always evolve into the state with the highest probability, it would reach it after some iterations and just sit there. Such criterion makes sure that the samples will be from different states, but with guidance of probability distribution function. Exactly what is needed to estimate GS energy.

During every iteration we compute and collect values of local energy and some other variables, and when VMC iterations are finished, we can compute sample mean value to get GS energy estimation. Afterwards, Gradient Descent is performed to compute more optimal variational parameters. The procedure stops when $L2$ norm of gradient vector of variational parameters becomes less than stopping criterion.

There is one variational parameter α in non-interacting part of wave function ansatz. This variational parameter is "responsible" for non-interacting problem. Once its optimal value is found, it can be used in further interacting computations. Jastrow factor J has $p = N(N-1)/2$ number of variational parameters β_{ij} with N being number of particles. All these parameters are being updated using GD after each full VMC cycle. When $N = 12$ there are 66 variational parameters, and optimizing all of them at the same time is quite challenging. Therefore we do not expect to achieve a great accuracy using the Jastrow factor. On the contrary, Pade-Jastrow factor has only one variational parameter β. It is much simpler to optimize one parameter at a time, compared to 66 variational parameters. However, we are trying to describe twelve particles with only one variational parameter, and therefore accuracy can be not great either.

Let us sum it up and write VMC burn-in algorithm in a schematic way. We will discuss implementation of VMC for model systems in respective sections. Burn-in algorithm is presented in Algorithm 3. Function BIVMC has four arguments: number of particles N, number of dimensions d, step size h and number of burn-in iterations. This function returns position matrix $\mathbf{X}$ after m burn-in iterations. The burn-in phase consists of running the first m iterations without recording any measurements, allowing the chain to converge to its stationary distribution so that subsequent samples are effectively independent draws.

Algorithm 3 Burn-in VMC Algorithm

```
 1: procedure BIVMC(N, d, h, m)
 2:     Initialize uniform random position of all particles as a 2D matrix $\mathbf{X}$ of size $N \times d$;
 3:     for (int i = 0; i < m; i++) do
 4:         Compute old state probability density $\text{rho}_{\text{old}} = |\Psi_T(\mathbf{X})|^2$;
 5:         Change position of all particle, so that new position matrix is $\mathbf{X}_n$;
 6:         Compute new state probability density $\text{rho}_{\text{new}} = |\Psi_T(\mathbf{X}_n)|^2$;
 7:         Compute acceptance rate $\alpha = \min\left(1, \frac{\text{rho}_{\text{new}}}{\text{rho}_{\text{old}}}\right)$;
 8:         Generate uniform random number $u$;
 9:         if $u < \alpha$ then
10:             Step is accepted, $\mathbf{X} \leftarrow \mathbf{X}_n$;
11:         else
12:             Step is rejected, $\mathbf{X}$ stays the same;
13:         end if
14:     end for
15:     return $\mathbf{X}$.
16: end procedure
```

3.5 Gradient Descent

As we discussed in Section 2.2.9, Gradient Descent is used to find optimal values of variational parameters, that minimize trial GS energy. Variational parameters are a part of non-interacting bosonic and fermionic trial wave functions, Jastrow factor and Pade-Jastrow factor. Non-interacting trial wave functions (3.2), (3.3) contain one variational parameter α, Pade-Jastrow factor (2.39) has one variational parameter β. Jastrow factor (2.40), on the opposite hand, has $N/2N(N-1)$ variational parameters β_{ij}. Let us derive GD update rules for all these wave functions.

After VMC is performed, we obtain VMC estimation of expectation value of local energy (2.31)

$$E_0 \approx \frac{1}{N}\sum_{i=1}^{N} E_L(\mathbf{R}_i, \alpha).$$

Analytical expression for the GS energy (2.29) is

$$E_0 = \langle E_L \rangle = \frac{\langle \Psi_T(\alpha)| E_L |\Psi_T(\alpha)\rangle}{\langle \Psi_T | \Psi_T \rangle}.$$

Trail wave functions contain variational parameters and they are used to compute local energy. Therefore local energy is also a function of variational parameters. GD algorithm 2 requires gradient vector of function that is optimized. In our case, this function is expectation value of local energy. That is, to optimize variational parameter α we need an expression for $d\langle E_L\rangle/d\alpha$. In Appendix 6.3 we derive this analytical expression, and it is given by (6.27):

$$\frac{\partial\langle E_L\rangle}{\partial\alpha} = 2\left(\left\langle \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)} E_L(\alpha)\right\rangle - \left\langle \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)}\right\rangle \langle E_L(\alpha)\rangle\right). \tag{3.11}$$

Here

$$\bar{\Psi}_T = \frac{\partial \Psi_T}{\partial \alpha}. \tag{3.12}$$

GD update rule for the variational parameter α is then

$$\alpha^{(k+1)} = \alpha^{(k)} - t^{(k)} \left(\frac{\partial\langle E_L\rangle}{\partial\alpha}\right)^{(k)}.$$

For simplicity, step $t^{(k)}$ can be chosen as a constant value, that can be tuned. Typical values for t are from 10^{-4} to 10^{-1}.

To optimize α we need to be able to compute equation (3.17) for a given trial wave function. This expression contains three different expectation values. The only one familiar term is expectation value of local energy, which represents VMC estimation of GS energy. To compute it, we sample local energies during every VMC iteration. Other two terms can be obtained exactly the same way – we need to sample and collect

$$O_1 = \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)} \quad \text{and} \quad O_2 = \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)} E_L(\alpha) \tag{3.13}$$

every VMC iteration. Variables O_1 and O_2 are referred to as observables or scores. We will discuss how to compute local energy and scores in respective sections for all model systems. Now we want to focus on deriving general GD update rules for all variational parameters.

When we consider interacting fermionic wave function, local energy becomes function of variational parameters α and β. Variational parameter α is related to non-interacting problem. Once its optimal value is found, we switch to interacting problem, where we need to optimize β. Pade-Jastrow factor contains only one variational parameter β, and therefore GD derivative for it remains the same:

$$\frac{\partial\langle E_L\rangle}{\partial\beta} = 2\left(\left\langle \frac{\bar{\Psi}_T(\alpha,\beta)}{\Psi_T(\alpha,\beta)} E_L(\beta)\right\rangle - \left\langle \frac{\bar{\Psi}_T(\alpha,\beta)}{\Psi_T(\alpha,\beta)}\right\rangle \langle E_L(\beta)\rangle\right). \tag{3.14}$$

Therefore, GD optimization rule for trial wave function with Pade-Jastrow factor is

$$\beta^{(k+1)} = \beta^{(k)} - t^{(k)} \left(\frac{d\langle E_L\rangle}{d\beta}\right)^{(k)}.$$

Finally, for trial wave function with Jastrow factor, we need to optimize $p = N/2(N-1)$ variational parameters β_{ij}. These variational parameters can be represented by vector $\beta = (\beta_0, ..., \beta_{p-1})$. We use 0 as a starting index instead of 1, because in C++ fist element of a vector has index zero. To store betas in vector in correct order, we need to transform 2D inex (i, j) into 1D index k. Since j is always bigger than i, in can be done by using

$$k = \frac{i}{2}(2N - i - 1) + (j - i - 1). \tag{3.15}$$

To show that this is a correct relation, consider a system of $N = 3$ particles. For such system, vector β contains $p = 3$ components. 2D indices for the components are $(0, 1)$, $(0, 2)$ and $(1, 2)$. We want for vector β to have a structure

$$\beta = (\beta_{01}, \beta_{02}, \beta_{12}) = (\beta_0, \beta_1, \beta_2).$$

Compute 1D index k for $(i, j) = (0, 1)$:

$$k = \frac{1}{2} \cdot 0(2 \cdot 3 - 1 - 1) + (1 - 0 - 1) = 0;$$

for $(0, 2)$:

$$k = \frac{1}{2} \cdot 0(2 \cdot 3 - 1 - 1) + (2 - 0 - 1) = 1$$

and for $(1, 2)$:

$$k = \frac{1}{2} \cdot 1(2 \cdot 3 - 1 - 1) + (2 - 1 - 1) = 2.$$

Therefore, for $N = 3$ expression (3.19) translates 2D index into 1D index correctly.

For trial wave function with Pade-Jastrow factor, GD algorithm is

$$\beta^{(k+1)} = \beta^{(k)} - t^{(k)} \nabla_\beta \langle E_L \rangle^{(k)}.$$

Vector β represents vector of p variational parameters and

$$\nabla_\beta E_L = \left(\frac{\partial \langle E_L \rangle}{\partial \beta_0}, ..., \frac{\partial \langle E_L \rangle}{\partial \beta_{p-1}} \right).$$

Derivative of expectation value of local energy is give by expression (3.18):

$$\frac{\partial \langle E_L \rangle}{\partial \beta_m} = 2 \left(\left\langle \frac{\bar{\Psi}_T^{(m)}(\alpha, \beta)}{\Psi_T(\alpha, \beta)} E_L(\beta) \right\rangle - \left\langle \frac{\bar{\Psi}_T^{(m)}(\alpha, \beta)}{\Psi_T(\alpha, \beta)} \right\rangle \langle E_L(\beta) \rangle \right),$$

where

$$\bar{\Psi}_T^{(m)}(\alpha, \beta) = \frac{\partial \Psi_T(\alpha, \beta)}{\partial \beta_m}. \tag{3.16}$$

To summarize, GD provides an easy wave to minimize trial GS energy by optimizing variational parameters. To optimize variational parameters, E_L, O_1 and O_2 need to be sampled during VMC.

Let us now discuss how importance sampling can be implemented in our research.

3.6 Metropolis–Hastings Algorithm

This section follows the lecture notes of Morten Hjorth-Jensen[26] for FYS4411. Langevin dynamics and the Fokker–Planck equation are tools for simulating physical processes on a computer. They are closely related to Markov chains and probability theory. In one dimension, the Langevin equation takes the form

$$\frac{\partial x(t)}{\partial t} = D\, F(x) + \eta,$$

which describes how the position $x(t)$ changes over time. From this, we get the update rule for a small time step Δt:

$$x_{\text{new}} = x_{\text{old}} + D\, F(x)\, \Delta t + \eta. \tag{3.17}$$

Here, x_{new} is the new position and x_{old} is the old position. The constant D equals $1/2$ in atomic units, η is a uniform random variable, Δt is a tunable time step, and $F(x)$ is the quantum force. The quantum force follows from the Fokker–Planck equation and is given by

$$\mathbf{F} \;=\; 2\, \frac{\nabla \Psi_T}{\Psi_T}. \tag{3.18}$$

This force guides the random walkers toward regions of higher probability, improving the VMC sampling. When using this update rule, the Metropolis–Hastings acceptance probability becomes

$$\alpha = \min\left(1, \frac{G(x,y,\Delta t)|\Psi_T(y)|^2}{G(y,x,\Delta t)|\Psi_T(x)|^2}\right), \tag{3.19}$$

where G is the Green's function for the diffusion process. For harmonic potential it has a form

$$G(y,x,\Delta t) = C\exp\left(\frac{-(y-x-D\Delta t F(x))^2}{4D\Delta t}\right). \tag{3.20}$$

In his article [27], Hastings introduced importance sampling for Monte Carlo methods in 1970.

From equation (3.19) we can see that only Green's function ratio is needed to compute acceptance probability. Since we work in two dimensions mostly, Green's function ratio is given by

$$\frac{G(\mathbf{R}_{\text{old}},\mathbf{R}_{\text{new}},dt)}{G(\mathbf{R}_{\text{new}},\mathbf{R}_{\text{old}},dt)} = \frac{(\mathbf{R}_{\text{new}}-\mathbf{R}_{\text{old}}-D\mathbf{F}_{\text{old}}dt)^2-(\mathbf{R}_{\text{old}}-\mathbf{R}_{\text{new}}-D\mathbf{F}_{\text{new}}dt)^2}{4Ddt}. \tag{3.21}$$

3.7 Local energy, Observables & Quantum Force For Model Systems

In this section we will provide a recipe to compute local energy E_L and observables O_1, O_2 and quantum force $\mathbf{F}$ during VMC iterations. We will also state analytical solution for systems where they can be derived.

3.7.1 Non-Interacting Bosons

We consider a 3D system of N non-interacting bosons trapped in harmonic potential and cooled, so that all particles occupy lowest energy state. In HO units, Hamiltonian for this system is given by equations (2.35) and (2.37).

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{1}{2}\Delta_i + \frac{1}{2}r_i^2\right).$$

Trial wave function ansatz has one variational parameter α and it is given by (3.2),

$$\Psi_T = \exp\left(-\alpha\sum_{i=1}^{N} r_i^2\right).$$

For such system GS energy and optimal variational parameter $\tilde{\alpha}$ can be found analytically, see Appendix 6.4 for derivation. GS energy of the system as a function of variational parameter is (6.31),

$$E_0(\alpha) = \frac{3N}{2}\left(\alpha + \frac{1}{4\alpha}\right),$$

optimal variational parameter is (6.32),

$$\tilde{\alpha} = \frac{1}{2}$$

and GS energy (6.33) is equal to

$$E(\tilde{\alpha}) = \frac{3N}{2}.$$

This result can be generalized for d-dimensional system:

$$E_0(\alpha) = \frac{dN}{2}\left(\alpha + \frac{1}{4\alpha}\right); \quad \tilde{\alpha} = \frac{1}{2}; \quad E(\tilde{\alpha}) = \frac{dN}{2}. \tag{3.22}$$

Recall that wave function that describes 1D quantum particle in harmonic potential is given by (2.19). For $n = 0$ in HO units it is proportional to

$$\psi_n(x) \sim e^{-x^2/2}.$$

Therefore, 3D wave function is proportional

$$\psi_n(\mathbf{r}) \sim e^{-r^2/2}.$$

By finding $\tilde{\alpha} = 1/2$ we just reconstructed original wave function, that a trial wave function Ψ_T is composed of.

This problem can be solved analytically due to relative simplicity of trial wave function and Hamiltonian of the system. Analogous fermionic problem has trial wave function of more complex form, and analytical solutions can be found only for a very simple cases like $N = 2$.

In Appendix 6.4 we also show that local energy (6.28) can be computed as

$$E_L = \frac{\hat{H}\Psi_T}{\Psi_T} = 3\alpha N + \left(\frac{1}{2} - 2\alpha^2\right)\sum_{i=1}^{N} r_i^2, \tag{3.23}$$

score O_1 is given by (6.34):

$$O_1 = \frac{\tilde{\Psi}_T}{\Psi_T} = -\sum_{i=1}^{N} r_i^2. \tag{3.24}$$

And score O_2 is just a product $E_L \times O_1$. These three variables need to be computed and stored on every VMC iteration to find their averages and perform a GD using them.

Expression for quantum force can be obtained using gradient of ψ_1 given by (6.9)

$$\mathbf{F}_i = 2\frac{\nabla_i \Psi_T}{\Psi_T} = \frac{-4\alpha\Psi_T\left(x_i \cdot \mathbf{e}_x + y_i \cdot \mathbf{e}_y + z_i \cdot \mathbf{e}_z\right)}{\Psi_T}, \tag{3.25}$$

therefore

$$\mathbf{F}_i = -4\alpha(x_i, y_i, z_i). \tag{3.26}$$

This system is the simplest of the system we study in this research. Mostly it comes from the relative simpleness of trial wave function.

3.7.2 Non-Interacting Fermions

As for non-interacting bosons, Hamiltonian is given by (2.35) and (2.37):

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{1}{2}\Delta_i + \frac{1}{2}r_i^2\right).$$

Trial wave function for N fermions in 2 dimensions is a product of spin-up and spin-down Slater determinants (3.3), (3.4) and (3.5),

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_N) = D_\uparrow(\mathbf{r}_1, ..., \mathbf{r}_{N/2}) D_\downarrow(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N).$$

For a system with trial wave function of this form, it is only possible to find analytical solutions for the first closed shell with two particles. For others closed shells, as we will show, it is not even possible to find proper analytical expression for local energy and scores – only expressions that can be computed numerically.

Let us first derive general expression for local energy,

$$E_L = \frac{\hat{H}\Psi}{\Psi} = \frac{1}{\Psi}\sum_{i=1}^{N}\left(-\frac{1}{2}\Delta_i\Psi + \frac{1}{2}r_i^2\Psi\right) = -\frac{1}{2}\sum_{i=1}^{N}\frac{\Delta_i\left(D_\uparrow D_\downarrow\right)}{D_\uparrow D_\downarrow} + \frac{1}{2}\sum_{i=1}^{N} r_i^2.$$

Second term can not be simplified further and it can be easily computed numerically. On the opposite hand, first term can be simplified. Spin-up SD is a function of $(\mathbf{r}_1, ..., \mathbf{r}_{N/2})$ and spin-down SD is a function of $(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N)$. Therefore, sum can be split in two:

$$\sum_{i=1}^{N}\frac{\Delta_i\left(D_\uparrow D_\downarrow\right)}{D_\uparrow D_\downarrow} = \sum_{i=1}^{N/2}\frac{D_\downarrow\Delta_i D_\uparrow}{D_\uparrow D_\downarrow} + \sum_{i=N/2+1}^{N}\frac{D_\uparrow\Delta_i D_\downarrow}{D_\uparrow D_\downarrow} = \sum_{i=1}^{N/2}\frac{\Delta_i D_\uparrow}{D_\uparrow} + \sum_{i=N/2+1}^{N}\frac{\Delta_i D_\downarrow}{D_\downarrow}.$$

The only thing left is to understand how to compute Laplacian of spin-up and spin-down Slater determinant. Let us consider spin-up SD, which is given by (3.7),

$$D_\uparrow(\mathbf{r}_1, ..., \mathbf{r}_{N/2}) = \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \cdots & \psi_{N/2}(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_{N/2}(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_{N/2}) & \psi_2(\mathbf{r}_{N/2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2}) \end{vmatrix}.$$

Each row of Slater determinant is dependent only on one coordinate. It means that gradient vector with respect to $\mathbf{r}_j$ of spin-up SD can be obtained from original spin-up SD by replacing all wave functions $\psi_i(\mathbf{r}_j)$ with respective gradient vectors $\nabla_j\psi_i(\mathbf{r}_j)$. For example,

$$\nabla_1 D_\uparrow(\mathbf{r}_1, ..., \mathbf{r}_{N/2}) = \begin{vmatrix} \nabla_1\psi_1(\mathbf{r}_1) & \nabla_1\psi_2(\mathbf{r}_1) & \cdots & \nabla_1\psi_{N/2}(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_{N/2}(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_{N/2}) & \psi_2(\mathbf{r}_{N/2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2}) \end{vmatrix}.$$

$\nabla_j D_\uparrow$ is a vector with two components:

$$\nabla_j D_\uparrow = \left(\frac{\partial D_\uparrow}{\partial x_j}, \frac{\partial D_\uparrow}{\partial y_j}\right).$$

Because the two determinants differ only in which particle coordinates they use, taking the gradient with respect to the j-th spin-down particle is done exactly as for spin-up: you form the usual Slater determinant and in its j-th row replace each orbital by its gradient. Laplacian of spin-up and spin-down Slater determinants can be obtained in a similar way – j-th row wave functions $\psi_i(\mathbf{r}_j)$ are replaced by respective Laplacians $\Delta_j\psi_i(\mathbf{r}_j)$. For example,

$$\Delta_N D_\downarrow(\mathbf{r}_{N/2},...,\mathbf{r}_N) = \begin{vmatrix} \psi_1(\mathbf{r}_{N/2+1}) & \psi_2(\mathbf{r}_{N/2+1}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+1}) \\ \psi_1(\mathbf{r}_{N/2+2}) & \psi_2(\mathbf{r}_{N/2+2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+2}) \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_N\psi_1(\mathbf{r}_N) & \Delta_N\psi_2(\mathbf{r}_N) & \cdots & \Delta_N\psi_{N/2}(\mathbf{r}_N) \end{vmatrix}.$$

Now we are able to compute local energy numerically using

$$E_L = -\frac{1}{2}\sum_{i=1}^{N/2}\frac{\Delta_i D_\uparrow}{D_\uparrow} - \frac{1}{2}\sum_{i=N/2+1}^{N}\frac{\Delta_i D_\downarrow}{D_\downarrow} + \frac{1}{2}\sum_{i=1}^{N} r_i^2. \tag{3.27}$$

Observable O_1 is given by (3.13) and (3.12),

$$O_1 = \frac{\bar{\Psi}_T}{\Psi_T} = \frac{1}{\Psi_T}\frac{\partial \Psi_T}{\partial \alpha}.$$

This expression can be simplified using the structure of trial wave function:

$$O_1 = \frac{1}{D_\uparrow D_\downarrow}\frac{\partial\left(D_\uparrow D_\downarrow\right)}{\partial \alpha} = \frac{1}{D_\uparrow}\frac{\partial D_\uparrow}{\partial \alpha} + \frac{1}{D_\downarrow}\frac{\partial D_\downarrow}{\partial \alpha}.$$

We can use expression for derivative from §2.1.1 [28]:

$$\frac{\partial \det(\mathbf{Y})}{\partial x} = \det(\mathbf{Y})\mathrm{Tr}\left(\mathbf{Y}^{-1}\frac{\partial \mathbf{Y}}{\partial x}\right)$$

Observable then simplifies to

$$\begin{aligned} O_1 &= \frac{1}{D_\uparrow}D_\uparrow \mathrm{Tr}\left(D_\uparrow^{-1}\frac{\partial D_\uparrow}{\partial \alpha}\right) + \frac{1}{D_\downarrow}D_\downarrow \mathrm{Tr}\left(D_\downarrow^{-1}\frac{\partial D_\downarrow}{\partial \alpha}\right) \\ &= \mathrm{Tr}\left(D_\uparrow^{-1}\frac{\partial D_\uparrow}{\partial \alpha}\right) + \mathrm{Tr}\left(D_\downarrow^{-1}\frac{\partial D_\downarrow}{\partial \alpha}\right). \end{aligned}$$

Since derivative of component wave functions with respect to α itroduce multiplicative factor $-r_i^2$, this can be written as

$$O_1 = \mathrm{Tr}\left(D_\uparrow^{-1}(-r^2)D_\uparrow\right) + \mathrm{Tr}\left(D_\downarrow^{-1}(-r^2)D_\downarrow\right),$$

so that the final expression is

$$O_1 = \sum_{i=1}^{N} -r_i^2. \tag{3.28}$$

General expression for quantum force can be obtained relatively easy:

$$\mathbf{F}_i = \begin{cases} 2\frac{\nabla_i D_\uparrow}{D_\uparrow} & \text{if} \quad i < N/2; \\ 2\frac{\nabla_i D_\downarrow}{D_\downarrow} & \text{if} \quad i > N/2+1. \end{cases} \tag{3.29}$$

The next step is to discuss how these variables can be computed for first three closed shells.

First Closed Shell

When there is only two non-interacting fermions, these fermions occupy lowest energy state $n = 0$. This system is exactly the same system as its bosonic analogue. In Appendix 6.5 we show that analytical solution can be found and that this solution is given by (6.37), (6.38) and (6.39):

$$E_0(\alpha) = 2\alpha + \frac{1}{2\alpha}; \quad \tilde{\alpha} = \frac{1}{2}; \quad E(\tilde{\alpha}) = 2.$$

This solution follows general solution (3.22) with $d = 2$ and $N = 2$.

Analytical expression for local energy is given by (6.35)

$$E_L = 4\alpha + (r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right).$$

Score O_1 can be computed as (6.36),

$$O_2 = -(r_1^2 + r_2^2)$$

and quantum force (3.29)is

$$\mathbf{F}_i = -4\alpha(x_i, y_i).$$

Second Closed Shells

Consider second closed shell first. For this system, spin-up and spin down Slater determinants (3.7), (3.8) are

$$D_\uparrow(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \psi_3(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \psi_3(\mathbf{r}_2) \\ \psi_1(\mathbf{r}_3) & \psi_2(\mathbf{r}_3) & \psi_3(\mathbf{r}_3) \end{vmatrix}; \tag{3.30}$$

$$D_\downarrow(\mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_6) = \begin{vmatrix} \psi_1(\mathbf{r}_4) & \psi_2(\mathbf{r}_4) & \psi_3(\mathbf{r}_4) \\ \psi_1(\mathbf{r}_5) & \psi_2(\mathbf{r}_5) & \psi_3(\mathbf{r}_5) \\ \psi_1(\mathbf{r}_6) & \psi_2(\mathbf{r}_6) & \psi_3(\mathbf{r}_6) \end{vmatrix}. \tag{3.31}$$

Here, spin orbitals are given by (6.7), (6.12) and (**??**):

$$\psi_1(\mathbf{r}) = e^{-\alpha r^2}; \quad \psi_2(\mathbf{r}) = xe^{-\alpha r^2}; \quad \psi_3(\mathbf{r}) = ye^{-\alpha r^2}.$$

Ansatz (3.6) for this system is

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_6) = D_\uparrow(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)D_\downarrow(\mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_6).$$

Local energy is computed using (3.20),

$$E_L = -\frac{1}{2}\sum_{i=1}^{3}\frac{\Delta_i D_\uparrow}{D_\uparrow} - \frac{1}{2}\sum_{i=4}^{6}\frac{\Delta_i D_\downarrow}{D_\downarrow} + \frac{1}{2}\sum_{i=1}^{6} r_i^2.$$

Laplasians of spin orbitals (6.10), (6.16), (6.17) are

$$\Delta\psi_1(x,y) = \left(-4\alpha + 4\alpha^2 r^2\right) e^{-\alpha r^2};$$
$$\Delta\psi_2(x,y) = x\left(-8\alpha + 4\alpha^2 r^2\right) e^{-\alpha r^2};$$
$$\Delta\psi_3(x,y) = y\left(-8\alpha + 4\alpha^2 r^2\right) e^{-\alpha r^2}.$$

Observable O_1 is computed using (3.32),

$$O_1 = \sum_{i=1}^{6} r_i^2.$$

Quantum force $\mathbf{F}_i$ is computed numerically using general expression (3.29).

Third Closed Shells

For a system with twelve fermions, components of spin-sector Slater determinants are given by

$$[D_{\uparrow}(\mathbf{r}_1, ..., \mathbf{r}_6)]_{ij} = \psi_i(\mathbf{r}_j); \quad [D_{\downarrow}(\mathbf{r}_7, ..., \mathbf{r}_{12})]_{ij} = \psi_i(\mathbf{r}_{j+6}), \tag{3.32}$$

where $i, j = 1, 2, 3, 4, 5, 6$. Spin orbitals $1-3$ are the same as for six particle case, while spin orbitals $4-6$ are given by (6.18), (6.19) and (6.20):

$$\psi_4(x,y) = (x^2-1)e^{-\alpha r^2}; \quad \psi_5(x,y) = (y^2-1)e^{-\alpha r^2}; \quad \psi_6(x,y) = xye^{-\alpha r^2}.$$

Local energy is computed using (3.20)

$$E_L = -\frac{1}{2}\sum_{i=1}^{6}\frac{\Delta_i D_{\uparrow}}{D_{\uparrow}} - \frac{1}{2}\sum_{i=7}^{12}\frac{\Delta_i D_{\downarrow}}{D_{\downarrow}} + \frac{1}{2}\sum_{i=1}^{12} r_i^2.$$

Laplacians of spin orbitals $1-3$ are the same, and Laplasinas of spin-orbitals $4-6$ are given by (6.24), (6.25) and (6.26):

$$\begin{aligned}
\Delta\psi_4(x,y) &= \Big(2 - 12\alpha x^2 + 4\alpha^2 x^2 r^2 + 4\alpha - 4\alpha^2 r^2\Big)e^{-\alpha r^2};\\
\Delta\psi_5(x,y) &= \Big(2 - 12\alpha y^2 + 4\alpha^2 y^2 r^2 + 4\alpha - 4\alpha^2 r^2\Big)e^{-\alpha r^2};\\
\Delta\psi_6(x,y) &= \Big(-12\alpha xy + 4\alpha^2 xyr^2\Big)e^{-\alpha r^2}.
\end{aligned}$$

Score O_1 is computed using (3.22),

$$O_1 = \sum_{i=1}^{12} -r_i^2,$$

and quantum force is computed numerically using general expression (3.29).

3.7.3 Interacting Fermions

To take interaction into account, Hamiltonial is modified with Coulomb's potential. In atomic units it is given by expressions (2.35), (2.36) and (2.37):

$$\hat{H} = \sum_{i=1}^{N}\left[-\frac{1}{2}\Delta_i + \frac{1}{2}B(\omega)r_i^2 + \sum_{j>i}^{N}\frac{1}{r_{ij}}\right].$$

As discussed earlier, $B(\omega)$ determines trap strength and it can be tuned by changing oscillator frequency ω.

To satisfy Kato's cusp condition, trial wave function is multiplied with Jastrow J or Pade-Jastrow P factors, so that ansatzes are given by (3.16):

$$\begin{aligned}
\Psi_J(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha, \beta) &= D_{\uparrow}(\mathbf{r}_1, ..., \mathbf{r}_{N/2}; \alpha)D_{\downarrow}(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N; \alpha)J(\mathbf{r}_1, ..., \mathbf{r}_N; \beta);\\
\Psi_P(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha, \beta) &= D_{\uparrow}(\mathbf{r}_1, ..., \mathbf{r}_{N/2}; \alpha)D_{\downarrow}(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N; \alpha,)P(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha,).
\end{aligned}$$

Where J and P are given by (2.39) and (2.40).

Including Jastrow factors of different form into wave function ansatzes make computation of local energy more complex. As we discussed earlier, optimal variational α is obtained by solving non-interatig problem. Therefore, for interacting case the goal is to optimize variational parameters in Jastrow or Pade-Jastrow factors.

Since wave function of non-interacting fermions describes non-interacting system, optimal variational parameter α is found for non-interacting system. Meaning that GD for interacting fermions tunes only variational parameters in Jastrow factors J and P and uses optimal α obtained from non-interacting case.

In Appendix 6.5 we derive analytical expressions for local energy E_L and scores $O_1^{(J)}$, $O_1^{(P)}$. Local energy is given by (6.8):

$$E_L = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3 + \hat{H}_I + \hat{V},$$

where $\mathcal{E}_1$ is given by (6.4),

$$\mathcal{E}_1 = -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\Delta_i D_\downarrow}{D_\downarrow}\right].$$

This quantity represents local energy of non-interacting system. We have discussed how to compute it in previous section.

$\mathcal{E}_2$ is given by (6.8),

$$\mathcal{E}_2 = -\left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow \nabla_i \ln I}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow \nabla_i \ln I}{D_\downarrow}\right].$$

This term contains gradient vectors of spin-up and spin-down Slater determinants. $\nabla_i D_\uparrow$ be obtained by replacing component wave functions $\psi_j(\mathbf{r}_i)$ with respective gradients $\nabla_i \psi_j(\mathbf{r}_i)$. Expressions for gradients of the wave functions $1-6$ were derived in Appendix 6.4 and they are given by equations

$$\nabla\psi_1 = -2\alpha e^{-\alpha r^2}(x, y);$$

$$\nabla\psi_2 = e^{-\alpha r^2}((1-2\alpha x^2), -2\alpha xy);$$

$$\nabla\psi_3 = e^{-\alpha r^2}(-2\alpha xy, (1-2\alpha y^2));$$

$$\nabla\psi_4 = e^{-\alpha r^2}((2x - 2\alpha x^3 + 2\alpha x), ((1-x^2)2\alpha y));$$

$$\nabla\psi_5 = e^{-\alpha r^2}((1-y^2)2\alpha x), (2y - 2\alpha y^3 + 2\alpha y));$$

and

$$\nabla\psi_6 = e^{-\alpha r^2}((y - 2\alpha x^2 y), (x - 2\alpha x y^2)).$$

and $\mathcal{E}_2$ is given by (6.8),

$$\mathcal{E}_3 = -\frac{1}{2}\sum_{i=1}^{N}\left(\Delta \ln I - (\nabla \ln I)^2\right).$$

By using this formulation, we never have to evaluate costly exponential functions. Instead, we only compute the exponent's argument, which is far cheaper computationally. Obviously, the same procedure can be done to non-interacting ansatz Φ, but we will not get the same benefit because Φ is a determinant constructed from exponential functions.

In expressions for $\mathcal{E}_2$ and $\mathcal{E}_3$ gradients and Laplacians of Jastrow and Pade-Jastrow factors are given by (6.50), (6.51), (6.52) and (6.53):

$$\nabla_i \ln J = \sum_{j=1\neq i}^{N} \beta_{ij} \frac{(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}};$$

$$\Delta_i \ln J = \sum_{j=1\neq i}^{N} \frac{\beta_{ij}}{r_{ij}};$$

$$\nabla_i \ln P = \sum_{j=1\neq i}^{N} \frac{a(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}(1+\beta r_{ij})^2};$$

$$\Delta_i \ln P = \sum_{j=1\neq i}^{N} \frac{a(1-\beta r_{ij})}{r_{ij}(1+\beta r_{ij})^3}.$$

The score $\mathbf{O}_1^{(J)}$ is a vector of dimensionality $p = N(N-1)/2$. It can be computed using (6.1),

$$\mathbf{O}_1^{(J)} = (r_0, r_1, ..., r_{p-1}, r_p).$$

Finally, the score $O_1^{(P)}$ is computed using (6.2):

$$O_1^{(P)} = \sum_{k=0}^{N-1} \frac{-a r_k^2}{(1+\beta r_k)^2}.$$

Quantum force (3.18) is computed as

$$\mathbf{F}_i = \begin{cases} \frac{\nabla_i D_\uparrow}{D_\uparrow} + \frac{\nabla I}{I} & \text{if} \quad i < N/2; \\ \frac{\nabla_i D_\downarrow}{D_\downarrow} + \frac{\nabla I}{I} & \text{if} \quad i < N/2. \end{cases}$$

3.8 Automatic Differentiation

Automatic differentiation is a powerful computational tool. It allow derivatives to be computed numerically. In Appendix 6.2 we derived gradients and Laplacians of all spin orbital wave functions. Even Laplacian of spin orbitals with $(n_x = 2, n_y = 0)$ has Laplacian of complex form. Instead of using analytical expressions, alternatively, we can use automatic differentiation to obtain numerical value of derivatives.

Benefit of this approach is that we do not need to find and program analytical derivatives. Downside - it is computationally expensive, more expensive then using analytical expressions. The best way to use automatic differentiation, if analytical expressions can be found, is to compare analytical result with automatic differentiation. This way, if results align, we can be sure that analytical expressions implemented correctly.

In our implementation we use `C++` library `autodiff` [29]. This library allows to compute derivatives numerically in two modes: forward and reverse. Forward mode computes both function and its derivative at the same time, while reverse mode computes only derivative itself. For our purposes, such as computing derivatives of spin orbitals, we chose to use reverse mode.

3.9 Parallelization

VMC computations can be very computationally expensive. Even for such a small system with only twelve particles, VMC can run hours, and even days. To obtain small relative error we need to run VMC for 10^6 iterations and 10^5 burn-in iterations. This means that computer needs to calculate 6×6 determinant more than 10^7 times. Normally, program is run on a computer's processor. Modern processors have several cores. Serial program uses only one core of the processor, while other cores are idle. It is efficient to use as many processor cores to perform computation simultaneously as possible. Parallel programming is designed to distribute work between processor cores.

There two type of systems: systems with shared memory and systems with distributed memory. As comes form its name, for systems with shared memory each core has access to the memory they use while running a program. On the opposite side, cores of distributed memory systems have their own memories, and program needs to be written in a specific way to coordinate cores' work properly.

`OpenMP` is an API that allows to parallelize programs run on shared memory systems, such as normal computers. `MPI` is an alternative API that allows to parallelize programs run on both distributed and shared memory systems. Example of a system with shared memory is a super computer or a cluster computer.

We have implemented both `OpenMP` and `MPI` VMC programs. One of the benefits of VMC is that it can be easily parallelized. Consider running a VMC program for N iterations using n cores. We can create n copies of the same system, and run VMC simulation on all n cores simultaneously. Using this approach, each core computes N/n VMC iteration. Each core evolve its system in their own way using different random numbers and sample local energy and other observables. After cores are done with VMC iterations, they send their sampled values to the "master" core, which then can compute averages of observables, perform GD, etc. Communication between cores take some time, but speedup can be huge.

G. Argentini provides a discussion on Amdahl's law, which provides a way to compute parallelization speedup [30]. When less then eight cores are used and program is parallelizable, speedup is almost linear. It mean that parallel program would run almost eight times faster than serial, which is very impressive.

3.10 Computational Implementation

We have discussed all necessary details needed to perform VMC estimation of GS energy. In this section we will discuss the way we implemented VMC and GD algorithms numerically. `C++` is used as the main programming language to perform calculations and `Python` is used to generate figures and to perform analysis of the data.

3.10.1 Object-Oriented Class Structure

Our implementation builds upon the Object-Oriented Template for Variational Monte Carlo by Morten Ledum and Øyvind Sigmundson Schøyen [31], which offers a modular class framework optimized for VMC calculations.

The object-oriented architecture employed in the research is illustrated in Figure 3.4. The top-level `VMC` directory contains the implementation of the Variational Monte Carlo algorithm and is organized into eight primary classes. Each class has a distinct responsibility, and some of them define additional subclasses. In the sections that follow, we describe the role and functionality of each class in turn.

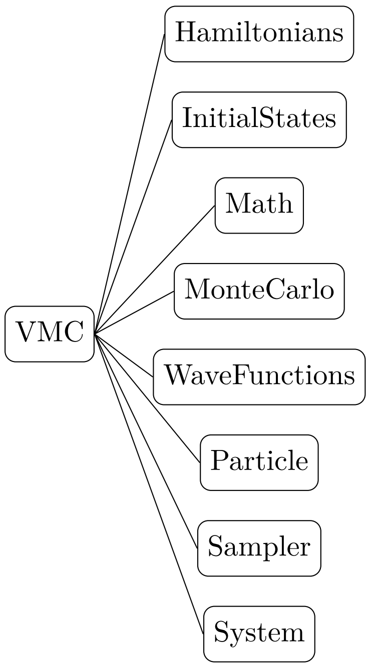


Figure 3.4: Class hierarchy of VMC directory.

3.10.2 Math Class

Class `Math` is composed only of one file that contains definition of functions

```
int nextInt(const int &lowerLimit, const int &upperLimit);
double nextDouble();
double nextGaussian(const double &mean, const double &standardDeviation).
```

In `C+` function is defined as

```
    return_type Function_name(arg_1_type arg_1, ..., arg_N_type arg_N).
```

`Function_name` is a name of the function. This function has `arg_1, ..., arg_N` arguments of type `arg_1_type, ..., arg_N_type`. `return_type` is a type of variable function returns.

Therefore, function `nextInt` generates an integer in range `lowerLimit, upperLimit`. `&lowerLimit` is a reference to `lowerLimit` variable – its location in memory. By providing reference to variable instead of variable itself we avoid copying this variable, which is less computationally expensive. Likewise, function `nextDouble` generates uniformly distributed random `double` in range `[0, 1)` and `nextGaussian` generates normally distributed random `double` in range `[mean - standardDeviation, mean + standardDeviation]`.

This class has a constructor, that generates object of `Random` class – random number generator (RNG).

`Random()` and `Random(seed)`.

If constructor is called without argument - it creates RNG that generates random sequence of numbers every time. If it is called with `seed` as its argument, RNG generates the same sequence of random numbers every time.

3.10.3 Particle Class

In `C++`, classes can have private and public variables. These variables characterize class object.

The `Particle` class represents a single particle's position in d-dimensional space. Internally, the class stores position vector and number of dimension of the particle

```
int m_numberOfDimensions;
std::vector<double> m_position.
```

Its constructor

```
Particle(const std::vector<double>& position)
```

takes a constant reference to a `std::vector<double>`, where each entry represents one Cartesian coordinate and sets private variable `m_position` equal to it argument `position`.

This class implements only one function, which shifts the d-th coordinate by Δ (see Algorithm 4).

Algorithm 4 Adjust Position

```
1: procedure ADJUSTPOSITION($d$, $\Delta$)
2:     m_position[$d$] += $\Delta$.
3: end procedure
```

In addition, two helping functions provide read-only access to the state:

- `std::vector<double>& getPosition()` Returns position of the particle `m_position`.
- `int getNumberOfDimensions()` Returns number of dimensions of the particle `m_numberOfDimensions`.

3.10.4 Initial States Class

Class `InitialStates` has only one function

```
std::vector<std::unique_ptr<Particle» setupRandomUniformInitialState
( int numberOfDimensions, int numberOfParticles, Random& rng ).
```

This function has three arguments: number of dimensions d, number of particles N and reference to RNG R. In Algorithm 5 we explain how this function is built using pseudocode. When we write R.`nextDouble()`, we use dot notation, because `nextDouble()` is a function, that only member of `Random` class can use.

Algorithm 5 Random Uniform Initial State

```
procedure SETUPRANDOMUNIFORMINITIALSTATE($N, d, R$)
    Create empty vector particles, which elements are objects of Particle class;
    Create vector pos of size $d$;
    for (int i = 0; i < $N$; i++) do
        for (int j = 0; j < $d$; j++) do
            pos[j] $\leftarrow$R.$nextDouble$();        end for
                Create object part of Particle class with pos coordinates;
                particles.push_back(part);
            end for
            return particles.
        end procedure
```

First, this function generates d uniform random numbers and stores them in vector `pos`. Then it creates object `part` of `Particle` class. In the end, it adds `part` to the end of vector `particles`. This process is repeated N times, one time for each particle. In the end, we get vector of size N that contains N objects of class `Particle`. This `particles` vector is then passed to another functions, that require position of the particles. Position and number of dimensions of each particle can be extracted using `getPosition` and `getNumberOfDimensions` functions of `Particle`. Essentially, vector `particles` is an object oriented way to store position of every particle.

From this point forward, `std::vector<std::unique_ptr<Particle»` will be denoted as `Pvec` and `std::vector <double>` will be denoted as `Dvec`.

3.10.5 Wave Function Class

`WaveFunctions` is the most complex and most important class. This class computes trial wave function and Laplacian of trial wave function for all model systems. From this point forward we will discuss only crucial parts of implementation.

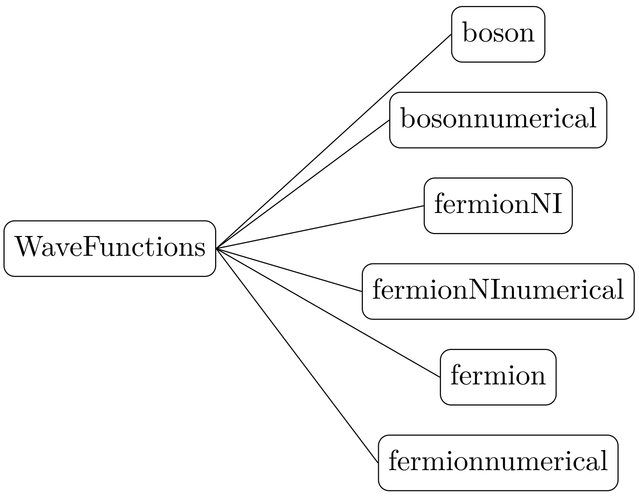


Figure 3.5: `WaveFunctions` class hierarchy.

On Figure 3.5 we present structure of `WaveFunctions` class. This class has four subclasses: `boson`, `bosonnumerical`, `fermionNI`, `fermionNInumerical`, `fermion` and `fermionnumerical`.

`WaveFunctions` has four private variables:

```
int m_numberOfParameters;
Dvec m_parameters;
int m_particles;
int m_mode.
```

First and third variables represent number of variational parameters and number of particles. `m_parameters` is a vector with all variational parameters. `int m_mode` is an integer equal to zero, if Jastrow factor is used and equal to one, if Pade-Jastrow factor is used. This class implements seven functions:

- `double r_squared(Pvec &particles, int i)` computes d-dimensional r^2;
- `double r_ij(Pvec &particles, int i, int j)` computes relative distance r_{ij};
- `int BetaIndex(int i, int j)` computes 1D beta index according to (3.15);
- `double a_ij(int i, int j)` computes constant a, which is needed for Pade-Jastrow factor (2.28) for the particles i and j.
- `GreensFunctionRatio(&R_new, &R_old, double dt, &F_new, &F_old)` computes ratio of Green's functions needed for Metropolis-Hastings algorithm. Every argument of this function is a d-dimensional vector `Dvec`, accept dt;
- `int getNumberOfParameters()` returns `m_numberOfParameters`;
- `Dvec getParameters()` returns `m_parameters` vector.

Each of these function are inherited by all `WaveFunctions` subclasses.

In addition, `WaveFunctions` declares three virtual functions:

- `virtual double evaluate(Pvec &particles)` – computes trial wave tunction;
- `virtual double computeDoubleDerivative(Pvec &particles) -` computes

$$\sum_{i=1}^{N} \frac{1}{\Psi_T} \Delta_i \Psi_T;$$

- `virtual Dvec quantumForce(Pvec&particles, int i)` computes quantum force for particle `i`.

These function are not defined in `WaveFunctions` class, only declared. Each subclass has their own definition of these three functions.

Simply said, `WaveFunctions` subclasses are designed to implement `evaluate` and `computeDoubleDerivative` functions for their own respective trial wave functions (and `quantumForce` when Mestopolis-Hastings is used).

Boson Class

The `boson` subclass is designed to compute trial wave function for N bosons in d dimensions, given by (3.2)

$$\Psi_T = \exp\left(-\alpha \sum_{i=1}^{N} r_i^2\right)$$

and Laplacian term, which is given by

$$\frac{1}{\Psi_T}\sum_{i=1}^{N}\Delta_i\Psi_T = -2d\alpha N + 4\alpha^2\sum_{i=1}^{n} r_i^2.$$

This is a generalized for d-dimensions version of formula (6.29).

`boson` is a subclass of `WaveFunctions` class, meaning that objects of `boson` class inheriat all functions and private variables of `WaveFunctions`.

This class implementation of `evaluate(Pvec &particles` function is presented in Algorithm 6. This algorithm allows to compute trial wave function relatively easy, compared to the other `WaveFunctions` subclasses.

Algorithm 6 Evaluate Function For Boson Class

```
1: procedure EVALUATE(Pvec &particles)
2:     $N \leftarrow$ m_particles;
3:     $\alpha \leftarrow$ m_parameters[0];
4:     $r2 \leftarrow 0.0$;
5:     for (int i = 0; i < $N$; i++) do
6:         $r2 \leftarrow r2+$ r_squared(particles, i);
7:     end for
8:     return $\exp(-\alpha \cdot r2)$.
9: end procedure
```

This class implementation of `computeDoubleDerivative(Pvec &particles)` is in the Algorithm 7

Algorithm 7 Compute Double Derivative Function For Boson Class

```
1: procedure COMPUTEDOUBLEDERIVATIVE($\mathbf{X}$)
2:     $N \leftarrow$ m_particles;
3:     $\alpha \leftarrow$ m_parameters[0];
4:     $r2 \leftarrow 0.0$;
5:     Particle $p \leftarrow$ particles[0];
6:     $d \leftarrow p$.getNumberOfDimensions();
7:     for (int i = 0; i < $N$; i++) do
8:         $r2 \leftarrow r2+$ r_squared(particles, i);
9:     end for
10:    return $-2Nd\alpha + 4\alpha^2 r2$.
11: end procedure
```

The last function that this subclass implements is `quantumForce`, which is based on formula (3.26). It is presented in Algorithm 8. Next, we will discuss similar class `bosonnumerical`, which implements the same formulas, but using automatic differentiation.

Algorithm 8 Quantum Force For Boson Class

```
1:  procedure QUANTUMFORCE(Pvec &particles, int i)
2:      $\alpha \leftarrow$ m_parameters[0];
3:      Particle $p \leftarrow$ particles[i];
4:      $d \leftarrow p$.getNumberOfDimensions();
5:      Dvec $r \leftarrow$ $p$[i].getPosition();
6:      Initialize $d$-dimensional Dvec $F$ with zeros;
7:      for (int i = 0; i < $d$; i++) do
8:          $F$[i] $\leftarrow -4\alpha r$[i];
9:      end for
10:     return $F$.
11: end procedure
```

Boson Numerical Class

This class is designed to estimate GS energy using the same ansatz as `boson` class, but uses automatic differentiation. This, and all "numerical" classes implement new function,

`VectorXvar fill_x(Pvec &particles).`

This function is designed to construct a one-dimensional vector, and use it for automatic differentiation instead of `Pvec particles`. `autodiff` library uses `VectorXvar` and `var` as variable types. It can compute numerical derivatives only if function's arguments have these types. Function iteslef is pretty trivial, see Algorithm 9. We will use a shorthand `Vvec` for `VectorXvar` from now on. Elements of `Vvec` are accessed via parentheses, not square brackets: `x(0)`. Vector `x` generated by this function has elements

$$\mathtt{x} = (x_0, y_0, x_1, y_1, ..., x_N, y_N)$$

for two-dimensional system, which we are most interested in.

Algorithm 9 Fill X Function

```
1:  procedure FILL_X(Pvec &particles)
2:      $N \leftarrow$ m_particles;
3:      $d \leftarrow$ particles[0].getNumberOfDimension();
4:      Initialize VectorXvar x($N \cdot d$);
5:      for (int i = 0; i < $N$; i++) do
6:          Dvec $r \leftarrow$particles[i].getPosition();
7:          for (int j = 0; j < $d$; j++) do
8:              x(i $\cdot d$ + j) $\leftarrow r$[j];
9:          end for
10:     end for
11:     return x.
12: end procedure
```

`computeDoubleDerivative` function for this class is pretty simple, see Algorithm 10. It uses function `LaplPsiTOverPsiT(Vvec &x)` to compute sum of Laplacians. Function `var PsiT(x)` computes trial wave function for a given `Vvec x` and its return type is `var`, so that it can be differentiated numerically. Function `hessian(psiT, x, g)` computes Hessian matrix. This is a matrix with all partial derivatives of ansatz, and therefore all derivatives that we need stay on diagonal. Method `val(var y)` returns value of type `double` of variable `y`.

Algorithm 10 Compute Double Derivative For Boson Numerical Class

```
procedure LaplPsiTOverPsiT(Vvec &x)
    Var psiT ← PsiT(x);
    Initialize Eigen::VectorXd g;
    Eigen::MatrixXd H ← hessian(psiT, x, g);
    Eigen::VectorXd LaplVector ← H.diagonal();
    double sum ← val(LaplVector.sum());
    return sum / val(psiT).
end procedure
procedure computeDoubleDerivative(Pvec &particles)
    x ← fill_x(particles);
    return LaplPsiTOverPsiT(x).
end procedure
```

`quantumForce` function implementation of this class is described in Algorithm 11. This function uses `derivatives` function to compute derivatives.

Algorithm 11 Quantum Force For Boson Numerical Class

```
procedure quantumForce(Pvec &particles, int i)
    x ← fill_x(particles);
    Var psiT ← PsiT(x);
    d ← x.size() / m_particles;
    Initialize $d$-dimensional Dvec $F$ with zeros;
    for (int j = 0; j < $d$; j++) do
        [gradx] = derivatives(psiT, wrt(x($d$·i + j)));
        $F$[i] = 2 * val(gradx) / (val(psiT));
    end for
    return $F$.
end procedure
```

We will omit discussion of `FermionNI` and `FermionNInumerical` classes, because they are simplified version of `Fermion` and `Fermionnumerical` classes. Therefore, we are moving on straight to `Fermion` class.

Fermion Class

This class is designed to compute trial ave function and its second derivative for interacting fermions. Interacting system is described by one or p variational parameters β, therefore constructor of this class is different:

`Fermion(double alpha, Dvec beta, bool mode, int` N`).`

This constructor takes two new arguments – vector with variational parameters `Dvec beta` and `bool mode`. `mode` is a variable that allows program to switch between Jastrow an Pade-Jastrow factors. Vector `m_parameters` is filler with beta parameters first, and then with alpha.

`Fermion` class contains functions `Psi`j`(Pvec & particles, int idx)`, $j = 1, ..., 6$ that compute spin orbital wave function. Functions `GradiPsi`j`(Pvec & particles, int idx)` compute gradients and `LapliPsi1`j`(Pvec & particles, int idx)` compute Laplacians. `SD` 12 is a multi-purpose function. It computes spin-up and spin-down Slater determinants, as well as their gradients and Laplacians. `SD` is hard-coded only for $2, 6$ and 12 particles. We presented only `m_particles == 6` case. Structure of two other cases follows the same logic.

Algorithm 12 Slater Determinant Function

```
procedure SD(Pvec &particles, int spin, int row, int der, int comp)
     size← m_particles/2;
     $d \leftarrow 0.0$;
    Initialize size×size-dimensional MatrixXd $A$ with zeros;
    if m_particles == 6 then
        for (int i = 0; i < $s$; i++) do
            if i == row then
                if der == 0 then
                    A(i, 0) = Psi1(particles, i + size * spin);
                    A(i, 1) = Psi2(particles, i + size * spin);
                    A(i, 2) = Psi3(particles, i + size * spin);
                else if der == 1 then
                    A(i, 0) = GradiPsi1(particles, i + size * spin)[comp];
                    A(i, 1) = GradiPsi2(particles, i + size * spin)[comp];
                    A(i, 2) = GradiPsi3(particles, i + size * spin)[comp];
                else if der == 2 then
                    A(i, 0) = LapliPsi1(particles, i + size * spin);
                    A(i, 1) = LapliPsi2(particles, i + size * spin);
                    A(i, 2) = LapliPsi3(particles, i + size * spin);
                end if
            else
                A(i, 0) = Psi1(particles, i + size * spin);
                A(i, 1) = Psi2(particles, i + size * spin);
                A(i, 2) = Psi3(particles, i + size * spin);
            end if
        end for
        $d \leftarrow$ A.determinant();
    else if m_particles == 2 then
        ⋮
        $d \leftarrow$ A.determinant();
    else if m_particles == 12 then
        ⋮
        $d \leftarrow$ A.determinant();
    end if
    return $d$.
end procedure
```

Let us discuss arguments of this function:

- `Pvec &particles` – reference to vector of particles;

- If `int spin == 0` function computes spin-up SD, and if `int spin == 1` function computes spi-down SD;
- `int row` represents index i in expressions $\nabla_i D_\uparrow$ and $\Delta_i D_\uparrow$. In other words, it represents row, in which spin orbitals are replaced with respective gradients or Laplacians;
- `int der` – order of derivative. 0 - normal SD, 1 - gradient, 2 - Laplacian;
- Gradient of SD is a vector with two components. `int comp` determines with of gradient components function computes.

The rest of functions just implement analytical expression that we derived earlier. We show a structure of `evaluate`, `computeDoubleDerivative` functions in Algorithm 13.

Algorithm 13 Quantum Force For Boson Numerical Class

```
procedure EVALUATE(Pvec &particles)
    J = (m_mode == 0) ?  Jastrow(particles) :  PadeJastrow(particles);
    return SD(particles, 0, 0, 0, 0) * SD(particles, 1, 0, 0, 0) * J;
end procedure
procedure COMPUTEDOUBLEDERIVATIVE(Pvec &particles)
    Lapl ← 0;
    GradGrad ← 0;
    if m_mode == 0 then
        for int i = 0; i < m_particles; i++ do
            Lapl += LapliJOverJ(particles, i);
            GradGrad += GradiPsiTGradiJOverPsiT(particles, i);
        end for
    else if m_mode == 1 then
        for int i = 0; i < m_particles; i++ do
            Lapl += LapliJOverJ(particles, i);
            GradGrad += GradiPsiTGradiJOverPsiT(particles, i);
        end for
    end if
    return LaplPsiTOverPsiT(particles) + Lapl + GradGrad
end procedure
```

`Fermion` class contains More then thirty different function. All of these function are written just to compute wave function and sun of it's second derivatives. For example, function `Jastow` computes Jastrow factor for a given particles, `LapliJOver` computes $\Delta_i \ln J$, and so forth.

Next class we want to discuss is `Hamiltonians`, which computes $\hat{H}\Psi_T$.

3.10.6 Hamiltonians Class

We designed `Hamiltonians` class in a general way. This class declares only one virtual function: `computeLocalEnergy`, which is then defined by its subclasses. We built the class this way, so that it is easy to add new terms to Hamiltonian, switch on and switch off some tems easily. To solve our problem, we have created class `HarmonicOscillator`, subclass of `Hamiltonians` class.

`HarmonicOscillator` class has two privat variables:

`double m_omega` *and* `bool m_Coulomb`.

First variable represents oscillator frequency of the trap ω, and `bool m_Coulomb` is used to switch between interacting and non-interacting Hamiltonians. Constructor used to create object of `HarmonicOscillator` class is

```
HarmonicOscillator::HarmonicOscillator(double omega, bool Coulomb)
```

and it just sets `m_omega = omega` and `m_Coulomb = Coulomb`.

`HarmonicOscillator`'s implementation of `computeLocalEnergy` function is presented in 14. It is pretty simple, because all heavy calculation is hidden inside `computeDoubleDerivative` function.

Algorithm 14 Compute Local Energy Function

```
1: procedure COMPUTELOCALENERGY(class WaveFunction &waveFunction, Pvec &particles)
2:     $N \leftarrow$particles.size();
3:     Initialize $r2 \leftarrow 0.0$;
4:     for int i = 0; i < $N$; i++ do
5:         $r2 \leftarrow r2$ + waveFunction.r_squared(particles, i);
6:     end for
7:     $T \leftarrow 0.5 \cdot r2 \cdot$m_omega$\cdot$m_omega;
8:     $V \leftarrow -0.5 \cdot$ waveFunction.computeDoubleDerivative(particles);
9:     $E \leftarrow T + V$;
10:    if m_Coulomb == 1 then
11:        $C \leftarrow 0.0$;
12:        for int i = 0; i < $N$; i++ do
13:            for int j = i; j < $N$; j++ do
14:                $C \leftarrow C + 1/$waveFunction.r_ij(particles, i, j);
15:            end for
16:        end for
17:        $E \leftarrow E + C$;
18:    end if
19:    return E.
20: end procedure
```

Now we are ready to proceed with presenting `Solvers` class, wich implements Metropolis VMC iteration and Metropolis-Hastings VMC iteration.

3.10.7 Monte Carlo Class

`Solver` is a directory name. This directory contains `montecarlo` class and its two subclasses: `metropolis` and `metropolishastings`. This structure was chosen to easily switch between these two algorithms. The `montecarlo` class has one privat variable, `m_rng` - object of `Random` class. Constructor of this class

```
MonteCarlo(std::unique_ptr<class Random> rng)
```

just sets `m_rng = rng`. This class declares just one virtual function,

```
bool step(double sl, class WaveFunction &waveFunction, Pvec &particles).
```

This function is implemented by `metropolis` and `metropolishastings` subclasses and it makes one VMC iteration (step).

Metropolis Class

The `metropolis` class is designed to implement VMC Metropolis algorithm and implement `step` function. In Algorithm 15 we schematically explain how this function works.

Algorithm 15 Metropolis Step Function

```
procedure STEP(sl, &waveFunction, &particles))
    Initialize $N$ and $d$;
    Initialize $\Psi_o \leftarrow 0$ and $\Psi_n \leftarrow 0$;
    Initialize Vvec<Vvec<double» rand of size $N \times d$;
    Compute old state wave function $\Psi_o \leftarrow$ waveFunction.evaluate(particles);
    for int i = 0; i < $N$; i++ do
        for int j = i; j < $d$; j++ do
            rand[i][j] = m_rng.nextDouble();
            particles[i].adjustPosition(sl·(rand[i][j] - 0.5), j);
        end for
    end for
    Compute new state wave function $\Psi_n \leftarrow$ waveFunction.evaluate(particles);
    Compute acceptance ratio, $\alpha \leftarrow$ min(1, $\Psi_n^2/\Psi_o^2$);
    if m_rng.nextDouble() > $\alpha$ then
        for int i = 0; i < $N$; i++ do
            for int j = i; j < $d$; j++ do
                particles[i].adjustPosition(-sl·(rand[i][j] - 0.5), j);
            end for
        end for
        return 0;
    else
        return 1;
    end if
end procedure
```

Essentially, this function represents our implementation of general Algorithm 1. To be able to compute new state wave function, first we need to move all particles. Therefore if step is rejected we move all particles back. To return particles to exact previous locations we save all uniform random numbers to `rand` vector, and them use elements of this vector to return particles back. If the step is accepted, this function returns `bool true`, adn if rejected it returns `bool false`. Accepted steps ratio is a very helful value. It allows us to determine if step size and learning rate (for GD) choice is good. Obviously, the choice is bad if acceptance rate is close to zero or one. We will see, that when systems become bigger, learning rate and step size need to be chosen as smaller values.

Metopolis Hastings Class

`metropolishastings` class is similar to `metropolis` class in the sense that it just implements the same function. However, this time according to Metropolis-Hastings algorithm. Algorithm 16 schematically describes Metropolis-Hastings algorithm. Here, approach is a little bit different. We move one particle at a time, and them compute quantum force and new wave with these new positions. Afterwards we compute Green's function ratio and accept or reject step only for one particle, and then repeat it for all particles. In this case, acceptance rate is not that useful, since it will be almost always close to one. However, it still can be used to judge step size and learning rate.

Algorithm 16 Metropolis Hastings Step Function

```
1: procedure STEP(sl, &waveFunction, &particles))
2:     Initialize $N$ and $d$;
3:     Initialize $\Psi_o \leftarrow 0$ and $\Psi_n \leftarrow 0$;
4:     $dt \leftarrow$ sl, $sdt \leftarrow$ sqrt(sl);
5:     Compute old state WF $\Psi_o \leftarrow$ waveFunction.evaluate(particles);
6:     Initialize Vvec<Vvec<double>> $F_o$ of size $N \times d$;
7:     for int i = 0; i < $N$; i++ do
8:         $F_o$[i] $\leftarrow$ waveFunction.quantumForce(particles, i);
9:     end for
10:    Initialize bool accept $\leftarrow$0;
11:    for int i = 0; i < $N$; i++ do
12:        Dvec $R_o \leftarrow$particles[i].getPosition();
13:        Initialize Dvec $R_n \leftarrow$0;
14:        for int j = 0; j < $d$; j++ do
15:            drift $\leftarrow$ D * $F_o$[i][j] * dt;
16:            gauss $\leftarrow$ nextGaussian(0, 1);
17:            $R_n \leftarrow R_o$ + drift$\cdot$gauss$\cdot sdt$;
18:        end for
19:        Compute new state WF $\Psi_n \leftarrow$ waveFunction.evaluate(particles);
20:        $F_n$ $\leftarrow$ waveFunction.quantumForce(particles, i);
21:        $G \leftarrow$ waveFunction.GreensFunctionRatio($R_n$, $R_n$, $dt$, $F_o$[i], $F_n$);
22:        Compute acceptance ratio, $\alpha \leftarrow$ min(1, $G \cdot \Psi_n^2/\Psi_o^2$);
23:        if m_rng.nextDouble()$\leq \alpha$ then
24:            $\Psi_o \leftarrow \Psi_n$;
25:            $F_o$[i] = $F_N$;
26:            accept $\leftarrow$1;
27:        else
28:            for int j = 0; j < $d$; j++ do
29:                particles[i].adjustPosition($R_o$[i] - $R_n$i], i);
30:            end for
31:        end if
32:    end for
33: end procedure
```

3.10.8 Sampler Class

This class is responsible for computing and storing all observables. Because of it, `sampler` class has a lot of private variables, some of them are

`int m_nPairs, int m_energy, int m_cumulativeEnergy, Dvec m_O1Jastow`

and a lot of others. Constructor of this class has a following structure:

```
Sampler( int $N$, int $d$, double $sl$, int $n$)
{
  m_N = $N$, m_n = $n$, m_d = $d$, m_sl = $sl$;
}.
```

All other variables are set to zero or zero vectors of respective size.

This class has three function:

- `sample(bool acceptedStep, System* system)` is responsible for computing all sampled values. All sampled values are added to respective `m_cumulative` private variables;
- `printOutputToTerminal(System& system)` is a helper function that outputs information about the system to terminal;
- `computeAverages()` computes mean values of `m_cumulative` variables.

In Algorithm 17 we outline structure of `sample` function. Basically, it just samples all observables: local energy, O_1 and O_2 for α, β and β_{ij} optimization. In algorithm we used a shorthand `m_c` for `m_cumulative` for all cumulative variables.

Algorithm 17 Sample Function

```
1: procedure SAMPLE(bool acceptedStep, System* system)
2:     Compute local energy, $E \leftarrow$ system.computeLocalEnergy();
3:     Sample local energy, m_cEnergy $\leftarrow$ m_cEnergy $+ E$;
4:     Compute sum of $r_i^2$ of all particles, $r2 \leftarrow$ system -> computer2();
5:     Sample $O_1$ for $\alpha$ optimization, m_cO1alpha $\leftarrow$ m_cO1alpha $+ r2$;
6:     Sample $O_2$ for $\alpha$ optimization, m_cO2alpha $\leftarrow$ m_cO2alpha $+ r2 \cdot E$;
7:     $N \leftarrow$ m_N, p$\leftarrow 0$, $O \leftarrow 0.0$;
8:     for int i = 0; i < $N-1$; i++ do
9:         for int j = i; j < $N$; j++ do
10:            $r_{ij} \leftarrow$ system.computerij(i, j);
11:            m_cO1Jastrow[p] += $r_{ij}$;
12:            m_cO2Jastrow[p] += $E \cdot r_{ij}$;
13:            $a \leftarrow 1$ if spins of $i$ and $j$ particles are idetical, $a \leftarrow 1/3$ otherwise;
14:            $\beta \leftarrow$ system.getWaveFunctionParameters()[0];
15:            $O \leftarrow O - ar_{ij}^2/(1+\beta r_{ij})^2$;
16:            p++;
17:        end for
18:    end for
19:    m_cO1Pade $\leftarrow$ m_O1cPade + $O$;
20:    m_cO2Pade $\leftarrow$ m_O2cPade + $O \cdot E$;
21:    m_stepnumber++;
22:    m_numberOfAcceptedSteps += acceptedStep.
23: end procedure
```

Function computeAverages() just divides all cumulative variable by number of VMC iterations n.

Additionally, Sampler class implements simple functions that return observables. For example, getEnergy() function returns m_energy, and getO1Jastow() returns vector m_O1Jastrow.

3.10.9 System Class

System is a class, that contains objects of classes Hamiltonian, WaveFunction, MonteCarlo and Pvec particles and uses them to perform VMC algorithm. This structure is very flexible. We developed six subclasses of WaveFunction, two subclasses of MonteCarlo and Hamiltonian has two modes.

Constructor of this class has the following structure:

```
System(Hamiltonian *H, WaveFunction *WF, MonteCarlo *MC, *Pvec parts)
{
  m_N = particles.size();
  m_d = particles[0].getNumberOfDimensions();
  m_H = move(H);
  m_WF = move(WF);
  m_MC = move(MC);
  m_particles = move(parts);
}
```

All arguments of constructor have type unique_ptr<class >. Method move() copies object from memory to a variable and then deletes old variable. As before variables that statr with m_ are private variables of the class.

This class implements runEquilibrationSteps function and runMetropolisSteps function. Whole VMC calculation is hidden inside these simple functions. In Algorithm 18 we present these functions. Essentially, runEquilibrationSteps function just runs first m VMC iterations without sampling and computes number of accepted steps. runMetropolisSteps runs n VMC iterations and samples all observables. It creates, and them returns sampler, which can be used in the main function of get all needed observables.

Algorithm 18 System Class functions

```
 1: procedure runEquilibrationSteps(double $sl$, int $m$)
 2:     int acceptedSteps $\leftarrow 0$;
 3:     for int i = 0; i < $m$; i++ do
 4:         acceptedSteps $\leftarrow$ m_MC.step($sl$, *m_WF, m_particles);
 5:     end for
 6:     return acceptedSteps;
 7: end procedure
 8: procedure runMetropolisSteps(double $sl$, int $n$)
 9:     Create Sampler class object:

                Sampler sampler = Sampler(m_N, m_d, $sl$, $n$);

10:     for int i = 0; i < $m$; i++ do
11:         bool acceptedSteps $\leftarrow$ m_MC.step($sl$, *m_WF, m_particles);
12:         sampler.sample(acceptedStep, this);
13:     end for
14:     sampler.computeAverages();
15:     return sampler.
16: end procedure
```

We have presented a class structure of VMC solver. In the following sections we will present serial and parallel main programs, that perform VMC and GD calculations.

3.10.10 Non-Interacting Serial VMC

Now we are ready to discuss how to actually perform a VMC calculation using presented class structure. We begin our discussion with non-interacting systems, i.e. systems with one variational parameter α. First thing is, obviously, to initialize parameters of the system:

- Number of burn-in VMC cycles m;
- Number of VMC cycles with sampling n;
- Number of particles N;
- Number of dimensions d;
- RNG Seed s;
- Initial variational parameter α;
- Step length sl.

This is the minimum parameters needed to run the simplest VMC calculation. In Algorithm we present the simplest VMC solver. Essentially, this algorithm creates objects of classes `Hamiltonian`, `WaveFunction` and `MonteCarlo` and initializes particle positions. Method `make_unique` creates a unique pointer to object of respective class. Afterward, we move all these object using `move` method. This way we avoid copying objects, and just copy-paste their memory locations to set up `System` class object. Then we begin countdown and run burn-in and VMC iterations. Finally, we output information about the system and GS energy to terminal. The class structure is complicated, but now it pays of: we can change ansatz, solver and Hamiltonian just by changing `H`, `WF` and `solver` objects. In the presented algorithm we chose non-interacting Hamiltonian, bosonic ansatz and Metropolis solver. If we want to switch to interacting Hamiltonian, we set `H` $\leftarrow$ `make_unique<HarmonicOscillator>(1,1)`. Recall, that second argument of `HarmonicOscillator` constructor is `bool Coulomb`. To change wave function, we set `WF` $\leftarrow$ `make_unique<Class>(`α`, `N`)`, where `Class` represents one one six `WaveFunction` subclasses. However, we need to be careful by doing so, because interacting ansatzes require two additional arguments. Finally, we can change solver by setting `solver` $\leftarrow$ `make_unique<MetropolisHastings>(move(rng))`.

Algorithm 19 Non-Interacting VMC solver

```
1: procedure VMC($N$, $d$, $n$, $m$, $\alpha$, $s$, $sl$)
2:     Construct RNG, rng ← make_unique<Random>($s$);
3:     particles ← setupRandomUniformInitialState($d$, $N$, *rng);
4:     Construct Hamiltonian, H ← make_unique<HarmonicOscillator>(1,0);
5:     Construct wave function, WF ← make_unique<Boson>($\alpha$, $N$);
6:     Construct solver, solver ← make_unique<Metropolis>(move(rng));
7:     Construct system,

           syst ← make_unique<System>(H, WF, solver, particles);

8:     Begin countdown high_resolution_clock start;
9:     Burn-in VMC: accsteps ← system.runEquilibrationSteps($sl$, $m$);
10:    VMC: sampler ← system.runMetropolisSteps($sl$, $n$);
11:    End countdown high_resolution_clock stop;
12:    Compute elapsed time, dur ← duration_cast(start - stop);
13:    Set sampler variable m_time ← dur;
14:    Print output to terminal, sampler.printOutputToTerminal(*syst).
15: end procedure
```

Possibility to change "block" of VMC so easily is the reason we implemented class structure. If we want to add additional ansatz, solver or Hamiltinian – we just need to implement respective subclasses and use them to create `System`, and perform VMC using it.

It is very easy to parallelize this VMC algorithm, and quite tricky to parallelize GD VMC algorithm. Therefore we will first present serial GD VMC algorithm, and then discuss how to parallelize it.

3.10.11 Non-Interacting Serial GD VMC

VMC algorithm estimates GS energy for a given value of variational parameter α. We aim to find optimal variational parameter, that minimizes GS energy. To optimize α we perform a GD Algorithm 20. This algorithm has two additional arguments: stopping criterion B and constant learning rate η. As we discussed earlier, GD algorithm is pretty simple. Sample O_1, O_2 and E and use them to find optimal variational parameter. In this algorithm we described GD algorithm for α optimization. Similar algorithm for Jastrow and Pade-Jastrow optimization will be discussed in the following sections.

Algorithm 20 Non-Interacting GD VMC solver

```
1: procedure GDVMC($N$, $d$, $n$, $m, \alpha$, $s$, $sl$, $B$, $\eta$,)
2:     Initialize l2 norm with a number bigger than $B$, l2 $\leftarrow$ 100;
3:     Initialize iteration counter, $i$ $\leftarrow$ 0 and max iterations number max;
4:     while grad > $B$ && $i <$ max do
5:         Perform VMC, VMC($N$, $d$, $n$, $m, \alpha$, $s$, $sl$);
6:         Collect local energy from sampler, $E \leftarrow$ sampler.getEnergy();
7:         Collect $O_1$ from sampler, $O_1 \leftarrow$ sampler.getO1alpha();
8:         Collect $O_2$ from sampler, $E \leftarrow$ sampler.getO2alpha();
9:         Compute gradient, grad $\leftarrow 2 \cdot (O_2 - EO_1)$;
10:        Update variational parameter, $\alpha \leftarrow \alpha - \eta \cdot$grad;
11:        Add one iteration to counter, $i$++;
12:        if grad $\leq B$ then
13:            print local energy $E$ to terminal.
14:        end if
15:    end while
16: end procedure
```

3.10.12 Interacting Serial GD VMC

Plain VMC algorithm for interacting systems is just Algorithm 19 with `Fermion` or `FermionNumerical` class. In addition, we need to specify `double` β for Pade-Jastrow ansatz or `Dvec` β_{ij} for Jastrow ansatz. We also need to set `bool mode = 0` to use Jastrow factor or `bool mode = 1` to use Pade-Jastrow factor.

- WF $\leftarrow$ `make_unique<Fermion>(`α,β_{ij}`, mode = 0` N`)` sets up interacting Jastrow fermionic ansatz;
- WF $\leftarrow$ `make_unique<Fermion>(`α,β`, mode = 1` N`)` sets up interacting Pade-Jastrow fermionic ansatz;
- WF $\leftarrow$ `make_unique<FermionNumerical>(`α,β_{ij}`, mode = 0` N`)` sets up numerical interacting Jastrow fermionic ansatz;
- WF $\leftarrow$ `make_unique<FermionNumerical>(`α,β`, mode = 1` N`)` sets up numerical interacting Pade-Jastrow fermionic ansatz.
- WF $\leftarrow$ `make_unique<FermionNumerical>(`α,β`, mode = 1` N`)` sets up numerical interacting Pade-Jastrow fermionic ansatz;
- H $\leftarrow$ `make_unique<HarmonicOscillator>(`ω`,1)` set up interacting HO Hamiltonial with oscillator frequncy ω.

`VMC(`N, d, n, m, α, s, sl, β, ω, `mode)` will denote Algorithm 19 with one of ansatzes described above and interacting Hamiltonian. We assume that for such system optimal parameter α is already found by solving non-interacting GD VMC. The goal of interacting GD VMC is to optimize variational parameters β or β_{ij}. GD VMC algorithm for Jastrow ansatz is presented in Algorithm 21. As we can see, it is pretty similar to non-ineracting GD VMC algorithm. The main difference is that now we update `np` parameters at a time, instead of just one. The other difference is that now l2 norm determines when we stop GD algorithm. Pade-Jastow GD algorithm is similar to non-interacting GD algorithm 20, but with Pade-Jastow ansatz and it is designed to minimize variational parameter β.

Algorithm 21 Interacting Jastrow GD VMC Solver

```
1:  procedure GDVMC($N$, $d$, $n$, $m, \alpha$, $s$, $sl$, $\beta_{ij}$, $\omega$, mode, $B$, $\eta$,)
2:      Initialize l2 norm with a number bigger than $B$, l2 ← 100;
3:      Initialize iteration counter, $i$ ← 0 and max iterations number max;
4:      Compute number of pairs, np ← $N(N-1)/2$;;
5:      while l2 > $B$ && $i <$ max do
6:          Initialize Dvec grad of size np with zeros;
7:          Perform VMC, VMC($N$, $d$, $n$, $m, \alpha$, $s$, $sl$, $\beta_{ij}$, $\omega$, mode);
8:          Collect local energy from sampler, $E \leftarrow$ sampler.getEnergy();
9:          Collect $O_1$ from sampler, $O_1$ ← sampler.getO1Jastow();
10:         Collect $O_2$ from sampler, $O_2$ ← sampler.getO2Jastow();
11:         Reset l2 ←0;
12:         for int k = 0; k < np; k++ do
13:             Compute gradient, grad[k] = $2 \cdot (O_2$[k] $- EO_1$[k]);
14:             Update variational parameters, $\beta_{ij}$[k] ← $\beta_{ij}$[k] - $\eta\cdot$ grad[k];
15:             Compute l2 norm squared, l2 ← l2 +grad[k]·grad[k];
16:         end for
17:         Compute l2 norm, l2 ← sqrt(l2);
18:         Add one iteration to counter, $i$++;
19:         if grad $\leq B$ then
20:             print local energy $E$ to terminal.
21:         end if
22:     end while
23: end procedure
```

3.10.13 Interacting OMP GD VMC

In this section we will discuss how GD VMC algorithm can be parallelized using OMP. OMP can be used only on shared memory systems, such as normal computers and notebooks. OMP provides an easy way to parallelize code. The easiest to parallelize are loops with independent iterations. This is exactly our case: we create `n_threads` copies of the system with different RNGs and let them run VMC in parallel. Benefit of this approach is that each system run only `n/n_threads` VMC iterations instead of `n` iterations. We require only two things from parallelized code – it runs faster and generates correct results. In Algorithm 22 we present OMP parallelized version of GD VMC algorithm for Jastrow ansatz. Inside **while** loop we create parallel environment using `#pragma omp parallel`. Inside this region, each thread will implement code inside the region. To make sure we create different systems, each thread get their own seeds `ts`. Threads use their seeds to create RNGs. RNGs created with different seed generate different sequences of random numbers. Afterward, each thread creates their own initial position of particles, and cretes their own solvers using their own RNGs. By doing so, we make sure that each thread generates different random numbers and theirs systems evolve in a different way. After threads are done with VMC computation, each thread collects observables.

Algorithm 22 Interacting Jastrow OMP GD VMC Solver

```
procedure OMPGDVMC($N$, $d$, $n$, $m$, $\alpha$, $s$, $sl$, $\beta_{ij}$, $\omega$, mode, $B$, $\eta$,)
    Initialize l2 norm with a number bigger than $B$, L2 $\leftarrow$ 100;
    Initialize iteration counter, $i \leftarrow$ 0 and max iterations number max;
    Compute number of pairs, np $\leftarrow N(N-1)/2$;
    #pragma omp parallel
        Nt = omp_get_num_threads();
    #pragma omp end parallel
    while L2 > $B$ && $i <$ max do
        Initialize Dvec grad of size np with zeros;
        Initialize Dvec O1, O2, TO1, TO2 of size np with zeros;
        Initialize double $E \leftarrow 0$, $TE \leftarrow 0$;
        #pragma omp parallel
            Get thread id, int tid = omp_get_thread_num();
            Create seeds for each thread, int ts = $s$ + tid;
            Perform VMC on each thread,

                VMC($N$, $d$, $n$/Nt, $m$/Nt, $\alpha$, ts, $sl$, $\beta_{ij}$, $\omega$, mode)

            Collect local energy on all threads , $E \leftarrow$ sampler.getEnergy();
            Collect O1 on all threads, O1 $\leftarrow$ sampler.getO1Jastow();
            Collect O2 on all threads, O2 $\leftarrow$ sampler.getO2Jastow();
            #pragma omp atomic
                $TE \leftarrow TE + E$;
            #pragma omp end atomic
            for int j = 0; j < np; j++ do
            #pragma omp atomic
                TO1[j] $\leftarrow$ TO1[j] + O1[j];
                TO2[j] $\leftarrow$ TO2[j] + O2[j];
            #pragma omp end atomic
            end for
        #pragma omp end parallel
        Reset L2 $\leftarrow$0;
        for int k = 0; k < np; k++ do
            Compute gradient, grad[k] = $2 \cdot$ (TO2[k]/Nt $-$ $E$TO1[k]/Nt);
            Update variational parameters, $\beta_{ij}$[k] $\leftarrow$ $\beta_{ij}$[k] - $\eta\cdot$ grad[k];
            Compute l2 norm squared, l2 $\leftarrow$ l2 +grad[k]$\cdot$grad[k];
        end for
        Compute L2 norm, L2 $\leftarrow$ sqrt(L2);
        Add one iteration to counter, $i$++;
    end while
end procedure
```

To obtain mean value of observables over all threads we need to somehow compute

$$\langle E\rangle = \frac{\sum_{i=\mathtt{tid}}^{\mathtt{Nt}} E^{(\mathtt{tid})}}{\mathtt{nt}}.$$

To do it correctly, we use `#pragma omp atomic`. This command makes sure that we compute sum of observable without double counting. We apply the same procedure for each element of observables O1 and O2. Finally, to obtain mean values of all observables we just divide them by Nt. After that we finalize parallel region and perform GD update as before.

3.10.14 Interacting MPI GD VMC

It is final section regrading computational implementation, and we will focus on MPI GD VMC algorithm. MPI is designed for distributed memory systems, meaning that all communication between processes need to be coded directly. For example, if first thread has computed value a and thread two need this value for its computation, thread one need to send it directly to thread two. We did not see such phenomenon for OMP, because it is designed for shared memory systems. This makes MPI parallelization more complicated, compared to OMP. However, if we manage to parallelize code using MPI, we can run this code on super computer or cluster. The most computation heavy programs and algorithms implement MPI parallelization. This is one of the reasons we use C++ as programming language, it can be relatively easy to parallelize, compared to other programming languages. Python, for example, can be paralllized, but it is done much harder. For C++ one just needs to install API and use it.

In Algorithm 23 we explain our MPI implementation. When using MPI, it is highly recommended to initialize parallel environment only one time. Therefore, first we create parallel environment, and then save number of threads and thread rank for each thread. Afterward, we create unique seed for each rank and use it to create unique systems. Each thread evolve their own system using their own RNGs, as in OMP case. When all threads done computing, each thread collects their observables. To send these observables from each thread to master thread we use MPI_Reduce() and MPI_Allreduce functions. After MPI_Reduce() is called, each thread sends their local energies E to master thread, and mater thread save their sum to TE variable. Function MPI_Allreduce does the same, but for a vector. Therefore, we use this function to fill vector TO1 in the following way

$$\texttt{TO1} = \left(\sum_{i=\texttt{rank}}^{\texttt{size}} \texttt{O1}_0^{(\texttt{rank})}, \cdots, \sum_{i=\texttt{rank}}^{\texttt{size}} \texttt{O1}_{\texttt{np}}^{(\texttt{rank})} \right).$$

Vector TO2 is filled the same way. After we send observables from each thread to master thread, it performs GD update. It is very important so send updated vector of variational parameters β_{ij} from master thread to all threads. To avoid double counting, we perform GD update on master thread, and then broadcast β_{ij} to all threads using MPI_Bcast(beta.data()). We do the same with L2 score L2. Be broadcasting L2 score we make sure that once L2 norm is smaller than B all threads exit the loop and parallel environment is finalized.

Algorithm 23 Interacting Jastrow OMP GD VMC Solver

```
1: procedure MPIGDVMC($N$, $d$, $n$, $m$,$\alpha$, $s$, $sl$, $\beta_{ij}$, $\omega$, mode, $B$, $\eta$,)
2:     Initialize L2 norm with a number bigger than $B$, L2 $\leftarrow$ 100;
3:     Initialize iteration counter, $i$ $\leftarrow$ 0 and max iterations number max;
4:     Create variables int size, rank;
5:     Set up parallel environment, MPI_Init(&argc, &argv);
6:     Assign number of threads to size variable,

                   MPI_Comm_size(MPI_COMM_WORLD, &size);

7:     Assign thread number of each thread to rank variable,

                   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

8:     Create seeds for each thread, $s \leftarrow s\cdot$rank;
9:     while L2 > $B$ && $i <$ max do
10:        Initialize Dvec grad of size np with zeros;
11:        Initialize Dvec O1, O2, TO1, TO2 of size np with zeros;
12:        Initialize double $E \leftarrow 0$, $TE \leftarrow 0$;
13:        Perform VMC on each thread,

               VMC($N$, $d$, $n$/size, $m$/size, $\alpha$, $s$, $sl$, $\beta_{ij}$, $\omega$, mode)

14:        Collect local energy on all threads , $E \leftarrow$ sampler.getEnergy();
15:        Collect O1 on all threads, O1 $\leftarrow$ sampler.getO1Jastow();
16:        Collect O2 on all threads, O2 $\leftarrow$ sampler.getO2Jastow();
17:        Save sum of $E$ from each thread to $TE$, MPI_Reduce(&$E$, &$TE$, MPI_SUM);
18:        Save sum of each O1 element to TO1, MPI_Allreduce(&O1, &TO1, MPI_SUM);
19:        Save sum of each O2 element to TO2, MPI_Allreduce(&O2, &TO2, MPI_SUM);
20:        if rank == 0 then
21:            Reset L2 $\leftarrow$0;
22:            for int k = 0; k < np; k++ do
23:                grad[k] = $2\cdot$(TO2[k]/size $- E\cdot$ TO1[k]/size);
24:                Update variational parameters, $\beta_{ij}$[k] $\leftarrow$ $\beta_{ij}$[k] - $\eta\cdot$ grad[k];
25:                Compute l2 norm squared, l2 $\leftarrow$ l2 +grad[k]$\cdot$grad[k];
26:            end for
27:            Send beta from master thread to all other threads, MPI_Bcast($\beta_{ij}$.data());
28:            Compute L2 norm, L2 $\leftarrow$ sqrt(L2);
29:            Add one iteration to counter, $i$++;
30:            Send L2 norm to all threads using MPI_Bcast(&l2_norm;
31:        end if
32:    end while
33:    Finalize parallel environment, MPI_Finalize();
34: end procedure
```

3.11 Methods

To perform this research a lot of helping software was used. In this section we explain what was used and for what purposes.

- `C++` was used as main programming language. `C++` programs were used to solve VMC and generate results;
- `OMP` and `MPI` were used to parallelize VMC solver;
- `Autodiff` API was used to develop VMC solvers with numerical differentiation in reverse mode;
- `Eigen` library was used to compute Slater determinants;
- `Python` was used as a helping programming language. All figures in 4 were generated using `matplotlib` library for python;
- `Pandas` python library was used to store the data;
- `Numpy` python library was used to perform arithmetic operation on vectors and perform analysis of the data;
- `Seaborn` python library was used to generate heatmaps;
- `Github` was used as a software version control and cloud space for the program directory;
- `Arxiv` was used to find scientific articles;
- `Arxivxplorer` was used to find scientific articles;
- `ChatGPT` was used to debug programs, improve language, cite sources using Biblitex and find sources.
- `LaTex` was used to write the thesis;
- `Overleaf` was used as a LaTex typesetting system;
- `Forest` LaTex library was used to generate diagrams 3.4 and 3.5;
- `Algorithmic` LaTex library was used to design all agorithms;
- `Method draw` open source SVG editor was used to generate diagrams 2.1, 3.1, 3.2 and 3.3.

All computation are made using `2,5 GHz Quad-Core Intel Core i7` processor with `16 GB 1600 MHz DDR3` memory. In all programs we use `seed = 2025`.

Chapter 4

Results and Discussion

4.1 Non-Interacting Bosons

First and the simplest system we study is a system of N non-interacting bosons. In Appendix 6.4 we derived analytical solution for 3D system, which is given by formula (6.32). In N dimensions GS energy as a function of variational parameter α is

$$E_0 = \frac{Nd}{2}\left(\alpha + \frac{1}{4\alpha}\right). \tag{4.1}$$

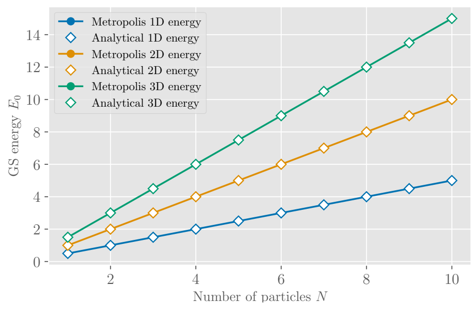
Additionally, optimal variational parameter alpha for any number of particles in any number of dimensions is $\tilde{\alpha} = 1/2$. GS energy is then given by

$$E_0(\tilde{\alpha}) = \frac{Nd}{2}. \tag{4.2}$$

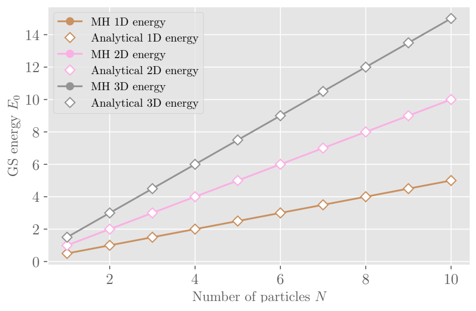
We chose this system as a starting point of the research, because it has analytical solution we can compare numerical results with. If these results align, then developed VMC simulation works as intended.

4.1.1 VMC algorithm

As a first step, we want to see if Metropolis and Metropolis-Hastings algorithms estimate GS energy correctly. On Figure 4.1 we present results of Metropolis and Metropolis-Hastings VMC simulation for $1 - 10$ particles in $1,\ 2$ and 3 dimensions.



(a) Metropolis GS energy



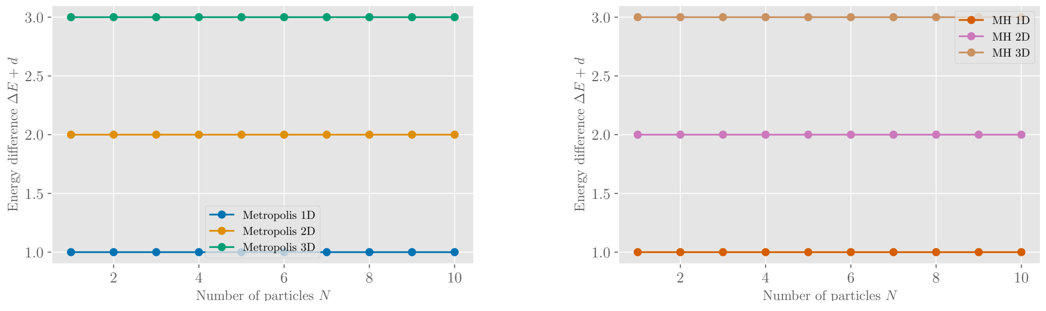
(b) Metropolis-Hastings GS energy

Figure 4.1: Metropolis, Metropolis-Hastings and analytical GS energy E_0 as a function of number of particles N in 1, 2 and 3 dimensions.

For this simulation we used step size $h = 1$, $m = 10^3$ burn-in iterations and $n = 10^4$ VMC iterations. Variational parameter was set to its optimal values $\tilde{\alpha} = 1/2$ and oscillator frequency was set to one, $\omega = 1$. In all upcoming simulations we will use $\omega = 1$, until we specify its value explicitly. As we can see from the data on these plots, numerical results align perfectly with analytical solution. These result were generated by `Boson` class that uses analytical expressions for derivatives.

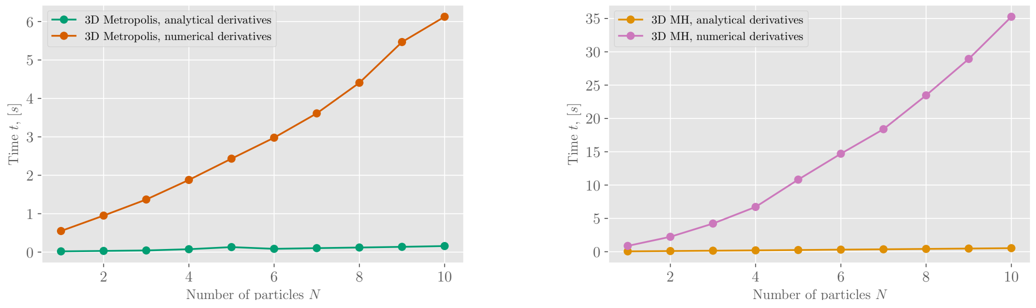
Next, we want to make sure `BosonNumerical` class generates the same correct result. On Figure 4.2 we show difference between GS energy computed using analytical derivatives and numerical derivatives $\Delta E_0 = E_0^{\text{AD}} - E_0^{\text{ND}}$ in $1-3$ dimensions. To distinguish dependencies we add number of dimensions d to energy difference ΔE_0. Data from the figures shows that we obtain the same GS energy using both analytical and numerical derivatives. This is a very important test, because if these two results misalign, there is a big probability that analytical derivatives were implemented incorrectly. However, if result align, it is still not a 100% guaranty that implementation is correct, so we need to be careful with result analysis.

From this point forward we will refer to algorithms that use analytical derivative expression as analytical algorithms and numerical algorithms for algorithms that use numerical derivatives.



(a) Metropolis $\Delta E_0 + d$ (b) Metropolis-Hastings $\Delta E_0 + d$

Figure 4.2: Metropolis and Metropolis-Hastings GS energy difference $\Delta E_0 = E_0^{\text{AD}} - E_0^{\text{ND}}$ plus number of dimensions d as a function of number of particles N in 1, 2 and 3 dimensions.

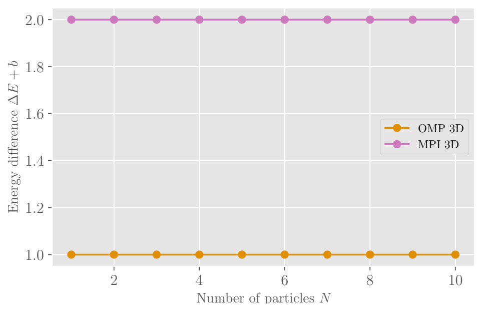


(a) 3D Metropolis execution time (b) 3D Metropolis-Hastings execution time

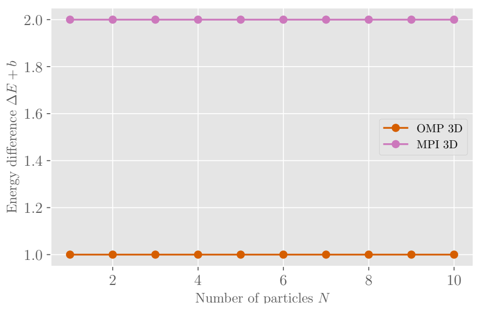
Figure 4.3: Metropolis and Metropolis-Hastings algorithms execution time t is seconds as a function of number of particles N in 3D.

We also want to study analytical and numerical algorithms execution time. Algorithms runtime for 3D systems is presented on figure 4.3. Although these algorithms generate the same correct results, numerical algorithms execution time grows much faster then analytical ones. This simulation used only $n = 10^4$ VMC iterations, and execution time pf numerical algorithms is enormous. And bosonic system is a system with the simplest ansatz, every system we will study later require more computations. Therefore, our strategy is to make sure that analytical and numerical algorithms generate the same results and use analytical ones for actual computations.

Now we want to focus on MPI and OMP parallel VMC algorithms. We want to see what speed they provide and make sure that they still generate correct results. Figure 4.4



(a) Metropolis $\Delta E_0 + b$

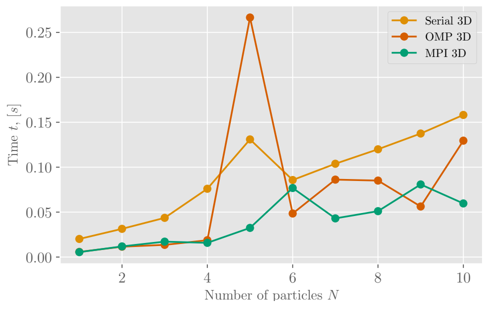


(b) Metropolis-Hastings $\Delta E_0 + b$

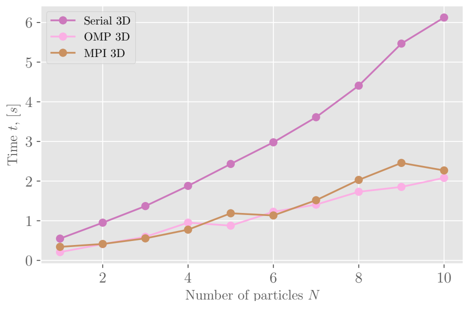
Figure 4.4: Metropolis and Metropolis-Hastings GS energy difference $\Delta E_0 = E_0^{\text{an}} - E_0^{\text{parallel}}$ plus parallel identifier b in 3D. $b = 1$ for OMP and $b = 2$ for MPI.

shows energy difference ΔE_0 between analytical and parallel algorithms. To distinguish data we added $b = 1$ to ΔE_0 for OMP and $b = 2$ for MPi. We can clearly see that parallel algorithms compute the same GS energy as analytical algorithms, meaning that they generate correct results.

Now, when we made sure that parallel algorithms generate correct results, we want to study algorithm runtime for all algorithms and compare them. We have seen already that numerical algorithms take more time to estimate GS energy. Parallel algorithms must speed computations up.



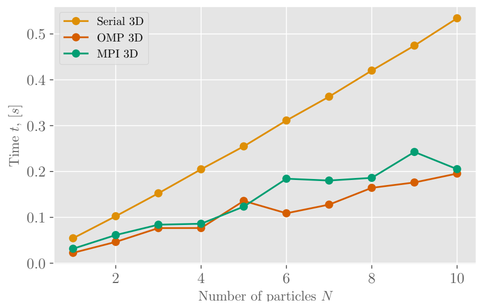
(a) Serial 3D Metropolis execution time



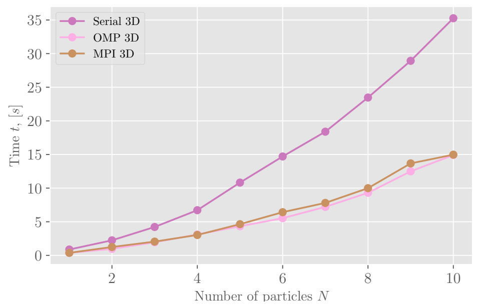
(b) Numerical 3D Metropolis execution time

Figure 4.5: Parallel and serial Metropolis algorithms execution time t is seconds as a function of number of particles N in 3D.

On figure 4.5 we show execution time for serial and parallel Metropolis algorithms for $n = 10^4$ VMC iterations. Analytical Metropolis algorithm with such small number of iterations is itself pretty fast, so form figure 4.5a we conclude that parallel algorithms do not give essential speed up when serial execution time is low. We can even see fluctuations, meaning that runtime of parallel algorithm can be bigger then its serial counterpart. When algorithm runtime is only 0.35 seconds, there is no need in parallelization. Numerical Metropolis algorithm is more computationally expensive, and from Figure 4.5b we see that for all N parallel runtime it approximately three times faster then of serial algorithm. We chose such small number of VMC iterations on purpose, to show that parallelization only make sense when algorithm is computationally expensive.



(a) Serial 3D MH execution time

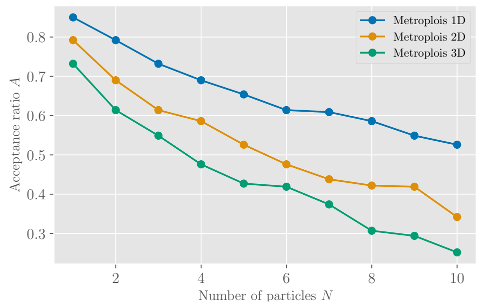


(b) Numerical 3D MH execution time

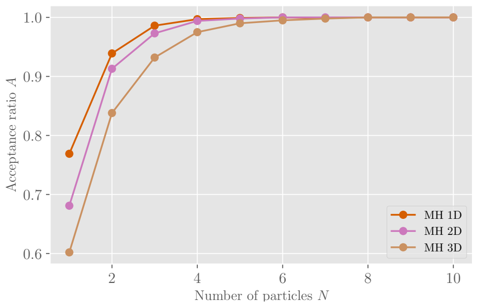
Figure 4.6: Parallel and serial Metropolis-Hastings algorithms execution time t is seconds as a function of number of particles N in 3D.

Metropolis-Hasting algorithm runtime is displayed on Figure 4.6. In this case, eve for analytical algorithm parallel versions give sufficient speedup. When we consider numerical algorithm, reducing runtime from 35 seconds to 15 is sufficient. In all cases, MPI and OMP parallelized algorithms give approximately the same speedup, so we can not choose "a better" way of parallelization, they both give great speedup.

Next point of interest is is how step size h and burn-in acceptance ratio A are related. This information will help us develop an intuition on what values of h should be chosen in order to obtain good result.



(a) Metropolis Acceptance rate



(b) Metropolis-Hastings Acceptance rate

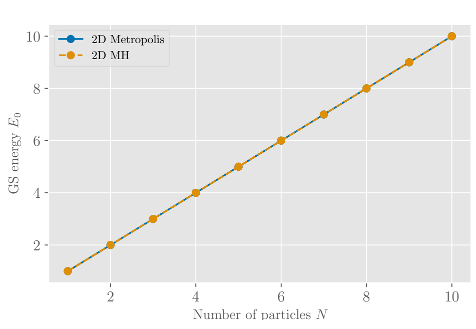
Figure 4.7: Metropolis and Metropolis-Hastings algorithms acceptance rate A as a function of number of particles N in 1, 2 and 3 dimensions for a fixed step size $h = 1$.

On Figure 4.7 we see two completely different dependencies. For Metropolis Algorithm acceptance rate decreases as number of particles increase, while Metropolis-Hastings' acceptance ration converges to one when N increases. There is nothing mysterious in this, because we compute acceptance ratio differently for these algorithms. Metropolis algorithm move all particles and then accepts or rejects the step. Om the other hand, Metropolis algorithm moves one particle, accepts or rejects this step and repets it for all particles. If only one particle's step has been accepted, we say that this step is accepted. Therefore, when N is big, we expect to see that at least one particle's step will be accepted and acceptance rate would be equal to one. For Metropolis algorithm we can make a conclusion that for larger systems smaller step size should be chose. Unfortunately, this data does not give us a hint on how to choose step size for Metropolis-Hastings algorithm.

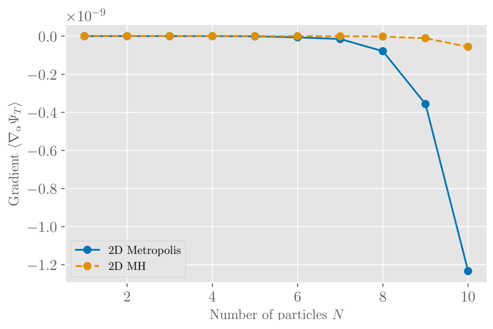
So far we only studied pure VMC algorithm with optimal variational parameter found analytically. It is time to switch to GD VMC algorithm and see if it able to find optimal variational parameter that minimizes GS energy.

4.1.2 GD VMC algorithm

First of all, we need to convince ourselves that GD algorithm converges when variational parameter is close to its optimal value. To do it, we execute GD VMC algorithm with $\alpha = 1/2$ and $n = 10^4$ VMC iterations. Figure 4.8 contains data of such simulation.



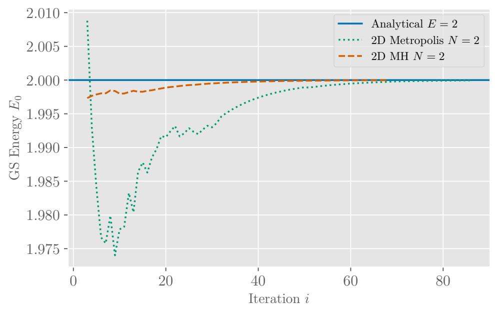
(a) 2D GS energy



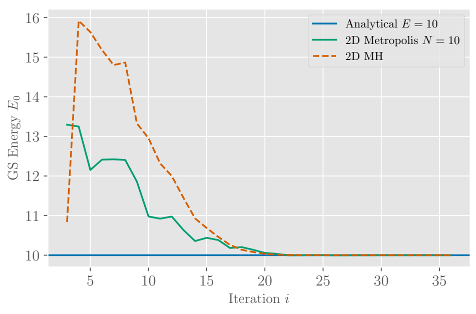
(b) 2D gradient

Figure 4.8: GS energy E_0 and gradient $\langle \nabla_\alpha \Psi_T \rangle$ as a function of number of particles N in 2D with optimal variational parameter $\alpha = \tilde{\alpha} = 1/2$.

For all N particles GD algorithm converged after one iteration, and GS energy aligns with analytical solution. Moreover, gradients, commuted for GD update are values proportional to 10^{-9}, which is a great result.



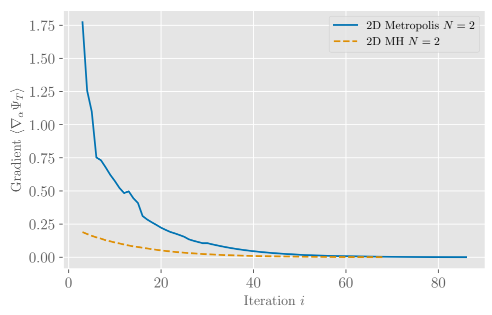
(a) $N = 2$ GS energy



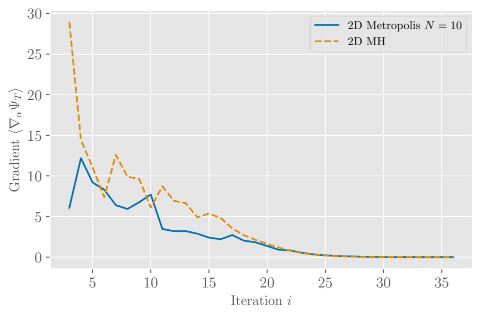
(b) $N = 10$ GS energy

Figure 4.9: GS energy E_0 of $N = 2$ and $N = 10$ particles as a function of number of iterations i in 2D. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $h = 1$, $\eta = 10^{-2}$ and $B = 10^{-4}$.

Now when we know that GD algorithm converges at optimal variational parameter, we are ready to perform real GD simulations, when initial variational parameter is not optimal. If we did not have any information about the system, the natural choice of initial variational parameter would be small value close to zero or one.

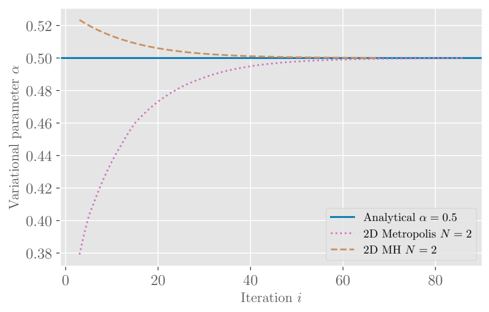


(a) $N = 2$ variational parameter α

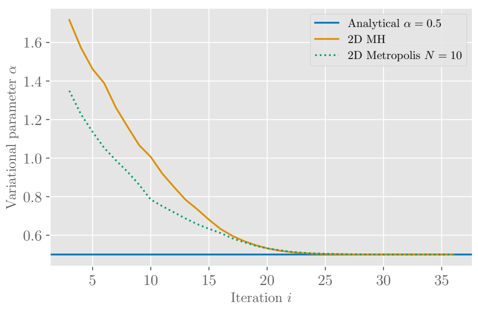


(b) $N = 10$ variational parameter α

Figure 4.10: Variational parameter α of $N = 2$ and $N = 10$ particles as a function of number of iterations i in 2D. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $h = 1$, $\eta = 10^{-2}$ and $B = 10^{-4}$.



(a) $N = 2$ gradient



(b) $N = 10$ gradient

Figure 4.11: Gradient $\langle \nabla_\alpha \Psi_T \rangle$ of $N = 2$ and $N = 10$ particles as a function of number of iterations i in 2D. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $h = 1$, $\eta = 10^{-2}$ and $B = 10^{-4}$.

Figures 4.9, 4.10 and 4.11 were generated by running OMP simulation for $N = 2$ and $N = 10$ 2D particles with the following parameters:

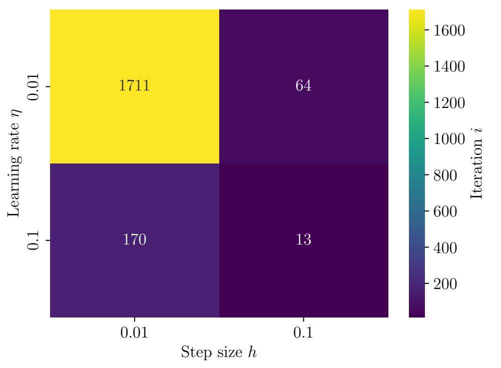
- Initial variational parameter $\alpha_0 = 0.1$;
- Number of VMC iterations $n = 10^3$;
- Number of burn-in iterations $m = 10^2$;
- Step size $h = 1$;
- Learning rate $\eta = 10^{-2}$;
- Stopping criterion $B = 10^{-3}$.

All plots exclude first two iterations for better visibility. First observation is that both $N = 2$ and $N = 10$ systems converged after small number of iterations. We chose such small number of burn-in and VMC iterations, because GD VMC is designed to find optimal parameters. If we are able to find optimal parameters using only 110 VMC iterations - it is a good sign. All plots show that all variables fluctuate strongly first iterations, and then slowly begin converging their true values. With such parameters Metropolis and Metropolis-Hastings algorithm need approximately the same number of iteration to converge. When Metropolis-Hastings algorithm does not converge in less iterations than Metropolis, it is better to use Metropolis algorithm, because it is less computationally expensive.

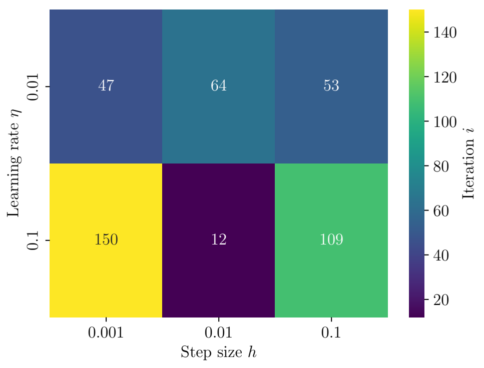
During first iteration gradient was big, and therefore variational parameter and GS energy changed their values radically. For example, for $N = 10$ energy jumped from $E \approx 11$ to $E \approx 16$, and started to converge slowly. In the end, GD found optimal variational parameter and GS energy with these parameters is very close to analytical GS energy with $\Delta E_0 = E_0^{\text{analyt}} - E_0^{\text{VMC}} = 10^{-4}$.

For this system value $\langle \nabla_\alpha \Psi_T \rangle = \langle O_2 \rangle - \langle E_L \rangle \langle O_1 \rangle$ is positive for $\alpha < \tilde{\alpha}$ and negative for $\alpha > \tilde{\alpha}$. This means that GD update of the form $\alpha \leftarrow \alpha - \langle \nabla_\alpha \Psi_T \rangle$ updates variational parameter incorrectly, new parameter is further from its optimal value than old. Therefore GD update for this system need to be changed to $\alpha \leftarrow \alpha + \langle \nabla_\alpha \Psi_T \rangle$. Using this rule variational parameter converges to its optimal value. We do not know apriori what value gradient has, we can only check it by running simulations.

Magnitude of fluctuations is determined by combination of step size h and learning rate η. This is the next topic of interest. We want to gain intuition for coming systems what values should we use for step size and learning rate in order to gain stable and correct results. If we choose relatively big parameters, GD will converge in small number of iteration or it will explode. Safe choice it to initialize parameters with small values, because GD will likely converge. But using small parameters can take thousands of iteration for GD to converge. This is not a problem for such simple system, but even one iteration of 12 interacting Fermions can take hours, therefore we need to find a way to pick parameters as big as possible, so that GD does not explode and does not take eternity to converge. We study GD convergence speed with different learning rates and



(a) Metropolis heatmap



(b) Metropolis-Hasting heatmap

Figure 4.12: GD iterations until convergence heatmap as a function of step size h and learning rate η for Metropolis and Metropolis-Hastings algorithms. Five particles in two dimensions.

step sizes for a system of five 2-dimensional particles with $n = 10^4$, $m = 10^3$, $\alpha_0 = 1$ and $B = 10^{-3}$. Heatmaps on Figure 4.12 show simulation results for such systems. Results are quite interesting. We can clearly see that Metropolis algorithm converges fastest with $\eta = 0.1$ and $h = 0.1$. All other configurations need more iterations to converge, especially when $h = 0.01$.

Situation with Metropolis-Hastings algorithm is not so clear. It converges in approximately 50 iterations for arbitrary h when $\eta = 0.01$. We can also see that for $\eta = h = 0.1$ MH converges in 109 iterations, while these parameters are best for Metropolis algorithm. When $\eta = 0.001$ and $h = 0.01$ MH does not converge, because we sat maximal number of iterations $i_{\text{max}} = 150$. Variational parameter gets stuck in the loop and jumps between $\alpha = 0.632668$ and $\alpha = 0.424532$ every iteration. This shows that GD algorithm with constant learning rate can fail if parameters we choose "unlucky" parameters.

From results of this simulation we can conclude that for Metropolis algorithm we need to choose the biggest η and h that do not blow up variational parameter. While Metropolis-Hastings works better with smaller values of h and η in range $(0.001, 0.1)$.

Finally, we want to make sure that all GD algorithms generate correct results. In Table 4.1 we present simulation of system with $N = 12$ particles in 2 dimension with $\eta = 10^{-2}$, $h = 10^{-1}$, $B = 10^{-3}$ and starting variational parameter $\alpha_0 = 0.6$. Numerical and analytical algorithms generate the same results and converge in the same number of iterations. Every algorithm computes correct GS energy $E_0 = 12$. Numerical algorithms runs approximately 3 times faster than analytical ones and MPI shows slightly better performance than OMP for every algorithm.

	Metropolis	Num. Metropolis	Metropolis-Hastings	Num. MH
Serial	0.16 s / 30	4.5 s / 30	0.58 s / 15	35 s / 15
OMP	0.05 s / 35	1.7 s / 25	0.2 s / 15	12 s / 15
MPI	0.048 s / 23	1.5 s / 23	0.16 s / 16	11.2 s / 16

Table 4.1: Comparison of GD runtimes and iterations until convergence for different Metropolis variants and parallel backends.

4.2 Non-Interacting Fermions

In this section we will consider first three shells of 2-dimensional non-interacting fermions. We derived analytical solution for the first closed shell, and they are given by formulas (6.38), (6.39) and (6.40):

$$E_0 = 2\alpha + \frac{1}{2\alpha}, \quad \tilde{\alpha} = \frac{1}{2}, \quad E_0(\tilde{\alpha}) = 2.$$

Salter determinant form of trial wave function makes it impossible to find expression for $E_0(\alpha)$ for higher closed shells. However, we know $E_0(\tilde{\alpha})$. Recall that for harmonic potential energy is given by formula (2.22),

$$E_{n_x,n_y} = (n_x + n_y + 1).$$

First shell is composed of two fermion in state $n = 0$ and four electrons in state $n = 1$. Therefore, GS energy of first shell is

$$E_0^{(N=6)} = 2 \cdot 1 + 4 \cdot 2 = 10.$$

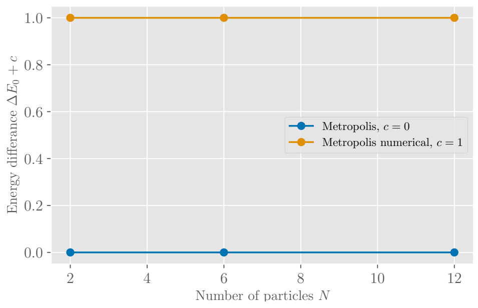
Likewise, GS energy of the third shell is

$$E_0^{(N=12)} = 2 \cdot 1 + 4 \cdot 2 + 6 \cdot 3 = 28.$$

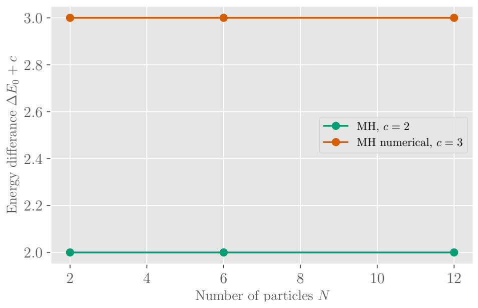
For idealized non-interacting case we can compare numerical results with analytical ones, which helps to develop VMC implementation. When VMC solver is tested on analytical system, we can move on to systems of interacting fermions.

4.2.1 VMC algorithm

We start by simulating all three closed shells with optimal variational parameters. Energy difference $\Delta E_0 = E_0^{(\mathrm{anayt})} - E_0^{(\mathrm{alg})}$ is plotted on the Figure 4.13. We ran this simulation with parameters $n = 10^4$, $m = 10^3$ and $h = 0.1$. For optimal value energy converges to its analytical value even in 10^4 VMC iterations. Data from this figure shows that both Metropolis and Metropolis-Hastings algorithm are implemented correctly and they generate correct results. During every VMC iteration program



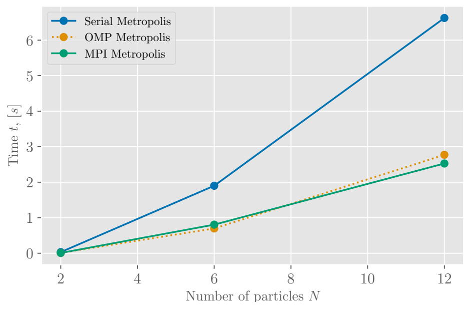
(a) Metropolis $\Delta E_0 + c$



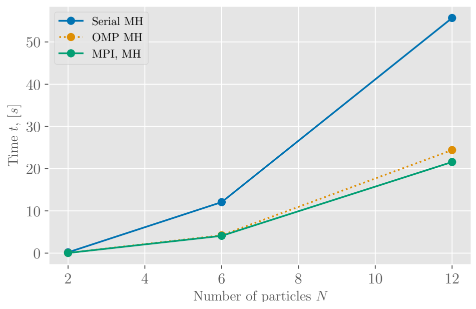
(b) Metropolis-Hastings $\Delta E_0 + c$

Figure 4.13: Metropolis and Metropolis-Hastings GS energy difference $\Delta E_0 = E_0^{\mathrm{an}} - E_0^{\mathrm{alg}}$ plus algorithm identifier c for analytical and numerical algorithms.

computes determinants, which takes much more time, compared to bosonic ansatz. On Figures 4.14 and 4.15 we show execution time of analytical and numerical algorithms using for the system with the same parameters. Data from these figures shows that



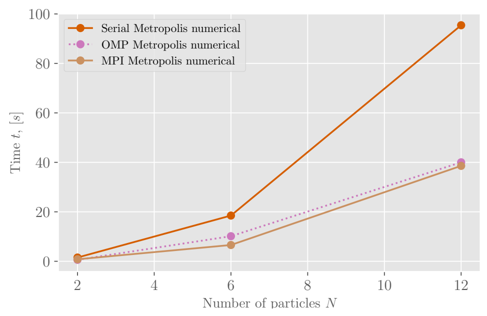
(a) Analytical Metropolis execution time



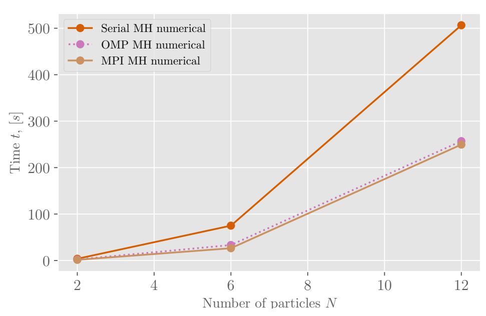
(b) Analytical MH execution time

Figure 4.14: Parallel and serial analytical Metropolis algorithms execution time t is seconds as a function of number of particles N.

MH algorithm is much more costly – its runtime is approximately ten times higher than Metropolis runtime. Moreover, execution time for both algorithms is approximately 100 times higher than for bosonic systems. However, parallelized programs give a decent speed up. Numerical algorithm have even bigger runtime. Numerical serial VMC computes 10^4 VMC iterations in 500 second, which is huge, compared with parallel analytical 20 seconds. Obviously, we want the algorithm to be executed as fast as possible, therefore parallel analytical algorithms are our choice. Once we make sure that numerical, analytical and parallel analytical results are correct, we will use parallel analytical algorithms for heavy computations.



(a) Numerical Metropolis execution time



(b) Numerical MH execution time

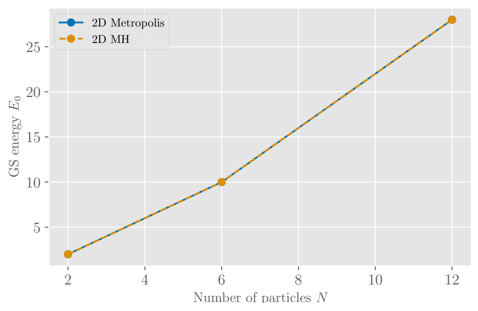
Figure 4.15: Parallel and serial numerical Metropolis algorithms execution time t is seconds as a function of number of particles N.

Another observation is that for non-interacting systems both Metropolis and MH algorithms generate **exact** results. In situations like this, it is more convenient to choose algorithm which runs faster. In this case Metropolis shows better performance.

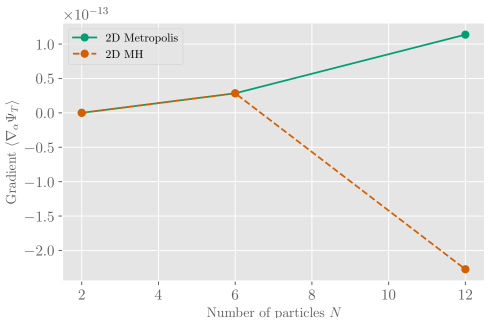
4.2.2 GD VMC algorithm

We start start to GD VMC algorithm by making sure it converges when $\alpha = \tilde{\alpha}$.

Figure 4.16 demonstrates data generated with parameters $h = 0.1$, $\eta = 0.01$, $n = 10^3$ and $m = 10^2$. GS energy is accurate using only 110 VMC iterations, it converges to the fanatical value. Gradient has values of 10^{-13} order. This switches that when gradient is minimized, GS energy gains correct value.



(a) GS energy



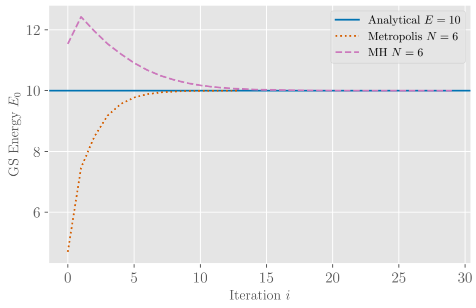
(b) Gradient

Figure 4.16: GS energy E_0 and gradient $\langle \nabla_\alpha \Psi_T \rangle$ as a function of number of particles N with optimal variational parameter $\alpha = \tilde{\alpha} = 1/2$.

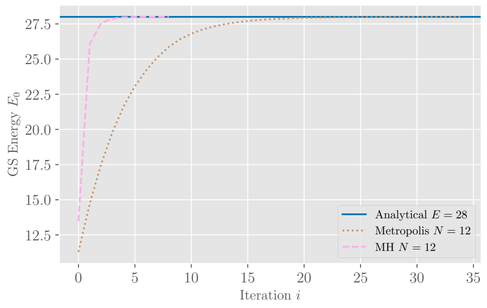
Next, we want to study how gradient, variational parameter and GS evolve during GD VMC algorithm. We will focus on second and third closed shells.

For ferminic system GD update of the form $\alpha \leftarrow \alpha - \eta \cdot \langle \nabla_\alpha E_L \rangle$ is correct. GD of this form minimizes α because $\langle \nabla_\alpha E_L \rangle$ is negative when $\alpha > \tilde{\alpha}$ and negative when $\alpha < \tilde{\alpha}$.

Figure 4.17 shows how GS energy changes during GD iterations. System parameters are specified in figure caption. For this simulation we used OMP algorithms. We chose different parameters for different configurations to show how many iteration GD needs to optimize variational parameter α. During first iterations energy is far from its true value, but only after $5 - 10$ GD iterations it starts to converge. The reason for such huge fluctuation during first iterations is that number of burn-in iterations is too small, and particles do not reach equilibrium state. However, as GD goes, particles reach equilibrium state and energy becomes accurate.



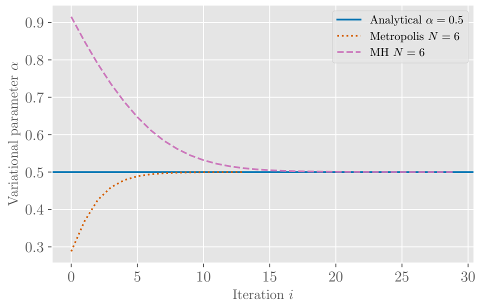
(a) $N = 6$ GS energy for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.01$



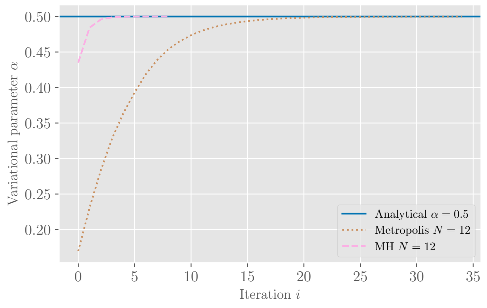
(b) $N = 12$ GS energy for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.001$

Figure 4.17: GS energy E_0 of $N = 2$ and $N = 12$ particles as a function of number of iterations i. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$ and $B = 10^{-3}$

Figures 4.18 and 4.19 demonstrate how variational parameter and sample mean gradient change during GD iterations. Essentially, we observe the same picture as for energy – far from optimal value during first iterations and start to converge after approximately 10 iterations. It is also important to note that after fist iterations convergence is smooth, we do see sufficient fluctuations after some q number of iterations. It means that after q-th iteration gradient becomes smaller with every iteration and variational parameter with energy becomes closed to their true values.



(a) $N = 6$ variational parameter α for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.01$

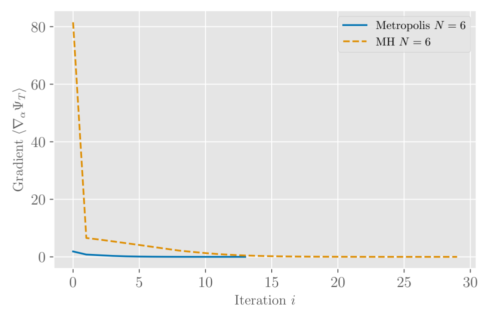


(b) $N = 10$ variational parameter α for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.001$

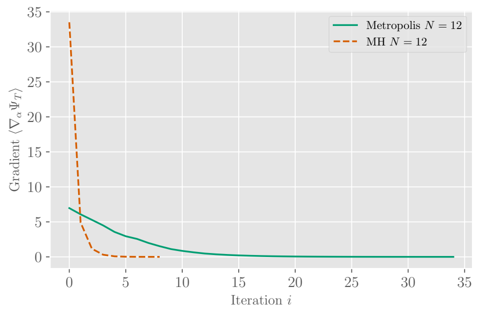
Figure 4.18: Variational parameter α of $N = 2$ and $N = 12$ particles as a function of number of iterations i. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$ and $B = 10^{-3}$

Additionally, we can note that for $N = 6$ Metropolis converges faster, while for $N = 12$ MH converges much faster. It is an important observation that even for $N = 12$ GD can converge in less than 10 iterations. However, it is really dependent on parameters of the systems: number of VMC steps, learning rate and step size. With number of VMC steps everything is clear, the more step we use – the better. Combination of learning rate and step size is a black box, we can only find how to choose parameters for a given system by tuning them.

If we compare step size and leaning rates for bosonic and fermionic systems, we can notice that for fermionic systems they are initialized with smaller values. The reason for that is that gradient explodes if parameters are chosen "too" big for a given system.



(a) $N = 6$ gradient for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.01$

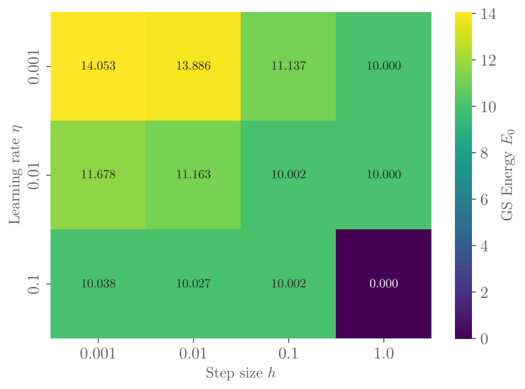


(b) $N = 10$ gradient for $h_{\mathrm{M}} = 0.1$, $h_{\mathrm{M}H} = 0.001$

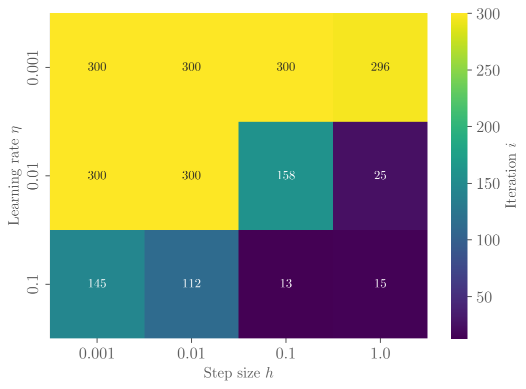
Figure 4.19: Gradient $\langle \nabla_\alpha \Psi_T \rangle$ of $N = 2$ and $N = 12$ particles as a function of number of iterations i. System parameters: $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$ and $B = 10^{-3}$

Therefore, now we will run simulations with fixed number of iterations, vary learning rate and step size and study how gradient, energy and variational parameters change. Following heatmaps were generated by running OMP GD VMC with $n = 10^3$ and $m = 10^2$.

We start by considering $N = 6$ system simulated with Metropolis algorithm. Heatmaps on Figure 4.20. If energy heatmap contains 0, it means that GD blew up and did not converge. If iteration heatmap contains 300, it means that GD did not converge in 300 iterations. Energy and iteration heatmaps help to determine optimal step size and learning rate for particular system and solving algorithm.



(a) $N = 6$ Metropolis energy heatmap

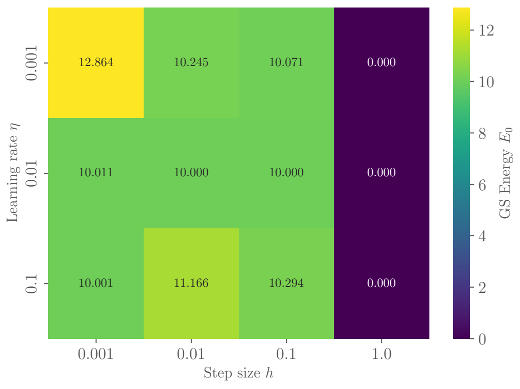


(b) $N = 6$ Metropolis iteration heatmap

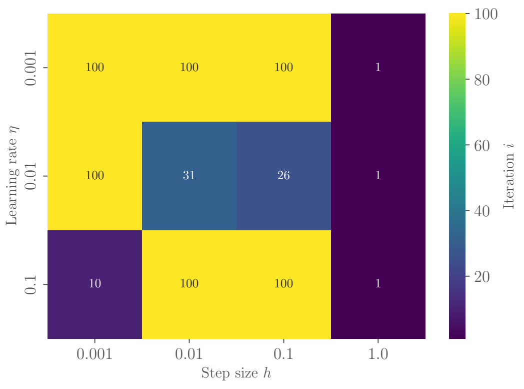
Figure 4.20: GD iterations until convergence and GS energy heatmaps as a functions of step size h and learning rate η for $N = 6$ Metropolis algorithm

Heatmaps analysis is pretty simple. Fist, we discard $(h,\ \eta)$ for with GD blew up. In this case it is $(1,\ 0.1)$ pair. Then we discard pairs for which GD did not converge and energy is far from it's optimal value. In this case it is all $(h,\ \eta)$ pairs for which $i = 300$. Now only "green" values in energy heatmaps left. All pairs $(h,\ \eta)$ are acceptable, they generate correct results. Data from iteration heatmap can help us choose such $(h,\ \eta)$ pairs that generate correct GS energy using fewest iterations. In this case, two pairs converge fast and generate correct results: $(h = 0.1,\ \eta = 0.1)$ and $(h = 1,\ \eta = 0.01)$.

The same heatmaps for $N = 6$ Metropolis-Hastings algorithm is presented on Figure 4.21. For this simulation we sat $i = 100$ as a maximal number of iterations. We can immediately see that step size $h = 1$ is too big for MH algorithm. Optimal pair in this case is $(h = 0.001,\ \eta = 0.1)$.



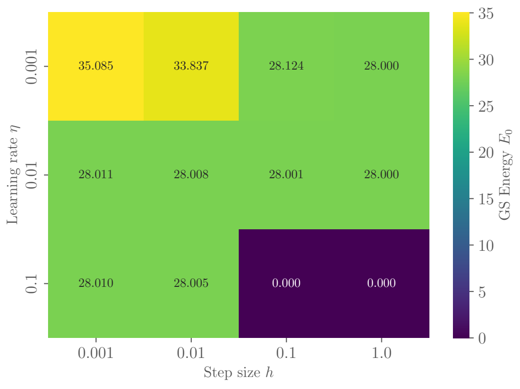
(a) $N = 6$ MH energy heatmap



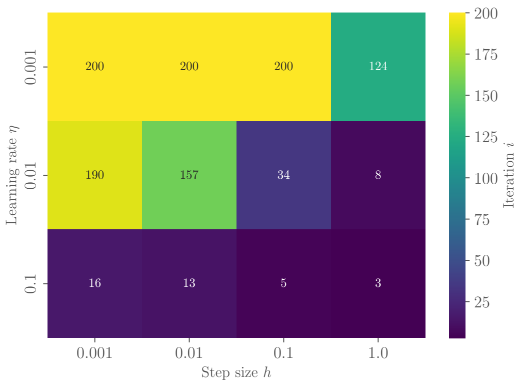
(b) $N = 6$ MH iteration heatmap

Figure 4.21: GD iterations until convergence and GS energy heatmaps as a functions of step size h and learning rate η for $N = 6$ Metropolis-Hastings algorithm

Continue with $N = 12$ Metropolis algorithm, heatmaps for which are on the Figure 4.22. For this system we have thee optimal pairs: $(h = 1,\ \eta = 0.01)$, $(h = 0.01,\ \eta = 0.1)$ and $(h = 0.001,\ \eta = 0.1)$.



(a) $N = 12$ Metropolis energy heatmap



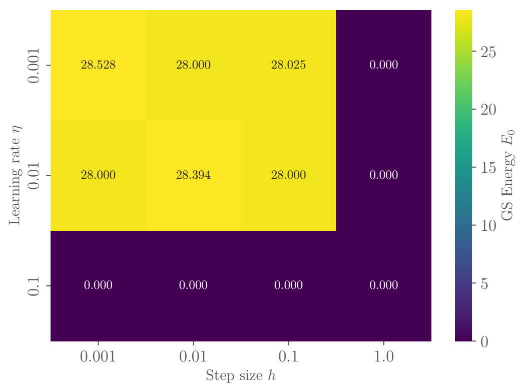
(b) $N = 12$ Metropolis iteration heatmap

Figure 4.22: GD iterations until convergence and GS energy heatmaps as a functions of step size h and learning rate η for $N = 12$ Metropolis algorithm

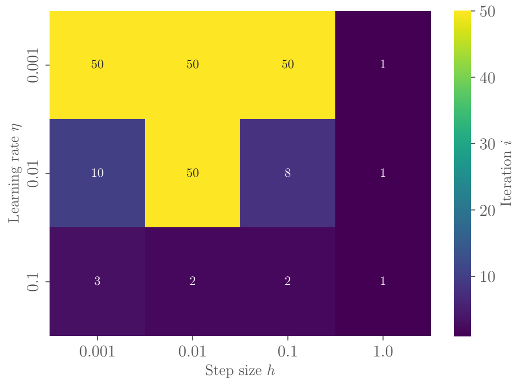
And finally, $N = 12$ Metropolis-Hastings heatmaps, Figure 4.23. Once again, we observe that MH is much sensitive for "big" learning rates and step sizes. For this system $(h = 0.1,\ \eta = 0.01)$ and $(h = 0.001,\ \eta = 0.01)$ are optimal combinations.

GD for non-interacting system can converge in less than ten iterations, if step size and learning rate are optimally chosen. Both Metropolis and Metropolis-Hastings generate exact GS energies. Therefore, we can conclude that both algorithms perform equally and better algorithm to use for non-interacting systems is Metropolis, simply because it is less computationally expensive.

Finally, we present Table 4.2 that contains number of iterations and runtime of all algorithms. All algorithms work correctly and converge in less than 11 iterations. In some cases even after $3-5$ iterations. Numerical and serial algorithms generate identical results, switching that they were implemented correctly. Algorithm execution times even for the largest system is very small, 0.17 seconds.



(a) $N = 12$ MH energy heatmap



(b) $N = 12$ MH iteration heatmap

Figure 4.23: GD iterations until convergence and GS energy heatmaps as a functions of step size h and learning rate η for $N = 12$ Metropolis-Hastings algorithm

$N = 2$	Metropolis	Num. Metropolis	Metropolis-Hastings	Num. MH
Serial	0.003 s / 11	0.140 s / 11	0.012 s / 5	0.369 s / 5
OMP	0.0007 s / 5	0.0575 s / 5	0.0033 s / 5	0.1534 s / 5
MPI	0.0007 s / 8	0.0722 s / 8	0.0031 s / 5	0.1259 s / 5
$N = 6$				
Serial	0.23 s / 8	1.77 s / 8	1.20 s / 3	7.40 s / 3
OMP	0.08 s / 10	0.50 s / 10	0.32 s / 7	2.82 s / 7
MPI	0.07 s / 9	0.52 s / 9	0.33 s / 5	2.30 s / 5
$N = 12$				
Serial	0.63 s / 5	9.58 s / 5	5.35 s / 5	46 s / 5
OMP	0.17 s / 5	2.76 s / 5	1.73 s / 4	14.58 s / 4
MPI	0.28 s / 4	2.67 s / 4	1.44 s / 5	14.72 s / 5

Table 4.2: Comparison of fermionic GD runtimes and iterations until convergence for different Metropolis variants and parallel backends. $n = 10^3$, $m = 10^2$, $B = 10^{-3}$.

Now, when we have studied non-interacting systems, we are ready to move on to interacting ones. So Far both algorithm generated correct results in small number of GD iterations. Let's see how they will perform for more complex systems.

4.3 Interacting Fermions

System of interacting fermions is much more complex than its non-interacting analogue. Complexity hides in the form of trial wave function. It needs to be modified with Jastow or Pade-Jastrow factor in order to nullify Coulomb's potential singularity. When sampling local energy during VMC iterations Laplacians of trial wave functions are calculated. Jastow factors introduce two new terms to local energy, and therefore computation becomes more expensive. Additionally, phase space where variational parameters belong alo becomes more complex, meaning that simple GD needs more VMC steps to converge. Local energy becomes even more sensible to the choice of step size and learning rate. Finally, we do not have analytical solution to compare numerical results with.

However, there is a fact that can help us with analyzing information. Interaction term add a positive value to the Hamiltonian. If we see that numerical interacting GS energy is smaller than non-interacting, it means that simulation generated incorrect results. This is a mathematical explanation. From physics perspective, every system want to be in the state that minimizes energy. When including interaction, entropy increases and energy of such system increases as well.

Having said that, let us begin with the simplest system of only two interacting fermions in $n = 0$ state.

4.3.1 First closed shell

When system is composed only of two interacting systems, we can estimate solution. Though process is very simple, we know that GS energy of non-interacting system is $E_0^{(\mathrm{NI})} = 2$. Hamiltonian introduces additional term r_{12}^{-1}. Particles repel one another, and we can estimate sample mean relative distance as $\langle r_{12} \rangle = 1/2$. Under these assumptions GS energy of interacting system is $E_0^{(\mathrm{I})} = 2 + 1 = 3$. However, this value is only an estimation, and it can be far from true GS energy.

In previous section we showed that optimal variational parameter α for non-interacting system is $\tilde{\alpha} = 1/2$. When considering interacting systems we need to optimize variational parameters β, which are a part of Jastrow factors. We will always set $\alpha = 1/2$ and focus on β optimization.

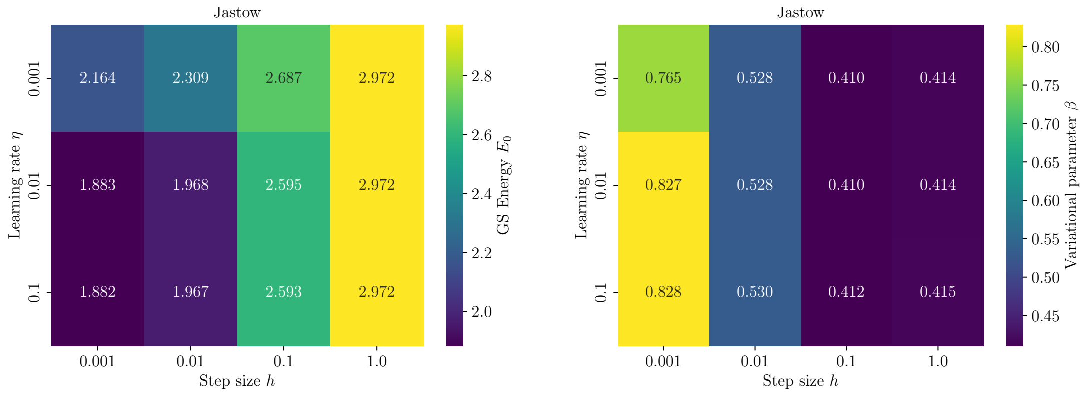
We need to change approach for the interacting system, for the reason of not having analytical solutions. One strategy is to run short simulations for different step size and learning rates and analyze results. These simulations can show us what parameter values are optimal. Additionally, even short simulations can give an overall understanding of GS value. Moreover, we run four simulations: Metropolis and Metropolis-Hastings algorithm for Jastow and Pade-Jastrow ansatzes. Having four different simulation we can find correlations, can help to determine GS energy.

First simulation parameters are: $m = 10^2$, $n = 10^3$ and $B = 10^{-2}$. Additionally, we set maximal number of GD iterations to $j = 3000$. OMP algorithm are used to generate results.

We will implement strategy similar to the one we used in previous section – generate heatmaps and find optimal step sizes and learning rates. But this time we need to consider not only $\langle E_0(h, \eta) \rangle$ and $\langle i(h, \eta) \rangle$, but also $\langle \beta \rangle$. In this case sample mean values of betas and energy is even more important than number of iterations until convergence.

Let us begin with Jastrow ansatz and Metropolis algorithm. Situlation results are depicted on Figure 4.24. Purple 2×2 box has energy less than energy of non-interacting system, and therefore we discard these results. The most promising results have $h = 1$. While studying non-interacting systems we noticed that Metropolis works well with relatively big step size. Variational parameters also have approximately equal values. For now we just mark pairs $(h = 1,\ \eta = 0.1)$, $(h = 1,\ \eta = 0.01)$ and $(h = 1,\ \eta = 0.001)$ as potentially good and GS energy to be $E_0 \approx 3$.

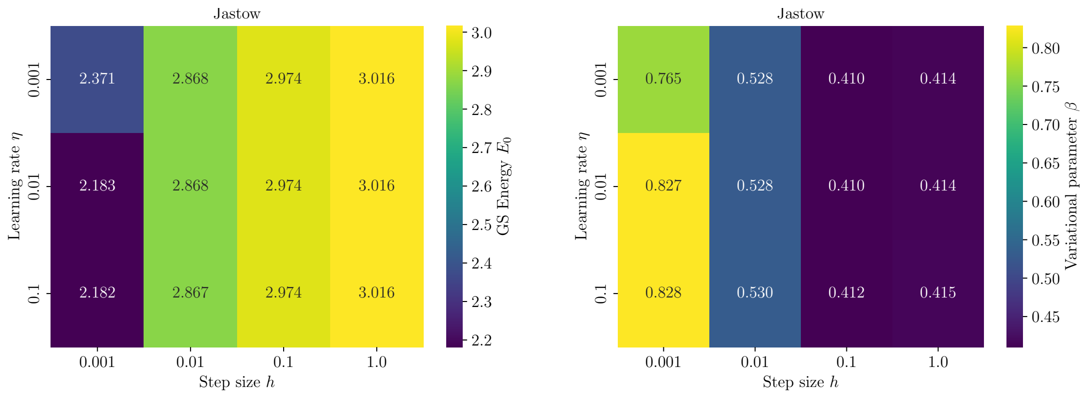
We continue with Metropolis-Hastings algorithm with Jastow ansatz, see Figure 4.25. Once again we see energy approximately equal to 3, which is a good sign. Simulations with $h = 1$ generated the same variational parameter, therefore we mark the same pairs of h and η for MH algorithm.



(a) Metropolis energy heatmap

(b) Metropolis variational parameter β heatmap

Figure 4.24: GD variational parameters and GS energy heatmaps as a functions of step size h and learning rate η for Metropolis algorithm with Jastow ansatz



(a) MH energy heatmap

(b) MH variational parameter β heatmap

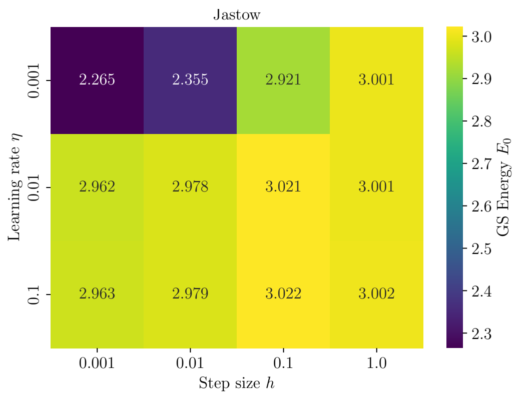
Figure 4.25: GD variational parameters and GS energy heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm with Jastow ansatz

Finally, we consider Metropolis and Metropolis-Hastings with Pade-Jastow anstzes depicted on figures 4.26 and 4.27. Once again, we see energy values are close to 3, and variational parameters are $\beta \approx 0.38$. Jastow and Pade-Jastow factors include variational parameters differently, therefore it is natural that they optimal parameters have different values. The most promising parameters are still $h = 1$ and all etas.

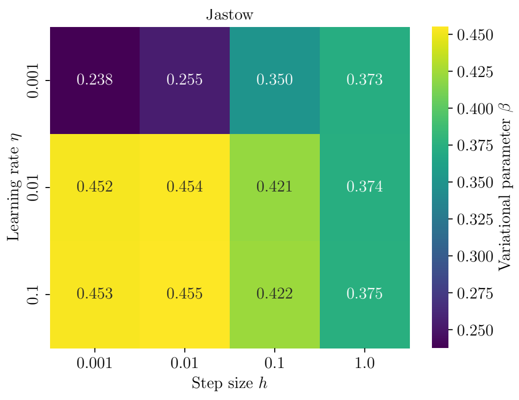
Next step is to determine what with what leaning rate algorithms converge fastest. To keep it short we will just state the results: for all algorithms GS with $(h = 1,\ \eta = 0.1)$ converge fastest (in approximately 20 iterations).

Now we already understand that GS energy is approximately equal to three. Next and final step is to run GD algorithm with $m_0$burn-in and n_0 VMC steps with optimal parameters and one final iteration with bigger number of burn-in steps m_1 VMC steps n_1.

Simulation with optimal parameters $(h = 1,\ \eta = 0.1)$, $m_0 = 10^4$, $n_0 = 10^5$, $m_1 = 10^5$ and $n_1 = 10^7$ showed the following results:

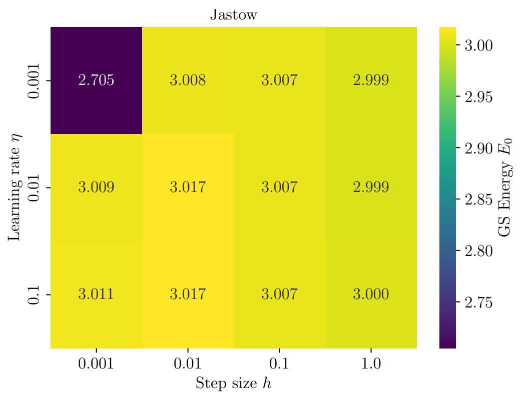


(a) Metropolis energy heatmap

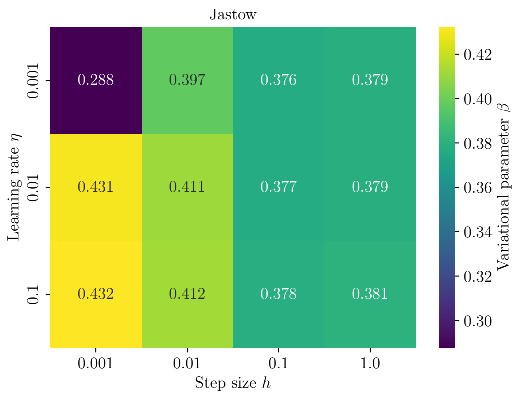


(b) Metropolis variational parameter β heatmap

Figure 4.26: GD variational parameters and GS energy heatmaps as a functions of step size h and learning rate η for Metropolis algorithm with Pade-Jastow ansatz



(a) MH energy heatmap



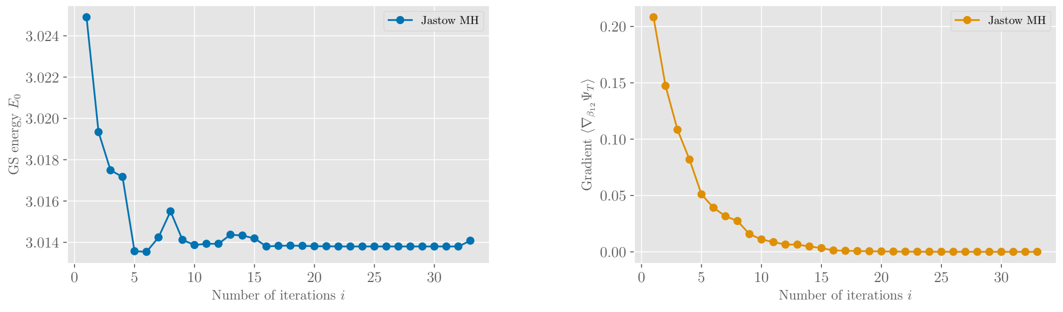
(b) MH variational parameter β heatmap

Figure 4.27: GD variational parameters and GS energy heatmaps as a functions of step size h and learning rate η for MH algorithm with Pade-Jastow ansatz

- Jastrow Metropolis GD simulation did not converge in 50 iterations, $\beta_{12} = 0.41398$, $E_0 = 3.01644$, $\langle \nabla_{\beta_{12}} \Psi_T \rangle = 0.0111455$;
- Jastrow MH GD simulation converged in 31 iterations, $\beta_{12} = 0.416592$, $E_0 = 3.01408$, $\langle \nabla_{\beta_{12}} \Psi_T \rangle = 9.7 \cdot 10^{-6}$;
- Pade-Jastrow Metropolis GD simulation did not converge in 50 iterations, $\beta_{12} = 0.341$, $E_0 = 3.00033$, $\langle \nabla_{\beta} \Psi_T \rangle = 1.6 \cdot 10^{-4}$;
- Pade-Jastrow MH GD simulation did not converge in 50 iterations, $\beta_{12} = 0.341$, $E_0 = 3.0011$, $\langle \nabla_{\beta} \Psi_T \rangle = 7 \cdot 10^{-5}$.

On Figure 4.28 we show how Gs energy and gradient change during GD VMC procedure for Jastow MH algorithm. Gradient smoothly decrees, while energy has some small pikes. Energy makes a small jump at the last iteration, because for this iteration we preformed more VMC steps.

We can conclude that GS energy for this system is $E_0 \approx 3$, and MH algorithm shows better performance – observable smoothly converge. During Metropolis algorithm



(a) MH Jastrow GS energy E_0 (b) MH Jastow Gradient $\langle\nabla_{\beta_{12}}\Psi_T\rangle$

Figure 4.28: Metropolis-Hastings with Jastrow ansatz GS energy and gradient as a function of number of iterations.

gradient decrees to some relatively small value and jumps back and forth. Also, our estimation of GS energy turned out to be correct.

4.3.2 Second Closed Shell

Approach chosen in previous section worked perfectly for first closed shell. In this section we will follow the same procedure:

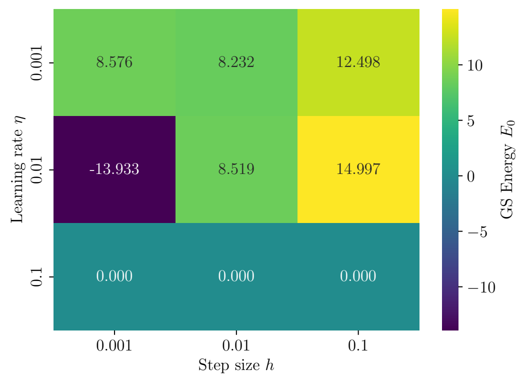
- Run short GD simulations for all algorithms;
- Analyze the data to estimate GS energy and optimal variational parameters;
- Determine optimal step size, learning rate and algorithm with the best performance;
- Run long GD simulation with algorithms that shown best performance to find GS energy.

For second closed shell $n_{\text{pairs}=15}$, and therefore Jastrow factor contains fifteen variational parameters β_{ij}. We update each variational parameter β_{ij} every GD iteration.

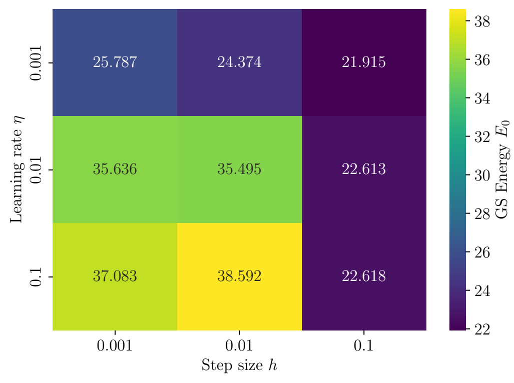
As a first step, we performed a short VMC simulation for all OMP parallelized algorithms with parameters $m = 10^2$, $n = 10^2$ and $\alpha = 1/2$. As initial Pade-Jastrow variational parameter we chose optimal parameter for $N = 2$ interacting system, $\beta = 0.4$. All Jastrow factor variational parameters were initialized as $\beta_{ij} = 0.385$.

During this simulation Metropolis algorithm showed bad performance, especially with Jastrow ansatz. It is impossible to determine optimal variational parameters or estimate GS energy by analyzing data obtained only by Metropolis simulations. On heatmaps depicted on figure 4.29. As we can see, Jastrow estimation of GS energy is bad. Pade-Jastrow simulation, on opposite hand, showed consistent results. For $h = 0.1$ energy is close to 22 for each learning rate. Bad performance of Jastrow factor does not say that it is a bad ansatz, it just says that 110 VMC iterations is not enough for it to generate good results. Therefore, to get better intuition for Metropolis algorithm with Jastrow ansatz we need to run GD VMC algorithm with more iterations.

Let us now study exactly the same simulation with Metropois-Hastings algorithm. Energy heatmaps for the same number of VMC iterations are depicted on Figure 4.30. Metropolis-Hastings shows consistency even with such small number of VMC iterations. It generates similar results for six variations of step sizes and learning rates. We can also see that MH algorithms with both Jastow and Pade-Jastow generate approximately the same energy: $E_0 \approx 20$. This switches that Jastrow factor works, it just needs more iterations in combination with Metropolis algorithm. But we want to estimate GS energy in the fastest possible way, and for this system this way turns out to be by using MH algorithm with Pade-Jastrow ansatz.

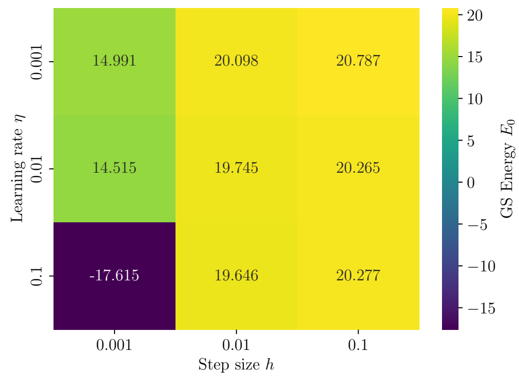


(a) Jastrow Metropolis energy heatmap

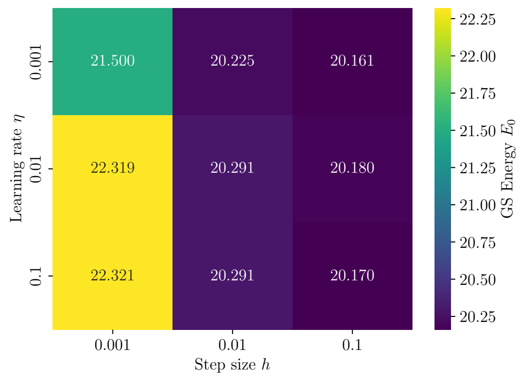


(b) Pade-Jastrow Metropolis energy heatmap

Figure 4.29: GS energy heatmaps as a functions of step size h and learning rate η for Metropolis algorithm



(a) Jastrow MH energy heatmap



(b) Pade-Jastrow MH energy heatmap

Figure 4.30: GS energy heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm

By analyzing similar heatmaps for L2 norm and number of iteration until convergence we conclude the following:

- Metropolis algorithm with Jastrow factor ansatz shows worse performance than Metropolis algorithm with Pade-Jastrow ansatz. Therefore we will study only Pade-Jastrow Metropolis algorithm. Optimal parameters are $(h = 0.1,\ \eta = 0.1)$;
- MH algorithm with Jastrow ansatz shows good performance. It converges to L2 $=$ 10^{-1}, which is a great result for a vector with fifteen entries. Optimal parameters are $(h = 0.1,\ \eta = 0.1)$;
- MH algorithm with Pade-Jastrow ansatz shows great performance. It converges to 10^{-2} after ≤ 7 iterations when $h = 0.1$. Optimal parameters are $(h = 0.1,\ \eta = 0.1)$.

For these algorithms and parameters we performed GD VMC simulation with $m_0 = 5 \cdot 10^3$, $n_0 = 5 \cdot 10^4$, $m_1 = 5 \cdot 10^4$ and $n_1 = 5 \cdot 10^5$ and obtained the following results:

- Metropolis-Hastings algorithm with Jastrow factor ansatz converged to $B = 10^{-1}$ after 9 iterations. Each GD iteration had a runtime $t \approx 10$ $[s]$. Estimated GS energy is $E_0 = 20.3582$. Optimal parameters are given by

$$\tilde{\beta}_k \approx \begin{cases} 1.0 & \text{for} \quad k = \{0, 1, 13, 14\}; \\ 2.5 & \text{else.} \end{cases}$$

- Metropolis algorithm with Pade-Jastrow factor ansatz converged to $B = 2.2 \cdot 10^{-2}$ after 25 iterations. Each GD iteration had a runtime $t \approx 25$ $[s]$. Estimated GS energy is $E_0 = 20.178$. Optimal variational parameter is $\tilde{\beta} = 0.443477$;

- Metropolis-Hastings algorithm with Pade-Jastrow factor ansatz converged to $B = 4 \cdot 10^{-3}$ after 25 iterations. Each GD iteration had a runtime $t \approx 35$ $[s]$. Estimated GS energy is $E_0 = 20.1958$. Optimal variational parameter is $\tilde{\beta} = 0.46676$.

From obtained data we can conclude that GS energy of second closed shell with interacting fermions is $E_0 \approx 20$. All algorithms accept Metropolis with Jastrow factor showed good convergence. MH algorithm with both ansatzes generates more stable and better result than Metropolis when number of VMC iterations is small. When number of VMC cycles is relatively big, both algorithm estimates GS energy great. Algorithms were able to converge so fast due to optimal choice of parameters, which was based on analysis of short GD VMC algorithms.

4.3.3 Third Closed Shell

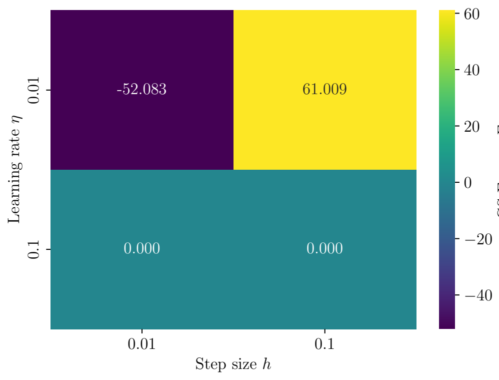
We continue with the last system - third closed shell. This is the most time consuming and the most complex system. To study it we follow the same path – short VMC, analize heatmaps, long VMC.

For this system we need to choose step size and learning rate carefully, because each VMC GD can take hours. We have seen that when system becomes bigger, solving algorithms work better with smaller parameters. Therefore we will test algorithms only for $h = \eta = 0.1, 0.01$.

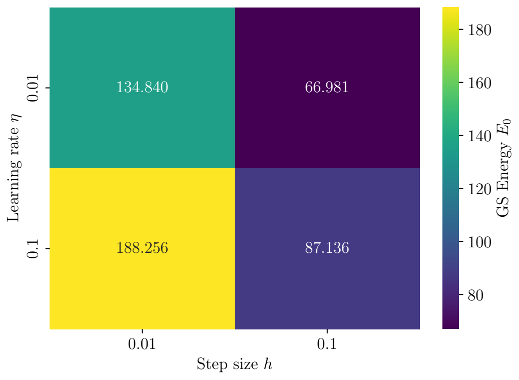
For this simulation we initialized parameters as $m = 10^3$, $n = 5 \cdot 10^3$, $\alpha = 1/2$ and 25 GD iterations. We initialize starting variational parameters as $\beta = 0.385$ and $\beta_{ij} = 0.4$. This time we derided not to choose optimal variational parameters for Jastrow factor to see if distinct values would generate better results.

Energy heatmaps are presented on figures 4.31 and 4.32. Unlike second closed shell, Metropolis energies do not provide us with helpful information regarding what true value they have. However, by analyzing L2 norm, number of iterations until convergence and energy together, we conclude that optimal parameters are $h = 0.1$ and $\eta = 0.01$. True GS energy then is somewhere in range $(60, 70)$.

Metropolis-Hastings algorithm, as expected, gives us better estimations and information. We can clearly see that true energy range narrows to $(66, 67)$. Once again we see that Metropolis-Hastings algorithm generates better results for interacting system. Recall, that for non-interacting systems both algorithms restored exact energy for every system.

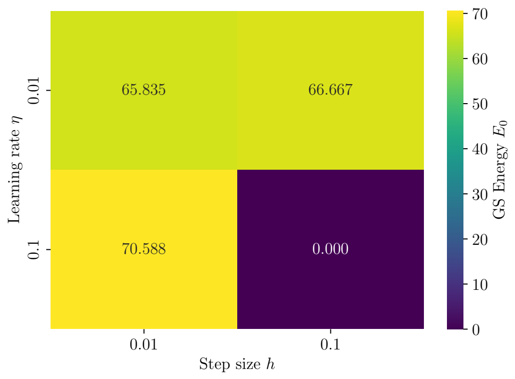


(a) Jastrow Metropolis energy heatmap

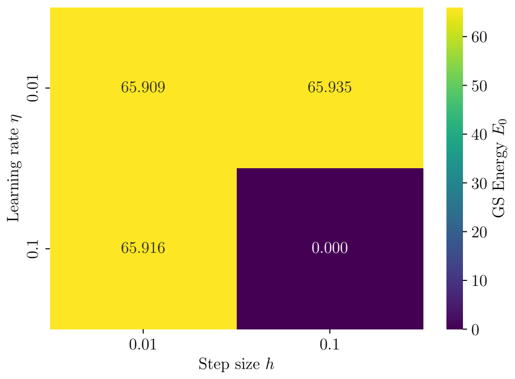


(b) Pade-Jastrow Metropolis energy heatmap

Figure 4.31: GS energy heatmaps as a functions of step size h and learning rate η for Metropolis algorithm



(a) Jastrow MH energy heatmap



(b) Pade-Jastrow MH energy heatmap

Figure 4.32: GS energy heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm

This time we will perform long GD VMC simulations only for Metropolis-Hastings algorithm. Reason for that is that Metropolis needs approximately $n = 10^5$ GD VMC iterations to generate decent results. Even with parallel algorithms, it would take days to simulate on a normal computer. And after all, we aim to estimate GS energy accurately and fast. Therefore MH is our choice for this system.

Our analysis of energy, iterations and L2 norm as functions of step size and learning rate indicate that optimal parameters are $h = \eta = 0.01$ for Jastrow ansatz and $h = 0.1, \eta = 0.01$ for Pade-Jastrow ansatz.

We choose $m_0 = 5 \cdot 10^3$, $n_0 = 5 \cdot 10^4$, $m_1 = 5 \cdot 10^4$ and $m_0 = 5 \cdot 10^5$ and step size with learning rate described earlier as parameters for long GD VMC simulation. Results of these simulations are:

- MH with Jastrow ansatz converged to L2$= 1.28$ after 26 iterations and GS energy is $E_0 = 66.3778$. In sense of convergence, this is a remarkable result. It means that gradient of each variational parameter is smaller than 10^{-1}. To achieve such great accuracy optimizing 66 variational parameters simultaneously is a great achievement. GS energy is consistent with result from shot VMC simulation;

- MH with Pade-Jastrow ansatz converged to L2$= 1.8 \cdot 10^{-3}$ after 12 iterations and GS energy is $E_0 = 65.83$. Once again, GS energy is consistent with different methods. The fact that energy obtained by different methods is approximately equal switches that this is the correct GS energy of the system. Convergence is also great, in the sense that we were able to achieve such accuracy after only twelve iterations.

Recall that each 2D GS energy estimated in this research was either an integer, or very close to it. We can assume that GS energy for third closed shell of interacting particles is equal to 66. Such a coincidence, 66 variational parameters and GS energy is equal to 66. Physics never ceases to amaze!

Chapter 5

Conclusion

In this research we have studied systems of bosons and fermions trapped in harmonic potential. The main focus was on development of flexible VMC algorithm. Computational implementation was made using `C++` as a main programming language. All programs were fully developed from scratch. VMC is computationally expensive, and therefore parallel-friendly programming languages like `C++` suit best. We were able to implement Metropolis and Metropolis-Hastings VMC algorithms for bosonic and fermionic systems. For each algorithm we developed three versions: serial, parallel and numerical. Parallelization was done using OMP and MPI. Both OMP and MPI showed great speedup while still generating correct results. Automatic differentiation was implemented for two reasons: comparison with analytical algorithms and to study its performance. For each algorithm numerical and analytical algorithms generated exactly the same results, which switches that local energy is compute correctly, and it is the most difficult part.

We studied system of non-interacting cooled bosons, namely BEC. We shown that for such system trial wave function is just a product of exponents, which is much simpler than fermionic wave function. We derived analytical solution for this system and numerical VMC estimates GS energy exactly even with small number of iterations. Both Metropolis and Metropolis-Hastings generate correct results. Since Metropolis algorithm is faster, we believe that for non-interacting systems Metropolis is an optimal choice. We modified trial wave function with variational parameter α and VMC simulation correctly showed that optimal varialtional parameter is $\alpha \tilde{=} 1/2$.

As for fermionic systems, we focused on studying both non-interacting and interacting closed shells - states with neutral spin in two dimensions. For non-interacting system both Metropolis and Metropolis-Hasting showed a great performance and generated correct energies for all systems. Interacting systems do not have a general solution, so they were the hardest test. For this system GD VMC also found that $\tilde{\alpha} = 1/2$, which aligns with analytical solution.

For the first closed shell we found GS energy to be equal to 3 with four digits precision. Before we obtained results, by assuming that mean value of relative energy is equal to half we estimated the energy to be equal to 3 and were correct. For this system Metropolis-Hastings algorithm with Jastrow ansatz showed the best performance, while other algorithm did good as well. Even though Jastrow factors have a very different form, we found optimal parameters to be approximately equal to 0.4.

Second closed shell was much harder to solve. First of all, it takes more time to run a simulation. Second is that Jastow factors contain relative distances of all combinations of particles, and therefore it is hard to optimize variational parameters. We were able to develop a strategy of finding optimal step sizes and leaning rates by creating different heatmaps and analyzing them. For this system Metropolis algorithm with Jastrow anatz turned out to generate bad results, while Pade-Jastow Metropolis generated good results. We concluded that the problem is not in Jastrow factor itself, it just requires more VMC iterations to generate correct result. In this regard Metropolis-Hastings was better. Both Jastrow and Pade-Jastrow Metropolis-Hastings simulations generated great results. We were able to conclude that GS energy is $E_0 = 20$.

Foe the third closed shell, once again, Metropolis-Hasting showed better results. If we had data only from Metropolis algorithm, it would be very hard or very long to find GS energy. By applying the same technique we found that $E_0 = 66$.

As a conclusion, we can say that we were able to develop VMC algorithms that work correctly and generate correct results. We can be sure that GS energies generated by VMC simulations are correct. We also can conclude that having two solving algorithms and two wave function ansatzes we really helpful. We were able to generate result only because we were using and analyzing combination of four algorithms.

There is very big prospects for the future researches in this field. For example, oscillator frequency dependency can be studied. Different forms of Jastrow factors can be used, as well as Coulolomb interaction with higher terms. A new science direction was popularized recently, which uses neural networks to generate trial wave functions. Even something simple as Boltzman machines can be used to this type of research. We have developed MPI version of the code, and therefore VMC for fourth and fifth closed shells can be developed and ran computer cluster.

Chapter 6

Appendix

6.1 Harmonic Oscillator Units and Atomic Units

The task is to make Hamiltonian dimensionless. If we consider non-interacting HO problem, Hamiltonian has a form

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m} \nabla_i^2 + \frac{m\omega^2}{2} r^2 \right).$$

Eigenvalue SE reads as

$$\hat{H}\psi(\mathbf{r}) - E\psi(\mathbf{r}) = 0.$$

Use explicit form of the Hamiltonian,

$$\sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m} \nabla_i^2 \psi(\mathbf{r}) + \frac{m\omega^2}{2} r^2 \psi(\mathbf{r}) \right) - E\psi(\mathbf{r}) = 0$$

Perform a change of variables in the form

$$\rho = \frac{\mathbf{r}}{a}, \quad \mathbf{r} = a\rho.$$

For such substitution, gradient changes to

$$\begin{aligned}\nabla &= \frac{\partial}{\partial x}\mathbf{e}_x + \frac{\partial}{\partial y}\mathbf{e}_y + \frac{\partial}{\partial z}\mathbf{e}_z = \frac{\partial}{\partial (a\rho_x)}\mathbf{e}_x + \frac{\partial}{\partial (a\rho_y)}\mathbf{e}_y + \frac{\partial}{\partial (a\rho_z)}\mathbf{e}_z \\ &= \frac{1}{a}\left(\frac{\partial}{\partial \rho_x}\mathbf{e}_{\rho_x} + \frac{\partial}{\partial \rho_y}\mathbf{e}_{\rho_y} + \frac{\partial}{\partial \rho_z}\mathbf{e}_{\rho_z} \right) = \frac{1}{a}\nabla_\rho.\end{aligned}$$

Here we have used the fact that for a linear transformation, basis vectors $\mathbf{e}_r = \mathbf{e}_\rho$ do not change. ∇_ρ represent gradient with respect to ρ coordinates. Putting this back to equation:

$$\sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m}\frac{1}{a^2} \nabla_{\rho_i}^2 \psi(\rho) + \frac{m\omega^2}{2} a^2 \rho^2 \psi(\rho) \right) - E\psi(\rho) = 0. \Big| \times \frac{ma^2}{\hbar^2};$$

$$\sum_{i=1}^{N} \left(-\frac{1}{2} \nabla_{\rho_i}^2 \psi(\rho) + \frac{1}{2}\frac{m^2\omega^2}{\hbar^2} a^4 \rho^2 \psi(\rho) \right) - \frac{ma^2}{\hbar^2} E\psi(\rho) = 0.$$

Choose constant a so that:

$$\frac{m^2\omega^2}{\hbar^2}a^4 = 1 \implies a = a_{\text{HO}} = \sqrt{\frac{\hbar}{m\omega}}.$$

a is called "Harmonic Oscillator length". In these variables energy changes to

$$\lambda = \frac{ma^2}{\hbar^2}E = \frac{m}{\hbar^2}\frac{\hbar}{m\omega}E = \frac{E}{\hbar\omega}.$$

In these scaled units the energy is a dimensionless number measured in quanta of $\hbar\omega$, and the coordinate is likewise dimensionless, measured in units of the oscillator length $a = \sqrt{\frac{\hbar}{m\omega}}$.

Dimensionless Hamiltonian becomes

$$\hat{H} = \sum_{i=1}^{N} -\frac{1}{2}\nabla_{\rho_i}^2 + \frac{1}{2}\rho^2. \tag{6.1}$$

Now consider Hamiltonian only with electrostatic Coulomb's potential

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\nabla_i^2 + \sum_{j>i}^{N}\frac{e^2}{4\pi\varepsilon_0}\frac{1}{r_{ij}}\right)$$

Performing the same substitution

$$\rho = \frac{\mathbf{r}}{a}, \quad \mathbf{r} = a\rho.$$

Relative distance between particles i and j r_{ij} in new coordinates changes to

$$\begin{aligned} r_{ij} &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \\ &= \sqrt{a^2(\rho_{x_i} - \rho_{x_j})^2 + a^2(\rho_{y_i} - \rho_{y_j})^2 + a^2(\rho_{z_i} - \rho_{z_j})^2} = a\rho_{ij}. \end{aligned}$$

Following the same steps:

$$\begin{aligned} &\sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\frac{1}{a^2}\nabla_{\rho_i}^2\psi(\rho) + \sum_{j>i}^{N}\frac{e^2}{4\pi\varepsilon_0}\frac{1}{a}\frac{1}{\rho_{ij}}\psi(\rho)\right) - E\psi(\rho) = 0 \;\Bigg|\; \times \frac{ma^2}{\hbar^2}; \\ &\sum_{i=1}^{N}\left(\nabla_{\rho_i}^2\psi(\rho) + \sum_{j>i}^{N}\frac{e^2 ma}{4\pi\varepsilon_0\hbar^2}\frac{1}{\rho_{ij}}\psi(\rho)\right) - \frac{ma^2}{\hbar^2}E\psi(\rho) = 0. \end{aligned}$$

Choose constant a so that

$$\frac{e^2 ma}{4\pi\varepsilon_0\hbar^2} = 1 \implies a = a_o = \frac{4\pi\varepsilon_0\hbar^2}{me^2}.$$

In this case, a_o is a Bohr's radius and these units are called atomic units. Energy changes to

$$\lambda = \frac{ma^2}{\hbar^2}E = \frac{m}{\hbar^2}\frac{16\pi^2\varepsilon_0^2\hbar^4}{m^2e^4}E = \frac{16\pi^2\varepsilon_0^2\hbar^2}{me^4}E = \frac{E}{E_h},$$

where E_h is a Hartree energy

$$E_h = \frac{me^4}{16\pi^2\varepsilon_0^2\hbar^2}.$$

In atomic units Hamiltonian becomes

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_{\rho_i}^2 + \sum_{j>i}^{N} \frac{1}{\rho_{ij}} \right) \tag{6.2}$$

When considering a system in both HO and Coulomb's potentials Hamiltonian is

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m}\nabla_i^2 + \frac{m\omega^2}{2}r^2 + \sum_{j>i}^{N} \frac{e^2}{4\pi\varepsilon_0}\frac{1}{r_{ij}} \right).$$

Perform change of variable

$$\rho = \frac{\mathbf{r}}{a}, \quad \mathbf{r} = a\rho.$$

SE becomes

$$\sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m}\frac{1}{a^2}\nabla_{\rho_i}^2 + \frac{m\omega^2}{2}a^2\rho^2 + \sum_{j>i}^{N} \frac{e^2}{4\pi\varepsilon_0}\frac{1}{a}\frac{1}{\rho_{ij}} \right)\psi - E\psi = 0 \Bigg| \times \frac{ma^2}{\hbar^2};$$

$$\sum_{i=1}^{N} \left(\nabla_{\rho_i}^2 + \frac{m^2\omega^2}{2\hbar^2}a^4\rho^2 + \sum_{j>i}^{N} \frac{mae^2}{4\pi\varepsilon_0\hbar^2}\frac{1}{\rho_{ij}} \right)\psi - \frac{ma^2}{\hbar^2}E\psi = 0.$$

We can not make all three terms equal to unity. By choosing HO units prefactor before Coulomb's term will appear. Likewise, by choosing atomic units, prefactor before harmonic term will appear.

Consider HO units first, $a_{\mathrm{HO}} = \sqrt{\frac{\hbar}{m\omega}}$. Coulomb's prefactor is equal to

$$A(\omega) = \frac{me^2}{4\pi\varepsilon_0\hbar^2}a_{HO} = \frac{me^2}{4\pi\varepsilon_0\hbar^2}\sqrt{\frac{\hbar}{m\omega}} = \frac{e^2}{4\pi\varepsilon_0}\sqrt{\frac{m}{\hbar^3\omega}}. \tag{6.3}$$

Here, all the constants are taken from CODATA. It's numerical value is

$$A(\omega) \approx \frac{(1.602\cdot 10^{-19})^2}{4\cdot 3.141\cdot 8.854\cdot 10^{-12}}\sqrt{\frac{9.109\cdot 10^{-31}}{(1.054\cdot 10^{-34})^3\omega}} = 2.034\cdot 10^8\omega^{-1/2}.$$

However, this choice is not logical. In these units we can tune interaction strength by varying oscillator frequency. More natural choice is atomic units with Bohr's radius $a_o = \frac{4\pi\varepsilon_0\hbar^2}{me^2}$. Prefactor then becomes

$$B(\omega) = \frac{m^2\omega^2}{2\hbar^2}a_o^4 = \frac{m^2\omega^2}{2\hbar^2}\frac{256\pi^4\varepsilon_0^4\hbar^8}{m^4e^8} = \frac{128\pi^4\varepsilon_0^4\hbar^6\omega^2}{m^2e^8}. \tag{6.4}$$

According to CODATA, numerical value of Bohr's radius is $a_o \approx 5.291\cdot 10^{-11}$. Numerical value of $B(\omega)$ is

$$B(\omega) \approx \frac{(9.109\cdot 10^{-31})^2}{2(1.054\cdot 10^{-34})^2}(5.291\cdot 10^{-11})^4\omega^2 = 2.926\cdot 10^{-34}\omega^2.$$

Using HO units, Hamiltonian becomes

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_{\rho_i}^2 + \frac{1}{2}\rho^2 + A(\omega)\sum_{j>i}^{N} \frac{1}{\rho_{ij}} \right). \tag{6.5}$$

In Atomic units it is

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_{\rho_i}^2 + B(\omega)\rho^2 + \sum_{j>i}^{N} \frac{1}{\rho_{ij}} \right) \tag{6.6}$$

6.2 Single-Particle Orbitals & Derivatives

In HO units single-particle orbital wave functions are given by equation (3.1):

$$\psi_n(x) = He_n(x)e^{-x^2/2},$$

while in atomic units wave functions are given by (3.3):

$$\psi_n(x) = He_n(x\sqrt{\omega})e^{-x^2\omega/2}.$$

They differ only by a rescaling of their argument. Hence we derive the general form with x as argument. When working in atomic units, we simply replace $x \to \sqrt{\omega}x$. If we work in two or three dimensions we replace $\mathbf{r} \to \sqrt{\omega}\mathbf{r}$.

For $n = 0$ wave function is equal to

$$\psi_{n=0}(x) = e^{-\alpha x^2}.$$

In two dimensions it becomes

$$\psi_{n=0}(x, y) = \psi_{n_x=0}(x)\psi_{n_y=0}(y) = e^{-\alpha x^2}e^{-\alpha y^2} = e^{-\alpha(x^2+y^2)} = e^{-\alpha r^2}.$$

In three dimensions it is

$$\psi_{n=0}(x, y, z) = \psi_{n_x=0}(x)\psi_{n_y=0}(y)\psi_{n_z=0}(z) = e^{-\alpha x^2}e^{-\alpha y^2}e^{-\alpha z^2} = e^{-\alpha(x^2+y^2+z^2)} = e^{-\alpha r^2}.$$

We define $\psi_1(\mathbf{r})$ as

$$\psi_1(\mathbf{r}) = e^{-\alpha r^2}. \tag{6.7}$$

Expression in this form applies in any spatial dimension. Here $\mathbf{r} = (x, y, z)$ in 3D so that $r^2 = x^2 + y^2 + z^2$; in 2D $\mathbf{r} = (x, y)$ and $r^2 = x^2 + y^2$; and in 1D $\mathbf{r} = (x)$ so $r^2 = x^2$.

We will need gradient and Laplacian of two- and three-dimensional ψ_1 for fermionic and bosonic systems respectively. Let us derive these expressions:

$$\nabla\psi_1(x, y) = \left(\frac{\partial}{\partial x}\left[e^{-\alpha r^2}\right]\cdot \mathbf{e}_x + \frac{\partial}{\partial y}\left[e^{-\alpha r^2}\right]\cdot \mathbf{e}_y\right) = \left(-2\alpha x e^{-\alpha r^2}\cdot \mathbf{e}_x - 2\alpha y e^{-\alpha r^2}\cdot \mathbf{e}_y\right),$$

so that

$$\nabla\psi_1(x, y) = -2\alpha e^{-\alpha r^2}\left(x\cdot \mathbf{e}_x + y\cdot \mathbf{e}_y\right). \tag{6.8}$$

Subsequently

$$\nabla\psi_1(x, y, z) = -2\alpha e^{-\alpha r^2}\left(x\cdot \mathbf{e}_x + y\cdot \mathbf{e}_y + z\cdot \mathbf{e}_z\right). \tag{6.9}$$

Laplacian can be computed as

$$\begin{aligned}\Delta\psi_1(x, y) = \frac{\partial}{\partial x}\left(-2\alpha x e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left(-2\alpha y e^{-\alpha r^2}\right) &= -2\alpha e^{-\alpha r^2} + 4\alpha^2 x^2 e^{-\alpha r^2} \\ &- 2\alpha e^{-\alpha r^2} + 4\alpha^2 y^2 e^{-\alpha r^2}.\end{aligned}$$

Final expression for the Laplacian is

$$\Delta\psi_1(x, y) = \left(-4\alpha + 4\alpha^2 r^2\right)e^{-\alpha r^2}. \tag{6.10}$$

In 3D Laplacian is

$$\Delta\psi_1(x, y, z) = \left(-6\alpha + 4\alpha^2 r^2\right)e^{-\alpha r^2}. \tag{6.11}$$

The following wave functions are needed for fermionic system, and therefore will only be derived for two dimensions. If $n = n_x + n_y = 1$ there is two possibilities: $(n_x, n_y) = (0,1)$ or $(1,0)$. We define $\psi_2(x,y)$ as

$$\psi_2(x,y) = \psi_{n_x=1}(x)\psi_{n_y=0}(y) = xe^{-\alpha r^2} \tag{6.12}$$

and $\psi_3(x,y)$ as

$$\psi_3(x,y) = \psi_{n_x=0}(x)\psi_{n_y=1}(y) = ye^{-\alpha r^2}. \tag{6.13}$$

Gradient of $\psi_2(x,y)$ can be calculated as

$$\begin{aligned}\nabla\psi_2(x,y) &= \left(\frac{\partial}{\partial x}\left[xe^{-\alpha r^2}\right]\cdot \mathbf{e}_x + \frac{\partial}{\partial y}\left[xe^{-\alpha r^2}\right]\cdot \mathbf{e}_y\right)\\ &= \left((1-2\alpha x^2)e^{-\alpha r^2}\cdot \mathbf{e}_x - 2\alpha xye^{-\alpha r^2}\cdot \mathbf{e}_y\right),\end{aligned}$$

so that

$$\nabla\psi_2(x,y) = \left((1-2\alpha x^2)\cdot \mathbf{e}_x - 2\alpha xy\cdot \mathbf{e}_y\right)e^{-\alpha r^2}. \tag{6.14}$$

Gradient of $\psi_3(x,y)$ can be computed similarly:

$$\begin{aligned}\nabla\psi_3(x,y) &= \left(\frac{\partial}{\partial x}\left[ye^{-\alpha r^2}\right]\cdot \mathbf{e}_x + \frac{\partial}{\partial y}\left[ye^{-\alpha r^2}\right]\cdot \mathbf{e}_y\right)\\ &= \left(-2\alpha xye^{-\alpha r^2}\cdot \mathbf{e}_x + (1-2\alpha y^2)e^{-\alpha r^2}\cdot \mathbf{e}_y\right),\end{aligned}$$

and it its final form is

$$\nabla\psi_3(x,y) = \left(-2\alpha xy\cdot \mathbf{e}_x + (1-2\alpha y^2)\cdot \mathbf{e}_y\right)e^{-\alpha r^2} \tag{6.15}$$

Laplacian of $\psi_2(x,y)$ is computed as

$$\begin{aligned}\Delta\psi_2(x,y) &= \frac{\partial}{\partial x}\left((1-2\alpha x^2)e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left(-2\alpha xye^{-\alpha r^2}\right)\\ &= \frac{\partial}{\partial x}\left(e^{-\alpha r^2} - 2\alpha x^2e^{-\alpha r^2}\right) - \frac{\partial}{\partial y}\left(2\alpha xye^{-\alpha r^2}\right)\\ &= -2\alpha xe^{-\alpha r^2} - 4\alpha xe^{-\alpha r^2} + 4\alpha^2x^3e^{-\alpha r^2} - 2\alpha xe^{-\alpha r^2} + 4\alpha^2xy^2e^{-\alpha r^2}\\ &= \left(-2\alpha x - 4\alpha x + 4\alpha^2x^3 - 2\alpha x + 4\alpha^2xy^2\right)e^{-\alpha r^2}\\ &= \left(-8\alpha x + 4\alpha^2x(x^2+y^2)\right)e^{-\alpha r^2}.\end{aligned}$$

Final expression is

$$\Delta\psi_2(x,y) = x\left(-8\alpha + 4\alpha^2r^2\right)e^{-\alpha r^2}. \tag{6.16}$$

We can obtain $\Delta\psi_3(x,y)$ from $\Delta\psi_2(x,y)$. Notice that $\psi_3(x,y) = \psi_2(y,x)$. Laplacian $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is symmetrical under substitution $x \to y$. It means that $\Delta\psi_3(x,y) = \Delta\psi_2(y,x)$ and it is equal to

$$\Delta\psi_3(x,y) = y\left(-8\alpha + 4\alpha^2r^2\right)e^{-\alpha r^2}. \tag{6.17}$$

There are three ways to construct wave function with $n = n_x + n_y = 2$: $(n_x, n_y) = (2,0)$, $(0,2)$ and $(1,1)$. We define

$$\psi_4(x,y) = \psi_{n_x=2}(x)\psi_{n_y=0}(y) = (x^2-1)e^{-\alpha r^2}, \tag{6.18}$$

$$\psi_5(x,y) = \psi_{n_x=0}(x)\psi_{n_y=2}(y) = (y^2-1)e^{-\alpha r^2} \tag{6.19}$$

and

$$\psi_6(x,y) = \psi_{n_x=1}(x)\psi_{n_y=1}(y) = xye^{-\alpha r^2}. \tag{6.20}$$

Let us find their gradients starting with $\psi_4(x,y)$:

$$\begin{aligned}\nabla\psi_4(x,y) &= \left(\frac{\partial}{\partial x}\left[(x^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[(x^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(2xe^{-\alpha r^2} - 2\alpha x^3e^{-\alpha r^2} + 2\alpha xe^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(2\alpha y(1-x^2)e^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right).\end{aligned}$$

So that

$$\nabla\psi_4(x,y) = \left(\mathbf{e}_x\left(2x - 2\alpha x^3 + 2\alpha x\right) + \mathbf{e}_y\left(2\alpha y - 2\alpha x^2y\right)\right)e^{-\alpha r^2}. \tag{6.21}$$

Continue with

$$\begin{aligned}\nabla\psi_5(x,y) &= \left(\frac{\partial}{\partial x}\left[(y^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[(y^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(2\alpha x(1-y^2)e^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(2ye^{-\alpha r^2} - 2\alpha y^3e^{-\alpha r^2} + 2\alpha ye^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right).\end{aligned}$$

Final form is

$$\nabla\psi_5(x,y) = \left(\left(2\alpha x(1-y^2)\right)\cdot\mathbf{e}_x + \left(2y - 2\alpha y^3 + 2\alpha y\right)\cdot\mathbf{e}_y\right)e^{-\alpha r^2}. \tag{6.22}$$

Final gradient is

$$\begin{aligned}\nabla\psi_6(x,y) = \nabla xye^{-\alpha r^2} &= \left(\frac{\partial}{\partial x}\left[xye^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[xye^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(ye^{-\alpha r^2} - 2\alpha x^2ye^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(xe^{-\alpha r^2} - 2\alpha xy^2e^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right),\end{aligned}$$

so that

$$\nabla\psi_6(x,y) = \left(\mathbf{e}_x\left(y - 2\alpha x^2y\right) + \mathbf{e}_y\left(x - 2\alpha xy^2\right)\right)e^{-\alpha r^2}. \tag{6.23}$$

Laplacian vectors can be derived as

$$\begin{aligned}\Delta\psi_4(x,y) &= \left(\frac{\partial}{\partial x}\left(2xe^{-\alpha r^2} - 2\alpha x^3e^{-\alpha r^2} + 2\alpha xe^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left((1-x^2)2\alpha ye^{-\alpha r^2}\right)\right)\\ &= \left(2e^{-\alpha r^2} - 4\alpha x^2e^{-\alpha r^2} - 6\alpha x^2e^{-\alpha r^2} + 4\alpha^2x^4e^{-\alpha r^2} + 2\alpha e^{-\alpha r^2} - 4\alpha^2x^2e^{-\alpha r^2}\right.\\ &\quad\left. + (1-x^2)2\alpha e^{-\alpha r^2} + (x^2-1)4\alpha^2y^2e^{-\alpha r^2}\right)\\ &= \left(2 - 4\alpha x^2 - 6\alpha x^2 + 4\alpha^2x^4 + 2\alpha - 4\alpha^2x^2 + (1-x^2)2\alpha + (x^2-1)4\alpha^2y^2\right)e^{-\alpha r^2}\\ &= \left(2 - 4\alpha x^2 - 6\alpha x^2 + 4\alpha^2x^4 + 2\alpha - 4\alpha^2x^2 + 2\alpha - 2\alpha x^2 + 4\alpha^2x^2y^2 - 4\alpha^2y^2\right)e^{-\alpha r^2}\end{aligned}$$

Finally,

$$\Delta\psi_4(x,y) = \left(2 - 12\alpha x^2 + 4\alpha^2x^2r^2 + 4\alpha - 4\alpha^2r^2\right)e^{-\alpha r^2} \tag{6.24}$$

Applying the same trick we used to obtain $\Delta\psi_3$ yields

$$\Delta\psi_5(x,y) = \left(2 - 12\alpha y^2 + 4\alpha^2y^2r^2 + 4\alpha - 4\alpha^2r^2\right)e^{-\alpha r^2}. \tag{6.25}$$

And the final Laplacian can be found as

$$\begin{aligned}
&\Delta\psi_6(x,y) = \Big(\frac{\partial}{\partial x}\Big(ye^{-\alpha r^2} - 2\alpha x^2 ye^{-\alpha r^2}\Big) + \frac{\partial}{\partial y}\Big(xe^{-\alpha r^2} - 2\alpha xy^2 e^{-\alpha r^2}\Big)\Big)\\
&= \Big(4\alpha^2 x^3 ye^{-\alpha r^2} - 2\alpha xye^{-\alpha r^2} - 4\alpha xye^{-\alpha r^2} + 4\alpha^2 xy^3 e^{-\alpha r^2} - 2\alpha xye^{-\alpha r^2} - 4\alpha xye^{-\alpha r^2}\Big)\\
&= \Big(4\alpha^2 x^3 y - 2\alpha xy - 4\alpha xy + 4\alpha^2 xy^3 - 2\alpha xy - 4\alpha xy\Big)e^{-\alpha r^2}
\end{aligned}$$

Final form of $\Delta\psi_6$ is

$$\Delta\psi_6(x,y) = \Big(-12\alpha xy + 4\alpha^2 xyr^2\Big)e^{-\alpha r^2}. \tag{6.26}$$

We will also need derivatives of all six wave functions with respect to variational parameter α:

$$\frac{\partial\psi_n}{\partial\alpha} = -r^2\psi_n \quad \text{,where} \quad n = 1,2,3,4,5,6. \tag{6.27}$$

6.3 Derivative of Expectation Value of Local Energy

Given expectational value of local energy

$$\langle E_L\rangle = \frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle}$$

find analytical expression for

$$\frac{\partial\langle E_L\rangle}{\partial\alpha}.$$

Here we omit using subindex in the trial wave function to keep notation clear. Partial derivatives are used because trial wave function is also a function of coordinates. Taking derivative of general expression:

$$\begin{aligned}
\frac{\partial\langle E_L\rangle}{\partial\alpha} &= \frac{\partial}{\partial\alpha}\left(\frac{\langle\Psi|\hat{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle}\right)\\
=&\frac{\frac{\partial}{\partial\alpha}\Big(\langle\Psi|\hat{H}|\Psi\rangle\Big)\langle\Psi|\Psi\rangle - \langle\Psi|\hat{H}|\Psi\rangle\frac{d}{d\alpha}\left(\langle\Psi|\Psi\rangle\right)}{\langle\Psi|\Psi\rangle^2}.
\end{aligned}$$

Compute terms with derivatives separately starting with

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\Big(\langle\Psi|\hat{H}|\Psi\rangle\Big) &= \frac{\partial}{\partial\alpha}\Big(\int\hat{H}|\Psi|^2 dV\Big) = \int\Big(\frac{\partial}{\partial\alpha}(\Psi)\hat{H}\Psi + \hat{H}\Psi\frac{\partial}{\partial\alpha}(\Psi)\Big)dV\\
&= 2\int\hat{H}\Psi\frac{\partial\Psi}{\partial\alpha}dV = 2\int\Psi^2\frac{1}{\Psi^2}\hat{H}\Psi\frac{\partial\Psi}{\partial\alpha}dV\\
&= 2\int\Psi^2\frac{\hat{H}\Psi}{\Psi}\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV = 2\int\Psi^2 E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV\\
&= \frac{2\int\Psi^2 E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV}{\langle\Psi|\Psi\rangle}\langle\Psi|\Psi\rangle = 2\left\langle E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}\right\rangle\langle\Psi|\Psi\rangle.
\end{aligned}$$

There are two forms of equation above. We can use logarithmic identity

$$\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha} = \frac{\partial\ln(\Psi)}{\partial\alpha}$$

so that it becomes

$$\frac{\partial}{\partial \alpha}\left(\langle\Psi|\,\hat{H}\,|\Psi\rangle\right) = 2\left\langle E_L \frac{\partial \ln(\Psi)}{\partial \alpha}\right\rangle \langle\Psi|\Psi\rangle\,.$$

Alternatively, we can define

$$\bar{\Psi} = \frac{\partial \Psi}{\partial \alpha}.$$

By using this convention,

$$\frac{\partial}{\partial \alpha}\left(\langle\Psi|\,\hat{H}\,|\Psi\rangle\right) = 2\left\langle E_L \frac{\bar{\Psi}}{\Psi}\right\rangle \langle\Psi|\Psi\rangle\,.$$

In our case, trial wave function is a real valued function, meaning $|\Psi|^2 = \Psi^2$. Second derivative then can be simplified as

$$\begin{aligned}\frac{\partial}{\partial \alpha}\langle\Psi_T|\Psi_T\rangle &= \frac{\partial}{\partial \alpha}\int |\Psi|^2 dV = \int \frac{\partial}{\partial \alpha}\left(\Psi\cdot\Psi\right)dV = 2\int \Psi\frac{\partial\Psi}{\partial\alpha}dV \\ &= 2\int \Psi^2\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV = \frac{2\int \Psi^2\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV}{\langle\Psi|\Psi\rangle}\langle\Psi|\Psi\rangle\,.\end{aligned}$$

In logarithmic form it is equal to

$$\frac{\partial}{\partial \alpha}\langle\Psi|\Psi\rangle = 2\int \Psi^2\frac{\partial \ln(\Psi)}{\partial \alpha}dV = 2\left\langle\frac{\partial \ln(\Psi)}{\partial\alpha}\right\rangle\langle\Psi|\Psi\rangle\,.$$

Second form is

$$\frac{\partial}{\partial \alpha}\langle\Psi|\Psi\rangle = 2\int \Psi^2\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV = 2\int |\Psi|^2\frac{\bar{\Psi}}{\Psi}dV = 2\left\langle\frac{\bar{\Psi}}{\Psi}\right\rangle\langle\Psi|\Psi\rangle\,.$$

Finally, the expression for local energy derivative becomes

$$\begin{aligned}\frac{\partial\langle E_L\rangle}{\partial\alpha} &= \frac{\frac{\partial}{\partial\alpha}\left(\langle\Psi|\,E_L\,|\Psi\rangle\right)\langle\Psi|\Psi\rangle - \langle\Psi|\,E_L\,|\Psi\rangle\frac{\partial}{\partial\alpha}\left(\langle\Psi|\Psi\rangle\right)}{\langle\Psi|\Psi\rangle^2} \\ &= \frac{2\left\langle E_L\frac{\bar{\Psi}}{\Psi}\right\rangle\langle\Psi|\Psi\rangle - 2\langle E_L\rangle\left\langle\frac{\bar{\Psi}}{\Psi}\right\rangle\langle\Psi|\Psi\rangle}{\langle\Psi|\Psi\rangle}\end{aligned}$$

and

$$\frac{\partial\langle E_L\rangle}{\partial\alpha} = 2\left\langle E_L\frac{\bar{\Psi}}{\Psi}\right\rangle - 2\langle E_L\rangle\left\langle\frac{\bar{\Psi}}{\Psi}\right\rangle. \tag{6.28}$$

6.4 Bosonic System: Analytical Solution, Local Energy & Observables

We will consider the most general case – N particles in 3D. Trial wave function is given by (3.5):

$$\Psi = e^{-\alpha\sum_{i=1}^{N} r_i^2} = \prod_{i=1}^{n} e^{-\alpha x_i^2}\cdot e^{-\alpha y_i^2}\cdot e^{-\alpha z_i^2}.$$

Hamiltonian is given by terms (2.35) and (2.37):

$$\hat{H} = \sum_{i=1}^{N}\left[-\frac{1}{2}\left(\frac{d^2}{dx_i^2}+\frac{d^2}{dy_i^2}+\frac{d^2}{dz_i^2}\right)+\frac{1}{2}\left(x_i^2+y_i^2+z_i^2\right)\right].$$

We find GS energy by solving equation (2.32)

$$E_0(\alpha) = \int P E_L dV,$$

where probability density function (2.30) is

$$P = \frac{|\Psi|^2}{\int |\Psi|^2 dV}$$

and local energy (2.31) is

$$E_L = \frac{\hat{H}\Psi}{\Psi}.$$

Laplacian of wave function with $n_x = n_y = n_z = 0$ is given by (6.11):

$$\Delta\psi_1(x,y,z) = \left(\frac{\partial^2}{\partial x^2}\left[e^{-\alpha r^2}\right] + \frac{\partial^2}{\partial y^2}\left[e^{-\alpha r^2}\right] + \frac{\partial^2}{\partial z^2}\left[e^{-\alpha r^2}\right]\right) = \left(-6\alpha + 4\alpha^2 r^2\right) e^{-\alpha r^2}.$$

Local energy can be expanded as

$$\begin{aligned} E_L = \frac{\hat{H}\Psi}{\Psi} &= \frac{1}{\Psi}\sum_{i=1}^{N}\left[-\frac{1}{2}\left(\frac{d^2}{dx_i^2} + \frac{d^2}{dy_i^2} + \frac{d^2}{dz_i^2}\right)\Psi + \frac{1}{2}\left(x_i^2 + y_i^2 + z_i^2\right)\Psi\right] \\ &= \sum_{i=1}^{N}\left[-\frac{1}{2}\left(-6\alpha + 4\alpha^2(x_i^2 + y_i^2 + z_i^2)\right) + \frac{1}{2}\left(x_i^2 + y_i^2 + z_i^2\right)\right] \\ &= \sum_{i=1}^{N}\left[3\alpha - 2\alpha^2(x_i^2 + y_i^2 + z_i^2) + \frac{1}{2}\left(x_i^2 + y_i^2 + z_i^2\right).\right]. \end{aligned}$$

Local energy during VMC iterations is computed using

$$\boxed{E_L = 3\alpha N + \left(\frac{1}{2} - 2\alpha^2\right)\sum_{i=1}^{N} r_i^2}. \tag{6.29}$$

Absolute value of wave function squared is equal to

$$\begin{aligned} |\Psi|^2 = &\int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha x_N^2} dx_N \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha y_N^2} dy_N \\ &\times \int_{-\infty}^{\infty} e^{-2\alpha z_1^2} dz_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N = \mathcal{I}_1^{3N}. \end{aligned}$$

Here we defined $\mathcal{I}_1$ as

$$\mathcal{I}_1(x) = \int_{-\infty}^{\infty} e^{-2\alpha x^2} dx.$$

Gaussian integral is given by [32]

$$G = \int_{-\infty}^{\infty} e^{-ax^2} dx = \sqrt{\frac{\pi}{a}}$$

where $a > 0$. So that

$$\mathcal{I}_1(x) = \int_{-\infty}^{\infty} e^{-2\alpha x^2} dx = \sqrt{\frac{\pi}{2\alpha}}. \tag{6.30}$$

Using this table integral, absolute value of wave function becomes

$$|\Psi|^2 = \mathcal{I}_1^{3N} = \left(\frac{\pi}{2\alpha}\right)^{3N/2}.$$

Therefore probability density function is

$$P = \left(\frac{2\alpha}{\pi}\right)^{3N/2} |\Psi|^2 = \left(\frac{2\alpha}{\pi}\right)^{3N/2} \prod_{i=1}^{N} e^{-2\alpha x_i^2} e^{-2\alpha y_i^2} e^{-2\alpha z_i^2}.$$

Ground State energy can be simplified as

$$\begin{aligned}
E_0 &= \left(\frac{2\alpha}{\pi}\right)^{3N/2} \\
&\times \int_{-\infty}^{\infty} \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} \sum_{i=1}^{N} \left[3\alpha - 2\alpha^2(x_i^2+y_i^2+z_i^2) + \frac{1}{2}\left(x_i^2+y_i^2+z_i^2\right)\right] dV \\
&= \left(\frac{2\alpha}{\pi}\right)^{3N/2} \sum_{i=1}^{N} \left[3\alpha \int_{-\infty}^{\infty} \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV \right. \\
&\left. + \left(-2\alpha^2 + \frac{1}{2}\right) \int_{-\infty}^{\infty} \left(x_i^2+y_i^2+z_i^2\right) \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV \right] \\
&= \left(\frac{2\alpha}{\pi}\right)^{3N/2} (\mathcal{T}_1 + \mathcal{T}_2).
\end{aligned}$$

Let us compute these terms separately,

$$\mathcal{T}_1 = 3\alpha \sum_{i=1}^{N} \int_{-\infty}^{\infty} \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV$$

These is no dependence of index i under the sum, and therefore it is a sum of N identical integrals

$$\begin{aligned}
\mathcal{T}_1 = 3N\alpha & \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha x_N^2} dx_N \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha y_N^2} dy_N \\
& \int_{-\infty}^{\infty} e^{-2\alpha z_1^2} dz_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N \ = 3N\alpha \mathcal{I}_1^N \cdot \mathcal{I}_1^N \cdot \mathcal{I}_1^N = 3N\alpha \left(\frac{\pi}{2\alpha}\right)^{3N/2}.
\end{aligned}$$

Second term:

$$\mathcal{T}_2 = \left(-2\alpha^2 + \frac{1}{2}\right) \sum_{i=1}^{N} \int_{-\infty}^{\infty} \left(x_i^2+y_i^2+z_i^2\right) \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV$$

Let us take a look at the first term of the sum with $i = 1$:

$$\begin{aligned}
& \int_{-\infty}^{\infty} x_1^2 e^{-2\alpha x_1^2} dx_1 \int_{-\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \cdots \int_{-\infty}^{\infty} e^{-2\alpha x_N^2} dx_N \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N \\
&+ \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} y_1^2 e^{-2\alpha y_1^2} dy_1 \int_{-\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N \\
&+ \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} z_1^2 e^{-2\alpha z_1^2} dz_1 \int_{-\infty}^{\infty} e^{-2\alpha z_2^2} dz_2 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N \\
&= \int_{-\infty}^{\infty} x_1^2 e^{-2\alpha x_1^2} dx_1 \cdot \mathcal{I}_1^{3N-1} + \int_{-\infty}^{\infty} y_1^2 e^{-2\alpha y_1^2} dy_1 \cdot \mathcal{I}_1^{3N-1} + \int_{-\infty}^{\infty} z_1^2 e^{-2\alpha z_1^2} dz_1 \cdot \mathcal{I}_1^{3N-1}.
\end{aligned}$$

We can define
$$\mathcal{I}_2 = \int_{-\infty}^{\infty} x^2 e^{-2\alpha x^2} dx.$$
This integral can be obtained from $\mathcal{I}_1$ by taking derivative with respect to α:
$$\frac{\partial}{\partial\alpha}\int_{-\infty}^{\infty} e^{-2\alpha x^2} dx = \frac{\partial}{\partial\alpha}\sqrt{\frac{\pi}{2\alpha}} \implies \int_{-\infty}^{\infty} -2x^2 e^{-2\alpha x^2} dx = -\sqrt{\frac{\pi}{8\alpha^3}}.$$
Therefore,
$$\mathcal{I}_2 = \int_{-\infty}^{\infty} x^2 e^{-2\alpha x^2} dx = \frac{1}{2}\sqrt{\frac{\pi}{8\alpha^3}}. \tag{6.31}$$
Term with $i = 1$ in $\mathcal{T}_2$ is equal to
$$\left(-2\alpha^2 + \frac{1}{2}\right)\cdot 3\mathcal{I}_{\in}\mathcal{I}_1^{3N-1} = \left(-2\alpha^2 + \frac{1}{2}\right)\cdot\frac{3}{2}\sqrt{\frac{\pi}{8\alpha^3}}\cdot\left(\sqrt{\frac{\pi}{2\alpha}}\right)^{3N-1}.$$
All other terms in the sum with $i = 2, 3, ..., N$ have a similar structure – three integrals $\mathcal{I}_2$ and $3(N-1)$ $\mathcal{I}_1$ integrals, so that
$$\mathcal{T}_2 = \left(-2\alpha^2 + \frac{1}{2}\right)\cdot\frac{3N}{2}\cdot\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}.$$
Finally, we can find analytical expression of local energy:
$$\begin{aligned}
E_0 = &\left(\frac{2\alpha}{\pi}\right)^{3N/2}\left[3N\alpha\left(\frac{\pi}{2\alpha}\right)^{3N/2} - 3\alpha^2 N\cdot\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}\right.\\
&\left.+\frac{3}{4}N\cdot\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}\right] = -3\alpha^2 N\left(\frac{2\alpha}{\pi}\right)^{3N/2}\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}\\
&3N\alpha\left(\frac{2\alpha}{\pi}\right)^{3N/2}\left(\frac{\pi}{2\alpha}\right)^{3N/2} + \frac{3}{4}N\left(\frac{2\alpha}{\pi}\right)^{3N/2}\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}\\
&= 3N\alpha - 3\alpha^2 N\left(\frac{2\alpha}{\pi}\right)^{1/2}\left(\frac{\pi}{8\alpha^3}\right)^{1/2} + \frac{3}{4}N\left(\frac{2\alpha}{\pi}\right)^{1/2}\left(\frac{\pi}{8\alpha^3}\right)^{1/2}\\
&= 3N\alpha - 3\alpha^2 N\frac{1}{2\alpha} + \frac{3}{4}N\frac{1}{2\alpha} = 3N\alpha - \frac{3}{2}N\alpha + \frac{3}{8}\frac{N}{\alpha} = \frac{3}{2}N\alpha + \frac{3}{8}\frac{N}{\alpha}.
\end{aligned}$$
Finally, GS energy for 3D BEC system as a function of α is
$$\boxed{E_0 = \frac{3N}{2}\left(\alpha + \frac{1}{4\alpha}\right)}. \tag{6.32}$$

We can find optimal variational parameter $\tilde{\alpha}$ by setting derivative of E_0 with respect to α:
$$\frac{dE_0}{d\alpha} = \frac{3}{2}N - \frac{3}{8\tilde{\alpha}^2}N = 0 \implies \tilde{\alpha}^2 = \frac{1}{4} \implies \tilde{\alpha} = \pm\frac{1}{2}.$$
We reject $\alpha = -1/2$, because Gaussian integral requires parameter α to be positive, and therefore
$$\boxed{\tilde{\alpha} = \frac{1}{2}}. \tag{6.33}$$

And GS energy is then
$$\boxed{E(\tilde{\alpha}) = N\left(\frac{3}{4} + \frac{6}{8}\right) = \frac{3N}{2}}. \tag{6.34}$$

Finally, score O_1 is given by (3.13):

$$O_1 = \frac{\tilde{\Psi}}{\Psi},$$

where $\tilde{\Psi}$ (3.12) is

$$\tilde{\Psi} = \frac{\partial \Psi}{\partial \alpha}.$$

It can be compututed as

$$O_1 = \frac{1}{\Psi}\frac{\partial \Psi}{\partial \alpha} = \frac{\partial}{\partial \alpha} e^{-\alpha \sum_{i=1}^{N} r_i^2} = -\sum_{i=0}^{N} r_i^2.$$

During VMC, observable O_1 is computed as

$$\boxed{O_1 = -\sum_{i=0}^{N} r_i^2}. \tag{6.35}$$

6.5 First Closed Shell: Analytical Solution & Observables

We ain to find analytical solution and analytical expressions for local energy and O_1. Trial wave function for the system is given by (6.3):

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = D_\uparrow(\mathbf{r}_1) D_\downarrow(\mathbf{r}_2).$$

In this case, Slater determinants are 1×1, meaning that

$$D_\uparrow(\mathbf{r}_1) = \psi_1(\mathbf{r}_1) = e^{-\alpha r_1^2} \quad \text{and} \quad D_\downarrow(\mathbf{r}_2) = \psi_1(\mathbf{r}_2),$$

where ψ_1 is given by (6.7):

$$\psi_1(\mathbf{r}) = e^{-\alpha r^2}.$$

Let us begin with deriving expression for local energy, using SD formulation (3.20)

$$\begin{aligned}
E_L &= -\frac{1}{2}\sum_{i=1}^{N/2} \frac{\Delta_i D_\uparrow}{D_\uparrow} - \frac{1}{2}\sum_{i=N/2+1}^{N} \frac{\Delta_i D_\downarrow}{D_\downarrow} + \frac{1}{2}\sum_{i=1}^{N} r_i^2 \\
&\quad - \frac{1}{2}\frac{\Delta_1 D_\uparrow}{D_\uparrow} - \frac{1}{2}\frac{\Delta_1 D_\downarrow}{D_\downarrow} + \frac{1}{2}(r_1^2 + r_2^2) \\
&\quad - \frac{1}{2}\frac{\Delta_1 \psi_1(\mathbf{r}_1)}{\psi_1(\mathbf{r}_1)} - \frac{1}{2}\frac{\Delta_2 \psi_1(\mathbf{r}_2)}{\psi_1(\mathbf{r}_2)} + \frac{1}{2}(r_1^2 + r_2^2).
\end{aligned}$$

Laplacian is given by (6.10):

$$\Delta \psi_1(x, y) = \left(-4\alpha + 4\alpha^2 r^2\right) e^{-\alpha r^2}.$$

Therefore local energy is

$$\begin{aligned}
E_L &= -\frac{1}{2}\left(-4\alpha + 4\alpha^2 r^2\right) - \frac{1}{2}\left(-4\alpha + 4\alpha^2 r^2\right) + \frac{1}{2}(r_1^2 + r_2^2) \\
&= 4\alpha - 2\alpha^2(r_1^2 + r_2^2) + \frac{1}{2}(r_1^2 + r_2^2) = 4\alpha + (r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right).
\end{aligned}$$

Numerically, local energy is computed using

$$\boxed{E_L = 4\alpha + (r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right)}. \tag{6.36}$$

Score O_1 is given by (3.22):

$$O_1 = \sum_{i=1}^{N/2}\sum_{j=1}^{N/2}(D_\uparrow^{-1})_{ij}\frac{\partial\psi_j(\mathbf{r}_i)}{\partial\alpha} + \sum_{k=1}^{N/2}\sum_{l=1}^{N/2}(D_\downarrow^{-1})_{kl}\frac{\partial\psi_l(\mathbf{r}_{N/2+k})}{\partial\alpha}$$

For $N = 2$, $D_\uparrow(\mathbf{r}_1) = \psi_1(\mathbf{r}_1)$ and $D_\downarrow(\mathbf{r}_2) = \psi_1(\mathbf{r}_2)$ it simplifies to

$$O_1 = \psi_1^{-1}(\mathbf{r}_1)\frac{\partial\psi_1(\mathbf{r}_1)}{\partial\alpha} + \psi_1^{-1}(\mathbf{r}_2)\frac{\partial\psi_1(\mathbf{r}_2)}{\partial\alpha}.$$

Derivative of ψ_1 with respect to α is given by (6.26):

$$\frac{\partial\psi_1(\mathbf{r})}{\partial\alpha} = -r^2\psi_1(\mathbf{r}),$$

so that

$$O_1 = \psi_1^{-1}(\mathbf{r}_1)r_1^2\psi_1(\mathbf{r}_1) + \psi_1^{-1}(\mathbf{r}_2)r_2^2\psi_1(\mathbf{r}_2) = r_1^2 + r_2^2.$$

Numerically score is computed as

$$\boxed{O_1 = r_1^2 + r_2^2}. \tag{6.37}$$

Now, let us derive analytical solution. It is very similar to the one obtained in Appendix 6.4. Probability density function (2.30) simplifies to

$$P = \frac{|\Psi|^2}{\int|\Psi|^2 dV} = \frac{e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}}{\int e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}dV}.$$

Compute normalization integrals separately:

$$\int e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}dV = \int_{-\infty}^{\infty} e^{-2\alpha x_1^2}dx_1 \int_{-\infty}^{\infty} e^{-2\alpha y_1^2}dy_1 \int_{-\infty}^{\infty} e^{-2\alpha x_2^2}dx_2 \int_{-\infty}^{\infty} e^{-2\alpha y_2^2}dy_2$$
$$= \mathcal{I}_1^4 = \left(\sqrt{\frac{\pi}{2\alpha}}\right)^4 = \frac{\pi^2}{4\alpha^2}.$$

Here, we used the same integral (6.29):

$$\mathcal{I}_1 = \sqrt{\frac{\pi}{2\alpha}}.$$

Probability distribution function is then

$$P = \frac{4\alpha^2}{\pi^2}e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}.$$

GS energy (2.32) is then equal to

$$E_0 = \int PE_L dV = \frac{4\alpha^2}{\pi^2}\int\left[e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}\left((r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right) + 4\alpha\right)\right]dV$$
$$= \frac{4\alpha^2}{\pi^2}\left[4\alpha\int_{\infty}^{\infty} e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}dV + \left(\frac{1}{2} - 2\alpha^2\right)\int_{\infty}^{\infty} e^{-2\alpha r_1^2}e^{-2\alpha r_2^2}(r_1^2 + r_2^2)dV\right]$$
$$= \frac{4\alpha^2}{\pi^2}\left(\mathcal{T}_1 + \mathcal{T}_2\right)$$

Compute $\mathcal{T}_1$ and $\mathcal{T}_2$ separately starting with the first one,

$$\begin{aligned}\mathcal{T}_1 &= 4\alpha \int_{\infty}^{\infty} e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} dV \\ &= 4\alpha \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \int_{-\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \int_{-\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \\ &= 4\alpha \mathcal{I}_1^4 = 4\alpha \frac{\pi^2}{4\alpha^2} = \frac{\pi^2}{\alpha}.\end{aligned}$$

Proceed with the second one,

$$\begin{aligned}\mathcal{T}_2 &= \left(\frac{1}{2} - 2\alpha^2\right) \int_{\infty}^{\infty} e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} (r_1^2 + r_2^2) dV \\ &= \left(\frac{1}{2} - 2\alpha^2\right) \Bigg[\int_{\infty}^{\infty} e^{-2\alpha(x_1^2+y_1^2)} e^{-2\alpha(x_2^2+y_2^2)} (x_1^2 + y_1^2) dx_1 dy_1 dx_2 dy_2 \\ &+ \int_{\infty}^{\infty} e^{-2\alpha(x_1^2+y_1^2)} e^{-2\alpha(x_2^2+y_2^2)} (x_2^2 + y_2^2) dx_1 dy_1 dx_2 dy_2 \Bigg] \\ &= \left(\frac{1}{2} - 2\alpha^2\right) \Bigg[\int_{\infty}^{\infty} x_1^2 e^{-2\alpha x_1^2} dx_1 \int_{\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \int_{\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \int_{\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \\ &+ \int_{\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \int_{\infty}^{\infty} y_1^2 e^{-2\alpha y_1^2} dy_1 \int_{\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \int_{\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \\ &+ \int_{\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \int_{\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \int_{\infty}^{\infty} x_2^2 e^{-2\alpha x_2^2} dx_2 \int_{\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \\ &+ \int_{\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \int_{\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \int_{\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \int_{\infty}^{\infty} y_2^2 e^{-2\alpha y_2^2} dy_2 \Bigg] \\ &= \left(\frac{1}{2} - 2\alpha^2\right) \cdot 4\mathcal{I}_1^3 \mathcal{I}_2.\end{aligned}$$

Here we have used definition of $\mathcal{I}_2$ (6.30):

$$\mathcal{I}_2 = \int_{\infty}^{\infty} x^2 e^{-2\alpha x^2} dx = \frac{1}{2}\sqrt{\frac{\pi}{8\alpha^3}}.$$

Second term then simplifies to

$$\begin{aligned}\mathcal{T}_2 &= \left(\frac{1}{2} - 2\alpha^2\right) \cdot 4\mathcal{I}_1^3 \mathcal{I}_2 = \left(\frac{1}{2} - 2\alpha^2\right) \cdot 4 \cdot \left(\frac{\pi}{2\alpha}\right)^{3/2} \cdot \frac{1}{2} \cdot \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \\ &= \left(\frac{1}{2} - 2\alpha^2\right) \frac{\pi^2}{4\alpha^3} = \frac{\pi^2}{8\alpha^3} - \frac{\pi^2}{2\alpha}.\end{aligned}$$

And ground state energy as a function of α is

$$E_0 = \frac{4\alpha^2}{\pi^2}\left(\frac{\pi^2}{\alpha} + \frac{\pi^2}{8\alpha^3} - \frac{\pi^2}{2\alpha}\right) = 4\alpha + \frac{1}{2\alpha} - 2\alpha = 2\alpha + \frac{1}{2\alpha}.$$

Therefore,

$$\boxed{E_0 = 2\alpha + \frac{1}{2\alpha}}. \tag{6.38}$$

Optimal variational parameter $\tilde{\alpha}$ is then

$$\frac{dE_0}{d\alpha} = 2 - \frac{1}{2\alpha} = 0 \implies \alpha^2 = \frac{1}{4} \implies \tilde{\alpha} = \pm\frac{1}{2}.$$

As before, we discard solution with minus sign due to Gaussian integrals restriction of its parameters and

$$\boxed{\tilde{\alpha} = \frac{1}{2}}. \tag{6.39}$$

Minimized GS energy is equal to

$$\boxed{E_0(\tilde{\alpha}) = 2}. \tag{6.40}$$

6.6 Interacting Fermions: General Expressions & Observables

Hamiltonian in atomic units is given by (2.35), (2.36) and (2.37):

$$\hat{H} = \sum_{i=1}^{N} \left[-\frac{1}{2}\Delta_i + \frac{1}{2}B(\omega)r_i^2 + \sum_{j>i}^{N} \frac{1}{r_{ij}} \right].$$

Trial wave functions can be written as a product

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha, \beta) = \Phi(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha) I(\mathbf{r}_1, ..., \mathbf{r}_N; \beta),$$

where I represents Jastrow factors in the form J (2.35) or in the form P (2.36):

$$J(\mathbf{r}_1, ..., \mathbf{r}_N; \beta_1, ..., \beta_p) = \exp\left(\sum_{i=1}^{N} \sum_{j>i}^{N} \beta_{ij} r_{ij} \right) = \exp\left(\sum_{k=1}^{p} \beta_k r_k \right);$$

$$P(\mathbf{r}_1, ..., \mathbf{r}_N; \beta) = \exp\left(\sum_{i=1}^{N} \sum_{j>i}^{N} \frac{a r_{ij}}{1+\beta r_{ij}} \right) = \exp\left(\sum_{k=1}^{p} \frac{a r_k}{1+\beta r_k} \right).$$

Here we have defined number of pairs as $p = N(N-1)/2$ and used transformation (3.19) of 2D index (i, j) into 1D index k. Φ represents non-interacting wave function (3.3):

$$\Phi(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha) = D_\uparrow(\mathbf{r}_1, ..., \mathbf{r}_{N/2}; \alpha) D_\downarrow(\mathbf{r}_{N/2+1}, ..., \mathbf{r}_N; \alpha).$$

Begin with the scores O_2 for Jastrow and Pade-Jastrow factors:

$$O_1 = \frac{\tilde{\Psi}_T}{\Psi_T} = \frac{1}{\Psi_T}\nabla_\beta \Psi_T = \frac{1}{\Phi I}\nabla_\beta (\Phi I) = \frac{\nabla_\beta I}{I} = \frac{\tilde{I}}{I} = \nabla_\beta \ln I.$$

Here we have used gradient vector instead of the simple partial derivative because Jastrow factor J contains $N/2(N-1)$ variational parameters β_{ij}, and O_2 as well as $\tilde{I}$ become vectors.

For Jastrow factor $I = J$ the score is

$$\mathbf{O}_1^{(J)} = \nabla_\beta \ln J = \nabla_\beta \left(\sum_{i=1}^{N} \sum_{j>i}^{N} \beta_{ij} r_{ij} \right) = \nabla_\beta \left(\sum_{k=0}^{N-1} \beta_k r_k \right) = r_0 \mathbf{e}_0 + \cdots + r_{N-1}\mathbf{e}_{N-1}.$$

In the component form score is

$$\mathbf{O}_1^{(J)} = (r_0, r_1, \cdots, r_{N-1}). \tag{6.41}$$

for Pade-Jastrow factor $I = P$:

$$O_1^{(P)} = \frac{\partial}{\partial \beta} \ln P = \frac{\partial}{\partial \beta} \left(\sum_{i=1}^{N} \sum_{j>i}^{N} \frac{a r_{ij}}{1+\beta r_{ij}} \right) = \sum_{k=0}^{N-1} \frac{\partial}{\partial \beta} \left(\frac{a r_k}{1+\beta r_k} \right) = \sum_{k=0}^{N-1} \frac{-a r_k^2}{(1+\beta r_k)^2}.$$

So that

$$O_1^{(P)} = \sum_{k=0}^{N-1} \frac{-ar_k^2}{(1+\beta r_k)^2}. \tag{6.42}$$

Proceed with deriving analytical expression for local energy. We can write local energy in the form

$$E_L = \frac{\hat{H}\Psi_T}{\Psi_T} = \frac{\left(\hat{H}_0 + \hat{H}_I + \hat{V}\right)\Psi_T}{\Psi_T}.$$

Let us consider these terms separately starting with

$$\frac{\hat{H}_0\Psi_T}{\Psi_T} = -\frac{1}{2}\cdot\frac{\sum_{i=1}^N \Delta_i(\Phi I)}{\Phi I}. \tag{6.43}$$

Laplacian can be expanded using the chain rule:

$$\begin{aligned}\Delta(\Phi I) &= \nabla(\nabla(\Phi I)) = \nabla(I\nabla\Phi + \Phi\nabla I) = \nabla(I\nabla\Phi) + \nabla(\Phi\nabla I)\\ &= I\Delta\Phi + \nabla I\nabla\Phi + \nabla\Phi\nabla I + \Phi\Delta I = I\Delta\Phi + 2\nabla I\nabla\Phi + \Phi\Delta I.\end{aligned}$$

Inserting it back to expression to (6.4) gives

$$\begin{aligned}\frac{\hat{H}_0\Psi_T}{\Psi_T} &= -\frac{1}{2}\cdot\frac{\sum_{i=1}^N(I\Delta_i\Phi + 2\nabla_i I\nabla_i\Phi + \Phi\Delta_i I)}{\Phi I}\\ &= -\frac{1}{2}\frac{\sum_{i=1}^N\Delta_i\Phi}{\Phi} - \frac{\sum_{i=1}^N\nabla_i\Phi\nabla_i I}{\Phi I} - \frac{1}{2}\frac{\sum_{i=1}^N\Delta_i I}{I}.\end{aligned}$$

In the expression above, first term represents local energy of non-interacting wave function Φ and additional two term appear because of more complex form of the ansatz. Fist two terms in expression above can be rewritten as

$$\begin{aligned}&-\frac{1}{2}\frac{\sum_{i=1}^N\Delta_i(D_\uparrow D_\downarrow)}{D_\uparrow D_\downarrow} - \frac{\sum_{i=1}^N\nabla_i(D_\uparrow D_\downarrow)\nabla_i I}{D_\uparrow D_\downarrow I}\\ &= -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^N\Delta_i D_\downarrow}{D_\downarrow}\right] - \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^N\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right]\end{aligned}$$

Putting in back to equation (6.3):

$$\begin{aligned}\frac{\hat{H}_0\Psi_T}{\Psi_T} &= -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^N\Delta_i D_\downarrow}{D_\downarrow}\right]\\ &\quad - \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^N\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right] - \frac{1}{2}\frac{\sum_{i=1}^N\Delta_i I}{I}\\ &= \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3.\end{aligned}$$

Here, we have defined

$$\mathcal{E}_1 = -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^N\Delta_i D_\downarrow}{D_\downarrow}\right], \tag{6.44}$$

$$\mathcal{E}_2 = \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^N\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right] \tag{6.45}$$

and

$$\mathcal{E}_3 = -\frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i I}{I}. \tag{6.46}$$

Coulomb's term simplifies to

$$\frac{\hat{H}_I \Psi_T}{\Psi_T} = \frac{\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}}\cdot\Psi_T}{\Psi_T} = \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} = \hat{H}_I$$

and harmonic term simplifies in the similar way to

$$\frac{\hat{V}\Psi_T}{\Psi_T} = \frac{B(\omega)}{2}\cdot\frac{\sum_{i=1}^{N} r_i^2\Psi_T}{\Psi_T} = \frac{B(\omega)}{2}\sum_{i=1}^{N} r_i^2 = \hat{V}.$$

Therefore, local energy can be computed as

$$E_L = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3 + \hat{H}_I + \hat{V}. \tag{6.47}$$

Jastrow factors are exponential functions. Local energy can be written is different form using

$$\nabla \ln I = \frac{\nabla I}{I}$$

and

$$\Delta \ln I = \nabla(\nabla \ln I) = \frac{\Delta I}{I} - \frac{(\nabla I)^2}{I^2} \implies \frac{\Delta I}{I} = \Delta \ln I - (\nabla \ln I)^2.$$

Alternative form of expression (6.5) is

$$\mathcal{E}_2 = -\left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow \nabla_i \ln I}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow \nabla_i \ln I}{D_\downarrow}\right]. \tag{6.48}$$

Likewise, alternative form of equation (6.6) is

$$\mathcal{E}_3 = -\frac{1}{2}\sum_{i=1}^{N}\left(\Delta \ln I - (\nabla \ln I)^2\right). \tag{6.49}$$

Continue with gradient of Jastrow factor:

$$\nabla_i (\ln J) = \nabla_i \sum_{j=1}^{N}\sum_{k>j}^{N}\beta_{jk} r_{jk} = \sum_{j=1\neq i}^{N}\beta_{ij}\nabla_i r_{ij}.$$

Gradient is equal to

$$\begin{aligned}\nabla_i r_{ij} &= \left(\frac{\partial}{\partial x_i}\cdot\mathbf{e}_x + \frac{\partial}{\partial y_i}\cdot\mathbf{e}_y\right)\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\\ &= \frac{2(x_i - x_j)}{2r_{ij}}\mathbf{e}_x + \frac{2(y_i - y_j)}{2r_{ij}}\mathbf{e}_y = \frac{\mathbf{r}_i - \mathbf{r}_j}{r_{ij}},\end{aligned}$$

and

$$\nabla_i (\ln J) = \sum_{j=1\neq i}^{N}\beta_{ij}\frac{(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}}. \tag{6.50}$$

Proceed with Laplacian of Jastrow factor:

$$\Delta_i \ln J = \nabla_i \left[\sum_{j=1\neq i}^{N} \beta_{ij} \frac{(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}} \right].$$

It can be computed using

$$\nabla_i \frac{(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}} = \frac{2r_{ij} - (\mathbf{r}_i - \mathbf{r}_j)(\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1}}{r_{ij}^2} = \frac{1}{r_{ij}},$$

so that

$$\Delta_i \ln J = \sum_{j=1\neq i}^{N} \frac{\beta_{ij}}{r_{ij}} = \sum_{m=1\neq i}^{p} \frac{\beta_m}{r_m}. \tag{6.51}$$

Next is gradient of Pade-Jastrow factor:

$$\begin{aligned}
\nabla_i \ln P &= \nabla_i \sum_{j=1}^{N} \sum_{k>j}^{N} \frac{a r_{jk}}{1+\beta r_{jk}} = \sum_{j=1\neq i}^{N} \nabla_i \left(\frac{a r_{ij}}{1+\beta r_{ij}} \right) \\
&= \sum_{j=1\neq i}^{N} \frac{(1+\beta r_{ij}) a (\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1} - a r_{ij} \beta (\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1}}{(1+\beta r_{ij})^2} \\
&= \sum_{j=1\neq i}^{N} \frac{a (\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1} + \beta r_{ij} a (\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1} - a r_{ij} \beta (\mathbf{r}_i - \mathbf{r}_j) r_{ij}^{-1}}{(1+\beta r_{ij})^2} \\
&= \sum_{j=1\neq i}^{N} \frac{a (\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}(1+\beta r_{ij})^2}.
\end{aligned}$$

Numerically gradient is computing using

$$\nabla_i \ln P = \sum_{j=1\neq i}^{N} \frac{a (\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}(1+\beta r_{ij})^2}. \tag{6.52}$$

Finally, Laplacian of Pade-Jastrow factor:

$$\Delta_i (\ln P) = \nabla_i \left(\sum_{j=1\neq i}^{N} \frac{a (\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}(1+\beta r_{ij})^2} \right) = \sum_{j=1\neq i}^{N} \nabla_i \left[(\mathbf{r}_i - \mathbf{r}_j) \Phi(\mathbf{r}_i) \right],$$

where we have defined

$$\Phi(\mathbf{r}_i) = \frac{a}{r_{ij}(1+\beta r_{ij})^2}.$$

Laplacian in this form can be computed as

$$\nabla_i \left[(\mathbf{r}_i - \mathbf{r}_j) \Phi(\mathbf{r}_i) \right] = \Phi(\mathbf{r}_i) \nabla_i \left[\mathbf{r}_i - \mathbf{r}_j \right] + (\mathbf{r}_i - \mathbf{r}_j) \nabla_i \left[\Phi(\mathbf{r}_i) \right].$$

First term is

$$\begin{aligned}
&\nabla_i \left[\mathbf{r}_i - \mathbf{r}_j \right] = \frac{\partial}{\partial x_i} (x_i - x_j) + \frac{\partial}{\partial y_i} (y_i - y_j) = 2; \\
&\Phi(\mathbf{r}_i) \nabla_i \left[\mathbf{r}_i - \mathbf{r}_j \right] = 2\Phi(\mathbf{r}_i);
\end{aligned}$$

Second term:

$$\nabla_i \left[\Phi(\mathbf{r}_i)\right] = \nabla_i \left[\frac{a}{r_{ij}(1+\beta r_{ij})^2}\right] = \frac{-a\nabla_i \left[r_{ij}(1+\beta r_{ij})^2\right]}{r_{ij}^2(1+\beta r_{ij})^4}.$$

Compute gradient separately:

$$\begin{aligned}\nabla_i \left[r_{ij}(1+\beta r_{ij})^2\right] &= \left(\mathbf{r}_i - \mathbf{r}_j\right) r_{ij}^{-1}(1+\beta r_{ij})^2 + r_{ij}\cdot 2(1+\beta r_{ij})\cdot \beta \left(\mathbf{r}_i - \mathbf{r}_j\right) r_{ij}^{-1}\\ &= \left(\mathbf{r}_i - \mathbf{r}_j\right) r_{ij}^{-1}(1+\beta r_{ij})\left[(1+\beta r_{ij}) + 2r_{ij}\beta\right],\end{aligned}$$

so that

$$\begin{aligned}&\left(\mathbf{r}_i - \mathbf{r}_j\right)\nabla_i \left[\Phi(\mathbf{r}_i)\right] = -a\left(\mathbf{r}_i - \mathbf{r}_j\right)\frac{\left(\mathbf{r}_i - \mathbf{r}_j\right) r_{ij}^{-1}(1+\beta r_{ij})\left[(1+\beta r_{ij}) + 2r_{ij}\beta\right]}{r_{ij}^2(1+\beta r_{ij})^4}\\ &= -\frac{a r_{ij}(1+\beta r_{ij})\left[(1+\beta r_{ij}) + 2r_{ij}\beta\right]}{r_{ij}^2(1+\beta r_{ij})^4}.\end{aligned}$$

Collecting everything back:

$$\begin{aligned}&\nabla_i \left[\left(\mathbf{r}_i - \mathbf{r}_j\right)\Phi(\mathbf{r}_i)\right] = \frac{2a}{r_{ij}(1+\beta r_{ij})^2} - \frac{r_{ij}(1+\beta r_{ij})\left[(1+\beta r_{ij}) + 2r_{ij}\beta\right]}{r_{ij}^2(1+\beta r_{ij})^4}\\ &= \frac{2a r_{ij}(1+\beta r_{ij})^2 - a r_{ij}(1+\beta r_{ij})\left[(1+\beta r_{ij}) + 2r_{ij}\beta\right]}{r_{ij}^2(1+\beta r_{ij})^4}\\ &= a\frac{2(1+\beta r_{ij}) - \left[(1+\beta r_{ij}) + 2r_{ij}\beta\right]}{r_{ij}(1+\beta r_{ij})^3} = \frac{a(1-\beta r_{ij})}{r_{ij}(1+\beta r_{ij})^3},\end{aligned}$$

and final expression for the Laplacian is

$$\Delta_i \left(\ln P\right) = \sum_{j=1\neq i}^{N} \frac{a(1-\beta r_{ij})}{r_{ij}(1+\beta r_{ij})^3} = \sum_{m=1\neq i}^{p} \frac{a(1-\beta r_m)}{r_m(1+\beta r_m)^3}. \tag{6.53}$$

Bibliography

[1] Morten Hjorth-Jensen. Project 1: Bec variational monte carlo. Course project report, Computational Physics II, 2024. URL https://compphysics.github.io/ComputationalPhysics2/doc/Projects/2024/Project1/pdf/Project1.pdf. Accessed: 11 May 2025.

[2] Philip L. Bowers. *Lectures on Quantum Mechanics: A Primer for Mathematicians.* Cambridge University Press, 2020.

[3] Daniel F. Styer. *The Strange World of Quantum Mechanics.* Cambridge University Press, 2000. Reprinted 2000, 2003.

[4] Donald C Chang. Physical interpretation of planck's constant based on the maxwell theory. *Chinese Physics B*, 26(4):040301, April 2017. ISSN 1674-1056. doi: 10.1088/1674-1056/26/4/040301. URL http://dx.doi.org/10.1088/1674-1056/26/4/040301.

[5] Jim Baggott. What einstein meant by 'god does not play dice'. Aeon Ideas, 2018. URL https://aeon.co/ideas/what-einstein-meant-by-god-does-not-play-dice. Accessed: 7 May 2025.

[6] David J. Griffiths. *Introduction to Quantum Mechanics.* Prentice Hall, 1994.

[7] Beyond the Horizon. When is quantum mechanics needed? URL https://medium.com/@prmj2187/determine-when-quantum-mechanics-is-needed-with-de-broglie-wavelength-compton-wavelength-and-8b054
Medium, 10 min read.

[8] Peter J. Mohr, David B. Newell, and Barry N. Taylor. Codata recommended values of the fundamental physical constants: 2018. *Rev. Mod. Phys.* 88, 035009, 2016.

[9] John A. Rock. A lecture on integration by parts, 2016. URL https://arxiv.org/abs/1606.04141.

[10] James Binney and David Skinner. *The Physics of Quantum Mechanics.* Cappella Archive, 2008. Revised printings 2009–2011.

[11] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* National Bureau of Standards, Applied Mathematics Series 55, 1964. URL https://personal.math.ubc.ca/~cbm/aands/abramowitz_and_stegun.pdf.

[12] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory.* Dover Publications, 1996. URL https://chemistlibrary.wordpress.com/wp-content/uploads/2015/02/modern-quantum-chemistry.pdf.

[13] C. J. Pethick and H. Smith. *Bose–Einstein Condensation in Dilute Gases.* Cambridge University Press, Cambridge, UK, 2002. ISBN 0-521-66194-3.

[14] Theoretical Physics Group, ETH Zurich. Chapter 3: Quantum gases. Lecture notes, Department of Theoretical Physics, ETH Zürich, 2021. URL https://ethz.ch/content/dam/ethz/special-interest/phys/theoretical-physics/cmtm-dam/documents/qg/Chapter_03.pdf. Accessed: 10 May 2025.

[15] Tosio Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Communications on Pure and Applied Mathematics*, 10(2):151–177, 1957. doi: 10.1002/cpa.3160100201.

[16] Morten Hjorth-Jensen. Project 2: Variational monte carlo. Course project report, Computational Physics II, 2024. URL https://compphysics.github.io/ComputationalPhysics2/doc/Projects/2024/Project2/Project2VMC/pdf/Project2VMC.pdf. Accessed: 11 May 2025.

[17] Morten Hjorth-Jensen. Week 1: Introduction and numerical methods. Lecture notes, Computational Physics II, 2024. URL http://compphysics.github.io/ComputationalPhysics2/doc/pub/week1/pdf/week1.pdf. Accessed: 11 May 2025.

[18] Martina Knoop, Niels Madsen, and Richard C. Thompson. Physics with trapped charged particles. arXiv:1311.7220 [physics.atom-ph], 2013. URL https://arxiv.org/abs/1311.7220.

[19] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice.* Chapman & Hall, London, 1996.

[20] Department of Mathematics, University of Oslo. Monte carlo methods. Course notes, MAT3110, University of Oslo, 2023. URL https://www.uio.no/studier/emner/matnat/math/MAT3110/h23/slides-and-lecture-notes/monte_carlo_v2023.pdf. Accessed: 2025-05-14.

[21] James E. Gentle. *Random Number Generation and Monte Carlo Methods.* Statistics and Computing. Springer, New York, NY, 2 edition, 2003. ISBN 978-0-387-00178-4. doi: 10.1007/b97336.

[22] Federico Becca and Sandro Sorella. *Quantum Monte Carlo Approaches for Correlated Systems.* Cambridge University Press, Cambridge, UK, 2017. ISBN 978-1-107-12993-1. doi: 10.1017/9781316417041.

[23] James E. Gubernatis, Naoki Kawashima, and Peter Werner. *Quantum Monte Carlo Methods: Algorithms for Lattice Models.* Cambridge University Press, Cambridge, UK, 2016. ISBN 978-1-107-00642-3. doi: 10.1017/CBO9781107050752.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, UK, 2004. ISBN 978-0-521-83378-3.

[25] J. W. Moskowitz and M. H. Kalos. A new look at correlations in atomic and molecular systems. i. application of fermion monte carlo variational method. *International Journal of Quantum Chemistry*, 20(5):1107–1119, 1981. doi: 10.1002/qua.560200508.

[26] Morten Hjorth-Jensen. Metropolis algorithm and markov chains, importance sampling, fokker–planck and langevin equations. Technical report, Department of Physics and Center for Computing in Science Education, University of Oslo, Oslo, Norway, february 2025. URL https://compphysics.github.io/ComputationalPhysics2/doc/pub/week3/pdf/week3.pdf. Lecture notes, Week 3.

[27] W. Kent Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. doi: 10.1093/biomet/57.1.97. URL https://academic.oup.com/biomet/article-abstract/57/1/97/2721936.

[28] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook. Tech. report, Technical University of Denmark, November 2012. http://www2.imm.dtu.dk/pubdb/p.php?3274.

[29] Allan M. M. Leal. autodiff, a modern, fast and expressive C++ library for automatic differentiation. `https://autodiff.github.io`, 2018. URL https://autodiff.github.io.

[30] Gianluca Argentini. A generalization of amdahl's law and relative conditions of parallelism, 2002. URL https://arxiv.org/abs/cs/0209029.

[31] mortele. variational-monte-carlo-fys4411. https://github.com/mortele/variational-monte-carlo-fys4411, 2025. GitHub repository.

[32] Lázaro Lima de Sales, Jonatas Arizilanio Silva, Eliângela Paulino Bento de Souza, Hidalyn Theodory Clemente Mattos de Souza, Antonio Diego Silva Farias, and Otávio Paulino Lavor. Gaussian integral by taylor series and applications. *REMAT: Revista Eletrônica da Matemática*, 7(2):e3001, July 2021. ISSN 2447-2689. doi: 10.35819/remat2021v7i2id4330. URL http://dx.doi.org/10.35819/remat2021v7i2id4330.