UNIVERSITY
OF OSLO

Master's thesis

Variational Monte Carlo Simulations of Quantum Many-Particle Systems

Yevhenii Volkov

Computational Science: Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

Summer 2025

Abstract

In this work, we develop Variational Monte Carlo (VMC) technique to estimate the Ground State (GS) energy of bosonic and fermionic particles confined in a harmonic oscillator potential. Although idealized, these model systems capture a wide range of quantum phenomena. On the bosonic side, we consider neutral alkali atoms in magnetic traps cooled below the critical temperature, forming a Bose–Einstein condensate (BEC). Since its first realization in 1995, BEC has continued to shed light on many-body behavior and interaction effects in dilute bosonic gases. On the fermionic side, we study two-dimensional semiconductor quantum dots – "artificial atoms", which are extensively used in both applied and fundamental research. In particular, we focus on the first three closed electronic shells (2, 6, and 12 electrons), revealing shell-structure effects and providing tests for many-body theories.

The core contribution of this work is a flexible, object-oriented VMC framework written in C++ from scratch. The code modularly separates the Hamiltonian, trial wave function, and solver routines, allowing each component to be interchanged or extended with minimal effort. We implement both Metropolis and Metropolis–Hastings sampling algorithms, using analytic and autodifferentiated [1] expressions for local energy and gradient evaluations. To handle large-scale simulations efficiently, all algorithms are parallelized with OpenMP [2] and MPI [3].

For N non-interacting bosons in d dimensions, VMC reproduces the analytic ground-state energy $E_0 = \frac{Nd}{2}$ exactly. For non-interacting fermions in a two-dimensional harmonic trap, we recover closed-shell energies of $E_0 = 2, 10$ and 28 (in $\hbar\omega = 1$ units) for $N = 2, 6, 12$ respectively, matching the exact sums of single-particle energies. Introducing Coulomb interactions for the fermionic case, we augment the Slater Determinant with either a linear Jastrow factor $J = \exp\left\{\sum_{j>i} \beta_{ij} r_{ij}\right\}$ or a Pade–Jastrow factor $P = \exp\left\{\sum_{j>i} \frac{a r_{ij}}{1+\beta r_{ij}}\right\}$ to take correlation into account. After variational optimization of the parameters β and β_{ij}, the interacting fermion GS energies become approximately 3.000, 20.178 and 65.83 for $N = 2, 6, 12$ respectively. These values agree within statistical error with established Diffusion Monte Carlo and coupled-cluster benchmarks [4], [5]. Finally, GS energies at $\omega = 0.28$ and $\omega = 0.5$ were computed, yielding results in agreement with the benchmarks [5].

By combining rigorous validation against analytic and high-precision numerical results with a highly modular and parallelized code base, this thesis establishes a robust platform for future investigations of strongly correlated regimes, alternative wave-function ansatzes (e.g., neural-network states), or extensions to spin–orbit coupling and time-dependent dynamics.

Contents

Preface

> "If it is correct, it signifies the end of physics as a science."
>
> —Albert Einstein, on quantum physics

Before entering Taras Shevchenko National University of Ukraine's physics faculty, I would never imagine myself have any business related to quantum physics. Everything seems so absurd. How can an electron be a particle and a wave at the same time? What do you mean by "if we observe experiment its outcome will change"?

Looking back on my journey, I want to thank all the people who influenced my life and made me a physicist. The changes began in 2015 when I met Alexander Fedko. He inspired my passion for mathematics and physics, and thanks to his guidance, I realized I wanted to learn more. I am deeply grateful to my high school physics teacher, Andriy Berezovsky. I also thank my Ukrainian university instructors – especially Andrey Semenov, Stanislav Vilchinsky, and Natalia Mayko.

A special thanks to my parents, who always believed in me and supported me. They never doubted that I would succeed, and that motivated me to live up to their expectations. To my grandfather Pavlo, thanks to whom I chose the University of Oslo, and who helped me improve both my bachelor's thesis and this one. And to my Irina and Yevheniia for their love and support.

Thank you to all my friends who have been with me throughout this entire time, even from afar — Kostya, Ilya, Nazarii, Danya, Senya and especially Maria, for motivating me to work on my thesis earlier than the last two weeks.

Thanks also to my friends and colleagues from the CCSE and my colleagues from Portal, in particular Yavar and Vytenis.

Feeling super lucky to be at the University of Oslo. The study environment and all university facilities here are truly top-notch, the best I have ever experienced. Before moving to Norway, I never knew a place could feel this cozy, with such awesome people, and the language is just så gøy!

I am extremely grateful to my supervisor, Morten Hjorth-Jensen. Every course I took with him as a teacher was incredibly interesting and challenging – especially FYS4411, which formed the basis of this thesis. Whenever I was worried that my program was not working or I did not understand something, Morten always knew how to find the right words to cheer me up and motivate me.

I feel that because of all these people, I am who I am today, and I write this thesis with a big smile on my face.

Chapter 1

Introduction

In this work, we tackle two goals at once: studying bosonic and fermionic systems and building the Variational Monte Carlo (VMC) tools to study these systems. In this thesis, we specialize on systems of particles confined to move in harmonic oscillator-like traps.

From a theoretical point of view, a system of quantum particles confined in harmonic potential is described by a Hamiltonian $\hat{H} = \hat{T}+\hat{H}_{HO}+\hat{V}$, where $\hat{T}$ represents the kinetic energies of all particles, $\hat{H}_{HO}$ represents the external Harmonic Oscillator (HO) potential and $\hat{V}$ represents the interaction between particles. The Hamiltonian itself represents the energy of the quantum system and it is the same for fermionic and bosonic particles.

Bosons have integer spin and their wave functions are symmetric under particle exchange, while fermions have half-integer spin and require antisymmetric wave functions. In practice, studying the same trapping potential for both types of particles simply means choosing the appropriate symmetry for the trial wave function.

Although a HO potential is an idealization, it accurately captures essential features of many real systems. In particular, HO confinement provides a good approximation for the behavior of dilute Bose–Einstein condensates in magnetic or optical traps and for electrons in semiconductor quantum dots, making it a valuable model for both bosonic and fermionic studies.

The Hamiltonian of the form $\hat{H} = \hat{T} + \hat{H}_{HO}$ describes a system of non interacting bosons. In this research we consider neutral alkali atoms confined in magnetic traps and cooled to a critical temperature [6]. The HO potential approximates the behavior of particles in such traps perfectly. This phenomenon is called Bose-Einstein condensation. It was first discovered in 1995, and it is studied up to date. Studying this system can help to understand better corelation between bosons, GS profile of such systems, thermodynamical properties, etc [7]. Unlike fermions, bosons can occupy the same quantum state, and in a Bose-Einstein condensate state all particles occupy a state with the lowest energy, which is the ground state. The wave function of such a system is just a product of exponential functions.

The Hamiltonian of the form $\hat{H} = \hat{T} + \hat{H}_{HO}$ describes a system of non-interacting fermions, and also interacting fermion if the interaction potential $\hat{V}$ is added. One of the real physical systems, that is described by this Hamiltonian is a two-dimensional quantum dot. This system is also called "artificial atom", because it can be created artificially, and the system has a shell structure that reminds of atomic shells. Quantum dots are intensively used in quantum technologies: they can be used to construct a qubit, nanoscale logic gates and switching devices. Since 2000, quantum dots were actively studied theoretically and experimentally, because they have a great potential in quantum technologies [8]. Fermionic particles can not occupy the same quantum state,

and therefore many-body fermionic systems have a shell structure. The wave function of such systems is given by a Slater Determinant (SD), which is composed of single-particle wave functions.

When including interactions among fermions, adding the Coulomb potential to the Hamiltonian produces electron–electron correlations. To address this, the trial wave function is multiplied by a Jastrow factor, which enforces the proper cusp behavior when particles are close and includes correlation effects, [9]. In this research, we use two forms of correlation factors:

- Jastrow factor: $J = \exp\Big\{\sum_{j>i} \beta_{ij} r_{ij}\Big\}$, which introduces a linear dependence on the interparticle separation r_{ij};
- Pade-Jastrow factor: $P = \exp\Big\{\sum_{j>i} \frac{a r_{ij}}{1+\beta r_{ij}}\Big\}$, which smoothly interpolates between the short-range cusp behavior and a decaying tail.

In this research, one of our main goals is to estimate the ground-state (GS) energy of these quantum systems. Experimentally, reaching the ground state simply requires cooling the system below its characteristic temperature (e.g., ultracold atomic gases are cooled via evaporative and laser cooling to achieve Bose–Einstein condensation [10]). Theoretically, the variational principle provides a straightforward way to compute GS observables: any trial wavefunction yields an energy that is an upper bound to the true ground-state energy. As a result, most quantum many-body methods—from Hartree–Fock to coupled-cluster theory—are built on variational foundations [11], §1.

On practical grounds, the GS energy is obtained by solving the time-independent Schrödinger equation, which leads to a high-dimensional integral over configuration space. For systems with many particles, direct numerical integration methods become infeasible because the number of grid points grows exponentially with dimension—a phenomenon known as the "curse of dimensionality" [12]. Monte Carlo techniques, by contrast, use random sampling to estimate such integrals with a computational cost that scales more gently with dimension. In the VMC approach, one applies the variational principle to construct a trial wave function whose parameters are optimized to minimize the expectation value of the Hamiltonian. Monte Carlo sampling is then used to evaluate that expectation value. Although VMC yields a reliable estimate of the ground-state energy, still higher accuracy can be achieved by feeding the optimized trial wave function into a Diffusion Monte Carlo calculation [9].

Monte Carlo methods require a sampling rule to move the system through configuration space. In this work, we use two main algorithms: Metropolis and Metropolis–Hastings. The Metropolis algorithm draws trial moves from a uniform distribution and accepts or rejects them based on the ratio of probability densities. Metropolis–Hastings generalizes this by allowing an arbitrary proposal distribution. It includes "drift" terms derived from the Fokker–Planck and Langevin equations, which guide particles toward regions of higher probability [13]. This importance sampling improves efficiency by concentrating sampling where probability density is large.

The systems we study have been analyzed previously with VMC, but our contribution is a flexible, object-oriented VMC framework that can handle both bosonic and fermionic cases together. Its modular design lets us swap Hamiltonians, trial wave functions, and sampling algorithms without rewriting core routines. This extensibility means new Hamiltonians or ansatzes can be added easily to study different systems. To evaluate derivatives, we derive analytic expressions and integrate the C++ Autodiff [1] library for automatic differentiation. Finally, we use OpenMP [2] and MPI [3] parallelization

to accelerate sampling and local-energy evaluations, ensuring efficient performance on modern distributed systems, as well as ordinary computers.

Chapter 2

Theory

This chapter covers all theory needed for our study of many-body systems in a harmonic-oscillator (HO) potential:

- First, we review the classical HO to build intuition for the quantum case;
- Then we discuss single-particle quantum basics—wave functions, the Schrödinger equation, operators, and how they apply to the HO;
- Next, we dive into many-body concepts: the Pauli principle, symmetric versus antisymmetric wave functions, and how correlation factors enter the picture;
- After that, we introduce the variational principle, derive the local energy, and define the specific systems we study;
- Finally, we explain key computational methods: Monte Carlo sampling, Metropolis and Metropolis–Hastings algorithms, and gradient-based optimization.

2.1 Classical Harmonic Oscillator

This section is based on Philip L. Bowers's book [14], Chapter 1.

Consider a one-dimensional particle moving in a harmonic potential:

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

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

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

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

where m represents the mass of the particle, t represents time, $\mathbf{r}$ represents the radius vector[1], which describes the 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 particle moves in the potential, the force acting on it can be found by using

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

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

Here, $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ is the gradient operator. 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 attempts to return the particle to the equilibrium state, $x = 0$. If the particle is not in the equilibrium state, the force acts on it and diminishes as the particle comes closer to $x = 0$. But the particle overshoots the equilibrium position. When the particle reaches equilibrium, it moves with a certain speed. At that point, no force is acting on the particle, so it keeps moving and starts feeling the restorative force again. It reaches the position of maximal displacement and then moves back toward equilibrium. This back-and-forth motion does not stop. Such movement is known as an *oscillation*, and the system is known as a *harmonic oscillator*.

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

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

The 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, and b and c are constants determined by the initial conditions $x(0) = b$ and $\dot{x}(0) = c\,\omega$.

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

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

Note that X is the magnitude of the maximal displacement. The 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 the potential energy is given by

$$V = \frac{k\,x^2(t)}{2} = \frac{k\,X^2}{2}\cos^2(\omega t).$$

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

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

Because $V(x)$ does not depend on time, the total energy remains constant. In other words, energy is conserved—an important property of the harmonic oscillator.

Before discussing the quantum harmonic oscillator, we first review fundamental concepts of quantum mechanics in the following sections. However, to set the stage, we begin with a brief historical overview of how quantum physics originated.

2.2 Quantum-Mechanical Foundation

2.2.1 The Dawn of Quantum Physics

Quantum physics as a field originated in the early twentieth century when Max Planck solved the ultraviolet catastrophe—the problem of black-body radiation [15], §1[2].

[2]From this point on, the symbol "§" will be used to indicate chapters or sections.

Classical physics seemed able to describe every phenomenon. However, there were a few problems that classical physics could not solve. These were three critical failures: the problem of black-body radiation, the photoelectric effect, and the lifetime of the Bohr atom. The failure of black-body radiation led to the quantum revolution. That classical physics could not solve a problem means that experimental results misaligned with classical predictions. If the measurement is done correctly, then obviously the problem is in the theory, which cannot explain such a phenomenon. But classical physics was correct in almost every other case, so how could it be wrong? The answer is 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 the ultraviolet catastrophe by assuming that light is radiated discretely, in packets (quanta) of light. A quantum of light propagates at the speed of light with energy equal to $\epsilon = h\nu$, where ν is the frequency and h is Planck's constant. By applying this idea, Planck's theory explained experimental data perfectly [16], §1. But the assumption that light was emitted discretely was unheard of from the classical point of view, where everything was continuous. Physicists of that time divided into two groups: those who supported quantum theory and those who did not. Most of the debates happened because quantum theory was probabilistic, as if nature is fundamentally probabilistic and random rather than deterministic. Even great physicists such as Albert Einstein were against quantum theory, especially its probabilistic interpretation. He even coined the famous phrase, "God does not play dice." Jim Baggott has a great article discussing that phrase and a possible explanation of what Einstein meant by saying it [17]. Although Einstein disagreed with some quantum postulates, he advanced the field greatly by explaining the photoelectric effect.

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

2.2.2 Schrödinger Equation and Planck's Constant

In what follows, the presentation closely follows David J. Griffiths' classic text *Introduction to Quantum Mechanics* [18], 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 nonrelativistic 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, $\Psi(x,t)$, obtained by solving one of the main equations of quantum physics—Schrödinger's equation (SE):

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

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

Planck's constant is one of the universal constants, like the gravitational constant G and the speed of light in vacuum c, as discussed in §1 of D. Chang's article [16].

He states that "Planck's constant was found to play a major role in many aspects of quantum physics." First of all, it is part of SE, and it appears in many other equations in quantum physics. For example, it is part of the de Broglie momentum relation, Dirac's fundamental quantum condition, and Heisenberg's uncertainty principle. The 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

The de Broglie wavelength is defined by $\lambda_{\mathrm{dB}} = h/p$, where p is the momentum of the particle. "Quantum mechanics becomes necessary when a particle's de Broglie wavelength $\lambda = \frac{h}{p}$ is comparable to the characteristic size d of the system" (Griffiths [18], §1.3).

According to CODATA's archive of physical constants by the Physical Measurement Laboratory of NIST [19],

Planck's Constant and Reduced Planck's Constant

Planck's constant $h \approx 6.626070 \times 10^{-34}\,\mathrm{J \cdot s}$, and
the reduced Planck's constant $\hbar = \frac{h}{2\pi} \approx 1.054573 \times 10^{-34}\,\mathrm{J \cdot s}$.

Equation (2.2) is a second-order partial differential equation. The solution to this equation is the wave function, which we will discuss in Section 2.2.3. Just by looking at SE (2.2), we can see that it is different from its classical analogue (2.1). First, this equation includes the imaginary unit i, which almost never appears in classical mechanics. Second, the unknown wave function is a function of both x and t. Finally, once SE (2.2) is solved, it yields a wave function representing probability amplitudes rather than a definite particle position, as in classical physics.

With SE (2.2) established, we now turn to the interpretation and properties of the wave function.

2.2.3 Wave Function and 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 [18] (§1.2–1.3 and 1.6) to develop its statistical interpretation.

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

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

Heisenberg's Uncertainty Principle

A spread in momentum corresponds to a spread in position, and in general, 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.

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

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

Born's Statistical Interpretation of the Wave Function

$|\Psi(x,t)|^2 = \Psi(x,t)\,\Psi^*(x,t)$ gives the probability of finding the particle at point x at time t.
More precisely:

$$P_{ab} \;=\; \int_a^b \left|\Psi(x,t)\right|^2 dx$$

is the probability of finding the particle between a and b at time t.

Statistical physics uses a specific name for $|\Psi(x,t)|^2$: the probability density, ρ, defined as

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

Thus, the wave function itself does not have a direct physical meaning; instead, the square of its norm, $|\Psi(x,t)|^2$, represents the probability density ρ. This introduces indeterminism into quantum physics: even if we know everything about the system, we cannot predict the outcome precisely. We can only state the probability with which an outcome is likely to occur. Quantum physics is fundamentally probabilistic. The outcome of an experiment, in general, cannot be predicted with 100% accuracy.

Three key statistical relations in quantum mechanics are:

1. The probability of finding the particle in all space is equal to 1 (another form of Born's rule):

$$\int_{-\infty}^{\infty} \left|\Psi(x,t)\right|^2 dx \;=\; 1. \tag{2.3}$$

2. The expectation value of any observable $f(x)$ is

$$\langle f(x)\rangle \;=\; \int_{-\infty}^{\infty} f(x)\left|\Psi(x,t)\right|^2 dx. \tag{2.4}$$

3. The definition of the standard deviation, which we met earlier while discussing the uncertainty principle, is

$$\sigma^2 \;=\; \left\langle (\Delta x)^2 \right\rangle \;=\; \langle x^2\rangle \;-\; \langle x\rangle^2. \tag{2.5}$$

Now that we know the physical meaning of the wave function, we are ready to discuss its important feature—normalization.

2.2.4 Wave Function Normalization

Following Griffiths [18] (§1.4), we require that the wave function satisfy Born's rule:

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

Without 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, a solution of the SE does not satisfy Born's rule—the wave function is not **normalized**. The general solution is determined only up to a multiplicative constant $A \in \mathbb{C}$, namely $\Phi(x,t) = A\,\Psi(x,t)$. Therefore, the wave function $\Phi(x,t)$ can be normalized so that it satisfies equation (2.3):

Normalization of the Wave Function

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

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

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

Once the wave function is normalized, it remains normalized for all time. Let us show that this statement is true by performing a short derivation, which we will need in the next section. Starting with

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

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

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

The term $\frac{\partial \Psi}{\partial t}$ can be obtained from 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 the term $\frac{\partial \Psi^*}{\partial t}$ can be obtained by taking the 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,

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

Substituting into the integral,

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

Since Ψ and Ψ^* are continuous functions, they both approach zero as $x \to \pm\infty$, which means that

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

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.

2.2.5 Position, Momentum, and Kinetic Energy in Quantum Mechanics

Following Griffiths [18] (§1.5), recall that Heisenberg's uncertainty principle states that position and momentum cannot be measured precisely at the same time. This means that we can no longer use an ordinary position x, but rather deal with its expectation value given by (2.4):

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

But what exactly does "expectation value of x" mean? If we perform an experiment on a system in the state Ψ to measure the position of the particle x, the system will no longer be in the state Ψ. It is said that measurement collapses the state. Thus, if we perform a series of consecutive measurements, we will not obtain the expectation value of x. To obtain the expectation value of x, we need a way to return the system to the state Ψ and then perform a measurement again. Alternatively, we can prepare many identical systems in the state Ψ and make one measurement on each of them. $\langle x \rangle$ is the average measurement for an ensemble of systems 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 understand what $\langle x \rangle$ means, it is worth studying $\frac{d\langle x \rangle}{dt}$, which resembles the classical velocity. By definition, it is given by

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

We can use the expression (2.6) for $\frac{\partial}{\partial t}\left|\Psi(x,t)\right|^2$ 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 (see the lecture [20] for a review of integration by parts):

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

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

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

since Ψ vanishes as $|x| \to \infty$, so the first term drops out. We can simplify the remaining integral by performing integration by parts once more on the second term with $u = \Psi$ and $dv = \frac{\partial \Psi^*}{\partial x}\, dx$:

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

Therefore,

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

This expression represents the velocity of the expectation value of x. It is not the same as the velocity of a single particle. Thus, this quantity is the expectation value of the particle's velocity:

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

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,

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

The expectation value of position can be written in a similar form:

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

The quantities between Ψ^* and Ψ in equations (2.7) and (2.8) represent operators:

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

Here, the hats over x and p signify that these are operators, not ordinary position or momentum. Operators are a very important concept in quantum mechanics, and we will discuss them in detail in Section 2.2.9. For now, it is sufficient to know that operators always act on the function immediately to their right, and you cannot swap their order. For example,

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

This is because $\hat{p}$ contains a derivative; applying it to ψ_2 gives a different result than applying it to ψ_1.

Now that we have the coordinate and momentum operators, we can construct the kinetic energy operator. In quantum mechanics, to obtain a quantum analogue of a classical expression, one replaces each classical quantity with its corresponding operator: observables are represented by Hermitian operators. Position, momentum, and kinetic

energy are observables; they can be measured in an experiment. Therefore, it is straightforward to obtain the kinetic energy operator:

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

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

2.2.6 Time-Independent Schrödinger Equation & Stationary States

The next step is to study the SE and its general solutions following Griffiths [18], §2.1. To illustrate how the time-independent SE follows from the general form (2.2), we provide a straightforward, pedagogical derivation.

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

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

Inserting this into SE (2.2) leads to

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

The left-hand side is a function of t, while the right-hand side is a function of x. They can be equal only if both sides equal the same constant. Let us denote this constant by E:

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

First, consider the left equation:

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

Multiplying by $\phi(t)$ on both sides yields

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

The right-hand side of the SE with separated variables now reads

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

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 the 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 and actually solve the equation, we need to know the potential $V(x)$. The main focus of this research is the HO potential.

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

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

Note that there is no integration constant. In general, there would be an integration constant C after integration, but ψ can be redefined as $\psi' = C\,\psi$ and the prime notation dropped. In general, any constant from ϕ can be absorbed into ψ.

Three important features of the time-independent SE:

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

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

is time-invariant. Moreover, every expectation value of an observable is

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

which is time-independent.

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

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

Its quantum analogue is obtained by replacing observables with their respective operators[3]. By doing that, we obtain the Hamiltonian operator, which describes the 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, the time-independent SE can be rewritten as

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

In this form, the SE shows that $\hat{H}$ acting on ψ yields the same function multiplied by E, i.e., it is an eigenvalue problem. In this research, however, our main focus is on using the variational principle to obtain the GS energy rather than solving the eigenvalue equation directly.

An important point is that the expectation value of $\hat{H}$ is E:

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

Moreover, the expectation value of $\hat{H}^2$ is E^2:

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

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

This implies that the standard deviation of energy is zero:

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

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

3. Normally, SE (2.14) has more than one solution, and in some cases even an infinite number of solutions. A solution to this equation is an eigenvalue E and eigenfunction ψ. They come as pairs. So, if the equation has infinitely many solutions, it has infinitely many eigenvalues $\{E_1, E_2, \dots\}$ and eigenfunctions $\{\psi_1, \psi_2, \dots\}$. The solutions to the time-dependent SE are

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

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

The SE is linear: if Ψ_1 and Ψ_2 are two solutions, then any linear combination $c_1\,\Psi_1 + c_2\,\Psi_2$ is also a solution. Consequently, once all independent solutions are known, the most general solution can be written as a linear combination of those basis functions:

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

Expansion coefficients have a physical meaning:

Meaning of Expansion Coefficients

$|c_n|^2$ is the probability that a measurement of the energy will yield 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 point to note:

Energy Conservation in Stationary States

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

Nonrelativistic quantum mechanics is built on the language of linear algebra. In the next section, we introduce the basic definitions and notation—vectors, operators, and inner products—that serve as the mathematical framework for our quantum-mechanical treatment.

2.2.7 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 [18], §3.1.

However, it is not the same linear algebra familiar from high school or the first years of university. In general, linear algebra studies vectors in vector spaces and how they transform. The difference between ordinary linear algebra and linear algebra in

quantum mechanics lies in the definition of vectors. To understand linear algebra in quantum mechanics, we introduce its simplest concepts and then translate them into quantum language.

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

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

Here, $|\alpha\rangle$ represents a vector α. The notation $|\cdot\rangle$ is referred to as a "ket vector"; it represents a column vector. A bra vector is obtained via Hermitian conjugation (complex conjugation and transposition) of the ket vector:

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

The "bra" and "ket" notation is known as the Dirac formalism; it is simply a way to denote vectors and is used extensively in quantum mechanics. We use J. Binney's explanation of the Dirac formalism from his book [21], §1.3.2. Basic linear algebra is introduced in this formalism so that it is easier to compare it to its quantum analogue.

From a mathematical point of view, the entries of bra and ket vectors are complex numbers, $\{a_n\} \in \mathbb{C}$, and the whole vector exists in the 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:

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

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

The inner product satisfies linearity, positivity, and conjugate symmetry (Binney [21], §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:
$$\langle\psi|\phi\rangle = \big(\langle\phi|\psi\rangle\big)^*.$$

In linear algebra, matrices represent linear transformations of vectors:

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

A matrix acts on a vector and transforms it into a different vector, in general. In quantum mechanics, those matrices correspond to operators acting on state vectors.

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

2.2.8 Hilbert Space

The wave function in quantum mechanics can be viewed as a vector. The reason the same approach as in the previous section cannot be used in quantum mechanics is that wave functions, in general, exist in infinite-dimensional spaces. This means that, for example, the inner product would be defined as an infinite sum, and matrices that act on the vector would be infinite-dimensional. Another approach is needed in quantum physics [18] §3.1.

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

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

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 a **Hilbert space**.

From a glance at equation (2.3) and the definition of Hilbert space, it is clear that the **wave function exists in a Hilbert space**.

The inner product in a Hilbert space can be defined as

$$\langle f(x)|g(x)\rangle = \int_a^b f^*(x)\, g(x)\, dx.$$

If both $f(x)$ and $g(x)$ are members of the Hilbert space (i.e., they are square-integrable), then the inner product is guaranteed to exist. This definition of the inner product is guaranteed to satisfy the three properties stated in the previous section[4]:

Normalization, Orthogonality & Orthonormality

If the inner product of a function with itself equals one,

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

the function is called **normalized**;
If the inner product of two functions equals zero,

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

the functions are called **orthogonal**;
If a set of functions $\{f_n\}$ is normalized and they are mutually orthogonal,

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

the set is said to be **orthonormal**. Here, $\delta_{nm} = 1$ if $n = m$ and $\delta_{nm} = 0$ if $n \neq m$ is the Kronecker delta function.

[4] For an operation to be called an "inner product" in mathematics, it must meet these three conditions.

It is very common for the set of wave functions to be orthonormal, as it greatly simplifies calculations. Instead of evaluating a complicated integral, one can immediately see what the integral equals.

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

2.2.9 Observables and Operators

We continue to follow Griffiths [18], §3.2. Recall that the expression for computing the expectation value of an observable is given by equation (2.12):

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

The observable Q is "sandwiched" between the wave functions in the integral. This expression can be rewritten as a scalar product in Hilbert space:

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

Consider the complex conjugate of the expression above:

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

The outcome of the measurement must be real, which implies

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

This equality holds when $Q(x,p)$ is an operator, not just an ordinary function—that is, $Q(x,p) \to \hat{Q}(x,p)$. In other words, observables in quantum mechanics are represented by operators. Moreover, operators that represent observables must satisfy

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

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

A **Hermitian 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 satisfies $\hat{Q} = \hat{Q}^\dagger$.

When the concept of operators is introduced, it is customary to write the expectation value as

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

In this notation, the operator acts on the ket to its right. This definition is equivalent to the previous one, but it emphasizes that the operator $\hat{Q}$ acts on the state $|\psi\rangle$. Taking the adjoint of the action of $\hat{Q}$ on a ket gives

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

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, it may be viewed as acting on either the ket or the bra, but in any single expectation value it appears in only one position.

Operators are the quantum analogs of matrices in linear algebra: they act on the quantum vector (the 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.

2.2.10 Quantum Harmonic Oscillator

This section draws on Philip L. Bowers's lecture §2 [14], Griffiths' treatment in §2.3 [18], and Binney & Skinner's §3.1 [21].

Recall that in Section 2.1 we discussed a classical 1D particle moving in an HO potential,

$$V(x) = \frac{k\,x^2}{2} = \frac{m\,\omega^2}{2}\,x^2. \tag{2.15}$$

In this section, we consider a quantum particle moving in an HO potential and generalize this concept to 2D and 3D.

For a quantum particle in an HO potential, we must solve the SE. Because the potential in (2.15) does not depend on time, we use the time-independent form (2.11) with that potential:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \frac{m\,\omega^2}{2}\,x^2\,\psi(x) = E\,\psi(x). \tag{2.16}$$

To see how this follows from the general Schrödinger eigenvalue equation (2.14),

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

we first write down the Hamiltonian operator. In classical physics, the Hamiltonian represents the sum of kinetic and potential energies:

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

Substituting $x \to \hat{x} = x$ and $p \to \hat{p} = -i\hbar\,\frac{d}{dx}$ 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 to equation (2.16):

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

This equation may seem straightforward, yet finding its solutions proves to be surprisingly involved. However, there is an analytical solution, which can be obtained by using second quantization or by making an appropriate substitution. Both methods

are described in detail in Griffiths §2.3. We will not derive the solution here but rather use the results from Griffiths:

$$\psi_n(x) = \left(\frac{m\,\omega}{\pi\,\hbar}\right)^{\frac{1}{4}} \frac{1}{\sqrt{2^n\,n!}}\, He_n(\xi)\, e^{-\xi^2/2}, \quad E_n = \left(n + \frac{1}{2}\right)\hbar\,\omega. \tag{2.17}$$

As discussed earlier, a solution to the SE is a pair of eigenvalue E_n and eigenfunction $\psi_n(x)$. Here, $\xi = \sqrt{\frac{m\,\omega}{\hbar}}\,x$ is a dimensionless coordinate, and

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

is the n-th probabilist's Hermite polynomial (Abramowitz & Stegun [22], §22.3).

In this work, we need Hermite polynomials up to second order. There are two common conventions—probabilist's and physicist's Hermite polynomials—and although the distinction can be subtle, we use the probabilist's version. The first three probabilist's Hermite polynomials are listed in Table 2.1 below.

$He_0(x)$	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, one can perform various analyses, and there are extensive texts devoted to the HO and its properties. The harmonic oscillator has infinitely many eigenstates. We focus on the ground state by virtue of the variational principle. Note that even the ground state has

$$E_0 = \frac{\hbar\,\omega}{2} \neq 0.$$

This is a feature of quantum physics: even in the lowest possible state, the energy is not zero.

It is straightforward to generalize the HO problem to 2D and 3D. First, we 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

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

In 3D, the momentum operator becomes

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

where $\boldsymbol{\nabla}$ is the gradient operator we met before. The operator $\boldsymbol{\nabla}^2$ is the Laplacian Δ, equal to

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

Similarly, the position operator becomes

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

Thus, the Hamiltonian operator reads

$$\hat{H} = -\frac{\hbar^2}{2m}\,\Delta + \frac{m\,\omega^2}{2}\,\mathbf{r}^2. \tag{2.18}$$

Now the wave function depends on three coordinates, $\psi = \psi(x, y, z)$, and the SE becomes

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

which expands to

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

This equation can be solved by changing to spherical coordinates or by separation of variables. A clear solution is presented in Binney & Skinner, Chapter 7.3, [21]. The solution of the 3D equation is

$$\psi_{n_x,n_y,n_z}(x, y, z) = \psi_{n_x}(x)\,\psi_{n_y}(y)\,\psi_{n_z}(z), \quad E_{n_x,n_y,n_z} = \left(n_x + n_y + n_z + \tfrac{3}{2}\right)\hbar\,\omega, \tag{2.19}$$

where $\psi_{n_x}(x)$, $\psi_{n_y}(y)$, and $\psi_{n_z}(z)$ are the 1D HO wave functions from (2.17).

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

$$\psi_{n_x,n_y}(x, y) = \psi_{n_x}(x)\,\psi_{n_y}(y), \quad E_{n_x,n_y} = (n_x + n_y + 1)\,\hbar\,\omega. \tag{2.20}$$

The subscripts n, n_x, n_y, and n_z are called principal quantum numbers; they denote the state in each dimension. The quantum state of the system is determined by these quantum numbers. By specifying n_x, n_y, and n_z, one immediately obtains the energy and wave function of the system.

Note that in 2D and 3D, energy levels become degenerate: different combinations of n_x, n_y, and n_z can yield the same energy. For example, in 2D,

$$E_{n_x=1,n_y=0} = 2\,\hbar\,\omega \quad \text{and} \quad E_{n_x=0,n_y=1} = 2\,\hbar\,\omega.$$

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

2.2.11 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 [18], §4.2)

$$V(r) = \frac{e^2}{4\pi\varepsilon_0\,r}. \tag{2.21}$$

Here $e \approx 1.602 \times 10^{-19}\,\mathrm{C}$ is the elementary charge, and $\varepsilon_0 \approx 8.854 \times 10^{-12}\,\mathrm{F \cdot m^{-1}}$ is the vacuum permittivity (CODATA [19]).

As we discussed earlier, to obtain the quantum expression we change each variable to its respective operator. In this case:

$$r \;\longrightarrow\; \hat{r} = r.$$

Thus, the potential remains unchanged.

For an N-electron system, each electron experiences the sum of Coulomb potentials from all the others. We return to this many-body interaction in Section 2.2.20.

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

2.2.12 Spin

Spin is not purely a 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 the motion of its center of mass, and **spin** ($\mathbf{S} = I\,\boldsymbol{\omega}$), associated with rotation about its own axis. Simply put, if an object rotates, it has spin. Earth, for example, has orbital angular momentum due to its rotation around the Sun and spin angular momentum due to its rotation about its north–south axis. In classical physics, spin is just the sum of all orbital angular momenta of the Earth's constituents (Griffiths [18], §4.4).

The concept of spin in quantum physics is similar in spirit but fundamentally different in nature. In the hydrogen atom, an electron carries orbital angular momentum due to its motion around the nucleus and has an intrinsic spin, which has nothing to do with motion in space but is analogous to classical spin. Quantum spin contributes to the 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.

For a quantum particle in an HO potential, the state of the system can be described by the principal quantum number n. Spin introduces two new quantum numbers: s, the spin value of the particle, and s_z, the projection of spin onto the z-axis. The projection s_z takes $2s+1$ possible values, i.e. $s_z \in \{-s, -s+1, \ldots, 0, \ldots, s-1, s\}$. All quantum particles have a fixed spin s, while the spin projection can take any value in $\{-s, -s+1, \ldots, 0, \ldots, s-1, s\}$.

An electron has spin $s = \frac{1}{2}\hbar$ and $s_z = \pm\frac{1}{2}\hbar$. A photon, on the other hand, has spin $s = \hbar$ and $s_z \in \{\hbar, 0, -\hbar\}$. Electrons and photons are fundamentally different particles because of their spin values.

Fermions & Bosons

A particle with half-integer spin is called a **fermion**;
a particle with integer spin is called a **boson**. (Griffiths [18], §5.1.1)

Following Szabo and Ostlund's introduction to many-body spin [11] (§2.2.1), to describe a quantum system completely we need to include spin. To keep it simple, consider an electron. An electron can be in the "spin-up" state, when its spin is parallel to the z-axis, or in the "spin-down" state, when its spin is anti-parallel to the z-axis. Define two orthogonal functions representing spin-up $\alpha(\xi)$ and spin-down $\beta(\xi)$ for an unspecified spin coordinate ξ, such that

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

and

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

In this formalism, an 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 must be specified. An electron can be in either the spin-up or spin-down state; therefore, a wave function ϕ that describes both its spatial and spin distributions takes the form

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

If a set of wave functions is orthonormal, then their spin components are also orthonormal:

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

Most quantum problems do not include spin dependence in the Hamiltonian, and the wave function can be written as a product of spatial and spin wave functions. Therefore, spin degrees of freedom are often omitted, with the understanding that they can be reintroduced later.

Since we study Bose–Einstein condensates (bosons) and quantum dots (fermions), understanding the distinction between bosons and fermions is crucial. Spin will also play a key role when defining wave functions for fermionic systems.

Having covered spin and one-body quantum foundations, we now move into many-body physics, beginning with the Pauli exclusion principle.

2.2.13 Many-Body Hamiltonian and Pauli Principle

We begin our discussion of many-body physics by writing down the Hamiltonian for N electrons interacting with M fixed nuclei. Examining this operator makes clear why the wave function must be antisymmetric under particle exchange. According to Szabo and Ostlund [11] (§2.1.1), the nonrelativistic many-electron Hamiltonian in atomic units is

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N} \Delta_i + \sum_{j=1}^{M}\sum_{i=1}^{N} \frac{Z_j}{r_{ij}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}}. \tag{2.22}$$

Here:

- Δ_i acts on the spatial coordinates $\mathbf{r}_i = (x_i, y_i, z_i)$ of electron i, giving its kinetic energy.
- Z_j is the charge of nucleus j, and r_{ij} is the relative distance between nucleus j and electron i.
- In atomic units, $\hbar = e = m_e = 4\pi\varepsilon_0 = 1$, so all prefactors become unity ([11] §2.1).

The Hamiltonian (2.22) depends only on the spatial coordinates of the electrons. The total wave function depends on N coordinates $\mathbf{x}_i$ ([11] §2.1.2):

$$\Psi = \Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N).$$

To take electron spin into account, we demand that the total wave function change sign whenever two electrons are exchanged. This requirement is the Pauli principle:

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 [11], §2.1.3):

$$\Psi(\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_N) = -\,\Psi(\mathbf{x}_1, \ldots, \mathbf{x}_j, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_N).$$

Enforcing this antisymmetry is essential for fermions. In practice, one constructs an antisymmetric spatial–spin wave function via a Slater determinant, which we introduce in the next section.

2.2.14 Fermionic Many-Body Wave Function

Following Griffiths [18] (§5.1.1), to understand how to construct a many-body wave function, consider a system of two fermions and then generalize to N fermions. Suppose we have two noninteracting 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 the 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 the particles are distinguishable. If there is no way to distinguish them, the expression above is incorrect, because one cannot know which particle occupies which state. In that case, all one can say is that one particle is in ϕ_a and the other is in ϕ_b. This is exactly the situation in the quantum world: all electrons are identical. We cannot label them "1" and "2" to tell them apart. The only way to distinguish two electrons is to perform a measurement, but that measurement collapses their state. Therefore, when constructing a wave function for two electrons, we must allow for both electrons to be in ϕ_a as well as in ϕ_b. There are two ways to construct a wave function that accounts for indistinguishability:

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

Here Ψ_+ is a bosonic wave function, while Ψ_- is fermionic. The factor $1/\sqrt{2} = 1/\sqrt{2!}$ comes from normalization.

Bosons are symmetric under exchange:

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

while fermions are antisymmetric:

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

This implies an important fact: the wave function of two fermions in the same state vanishes:

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

If the wave function is identically zero, the probability of finding the electrons anywhere is zero. But there are two electrons, so they must be somewhere. Therefore, two fermions cannot occupy the same state. This is the **Pauli exclusion principle**: two fermions cannot share the same spin–orbital. Recall that a quantum state of a particle is specified by its quantum numbers; thus, no two fermions in the system can have identical quantum numbers—at least one must differ.

To generalize to N particles, the total wave function must satisfy the antisymmetry requirement of the Pauli principle and be constructed from single-particle wave functions. A determinant naturally satisfies these conditions. The fermionic wave function is given by

$$\Psi_- = \frac{1}{\sqrt{N!}}\begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{vmatrix}. \tag{2.23}$$

Here, $\phi_i(\mathbf{x}_j)$ is a spin–orbital, a function of the spatial and spin coordinates of the j-th electron. The factor $1/\sqrt{N!}$ ensures proper normalization, $\langle\Psi_-|\Psi_-\rangle = 1$. This determinant is called a **Slater determinant** (Shavitt & Bartlett [23], §1.3).

Even for $N = 3$, working with a Slater determinant is challenging. Since this wave function must be used for any calculation, analytical solutions are rare in many-body physics; numerical and approximate methods replace them.

From a computational standpoint, evaluating the determinant of an $N \times N$ matrix via LU factorization requires $\mathcal{O}(N^3)$ operations, which becomes expensive for large N (Süli & Mayers [24], §2.6). Therefore, we either keep N small or employ methods to approximate the determinant. For our system of interest, a Slater determinant of size N can be approximated by the product of two determinants of size $N/2$. We discuss this in detail in Section 3.3.

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

2.2.15 Bosonic Many-Body Wave Function

Like fermions, indistinguishable bosons require symmetrized many-body states. Thus, one must construct a wave function analogous to that for fermions. In the previous section, it was shown that for two bosons the wave function is

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

and it is symmetric under exchange of the two particles:

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

Let us extend this concept to three bosons. There are $6 = 3!$ permutations of three bosons:

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

where the sum runs over all permutations.

It is straightforward to generalize the bosonic wave function to N bosons:

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

where

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

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

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

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 [25], §8.3.1.

Since bosons do not follow the antisymmetric Pauli principle, two or more bosons can occupy the same quantum state. When the system is cooled below the critical temperature, all bosons condense into the ground state and form a BEC.

For a BEC, the total wave function has a simpler form. We begin by considering the $N = 2$ case for simplicity and then generalize to N bosons. When there are only two bosons,

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

This is possible because all bosons occupy the lowest-energy state a.

This expression generalizes to N bosons:

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \phi_a(\mathbf{x}_1)\,\phi_a(\mathbf{x}_2)\,\cdots\,\phi_a(\mathbf{x}_N). \tag{2.24}$$

Note that Ψ_+ does not include a normalization constant, since each component wave function is already normalized.

We choose the BEC trial wave function in Gaussian form because it follows from the noninteracting limit of the Gross–Pitaevskii equation, as derived in the ETH Zurich lecture notes [26].

Next, we consider the issues introduced by the electrostatic potential and how to address them.

2.2.16 Kato's Cusp Condition

Certain potentials—most notably the Coulomb potential—become singular when two charged particles coincide. To illustrate this, consider a simple two-electron system in atomic units. The Hamiltonian is

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

Here, the Coulomb potential term has a singularity: if the electrons approach each other arbitrarily closely, i.e. $r_{12} \to 0$, that term diverges and the energy of the system would become infinite. However, this issue can be resolved because the Hamiltonian also contains two kinetic-energy terms, each involving second derivatives. If the wave function is modified so that its second derivative cancels the divergent Coulomb term, the total energy remains finite.

In his original article [27], Tosio Kato addressed these divergences and showed that the wave function must satisfy what is now known as Kato's cusp condition:

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

If the wave function satisfies this condition, the divergence is canceled. Here, $\mu = \frac{m_i\, m_j}{m_i + m_j}$ is the reduced mass of the two particles, and Z is the charge number.

In the next section, we will introduce correlation factors that both satisfy Kato's theorem and account for electron correlation.

2.2.17 Jastrow Factor and Pade–Jastrow Factor

"*In quantum-chemical methods it is common to express many-body wave functions as linear combinations of determinants. However, such expansions converge very slowly because of the difficulty in describing the cusps that occur whenever two electrons come into contact. Quantum Monte Carlo simulations of solids require a much more compact representation and normally use trial wave functions of the Slater–Jastrow type (Jastrow, 1955), consisting of a single SD multiplied by a totally symmetric nonnegative Jastrow correlation factor that includes the cusps. The orbitals in the SD are usually obtained from accurate Hartree–Fock calculations, while the Jastrow factor is chosen to have some specific functional form and optimized*" [9], Chapter IV.

Choosing an appropriate Jastrow factor is nontrivial. For a comprehensive discussion, see Hammond and Reynolds [28]. In general, one writes the Jastrow factor as

$$J = \exp\bigl[u(r_{ij})\bigr],$$

where $u(r_{ij})$ encodes two-body correlations. In this work, we consider two common choices:

- **Linear Jastrow:**
$$J = \exp\Bigl[\sum_{i=1}^{N}\sum_{j>i}^{N} \beta_{ij}\, r_{ij}\Bigr], \tag{2.25}$$
where each β_{ij} is a variational parameter corresponding to the pair (i, j);
- **Pade–Jastrow:**
$$P = \exp\Bigl[\sum_{i=1}^{N}\sum_{j>i}^{N} \frac{a\, r_{ij}}{1+\beta\, r_{ij}}\Bigr], \tag{2.26}$$
where β is a variational parameter, as discussed in Foulkes [9] and in [29].

Here β (for Pade–Jastrow) and β_{ij} (for linear Jastrow) are variational parameters to be optimized. Both forms explicitly incorporate electron–electron correlations and ensure that the trial wave function satisfies the Kato cusp when electrons are close.

The constant a represents correlation between particles with different spins. Huang [30] derived that in d dimensions it is

$$a = \begin{cases} \frac{1}{d+1}, & \text{for same-spin particles,} \\ \frac{1}{d-1}, & \text{for opposite-spin particles.} \end{cases} \tag{2.27}$$

With the essential quantum mechanics now in place, we can turn to one of the most crucial topics of this research—the variational principle.

2.2.18 Variational Principle

The variational principle provides a straightforward upper bound to the ground-state energy. It is easy to derive, so let us follow Griffiths's [18] approach in §7.

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

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

There exists a set of eigenfunctions $|\psi_n\rangle$ and eigenvalues E_n that solve this equation, but they are unknown. Then the general solution $|\psi\rangle$ can be written as a linear combination of the $|\psi_n\rangle$:

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

Here, the c_n are the expansion coefficients discussed in Section 2.2.6. Each $|\psi_n\rangle$ is an eigenstate of the Hamiltonian:

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

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

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

The expectation value of the energy is

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

Since E_0 is the smallest eigenvalue, replacing each E_n with E_0 in the sum gives a lower bound:

$$\sum_n E_0 |c_n|^2 \leq \sum_n E_n |c_n|^2.$$

But $\sum_n |c_n|^2 = 1$, so

$$\sum_n E_0 |c_n|^2 = E_0 \sum_n |c_n|^2 = E_0.$$

Putting everything together:

Variational Principle

For any normalized $|\psi\rangle$, the ground-state energy satisfies

$$E_0 \leq \langle\hat{H}\rangle = \langle\psi| \hat{H} |\psi\rangle .$$

This principle may seem trivial, since it states that the ground-state energy is always less than or equal to the expectation value of the Hamiltonian. At the same time, it offers great freedom because the trial wave function can be chosen arbitrarily. The quantity

$$\Delta E = \langle\hat{H}\rangle - E_0$$

measures the quality of the ansatz wave function. If the ansatz is exact, then equality holds: $\langle\hat{H}\rangle = E_0$. If the ansatz is far from the true wave function, ΔE can be large. Minimizing $\langle\hat{H}\rangle$ over variational parameters drives $\Delta E \to 0$. The variational principle motivates studying the ground-state energy, since it provides a practical method to compute it.

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

The recipe is simple: choose a wave function ansatz (generally with variational parameters), compute the expectation value of the Hamiltonian using this wave function, and tune the parameters to minimize the energy.

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

2.2.19 Local Energy and Variational Monte Carlo

Following Morten Hjorth-Jensen's lecture notes [31], consider a general time-independent system of N particles described by a Hamiltonian $\hat{H}$. Here, we do not assume any specific form for $\hat{H}$ or for the wave function. This level of abstraction highlights the essence of the variational approach and sets the stage for developing the Variational Monte Carlo framework.

According to the variational principle, the ansatz can be chosen arbitrarily:

$$\Psi_T(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N; \boldsymbol{\alpha}) = \Psi_T(\mathbf{R}, \boldsymbol{\alpha}).$$

We assume that the trial wave function is not normalized and that it contains m variational parameters $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_m\}$. The variational energy E_T can be obtained from the time-independent SE (2.14):

$$\hat{H}\,\Psi_T(\mathbf{R}, \boldsymbol{\alpha}) = E_T\,\Psi_T(\mathbf{R}, \boldsymbol{\alpha}).$$

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

$$\int \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha})\,\hat{H}\,\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\,dV = E_T \int \left|\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\right|^2 dV.$$

Hence, the variational energy can be written as

$$E_T = \frac{\displaystyle\int \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha})\,\hat{H}\,\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\,dV}{\displaystyle\int \left|\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\right|^2 dV}. \tag{2.28}$$

Equation (2.28) may not seem daunting at first glance, but let us examine the integrals more closely. For simplicity, consider a BEC system where the ansatz is given by (2.24):

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

Then the denominator in (2.28) simplifies to

$$\int \left|\Psi_T(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N; \boldsymbol{\alpha})\right|^2 dV = \int_{-\infty}^{\infty} \phi(\mathbf{x}_1; \boldsymbol{\alpha})^2\,d\mathbf{x}_1 \cdots \int_{-\infty}^{\infty} \phi(\mathbf{x}_N; \boldsymbol{\alpha})^2\,d\mathbf{x}_N.$$

This expression involves N separate d-dimensional integrals over $(-\infty, \infty)$. For ten particles with just ten grid points per coordinate, the required number of evaluations becomes astronomically large, far beyond any realistic computing capability—making a direct evaluation effectively impossible.

Because a direct evaluation of the integral in (2.28) is infeasible, we employ Variational Monte Carlo to estimate it. For a trial wave function, the probability distribution function P is defined by

$$P(\mathbf{R}; \boldsymbol{\alpha}) = \frac{\left|\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\right|^2}{\displaystyle\int \left|\Psi_T(\mathbf{R}, \boldsymbol{\alpha})\right|^2 dV}. \tag{2.29}$$

Note that this is exactly the same probability density ρ discussed in Section 2.2.3 for a non-normalized wave function. We can then define the **local energy**:

$$E_L(\mathbf{R}, \boldsymbol{\alpha}) = \frac{1}{\Psi_T(\mathbf{R}, \boldsymbol{\alpha})}\,\hat{H}\,\Psi_T(\mathbf{R}, \boldsymbol{\alpha}). \tag{2.30}$$

The expectation value of the local energy is

$$\langle E_L \rangle = \int P(\mathbf{R};\boldsymbol{\alpha})\, E_L(\mathbf{R},\boldsymbol{\alpha})\, dV = \int \Big(\frac{|\Psi_T|^2}{\int |\Psi_T|^2\, dV}\, E_L \Big) dV = \frac{\int |\Psi_T|^2\, E_L\, dV}{\int |\Psi_T|^2\, dV}$$
$$= \frac{\int \Psi_T^*\, \Psi_T\, E_L\, dV}{\int |\Psi_T|^2\, dV} = \frac{\int \Psi_T^*\, \hat{H}\, \Psi_T\, dV}{\int |\Psi_T|^2\, dV}.$$

Thus, $\langle E_L \rangle$ coincides with the variational energy defined in (2.28). In practice, one writes:

$$E_T = \langle E_L \rangle = \int P\, E_L\, dV \approx \frac{1}{n} \sum_{i=1}^{n} E_L(\mathbf{R}_i, \boldsymbol{\alpha}), \tag{2.31}$$

where n is the number of Monte Carlo samples. By replacing the high-dimensional integral $\int P\, E_L\, dV$ with its Monte Carlo estimate, VMC drastically reduces computational cost while retaining high accuracy. Moreover, the variational principle guarantees

$$E_T(\boldsymbol{\alpha}) \;\geq\; E_0.$$

Crucially, the value E_T becomes an estimate of the true ground-state energy only after the variational parameters $\boldsymbol{\alpha}$ have been optimized. The entire VMC procedure is therefore aimed at minimizing $E_T(\boldsymbol{\alpha})$ so that it approaches E_0 as closely as possible.

With the necessary quantum foundations now established, we can proceed to introduce our model systems using the formal language of quantum mechanics.

2.2.20 Model Systems

In this research, we study three distinct model systems:

- **N non-interacting bosons in d dimensions:**
  A gas of N non-interacting bosons confined in d dimensions, which serves as a simple model for a Bose–Einstein condensate of alkali atoms.

- **Two-dimensional non-interacting fermions ($\mathbf{N = 2, 6, 12}$):**
  An idealized, non-interacting quantum dot containing $N = 2, 6$, or 12 electrons in two dimensions. These choices correspond to the first three closed-shell configurations, each with total spin $S = 0$;

- **Two-dimensional interacting fermions ($\mathbf{N = 2, 6, 12}$):**
  A more realistic quantum dot model in two dimensions, where $N = 2, 6$, or 12 electrons experience mutual interactions. Again, we focus on the first three closed-shell configurations.

A quantum system is fully specified by its Hamiltonian. In our research, we use the following Hamiltonians:

- **Non-interacting particles in a HO potential:**

$$\hat{H} = -\frac{\hbar^2}{2m} \sum_{i=1}^{N} \Delta_i + \frac{m\omega^2}{2} \sum_{i=1}^{N} r_i^2; \tag{2.32}$$

- **Interacting electrons in a HO potential:**

$$\hat{H} = -\frac{\hbar^2}{2m}\sum_{i=1}^{N}\Delta_i + \frac{m\omega^2}{2}\sum_{i=1}^{N} r_i^2 + \frac{e^2}{4\pi\varepsilon_0}\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}}. \tag{2.33}$$

We will work in atomic units, $\hbar = m = e = 4\pi\varepsilon_0 = 1$ (Szabo and Ostlund [11], §2.1). In these units, Hamiltonians (2.32) and (2.33) reduce to

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N}\Delta_i + \frac{1}{2}\,\omega^2\sum_{i=1}^{N} r_i^2, \tag{2.34}$$

and

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N}\Delta_i + \frac{1}{2}\,\omega^2\sum_{i=1}^{N} r_i^2 + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}}. \tag{2.35}$$

In Appendix 6.1, we derive these Hamiltonians in SI units by rescaling the coordinates.

For the variational principle, we need to choose an ansatz and modify it with variational parameters. For now, we specify the general form of the bosonic and fermionic many-body wave functions:

- **Bosonic many-body wave function** (equation (2.24)):

$$\Psi_{\mathrm{B}} = \prod_{i=1}^{N}\phi_a(\mathbf{x}_i); \tag{2.36}$$

- **Fermionic many-body wave function** (equation (2.23)):

$$\Psi_{\mathrm{F}} = \frac{1}{\sqrt{N!}}\begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{vmatrix}. \tag{2.37}$$

When considering interacting fermions, the fermionic many-body wave function is modified with correlation Jastrow factors of the form:

- **Linear Jastrow factor** (equation (2.25)):

$$J = \exp\Big[\sum_{i=1}^{N}\sum_{j>i}^{N}\beta_{ij}\,r_{ij}\Big]; \tag{2.38}$$

- **Pade–Jastrow factor** (equation (2.26)):

$$P = \exp\Big[\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{a\,r_{ij}}{1+\beta\,r_{ij}}\Big]. \tag{2.39}$$

With the Hamiltonians and corresponding many-body wave functions defined above, we can perform a VMC estimation of the ground-state energy. The precise form of each variational ansatz will be detailed in Chapter 3.

Having established the physical framework, we now turn to the computational foundations required to implement VMC. In particular, we will review the statistical principles underlying sampling methods, delve into Monte Carlo integration techniques, and outline how gradient descent can be used to adjust variational parameters toward minimizing the energy estimate.

2.3 Computational Foundation

2.3.1 Bayesian Statistics

Following "Markov Chain Monte Carlo in Practice" [32] by W. R. Gilks, S. Richardson, and D. J. Spiegelhalter (§1.2), Markov Chain Monte Carlo 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 then requires constructing the joint distribution $P(D, \theta)$ over all random quantities:

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

This probability distribution represents a full probability model. Here, $P(\theta)$ is a prior distribution and $P(D \mid \theta)$ is a likelihood.

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

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

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

$$\mathbb{E}\big[f(\theta) \mid D\big] = \frac{\int f(\theta)\, P(\theta)\, P(D \mid \theta)\, d\theta}{\int P(\theta)\, P(D \mid \theta)\, d\theta}.$$

This expectation value is impossible to evaluate analytically in high dimensions, so numerical methods are used. Markov Chain Monte Carlo techniques are the most widely used and stable.

In general terms, let X denote a vector of k variables with distribution $\pi(x) = P(D \mid \theta)\, P(\theta)$. In these terms, the expectation value is

$$\mathbb{E}\big[f(X)\big] = \frac{\int f(x)\, \pi(x)\, dx}{\int \pi(x)\, dx}.$$

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

2.3.2 Monte Carlo Integration

According to Gilks [32] (§1.3.1), numerical Monte Carlo methods evaluate $\mathbb{E}[f(X)]$ by drawing samples $\{X_t,\ t = 1, \ldots, n\}$ from $\pi(x)$ and then approximating

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

In other words, the population mean of $f(X)$ is replaced by the sample mean. The law of large numbers ensures that this estimate becomes arbitrarily accurate as the number of Monte Carlo samples n increases.

The distribution $\pi(x)$ can be complicated, making it impossible to draw samples $\{X_t\}$ independently from $\pi(x)$. However, the samples need not be independent draws from $\pi(x)$. Instead, $\{X_t\}$ can be generated so that they cover the support of $\pi(x)$ in the correct proportions. The most frequently used way to generate $\{X_t\}$ is via a Markov chain that has $\pi(x)$ as its stationary distribution.

2.3.3 Markov Chains

Following Gilks [32], §1.3.2, suppose that we generate a sequence of $t+1$ random variables $\{X_0, X_1, \ldots, X_t\}$ such that at each time $t > 0$, the next state X_{t+1} is sampled from the distribution $P(X_{t+1} \mid X_t)$. This means that the state X_{t+1} depends only on the previous state X_t and not on the history of the chain $\{X_0, \ldots, X_{t-1}\}$. This sequence is called a Markov chain, and $P(X_{t+1} \mid X_t)$ is called the transition kernel of the chain. Generally, the kernel $P(X_{t+1} \mid X_t)$ can be time-dependent, but we assume it is time-independent.

A natural question may arise: how does the initial state X_0 affect X_t? In this scenario, we know nothing about the intermediate states $\{X_1, \ldots, X_{t-1}\}$, and the question concerns the distribution of X_t given X_0, denoted $P^{(t)}(X_t \mid X_0)$. Under regularity conditions, the chain will eventually "forget" its initial state, and $P^{(t)}(X_t \mid X_0)$ will converge to a unique stationary distribution $\phi(x)$. After a sufficient number of steps, as t increases, the samples $\{X_t\}$ will increasingly resemble dependent draws from $\phi(x)$.

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

$$\mathbb{E}\left[f(X)\right] = \frac{1}{n-m} \sum_{t=m+1}^{n} f(X_t). \tag{2.40}$$

2.3.4 Monte Carlo Error Estimate

According to the article by U. S. Fjordholm [33], the measure of Monte Carlo error is the root mean square error

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

where $I_M = \frac{1}{n}\sum_{t=1}^{n} f(X_t)$ is the Monte Carlo estimate. It can be shown 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 constant, the root mean square error of the Monte Carlo estimator scales as $M^{-1/2}$; the error decreases as the number of Monte Carlo samples increases.

2.3.5 Metropolis and Metropolis–Hastings Algorithms

Equation (2.40) provides a recipe to compute the Monte Carlo estimator $\mathbb{E}[f] = \langle f \rangle$, drawing samples from the unitary distribution $\phi(x)$. The Metropolis–Hastings algorithm ensures 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 $q(\cdot \mid X_t)$*" – Gilks [32] (§1.3.3). The proposal distribution q can depend on the current state X_t, and it is frequently chosen as a uniform distribution $U(0,1)$, although it can take any form. The candidate state Y is accepted with probability $\alpha(X_t, Y)$:

$$\alpha(X_t, Y) = \min\Big(1,\ \frac{\pi(Y)\, q(X_t \mid Y)}{\pi(X)\, q(Y \mid X_t)}\Big).$$

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 step is presented in Algorithm 1.

Algorithm 1 Metropolis–Hastings step

procedure MHSTEP(X_t, π, q)
  Sample point Y from q;
  Sample a uniform random variable $u \sim U(0,1)$;
  Compute acceptance ratio

$$\alpha(X_t, Y) \leftarrow \min\Big(1,\ \frac{\pi(Y)\, q(X \mid Y)}{\pi(X)\, q(Y \mid X)}\Big);$$

  if $u \leq \alpha$ **then**
    $X_{t+1} \leftarrow Y$;
  else
    $X_{t+1} \leftarrow X_t$;
  end if
  return X_{t+1}.
end procedure

The Metropolis algorithm is a special case of Metropolis–Hastings in which the proposal densities $q(X \mid Y)$ and $q(Y \mid X)$ are taken to be constant. In that case, the acceptance probability reduces to

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

We will discuss in detail how we implement both Metropolis (constant-q) and the more general Metropolis–Hastings algorithms in Sections 3.4 and 3.6.

2.3.6 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 choice would represent a random number. However, we cannot ask a computer to pick a random number. A computer is a machine that performs arithmetic operations; it

cannot pick a purely random number. Therefore, scientists developed pseudo-random number generation algorithms that generate pseudo-random numbers.

"Random Number Generation and Monte Carlo Methods" [34] by James E. Gentle provides a thorough description of pseudo-random number generation algorithms. We will discuss the very basics of random number generation according to Gentle's §1.1 and §1.2.

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

$$a = b \bmod m.$$

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

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

$$x_i = (a\, x_{i-1} + c) \bmod m$$

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

One feature of pseudo-random number generation is that the sequences are generally defined by a set of constants. Sequences generated with the same constants are obviously identical. These parameters are called a seed. Using the same seed allows one to generate the same sequence of pseudo-random numbers, which is very helpful when developing code, since computations that include random numbers can be reproduced.

2.3.7 Direct Sampling and Importance Sampling

While discussing the Metropolis–Hastings algorithm, it was stated that the random variable is sampled from a uniform distribution. In this section, we will follow F. Becca's [35] approach in §3.3 and §3.4 to discuss the concepts of direct sampling and importance sampling.

Consider estimating the value of π stochastically. In this case, we draw a circle of radius $r = 1$ and the square that contains it; see Figure 2.1. We can then randomly "shoot bullets" inside the square, counting each trial. By keeping track of trials and hits, we can perform a Monte Carlo integration to estimate π:

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

where

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

and $\mathcal{P}(x,y) = \left(\int_{\text{square}} dx\, dy\right)^{-1}$ represents the probability density 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; it is not counted as a hit otherwise. Thus, π can be estimated as

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

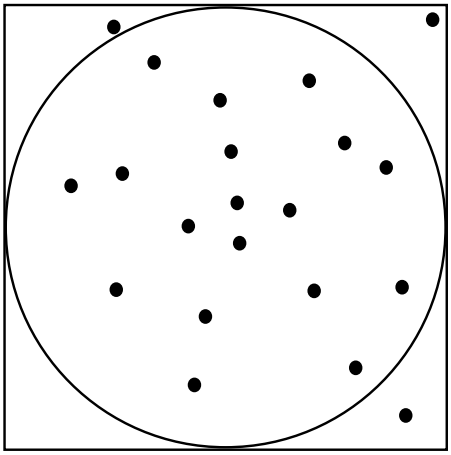


Figure 2.1: Simulation of 20 random shots inside the circle. The scheme is inspired by Figure 3.1 in F. Becca's book.

In this case, direct uniform sampling works perfectly well. However, if $f(x)$ is a sharply peaked function, uniformly drawing random numbers is not efficient. The reason is that we would compute many points x_i that contribute almost nothing to the integral. In this situation, a technique known as importance sampling must be implemented.

Consider the stochastic computation of the integral

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

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

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

This integral can be estimated by:

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

where the x_i are drawn from $\mathcal{P}(x)$. If the function $\mathcal{P}(x)$ is chosen to be close to $F(x)$, then the statistical error is drastically reduced.

2.3.8 Variational Monte Carlo Revisited

As the name suggests, Variational Monte Carlo is a combination of two techniques: the variational principle, which provides a recipe to estimate the GS energy, and the Monte Carlo technique, which is used to compute the integral estimation of the GS energy. Let us revisit Section 2.2.19 in a different manner.

The variational principle states that for a given wave-function ansatz Ψ_T, the GS energy estimate is

$$E_T \;=\; \frac{\langle\Psi_T|\,\hat{H}\,|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} \;\geq\; E_0.$$

According to J. Gubernatis's book on quantum Monte Carlo methods [36] (§9.1), the goal of the VMC method is to make E_T as close as possible to E_0.

Suppose that the state $|\Psi_T\rangle$ is a good estimate of some eigenstate of a Hamiltonian. This eigenstate may or may not be the ground state. In general, the trial state Ψ_T is not an eigenstate of the Hamiltonian, meaning we cannot use the standard relationship $\hat{H}\,|\Psi_T\rangle = E_T\,|\Psi_T\rangle$. 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 energy E_T. Obviously, the residual state is zero if $|\Psi_T\rangle$ is an eigenstate of $\hat{H}$. If $|\Psi_T\rangle$ is not an eigenstate, we want to find a value E_T that minimizes

$$\langle\phi|\phi\rangle \;=\; \Big\langle (\hat{H} - E_T)\Psi_T \Big| (\hat{H} - E_T)\Psi_T \Big\rangle\,.$$

Expanding the expression above yields:

$$\langle\phi|\phi\rangle \;=\; \Big\langle \hat{H}\Psi_T \Big| \hat{H}\Psi_T \Big\rangle \;-\; E_T \Big\langle \hat{H}\Psi_T \Big| \Psi_T \Big\rangle \;-\; E_T \Big\langle \Psi_T \Big| \hat{H}\Psi_T \Big\rangle \;+\; E_T^2.$$

Define $\lambda = \Big\langle \Psi_T \Big| \hat{H}\Psi_T \Big\rangle \;/\; \langle\Psi_T|\Psi_T\rangle$, so that the equation takes the form

$$\langle\phi|\phi\rangle \;=\; \Big\langle \hat{H}\Psi_T \Big| \hat{H}\Psi_T \Big\rangle \;+\; \langle\Psi_T|\Psi_T\rangle\;[(E_T - \lambda)^2 - \lambda^2].$$

We want to minimize this expression, and since both inner products are positive, this is achieved by setting

$$\lambda \;=\; E_T \;=\; \frac{\langle\Psi_T|\,\hat{H}\,|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle}.$$

The ratio on the right-hand side is called Rayleigh's quotient. We have shown that the lower bound of the Rayleigh quotient is the GS energy; the upper bound is the highest energy eigenstate E_N.

If the trial state is accurate to $\mathcal{O}(\varepsilon)$, then the trial energy is accurate 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, the trial energy becomes

$$E_T \;=\; \frac{\sum_C \langle\Psi_T|C\rangle\;\langle C|\,\hat{H}\,|\Psi_T\rangle}{\sum_C \big|\langle C|\Psi_T\rangle\big|^2}.$$

This can be rewritten as

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

where

$$E(C) \;=\; \frac{\langle C|\,\hat{H}\,|\Psi_T\rangle}{\langle C|\Psi_T\rangle} \quad\text{and}\quad P(C) \;=\; \frac{\big|\langle C|\Psi_T\rangle\big|^2}{\sum_C \big|\langle C|\Psi_T\rangle\big|^2}.$$

Note that this is the discrete case. It reduces to the continuous case discussed in Section 2.2.19 in the continuous limit.

The variational 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 the Metropolis or Metropolis–Hastings algorithm to sample C_i from $P(C)$. This algorithm requires a proposal probability $T(C' \mid C)$, and the acceptance probability becomes

$$\alpha(C, C') \;=\; \min\Big(1,\, \frac{P(C')\,T(C \mid C')}{P(C)\,T(C' \mid C)}\Big) \;=\; \min\Big(1,\, \frac{\big|\langle C'|\Psi_T\rangle\big|^2\,T(C \mid C')}{\big|\langle C|\Psi_T\rangle\big|^2\,T(C \mid C')}\Big).$$

2.3.9 Gradient Descent

In this research, we need to optimize the variational parameters of a trial wave-function ansatz, i.e., find parameters that minimize equation (2.28). It can be shown that E_T is a convex function, and various methods can be used to minimize it. One of the standard methods is gradient descent.

Following [37] (§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 assume there exists a solution x^* to this problem, so that $p^* = f(x^*)$ is the optimal value.

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

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

For a general f, the equation cannot be solved analytically, and an iterative algorithm must be used. By an iterative algorithm, we mean a sequence $x^{(0)}$, $x^{(1)}$, ... with $f\big(x^{(k)}\big) \to p^*$ as $k \to \infty$. The algorithm requires a stopping criterion B, and it terminates when

$$f\big(x^{(k)}\big) - p^* < B.$$

The descent family of methods produces a minimizing sequence

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

Here, $t^{(k)}$ is a step size, $\Delta x^{(k)}$ is a search direction, and $k = 0, 1, \ldots$ is the iteration index. For this family of methods, $f\big(x^{(k+1)}\big) < f\big(x^{(k)}\big)$.

Gradient descent is a member of the descent family with $\Delta x = -\nabla f(x)$. The minimizing sequence then becomes

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

The gradient descent procedure is presented in Algorithm 2. Here, $\|x\|_2$ denotes the ℓ_2 norm of a vector x, defined by $\|x\|_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}$. The function GD has only one argument—the starting position x_0—and returns x when $\|\nabla f(x)\|_2 < B$.

Algorithm 2 Gradient Descent

procedure GD(x_0)
    Choose step t;
    $x \leftarrow x_0$;
    while $\|\nabla f(x)\|_2 > B$ **do**
        $\Delta x \leftarrow -\nabla f(x)$;
        $x \leftarrow x + t\,\Delta x$;
    end while
    return x.
end procedure

Chapter 3

Methods and implementations

3.1 Single-Particle Orbitals

As we saw in Sections 2.2.14 and 2.2.15, many-body wave functions are built from one-body orbitals. In this section, we specify the precise form of those single-particle orbitals and discuss the unit systems we use.

Because the systems of interest use a HO potential, the one-body wave functions that compose the many-body state are the single-particle orbitals (2.17). We first consider the one-dimensional case:

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

with

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

and the energy of the nth single-particle orbital is

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

One feature of VMC is that only ratios of wave functions enter into the sampling and acceptance criteria. As a result, any constant prefactor in Ψ_T cancels out and can be dropped, meaning

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

To perform a VMC, these orbitals need to be modified by introducing a variational parameter α. One way to do this is:

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

With this choice, taking spatial derivatives does not bring in extra normalization constants, which keeps the analytical expressions simpler and reduces computational overhead.

In Appendix 6.1, we show how Hamiltonians of the model systems can be made dimensionless using a change of variables. There are two natural systems of units for the model systems:

- When we consider the non-interacting problem and do not aim to study ω-dependence, we use HO units. HO units set $\hbar = m = \omega = 1$, so that the single-particle orbital becomes

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

and the energy of the one-dimensional nth orbital is

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

The Hamiltonian in these units is given by (2.34) with $\omega = 1$:

$$\hat{H} = -\frac{1}{2}\sum_{i=1}^{N} \Delta_i + \frac{1}{2}\sum_{i=1}^{N} r_i^2. \tag{3.3}$$

- If we wish to study ω-dependence or interacting fermions, we use atomic units. Atomic units set $\hbar = m = e = 4\pi\epsilon_0 = 1$. In these units, HO wave functions are given by:

$$\psi_n(x) = He_n(\sqrt{\omega}\, x)\, e^{-\alpha\omega x^2}, \tag{3.4}$$

and the energy is

$$E_n = \left(n + \frac{1}{2}\right)\omega. \tag{3.5}$$

The Hamiltonians of non-interacting and interacting systems in atomic units are given by equations (2.34) and (2.35).

Notice that

$$\psi_n^{\text{au}}(x) = \psi_n^{\text{HO}}(\sqrt{\omega}\, x) \quad \text{and} \quad E_n^{\text{au}} = \omega\, E_n^{\text{HO}}.$$

Therefore, for notational simplicity we will use HO units, with the understanding that the ω-dependence can be easily restored.

Recall that in two dimensions, a harmonic-oscillator orbital is simply the product of two one-dimensional orbitals with quantum numbers n_x and n_y:

$$\psi_{n_x,n_y}(x,y) = \psi_{n_x}(x)\, \psi_{n_y}(y).$$

Similarly, in three dimensions:

$$\psi_{n_x,n_y,n_z}(x,y,z) = \psi_{n_x}(x)\, \psi_{n_y}(y)\, \psi_{n_z}(z).$$

Orbit energies in 2D and 3D are

$$E_{n_x,n_y} = n_x + n_y + 1; \quad E_{n_x,n_y,n_z} = n_x + n_y + n_z + \frac{3}{2}.$$

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

3.2 Bosonic Ansatz

In Sections 2.2.15 and 2.2.14, we discussed a general way of constructing many-body wave functions from single-particle wave functions for fermions and bosons. Two factors impact the form of the wave function: the variational principle and the harmonic potential. The variational principle implies that the trial GS energy obtained using a given trial wave function must be minimized. Therefore, the trial wave function needs to contain a number of variational parameters. By optimizing these parameters, the variational energy is minimized and estimates the true GS energy. As we discussed earlier, the trial wave function can be chosen arbitrarily; the closer it is to the true wave function, the better.

Consider the bosonic trial wave-function ansatz first. Recall that for a BEC the general wave function is given by equation (2.36):

$$\Psi_{\mathrm{B}}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \prod_{i=1}^{N} \phi_a(\mathbf{x}_i).$$

Since bosons are not subject to the Pauli principle, they can all occupy the same single-particle state. In a BEC, all N particles reside in the ground-state orbital, so

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

Spin does not enter the Hamiltonian for our bosonic system; hence, we drop the spin coordinate and set $\phi_a(\mathbf{x}) \to \psi_1(\mathbf{r})$. Thus, the trial wave-function ansatz for N bosons becomes

$$\Psi_T(\mathbf{r}_1, \ldots, \mathbf{r}_N) = \prod_{i=1}^{N} \psi_1(\mathbf{r}_i) = e^{-\alpha \sum_{i=1}^{N} r_i^2}. \tag{3.6}$$

This expression 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 Ansatz

A 2D quantum state in a harmonic trap is defined by two quantum numbers—principal quantum numbers n_x and n_y. Particles trapped in such a potential have an intrinsic quantum number: the projection of the spin on the z-axis, s_z. Therefore, a fermion trapped in the quantum potential is described by the quantum numbers n_x, n_y, and s_z. Since we are interested in the ground-state energy, we need to construct a trial wave function that describes a many-body state with the lowest energy.

We will consider only "closed-shell" states. A closed-shell state means that all available states up to $n = n_x + n_y$ are occupied. The first shell has $n = 0$, which occurs only when $n_x = n_y = 0$. The first closed shell ($n = 0$) is filled by two fermions with opposite spin projections occupying the only available state. No more fermions can be added to this shell. This configuration is schematically presented in Figure 3.1.

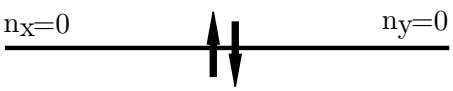


Figure 3.1: Schematic representation of the first closed shell. Here ↑ represents a fermion with $s_z = +\frac{1}{2}$ and ↓ represents a fermion with $s_z = -\frac{1}{2}$ (in atomic units).

The next shell has $n = 1$. This shell includes 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, the second closed shell contains six fermions. This configuration is shown schematically in Figure 3.2.

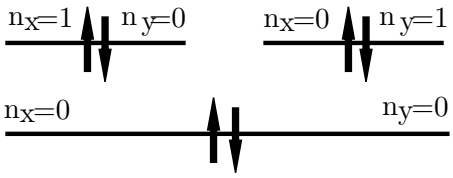


Figure 3.2: Schematic representation of the second closed shell. Here ↑ represents a fermion with $s_z = +\frac{1}{2}$ and ↓ represents a fermion with $s_z = -\frac{1}{2}$ (in atomic units).

The third shell has $n = 2$. There are six fermions that occupy states up to $n = 1$. In addition, there are two fermions in the state $(n_x = 2, n_y = 0)$, two fermions in the state $(n_x = 0, n_y = 2)$, and two fermions in the state $(n_x = 1, n_y = 1)$. Overall, the third closed shell contains twelve fermions. A schematic representation of this shell is shown in Figure 3.3.

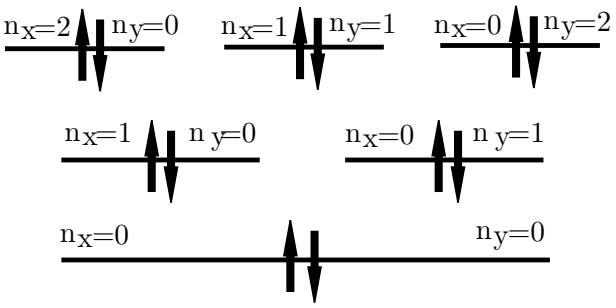


Figure 3.3: Schematic representation of the third closed shell. Here ↑ represents a fermion with $s_z = +\frac{1}{2}$ and ↓ represents a fermion with $s_z = -\frac{1}{2}$ (in atomic units).

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

In general, the wave function that represents such states is a Slater determinant (SD) of the form (2.37). However, for systems like ours, it can be simplified. In their article, J. W. Moskowitz and M. H. Kalos proposed that an SD can be factorized into separate components [38] (Gubernatis [36] follows this approach in §9.2.1):

$$\Psi_T = D_\uparrow(\mathbf{r}_1, \ldots, \mathbf{r}_{N/2}) \cdot D_\downarrow(\mathbf{r}_{N/2+1}, \ldots, \mathbf{r}_N). \tag{3.7}$$

Here,

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

and

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

A normalization constant is generally required, but the VMC machinery uses only ratios of wave functions, so we can omit it.

In the definitions of $D_\uparrow$ and $D_\downarrow$, we label all spin-up particles as $\{1, 2, \ldots, N/2\}$ and all spin-down particles as $\{N/2+1, N/2+2, \ldots, N\}$. This choice is merely a convention; we could have labeled the first $N/2$ particles as spin-down and the remaining as spin-up. Throughout this work, $D_\uparrow$ denotes the spin-up Slater determinant (built from particles 1 through $N/2$), and $D_\downarrow$ denotes the spin-down Slater determinant (built from particles $N/2+1$ through N). Both determinants share the same functional form and differ only in their particle indices.

After splitting the original determinant, we can replace the spin-dependent wave functions $\phi_i(\mathbf{r}_j, \xi)$ with spin-independent orbitals $\psi_i(\mathbf{r}_j)$. Spin serves only to specify the occupancy structure of the many-body ansatz, and since the Hamiltonian is spin-independent, we can omit the spin coordinate. Therefore, the spin-up and spin-down Slater determinants become

$$D_\uparrow(\mathbf{r}_1, \ldots, \mathbf{r}_{N/2}) = \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \cdots & \psi_{N/2}(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \cdots & \psi_{N/2}(\mathbf{r}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_{N/2}) & \psi_2(\mathbf{r}_{N/2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2}) \end{vmatrix} \tag{3.8}$$

and

$$D_\downarrow(\mathbf{r}_{N/2+1}, \ldots, \mathbf{r}_N) = \begin{vmatrix} \psi_1(\mathbf{r}_{N/2+1}) & \psi_2(\mathbf{r}_{N/2+1}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+1}) \\ \psi_1(\mathbf{r}_{N/2+2}) & \psi_2(\mathbf{r}_{N/2+2}) & \cdots & \psi_{N/2}(\mathbf{r}_{N/2+2}) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_N) & \psi_2(\mathbf{r}_N) & \cdots & \psi_{N/2}(\mathbf{r}_N) \end{vmatrix}. \tag{3.9}$$

A more compact way to write the Slater determinants is in component form:

$$[D_\uparrow]_{ij} = \psi_i(\mathbf{r}_j), \tag{3.10}$$

where $i = 1, \ldots, N/2$ labels the single-particle orbital (row index) and $j = 1, \ldots, N/2$ labels the particle coordinate (column index). Similarly, for the spin-down determinant $D_\downarrow$ we have

$$[D_\downarrow]_{ij} = \psi_i(\mathbf{r}_{N/2+j}), \tag{3.11}$$

with the same indices i and j.

Let us now consider the first three closed-shell systems. The first system has only two particles. In this case, the 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 a HO wave function with $n_x = n_y = 0$, given by formula (6.7):

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

The trial wave function then has the simple form:

$$\Psi_T(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha\left(r_1^2 + r_2^2\right)}. \tag{3.12}$$

Notice that this wave function is equivalent to the bosonic trial wave function given by equation (3.6) with $N = 2$. Both wave functions describe particles (fermions or bosons)

occupying the lowest shell with $n = 0$, so it is not surprising that they have the same functional form. Note that this spatial wave function is symmetric under exchange of the two particle coordinates. Only the full many-body wave function, including spin, must be antisymmetric for fermions; in this case, the antisymmetry is provided by the spin part.

For the second closed shell, the trial wave function is constructed from spin-up and spin-down Slater determinants. The orbital $\psi_1(x, y)$ is the same as for the two-particle system. The orbitals $\psi_2(x, y)$ and $\psi_3(x, y)$, corresponding to $(n_x = 1, n_y = 0)$ and $(n_x = 0, n_y = 1)$, are given by formulae (6.12) and (6.13):

$$\psi_2(x, y) = \psi_{n_x=1}(x)\,\psi_{n_y=0}(y) = x\,e^{-\alpha r^2}, \quad \psi_3(x, y) = \psi_{n_x=0}(x)\,\psi_{n_y=1}(y) = y\,e^{-\alpha r^2}.$$

The trial wave function for the third closed shell is also a product of spin-up and spin-down Slater determinants. The orbitals ψ_1–ψ_3 are as before, and the orbitals ψ_4, ψ_5, ψ_6 are given by (6.18), (6.19), and (6.20):

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

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

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

The ansatzes we have constructed describe two, six, and twelve non-interacting fermions. To account for interactions among the fermions, these ansatzes must be multiplied by a Jastrow factor (2.25) or a Pade-Jastrow factor (2.26):

$$\Psi_J = D_\uparrow\,D_\downarrow\,J \quad \text{and} \quad \Psi_P = D_\uparrow\,D_\downarrow\,P. \tag{3.13}$$

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

3.4 Metropolis VMC Algorithm

Recall that the VMC machinery is designed to estimate the GS energy as (2.31):

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

Let us go through the VMC algorithm step by step. The first step is to choose the system we will study. In practice, this means choosing the parameters of the system: the number of particles, the number of dimensions, the system Hamiltonian, and a trial wave function with variational parameters. Variational parameters can be initialized with values in the half-open interval $(0, 1]$. The quality of the initial variational parameters determines how many VMC steps will be needed to find the optimal parameters and the minimum trial energy. For now, we will not vary the oscillator frequency. With all parameters set, we can begin the VMC algorithm.

In order to start the algorithm, we need to set the positions of all particles. Recall that VMC has a burn-in time, meaning that the first m VMC iterations will be discarded, so the initial positions of the particles do not matter. A popular choice is to sample particle positions from the uniform distribution $U(0, 1)$. This means that each particle will have a random position in the interval $[0, 1)$ in each dimension.

When the particles have starting positions, the VMC procedure can be initialized. The general idea is simple: compute the local energy at the current positions, accept or reject the proposed state, update positions if the step is accepted, and repeat. In practice, computing the local energy is a complicated task due to the complexity of the trial wave function and the Hamiltonian.

New particle positions are generated using random numbers $\mathbf{u} = (u_x, u_y, u_z)$ drawn from the uniform distribution $U(0,1)$. Let $\mathbf{r}^{(t)} = (x^{(t)}, y^{(t)}, z^{(t)})$ denote the position of one particle at iteration t. The new position of that particle, $\mathbf{r}^{(t+1)} = (x^{(t+1)}, y^{(t+1)}, z^{(t+1)})$, can be computed as

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

Here, h is a step size that determines how far particles are moved each iteration. Since $u \in [0,1)$, $u - 0.5\,h$ can be positive or negative, so the position can increase or decrease. In the long run, each particle moves around the origin $(0,0,0)$. The same procedure is applied to all particles at every VMC iteration. A new random number is generated for each dimension of each particle. At every iteration, either the set of proposed random displacements or the set of previous positions must be saved so that, if the step is rejected, the old positions can be restored.

Let $\Psi^{(t+1)}$ denote the proposed wave function with new coordinates and $\Psi^{(t)}$ denote the previous wave function. The step is accepted if a random number $u \sim U(0,1)$ satisfies

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

This acceptance criterion ensures that the system evolves not only toward a state with high probability but sometimes toward states with lower probability. If the system always evolved to the state with the highest probability, it would reach that state after some iterations and then remain there. This criterion ensures that the samples span different states while being guided by the probability distribution function. This is exactly what is needed to estimate the GS energy.

During every iteration, we compute and collect values of the local energy and other relevant quantities. When the VMC iterations are finished, we compute the sample mean of the local energy to obtain the GS energy estimate. Afterwards, gradient descent is performed to find more optimal variational parameters. The procedure stops when the ℓ_2 norm of the gradient vector of variational parameters becomes less than the stopping criterion.

There is one variational parameter α in the non-interacting part of the wave-function ansatz. This variational parameter governs the non-interacting problem. Once its optimal value is found, it can be used in subsequent interacting computations. The Jastrow factor J has $p = N(N-1)/2$ variational parameters β_{ij}, where N is the number of particles. All these parameters are updated using gradient descent after each full VMC cycle. When $N = 12$, there are 66 variational parameters, and optimizing all of them simultaneously is quite challenging. Consequently, a Jastrow-factor ansatz requires many VMC iterations to yield an accurate estimate of the local energy. In contrast, a Padé-Jastrow factor has only one variational parameter β. It is much simpler to optimize a single parameter rather than 66 parameters at once. However, because we attempt to describe twelve particles with a single variational parameter, it also requires many VMC iterations to converge.

Let us summarize and present the VMC burn-in algorithm schematically. We discuss the implementation of VMC for model systems in detail in the Computational

Implementation (Section 3.10). The burn-in algorithm is presented in Algorithm 3. The function BIVMC has four arguments: the number of particles N, the number of dimensions d, the step size h, and the number of burn-in iterations m. This function returns the 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

procedure BIVMC(N, d, h, m)
  Initialize uniform random positions of all particles in a 2D matrix $\mathbf{X}$ of size $N \times d$;
  for `int` $i = 0; i < m; i++$ **do**
    Compute old state probability density $\rho_{\text{old}} \leftarrow \left|\Psi_T(\mathbf{X})\right|^2$;
    Propose new positions for all particles, obtaining the new position matrix $\mathbf{X}_n$;
    Compute new state probability density $\rho_{\text{new}} \leftarrow \left|\Psi_T(\mathbf{X}_n)\right|^2$;
    Compute acceptance probability $\alpha \leftarrow \min\left(1, \frac{\rho_{\text{new}}}{\rho_{\text{old}}}\right)$;
    Generate a uniform random number $u \sim U(0, 1)$;
    if $u < \alpha$ **then**
      Accept the step: $\mathbf{X} \leftarrow \mathbf{X}_n$;
    else
      Reject the step: $\mathbf{X}$ remains unchanged;
    end if
  end for
  return $\mathbf{X}$.
end procedure

3.5 Gradient Descent

As we discussed in Section 2.3.9, Gradient Descent is used to find optimal values of variational parameters that minimize the variational energy. Variational parameters are a part of non-interacting bosonic and fermionic trial wave functions, the Jastrow factor, and the Pade-Jastrow factor. Non-interacting trial wave functions (3.6) and (3.7) contain one variational parameter α; the Pade-Jastrow factor (2.39) has one variational parameter β. The Jastrow factor (2.38), on the other hand, has $N(N-1)/2$ variational parameters β_{ij}. Let us derive GD update rules for all these ansatzes.

After performing VMC, we obtain a VMC estimate of the expectation value of the local energy (2.31):

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

The analytical expression for the variational energy (2.28) in bra-ket form is:

$$E_T = \langle E_L \rangle = \frac{\langle \Psi_T(\alpha) | \hat{H} | \Psi_T(\alpha) \rangle}{\langle \Psi_T(\alpha) | \Psi_T(\alpha) \rangle}.$$

Trial wave functions contain variational parameters, and they are used to compute the local energy. Therefore, the local energy is also a function of the variational parameters. The GD Algorithm 2 requires the gradient vector of the function being optimized. In our case, this function is the expectation value of the local energy. That is, to optimize

the 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 (6.28):

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

Although the derivation presented here is original, this result is well known and can be obtained via alternative methods. For example, Lüchow [39] derives the same formula using both Newton's method and the linear method.

Here,

$$\bar{\Psi}_T = \frac{\partial\Psi_T}{\partial\alpha}. \tag{3.15}$$

The 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, the step size $t^{(k)}$ can be chosen as a constant value that is tuned. Typical values for t range from 10^{-4} to 10^{-1}.

To optimize α, we need to be able to compute equation (3.14) for a given trial wave function. This expression contains three different expectation values. The only familiar term is the expectation value of the local energy, which represents the VMC estimate of the ground-state (GS) energy. To compute it, we sample the local energies during each VMC iteration. The other two terms can be obtained in exactly the same way: we need to sample and collect

$$O_1 = \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)} \quad\text{and}\quad O_2 = \frac{\bar{\Psi}_T(\alpha)}{\Psi_T(\alpha)}E_L(\alpha) \tag{3.16}$$

during each VMC iteration. Variables O_1 and O_2 are referred to as observables or scores. We will discuss how to compute the local energy and scores for all model systems in Section 3.7. Now we focus on deriving general GD update rules for all variational parameters.

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

$$\frac{\partial\langle E_L\rangle}{\partial\beta} = 2\left(\left\langle\frac{\bar{\Psi}_T(\alpha,\beta)}{\Psi_T(\alpha,\beta)}E_L(\beta)\right\rangle - \left\langle\frac{\bar{\Psi}_T(\alpha,\beta)}{\Psi_T(\alpha,\beta)}\right\rangle\langle E_L(\beta)\rangle\right). \tag{3.17}$$

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

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

Finally, for a trial wave function with a Jastrow factor, we need to optimize $p = N(N-1)/2$ variational parameters β_{ij}. These variational parameters can be represented by the vector $\beta = (\beta_0, \ldots, \beta_{p-1})$. We use 0 as the starting index instead of 1, because in `C++` the first element of a vector has index zero. To store beta values in

the vector in the correct order, we need to transform the 2D index (i, j) into a 1D index k. Since j is always larger than i, this can be done by using

$$k = \frac{i}{2}(2N - i - 1) + (j - i - 1). \tag{3.18}$$

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

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

Compute the 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 (3.18) correctly translates the 2D index into the 1D index.

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

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

Here, β is the vector of p variational parameters, and

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

The derivative of the expectation value of the local energy is given by expression (3.17):

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

where

$$\bar{\Psi}_T^{(m)}(\alpha, \beta) = \frac{\partial \Psi_T(\alpha, \beta)}{\partial \beta_m}. \tag{3.19}$$

To summarize, GD provides an easy way to minimize the trial ground-state energy by optimizing variational parameters. To optimize the variational parameters, E_L, O_1, and O_2 need to be sampled during VMC.

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

3.6 Metropolis–Hastings Algorithm

This section follows the lecture notes of Morten Hjorth-Jensen[40] 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 obtain the update rule for a small time step Δt:

$$x_{\text{new}} = x_{\text{old}} + D\,F(x)\,\Delta t + \eta. \tag{3.20}$$

Here, x_{new} is the new position and x_{old} is the previous position. The constant D equals $1/2$ in atomic units, η is a Gaussian 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

$$\mathbf{F} \;=\; 2\,\frac{\nabla\Psi_T}{\Psi_T}. \tag{3.21}$$

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

$$\alpha = \min\left(1,\; \frac{G(x,y,\Delta t)\,|\Psi_T(y)|^2}{G(y,x,\Delta t)\,|\Psi_T(x)|^2}\right), \tag{3.22}$$

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

$$G(y,x,\Delta t) = C\exp\left(\frac{-(y-x-D\,\Delta t\,F(x))^2}{4D\,\Delta t}\right). \tag{3.23}$$

In his article,[13] Hastings introduced importance sampling for Monte Carlo methods in 1970.

Recall that in Section 2.3.5, the acceptance probability involves the proposal densities $q(X|Y)$ and $q(Y|X)$. In our harmonic-oscillator systems, we take the corresponding Green's functions to serve as these proposal distributions.

From equation (3.22), we see that only the ratio of Green's functions is needed to compute the acceptance probability. The natural logarithm of the Green's function ratio is given by

$$\ln\frac{G_{\text{new}}}{G_{\text{old}}} = \frac{\left(\mathbf{R}_{\text{new}} - \mathbf{R}_{\text{old}} - D\,\mathbf{F}_{\text{old}}\,\Delta t\right)^2 \;-\; \left(\mathbf{R}_{\text{old}} - \mathbf{R}_{\text{new}} - D\,\mathbf{F}_{\text{new}}\,\Delta t\right)^2}{4D\,\Delta t}. \tag{3.24}$$

3.7 Local energy, Observables and Quantum Force For Model Systems

In this section, we will provide a recipe to compute local energy E_L, observables O_1, O_2, and quantum force $\mathbf{F}$ during VMC iterations. We will also find analytical solutions 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 a harmonic potential and cooled so that all particles occupy the lowest energy state. In HO units (or atomic units with $\omega = 1$), the Hamiltonian for this system is given by equation (3.3):

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

The trial wave function ansatz has one variational parameter α and is given by (3.6):

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

For such a system, the GS energy and the optimal variational parameter $\tilde{\alpha}$ can be found analytically; see Appendix 6.4 for the derivation. Although derived independently here via direct evaluation of the variational integral using Gaussian integrals, this result is not new and has been obtained previously by others using the same and different approaches.

The variational energy of the system as a function of the variational parameter is (6.32):

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

the optimal variational parameter is (6.33):

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

and the GS energy (6.34) is

$$E_0(\tilde{\alpha}) = \frac{3N}{2}.$$

This result can be generalized for a d-dimensional system:

$$\begin{aligned} E_T(\alpha) &= \frac{dN}{2}\left(\alpha + \frac{1}{4\alpha} \right), \\ \tilde{\alpha} &= \frac{1}{2}, \\ E_0(\tilde{\alpha}) &= \frac{dN}{2}. \end{aligned} \tag{3.25}$$

Recall that the wave function that describes a 1D quantum particle in a harmonic potential is given by (2.17). For $n = 0$ in HO units, it is proportional to

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

Therefore, the 3D wave function is proportional to

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

By finding $\tilde{\alpha} = 1/2$, we have reconstructed the original wave function from which the trial wave function Ψ_T is composed.

This problem can be solved analytically due to the relative simplicity of the trial wave function and Hamiltonian of the system. The analogous fermionic problem has a

trial wave function of more complex form, and analytical solutions can be found only for very simple cases like $N = 2$.

In Appendix 6.4 we also show that the local energy for the 3D system (6.29) can be computed as

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

In d dimensions it can be generalized as

$$E_L = \frac{\hat{H}\Psi_T}{\Psi_T} = \alpha N d + \left(\frac{1}{2} - 2\alpha^2\right)\sum_{i=1}^{N} r_i^2. \tag{3.26}$$

The score O_1 is given by (6.35):

$$O_1 = \frac{\tilde{\Psi}_T}{\Psi_T} = -\sum_{i=1}^{N} r_i^2. \tag{3.27}$$

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

The expression for the quantum force can be obtained using the gradient of ψ_1 given by (6.9):

$$\mathbf{F}_i = 2\,\frac{\nabla_i \Psi_T}{\Psi_T} = \frac{-4\alpha\,\Psi_T\,(x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z)}{\Psi_T}, \tag{3.28}$$

therefore,

$$\mathbf{F}_i = -4\alpha\,(x_i, y_i, z_i). \tag{3.29}$$

This system is the simplest of the systems we study in this research. This simplicity stems primarily from the relative simplicity of the trial wave function.

3.7.2 Non-Interacting Fermions

As with non-interacting bosons, the Hamiltonian is given by (3.3):

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

The trial wave function for N fermions in two dimensions is a product of spin-up and spin-down Slater determinants (3.7):

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

For a system with a trial wave function of this form, analytical solutions exist for the first closed shell with two particles. For higher closed-shell configurations, as we will show, it is not possible to obtain a closed-form expression for the local energy—only formulas that must be evaluated numerically.

Let us first derive the general expression for the local energy:

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

The second term cannot be simplified further and can be easily computed numerically. On the other hand, the first term can be simplified. The spin-up SD is a function of $(\mathbf{r}_1, \ldots, \mathbf{r}_{N/2})$ and the spin-down SD is a function of $(\mathbf{r}_{N/2+1}, \ldots, \mathbf{r}_N)$. Therefore, the sum can be split into two terms:

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

The only remaining task is to understand how to compute the Laplacian of the spin-up and spin-down Slater determinants. Let us consider the spin-up SD, which is given by (3.8):

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

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

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

Here, $\nabla_j D_\uparrow$ 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 the particle coordinates they use, taking the gradient with respect to the j-th spin-down particle is done exactly as for spin-up: we form the usual Slater determinant and, in its j-th row, replace each orbital by its gradient.

The Laplacian of the spin-up and spin-down Slater determinants can be obtained in a similar way: the j-th row wave functions $\psi_i(\mathbf{r}_j)$ are replaced by their respective Laplacians $\Delta_j \psi_i(\mathbf{r}_j)$. For example,

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

We can now compute the local energy numerically using

$$E_L = -\frac{1}{2} \sum_{i=1}^{N/2} \frac{\Delta_i D_\uparrow}{D_\uparrow} - \frac{1}{2} \sum_{i=N/2+1}^{N} \frac{\Delta_i D_\downarrow}{D_\downarrow} + \frac{1}{2} \sum_{i=1}^{N} r_i^2. \tag{3.30}$$

The observable O_1 is given by (3.16) and (3.15):

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

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

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

We can use the expression for the derivative from §2.1.1 [41]:

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

The observable then simplifies to

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

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

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

so that the final expression is

$$O_1 = \sum_{i=1}^{N} -r_i^2. \tag{3.31}$$

A general expression for the quantum force can be obtained relatively easily:

$$\mathbf{F}_i = \begin{cases} 2\frac{\nabla_i D_\uparrow}{D_\uparrow}, & \text{if } i \le N/2, \\ 2\frac{\nabla_i D_\downarrow}{D_\downarrow}, & \text{if } i > N/2. \end{cases} \tag{3.32}$$

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

First Closed Shell

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

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

This solution follows the general solution (3.25) with $d = 2$ and $N = 2$.

The analytical expression for the local energy is given by (6.36):

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

Score O_1 can be computed from (3.31):

$$O_1 = -(r_1^2 + r_2^2),$$

and the quantum force (3.32) is

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

Second Closed Shells

For this system, spin-up and spin-down Slater determinants (3.8), (3.9) are

$$D_\uparrow(\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3) = \begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_2(\mathbf{r}_1) & \psi_3(\mathbf{r}_1) \\ \psi_1(\mathbf{r}_2) & \psi_2(\mathbf{r}_2) & \psi_3(\mathbf{r}_2) \\ \psi_1(\mathbf{r}_3) & \psi_2(\mathbf{r}_3) & \psi_3(\mathbf{r}_3) \end{vmatrix}, \tag{3.33}$$

$$D_\downarrow(\mathbf{r}_4,\mathbf{r}_5,\mathbf{r}_6) = \begin{vmatrix} \psi_1(\mathbf{r}_4) & \psi_2(\mathbf{r}_4) & \psi_3(\mathbf{r}_4) \\ \psi_1(\mathbf{r}_5) & \psi_2(\mathbf{r}_5) & \psi_3(\mathbf{r}_5) \\ \psi_1(\mathbf{r}_6) & \psi_2(\mathbf{r}_6) & \psi_3(\mathbf{r}_6) \end{vmatrix}. \tag{3.34}$$

Here, the spin orbitals are given by (6.7), (6.12), and (6.13):

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

The ansatz (3.7) for this system is

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

The local energy is computed using (3.30):

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

The Laplacians of the spin orbitals (6.10), (6.16), and (6.17) are

$$\begin{aligned} \Delta\psi_1(x,y) &= (-4\alpha + 4\alpha^2 r^2)e^{-\alpha r^2}, \\ \Delta\psi_2(x,y) &= x(-8\alpha + 4\alpha^2 r^2)e^{-\alpha r^2}, \\ \Delta\psi_3(x,y) &= y(-8\alpha + 4\alpha^2 r^2)e^{-\alpha r^2}. \end{aligned}$$

The observable O_1 is computed using (3.31):

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

The quantum force $\mathbf{F}_i$ is computed numerically using the general expression (3.32).

Third Closed Shells

For a system with twelve fermions, the components of the spin-sector Slater determinants are given by (3.10) and (3.11):

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

where $i,j = 1,2,3,4,5,6$.

Spin orbitals 1–3 are the same as for the six-particle case, while spin orbitals 4–6 are given by (6.18), (6.19), and (6.20):

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

The local energy is computed using (3.30):

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

The Laplacians of spin orbitals 1–3 are the same as before, and the Laplacians of spin orbitals 4–6 are given by (6.24), (6.25), and (6.26):

$$\Delta\psi_4(x,y) = (2 - 12\alpha x^2 + 4\alpha^2 x^2 r^2 + 4\alpha - 4\alpha^2 r^2)e^{-\alpha r^2},$$
$$\Delta\psi_5(x,y) = (2 - 12\alpha y^2 + 4\alpha^2 y^2 r^2 + 4\alpha - 4\alpha^2 r^2)e^{-\alpha r^2},$$
$$\Delta\psi_6(x,y) = (-12\alpha\, x\, y + 4\alpha^2 x\, y\, r^2)e^{-\alpha r^2}.$$

The score O_1 is computed using (3.31):

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

and the quantum force is computed numerically using the general expression (3.32).

3.7.3 Interacting Fermions

To include interactions, the Hamiltonian is modified to include the Coulomb potential. In atomic units with $\omega = 1$, it is given by expression (2.35):

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

To satisfy Kato's cusp condition and account for electron correlations, the trial wave function is multiplied by the Jastrow factor J or the Pade-Jastrow factor P, yielding the ansatzes in (3.13):

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

where J and P are given by (2.38) and (2.39).

Including Jastrow factors of different forms in the wave function ansatz makes the computation of the local energy more complex. As discussed earlier, the optimal variational parameter α is obtained by solving the non-interacting problem. Therefore, for the interacting case, the goal is to optimize the variational parameters in the Jastrow or Pade-Jastrow factors.

In Appendix 6.5, we derive analytical expressions for the local energy E_L and the scores $O_1^{(J)}$ and $O_1^{(P)}$. As before, we present these expressions in atomic units with $\omega = 1$, since the ω-dependence can be easily restored.

The local energy is derived from the general expression (2.30):

$$E_L = \frac{\hat{H}\Psi_T}{\Psi_T} = \frac{(\hat{H}_0 + \hat{H}_I + \hat{V})\Psi_T}{\Psi_T}.$$

It can be written as the sum of five terms (6.46):

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

Let us go through each term, starting with $\hat{H}_I$. This term represents the electrostatic interaction between electrons, and it is the third term in (2.35):

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

This term retains its form because it is computed as

$$\Psi_T^{-1}\hat{H}_I\Psi_T.$$

Since $\hat{H}_I$ contains only the $1/r_{ij}$ interaction, its action on the ansatz does not change it, and the ansatz factors cancel out.

The $\hat{V}$ term represents the external HO potential, and it is given by the second term in (2.35):

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

This term is computed as

$$\Psi_T^{-1}\hat{V}\Psi_T.$$

Since $\hat{V}$ acts purely by multiplication, it does not alter the structure of the ansatz and thus retains its original form.

The terms $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ arise from the kinetic-energy operator acting on the trial wave function

$$\Psi_T = D_\uparrow D_\downarrow I,$$

where I denotes a correlation factor (either the linear Jastrow or the Pade-Jastrow) for generality. Because the ansatz is a product of the spin-up SD, spin-down SD, and the correlation factor, the Laplacian for each particle i acting on this ansatz gives three contributions:

$$\Delta_i\Psi_T = \Delta_i(D_\uparrow D_\downarrow I) = \Delta_i(D_\uparrow D_\downarrow) + 2\,\nabla_i(D_\uparrow D_\downarrow)\cdot\nabla_i(I) + \Delta_i(I).$$

Therefore, $\Psi_T^{-1}\hat{H}_0\Psi_T$ expands as

$$\Psi_T^{-1}\hat{H}_0\Psi_T = \sum_{i=1}^{N}\left[\underbrace{-\frac{1}{2}\Delta_i(D_\uparrow D_\downarrow)}_{=\mathcal{E}_1} - \underbrace{\nabla_i(D_\uparrow D_\downarrow)\cdot\nabla_i(I)}_{=\mathcal{E}_2} - \underbrace{\frac{1}{2}\Delta_i(I)}_{=\mathcal{E}_3}\right].$$

In Appendix 6.6, we show that $\mathcal{E}_1$ is given by (6.43):

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

This quantity represents the local energy of the non-interacting system, which we discussed how to compute in the previous section.

$\mathcal{E}_2$ is given by (6.47):

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

And $\mathcal{E}_3$ is given by (6.48):

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

The $\mathcal{E}_2$ term contains gradient vectors of the spin-up and spin-down Slater determinants. $\nabla_i D_\uparrow$ can be obtained by replacing the component wave functions $\psi_j(\mathbf{r}_i)$

with their respective gradients $\nabla_i \psi_j(\mathbf{r}_i)$. Expressions for the gradients of the wave functions 1–6 are derived in Appendix 6.1 and are given by:

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

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

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

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

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

and

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

By using the logarithmic form of the correlation factors, we never have to evaluate computationally costly exponential functions. Instead, we only compute the exponent's arguments, which is far cheaper computationally. Obviously, the same procedure can be applied to the spin-up and spin-down SDs, but we do not gain the same benefit because they are determinants constructed from exponential functions.

In the expressions for $\mathcal{E}_2$ and $\mathcal{E}_3$, the gradients and Laplacians of the Jastrow and Pade-Jastrow factors are given by (6.49), (6.50), (6.51), and (6.52), respectively:

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

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

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

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

The score $\mathbf{O}_1^{(J)}$ is a vector of dimensionality $p = N(N-1)/2$. It can be computed using (6.40):

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

where r_k represents the relative distance between particles i and j, and the 2D index is transformed to 1D using (3.18).
The score $O_1^{(P)}$ is computed using (6.41):

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

Finally, the quantum force (3.21) is computed as

$$\mathbf{F}_i = \begin{cases} 2\,\frac{\nabla_i D_\uparrow}{D_\uparrow} + 2\,\nabla_i \ln I, & \text{if } i \leq N/2, \\ 2\,\frac{\nabla_i D_\downarrow}{D_\downarrow} + 2\,\nabla_i \ln I, & \text{if } i > N/2. \end{cases}$$

3.8 Automatic Differentiation

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

The benefit of this approach is that we do not need to find and program analytical derivatives. The downside is that it is computationally expensive—more expensive than using analytical expressions. The best way to use automatic differentiation, when analytical expressions can be found, is to compare the analytical result with the result from automatic differentiation. This way, if the results align, we can be sure that the analytical expressions are implemented correctly.

In our implementation, we use the `C++` library `autodiff` [1]. This library allows us to compute derivatives numerically in two modes: forward and reverse. Forward mode computes both the function and its derivative at the same time, while reverse mode computes only the derivative. For our purposes, such as computing derivatives of single-particle orbitals, we choose to use reverse mode.

3.9 Parallelization

The VMC procedure can be very computationally expensive. Even for a small system with only twelve particles, VMC can run for hours or even days. To obtain a small relative error, we need to run at least 10^5 iterations and 10^4 burn-in iterations. This means that the computer needs to calculate 6×6 determinants more than 10^6 times. Normally, the program runs on a computer's processor. Modern processors have several cores. A serial program uses only one core of the processor, while the other cores remain idle. It is efficient to use as many processor cores as possible to perform computations simultaneously. Parallel programming is designed to distribute work between processor cores.

There are two types of systems: systems with shared memory and systems with distributed memory. As the name implies, in systems with shared memory, each core has access to the same memory while running the program. On the other hand, cores in distributed memory systems have their own separate memories, and the program needs to be written in a specific way to coordinate the cores' work properly.

`OpenMP` (`OMP`) [2] is an API that allows parallelization of programs running on shared memory systems, such as typical workstations. `MPI` [3] is an alternative API that allows parallelization of programs running on both distributed and shared memory systems. An example of a shared memory system is a supercomputer 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 the VMC simulation on all n cores simultaneously. Using this approach, each core computes N/n VMC iterations. Each core evolves its system independently using different random numbers while sampling local energy and other observables. After the cores complete their VMC iterations, they send their sampled values to the master core, which can then compute averages of observables, perform GD, etc. Communication between cores takes some time, but the speedup can be substantial.

G. Argentini provides a discussion of Amdahl's law, which offers a way to estimate parallelization speedup [42]. When fewer than eight cores are used and the program is parallelizable, the speedup is almost linear with the number of cores. This means that the parallel program can run almost eight times faster than the serial one, which is very impressive.

3.10 Computational Implementation

We have described all necessary details required to perform VMC estimation of the GS energy. In this section, we discuss how we implemented the VMC and GD algorithms numerically. We use `C++` as the main programming language to perform calculations, and `Python` to generate figures and analyze 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 [43], which provides a framework for VMC calculations. We have extended and redesigned that structure to introduce a more flexible, modular set of classes tailored to our needs. This expanded architecture is original to our approach and is one of the key factors that make this research unique.

The object-oriented architecture employed in this work 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 define additional subclasses. In the sections that follow, we describe the role and functionality of each class in turn.

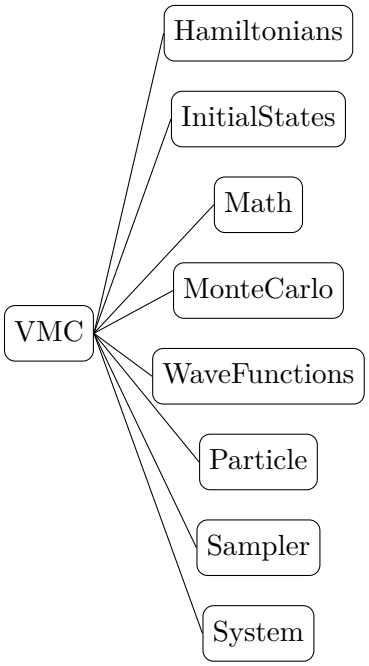


Figure 3.4: Class hierarchy of the `VMC` directory.

3.10.2 Math Class

The `Math` class is composed of a single file that contains definitions of the following functions:

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

In `C++`, a function is defined as

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

`Function_name` is the name of the function. This function has arguments `arg_1`, ..., `arg_N` of types `arg_1_type`, ..., `arg_N_type`. `return_type` is the type of the value returned by the function.

The function `nextInt` generates an integer in the range `[lowerLimit, upperLimit]`. `&lowerLimit` yields the memory address of `lowerLimit`. If you store that address in a pointer (for example, `int *p = &lowerLimit`), then dereferencing the pointer with `*p` gives you the value of `lowerLimit`. By passing the address `&lowerLimit` instead of the variable itself, we avoid copying the variable, which is more efficient.

Likewise, `nextDouble` generates a uniformly distributed random `double` in the range `[0, 1)`, and `nextGaussian` generates a normally distributed random `double` in the range `[mean - standardDeviation, mean + standardDeviation]`.

This class has a constructor that creates an object of the `Random` class—the random number generator (RNG):

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

If the constructor is called without an argument, it creates an RNG that generates a new random sequence each time. If it is called with `seed` as its argument, the 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 the class's objects.

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

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

From this point forward, we will use the shorthand notation `Dvec` to refer to `std::vector<double>`.

The constructor of the `Particle` class

```
Particle(const Dvec& position)
```

takes a constant reference to `Dvec position` (where each entry represents one Cartesian coordinate) and sets the private variable `m_position` equal to its argument `position`.

This class implements a single function, which shifts the d-th coordinate by `change` (see Algorithm 4).

Algorithm 4 Adjust Position Function

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

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

- `Dvec& getPosition()` returns the position of the particle `m_position`.
- `int getNumberOfDimensions()` returns the number of dimensions of the particle `m_numberOfDimensions`.

3.10.4 Initial States Class

The class `InitialStates` has only one function:

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

This function has three arguments: the number of dimensions `d`, the number of particles `N`, and a reference to the RNG `R`. For clarity, we use the shorthand notations `d`, `N`, and `R`. We also use the shorthand `Pvec` to denote `std::vector<std::unique_ptr<Particle»`.

Algorithm 5 shows how this function is implemented in pseudocode. When we write `R.nextDouble()`, we use dot notation because `nextDouble()` is a member function of the `Random` class.

Algorithm 5 Set Up Random Uniform Initial State Function

```
1: procedure SETUPRANDOMUNIFORMINITIALSTATE(d, N, R)
2:     Create empty Pvec particles, whose elements are Particle objects;
3:     Create empty Dvec pos of size d;
4:     for int i = 0; i < N; i++ do
5:         for int j = 0; j < d; j++ do
6:             pos[j] = R.nextDouble();
7:         end for
8:         Create part using the Particle constructor:

                auto part = std::make_unique<Particle>(pos);

9:         particles.push_back(part);
10:    end for
11:    return particles.
12: end procedure
```

First, this function generates `d` uniform random numbers and stores them in the vector `pos`. Then it creates a `Particle` object `part`. Finally, it appends `part` to the

end of the vector `particles`. This process is repeated `N` times, once for each particle. At the end, we obtain a vector of size `N` containing `Particle` objects. This `particles` vector is then passed to other functions that require the particles' positions. The position and number of dimensions of each particle can be accessed using `getPosition()` and `getNumberOfDimensions()` from the `Particle` class. In essence, the vector `particles` provides an object-oriented way to store the position of every particle.

3.10.5 Wave Function Class

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

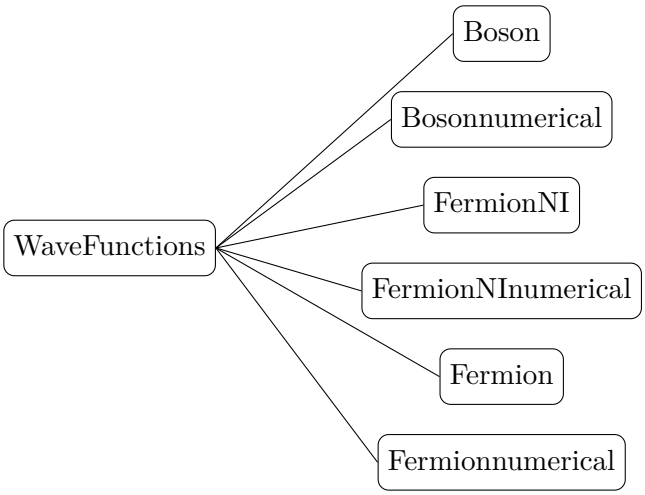


Figure 3.5: `WaveFunctions` class hierarchy.

In Figure 3.5, we present the structure of the `WaveFunctions` class. This class has six subclasses: `Boson`, `Bosonnumerical`, `FermionNI`, `FermionNInumerical`, `Fermion`, and `Fermionnumerical`.

`WaveFunctions` has five private variables:

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

The first and third variables represent the number of variational parameters and the number of particles, respectively. `m_parameters` is a vector containing all variational parameters. `int m_mode` is zero if the linear Jastrow factor is used and one if the Pade-Jastrow factor is used. And `double m_omega` is a private frequency ω.

This class implements seven functions:

- `double r_squared(Pvec& particles, int i)` computes the d-dimensional r^2.
- `double r_ij(Pvec& particles, int i, int j)` computes the relative distance r_{ij}.

- `int BetaIndex(int i, int j)` computes the 1D beta index according to (3.18).
- `double a_ij(int i, int j)` computes the constant a for particles i and j according to (2.27).
- `double GreensFunctionRatio(Dvec& R_new, Dvec& R_old, double dt, Dvec& F_new, Dvec& F_old)` computes the ratio of Green's functions needed for the Metropolis–Hastings algorithm according to (3.24). Every argument of this function is a d-dimensional vector `Dvec`, except for `dt`.
- `int getNumberOfParameters()` returns `m_numberOfParameters`.
- `Dvec getParameters()` returns the `m_parameters` vector.

Each of these functions is inherited by all `WaveFunctions` subclasses.

In addition, `WaveFunctions` declares three virtual functions:

- `virtual double evaluate(Pvec& particles)` computes the trial wave function.
- `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 the quantum force for particle `i`.

These functions are declared in `WaveFunctions` but not defined. Each subclass provides its own definition of these three functions.

In summary, each `WaveFunctions` subclass implements `evaluate` and `computeDoubleDerivative` for its respective trial wave function (and `quantumForce` when Metropolis–Hastings is used).

Boson Class

The `Boson` subclass is designed to compute trial wave function for `N` bosons in `d` dimensions, as given by (3.6):

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

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 the `WaveFunctions` class, meaning that objects of the `Boson` class inherit all functions and private variables of `WaveFunctions`.

The implementation of `evaluate(Pvec &particles)` in this class is shown in Algorithm 6. This algorithm computes the trial wave function relatively easily compared to the other `WaveFunctions` subclasses.

Algorithm 6 Evaluate Function For `Boson` Class

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

The implementation of `computeDoubleDerivative(Pvec &particles)` is given in Algorithm 7.

Algorithm 7 Compute Double Derivative Function For `Boson` Class

```
1: procedure COMPUTEDOUBLEDERIVATIVE(Pvec &particles)
2:     int N = m_particles;
3:     double alpha = m_parameters[0];
4:     double r2 = 0.0;
5:     Particle p = particles[0];
6:     int d = p.getNumberOfDimensions();
7:     for int i = 0; i < N; i++ do
8:         r2 += r_squared(particles, i);
9:     end for
10:    return -2 * N * d * alpha * m_omega + 4 * m_omega^2 * alpha^2 * r2.
11: end procedure
```

The final function that this subclass implements is `quantumForce`, based on formula (3.29). It is presented in Algorithm 8. Next, we discuss the similar class `Bosonnumerical`, which implements the same formulas using automatic differentiation.

Algorithm 8 Quantum Force For `Boson` Class

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

Boson Numerical Class

This class is designed to compute the same quantities as the `Boson` class, but using automatic differentiation. All "numerical" classes implement an additional function:

`VectorXvar fill_x(Pvec &particles).`

This function constructs a one-dimensional vector and uses it for automatic differentiation instead of `Pvec particles`. The `autodiff` library requires `VectorXvar` and `var` types for function arguments to compute numerical derivatives. The implementation of `fill_x` is shown in Algorithm 9. From now on, we use the shorthand `Vvec` for `VectorXvar`. Elements of `Vvec` are accessed via parentheses, not square brackets: `x(0)`. The vector `x` generated by this function has elements

$$\mathtt{x} = (x_0, y_0, x_1, y_1, \ldots, x_{N-1}, y_{N-1})$$

for a two-dimensional system, which is our primary interest.

Algorithm 9 Fill `x` Function

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

The `computeDoubleDerivative` function for this class is given in Algorithm 10. It uses the helper function `LaplPsiTOverPsiT(Vvec &x)` to compute the sum of Laplacians. The call `var psiT = PsiT(x)` computes the trial wave function for a given `Vvec x` and returns a `var` so that it can be differentiated numerically. The function `hessian(psiT, x, g)` computes the Hessian matrix, whose diagonal entries are the second derivatives of the ansatz with respect to each variable. The call `val(y)` returns the `double` value of a `var y`.

The implementation of `quantumForce` for this class is shown in Algorithm 11. It uses the `derivatives` function to compute first derivatives.

We omit discussion of the `FermionNI` and `FermionNInumerical` classes, since they are simplified versions of `Fermion` and `Fermionnumerical`. We now proceed to describe the `Fermion` class.

Fermion Class

This class is designed to compute trial wave function and sum of its 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`).`

Algorithm 10 Compute Double Derivative For `BosonNumerical` Class

```
1: procedure LaplPsiTOverPsiT(Vvec &x)
2:     var psiT = PsiT(x);
3:     Initialize Eigen::VectorXd g;
4:     Eigen::MatrixXd H = hessian(psiT, x, g);
5:     Eigen::VectorXd LaplVector = H.diagonal();
6:     double sum = val(LaplVector.sum());
7:     return sum / val(psiT);
8: end procedure
9: procedure computeDoubleDerivative(Pvec &particles)
10:    Vvec x = fill_x(particles);
11:    return LaplPsiTOverPsiT(x);
12: end procedure
```

Algorithm 11 Quantum Force For `BosonNumerical` Class

```
1: procedure quantumForce(Pvec &particles, int i)
2:     Vvec x = fill_x(particles);
3:     var psiT = PsiT(x);
4:     int d = x.size() / m_particles;
5:     Initialize Dvec F of size d with zeros;
6:     for int j = 0; j < d; j++ do
7:         var gradx = derivatives(psiT, wrt(x(d × i + j)));
8:         F[j] = 2 * val(gradx) / val(psiT);
9:     end for
10:    return F;
11: end procedure
```

This constructor takes two new arguments – vector with initial 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 filled with beta parameters first, and then with alpha.

`Fermion` class contains functions `Psi`j`(Pvec & particles, int idx)`, $j = 1, ..., 6$ that compute single-particle wave function. Functions `GradiPsi`j`(Pvec & particles, int idx)` compute gradients and `LapliPsi1`j`(Pvec & particles, int idx)` compute Laplacians of single-particle wave functions. `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.

Let us discuss arguments of this function:

- `Pvec &particles` is reference to vector of particles;
- If `int spin == 0` function computes spin-up SD, and if `int spin == 1`, function computes spin-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 orbitals are replaced with respective gradients or Laplacians;
- `int der` is order of derivative. 0 - normal SD, 1 - gradient, 2 - Laplacian;

- Gradient of SD is a vector with two components. `int comp` determines which of gradient components function computes.

We show a structure of `evaluate` and `computeDoubleDerivative` functions in Algorithm 13.

`Fermion` class contains more then thirty different functions. All these function are written just to compute trial wave function and sum of its 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 $\hat{H}\Psi_T$.

Algorithm 12 Slater Determinant Function

```
1: procedure SD(Pvec &particles, int spin, int row, int der, int comp)
2:     int size = m_particles/2;
3:     double det = 0.0;
4:     Initialize MatrixXd A of size size×size with zeros;
5:     if m_particles == 6 then
6:         for (int i = 0; i < size; i++) do
7:             if i == row then
8:                 if der == 0 then
9:                     A(i, 0) = Psi1(particles, i + size · spin);
10:                    A(i, 1) = Psi2(particles, i + size · spin);
11:                    A(i, 2) = Psi3(particles, i + size · spin);
12:                else if der == 1 then
13:                    A(i, 0) = GradiPsi1(particles, i + size · spin)[comp];
14:                    A(i, 1) = GradiPsi2(particles, i + size · spin)[comp];
15:                    A(i, 2) = GradiPsi3(particles, i + size · spin)[comp];
16:                else if der == 2 then
17:                    A(i, 0) = LapliPsi1(particles, i + size · spin);
18:                    A(i, 1) = LapliPsi2(particles, i + size · spin);
19:                    A(i, 2) = LapliPsi3(particles, i + size · spin);
20:                end if
21:            else
22:                A(i, 0) = Psi1(particles, i + size · spin);
23:                A(i, 1) = Psi2(particles, i + size · spin);
24:                A(i, 2) = Psi3(particles, i + size · spin);
25:            end if
26:        end for
27:        det = A.determinant();
28:    else if m_particles == 2 then
29:        ⋮
30:        det = A.determinant();
31:    else if m_particles == 12 then
32:        ⋮
33:        det = A.determinant();
34:    end if
35:    return det.
36: end procedure
```

Algorithm 13 Evaluate and Compute Double Derivative Functions For Fermion Class

```
1: procedure EVALUATE(Pvec &particles)
2:     double J = (m_mode == 0) ? Jastrow(particles) : PadeJastrow(particles);
3:     return SD(particles, 0, 0, 0, 0) · SD(particles, 1, 0, 0, 0) · J;
4: end procedure
5: procedure COMPUTEDOUBLEDERIVATIVE(Pvec &particles)
6:     double Lapl = 0.0;
7:     double GradGrad 0.0;
8:     if m_mode == 0 then
9:         for int i = 0; i < m_particles; i++ do
10:            Lapl += LapliJOverJ(particles, i);
11:            GradGrad += GradiPsiTGradiJOverPsiT(particles, i);
12:        end for
13:    else if m_mode == 1 then
14:        for int i = 0; i < m_particles; i++ do
15:            Lapl += LapliPJOverPJ(particles, i);
16:            GradGrad += GradiPsiTGradiJOverPsiT(particles, i);
17:        end for
18:    end if
19:    return LaplPsiTOverPsiT(particles) + Lapl + GradGrad
20: end procedure
```

3.10.6 Hamiltonians Class

We designed the `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 the Hamiltonian and to switch terms on and off. To solve our problem, we have created the class `HarmonicOscillator`, a subclass of the `Hamiltonians` class.

The `HarmonicOscillator` class has two private variables:

`double m_omega` and `bool m_Coulomb`.

The first variable represents the oscillator frequency of the trap, ω, and `bool m_Coulomb` is used to switch between interacting and non-interacting Hamiltonians. The constructor used to create an object of the `HarmonicOscillator` class is

`HarmonicOscillator::HarmonicOscillator(double omega, bool Coulomb)`,

and it simply sets `m_omega = omega` and `m_Coulomb = Coulomb`.

`HarmonicOscillator`'s implementation of the `computeLocalEnergy` function is presented in Algorithm 14. It is quite simple because all heavy calculations are hidden inside the `computeDoubleDerivative` function.

Now we are ready to proceed with presenting the `Solvers` class, which implements Metropolis VMC iteration and Metropolis–Hastings VMC iteration.

3.10.7 Monte Carlo Class

The `MonteCarlo` class is a general class that has two subclasses: `Metropolis` and `MetropolisHastings`. This structure was chosen to allow easy switching between these

Algorithm 14 Compute Local Energy Function

```
1: procedure computeLocalEnergy(class WaveFunction &waveFunction, Pvec
   &particles)
2:     int N = particles.size();
3:     double r2 = 0.0;
4:     for int i = 0; i < N; i++ do
5:         r2 += waveFunction.r_squared(particles, i);
6:     end for
7:     double T = 0.5 · r2 · m_omega^2;
8:     double V = -0.5 · waveFunction.computeDoubleDerivative(particles);
9:     double E = T + V;
10:    if m_Coulomb == 1 then
11:        double C = 0.0;
12:        for int i = 0; i < N; i++ do
13:            for int j = i; j < N; j++ do
14:                C += waveFunction.r_ij(particles, i, j);
15:            end for
16:        end for
17:        E += C;
18:    end if
19:    return E.
20: end procedure
```

two algorithms. The `MonteCarlo` class has one private variable, `m_rng`, which is an object of the `Random` class. The constructor of this class,

`MonteCarlo(std::unique_ptr<class Random> rng)`,

simply sets `m_rng = rng`. This class declares only one virtual function:

`bool step(double sl, class WaveFunction &waveFunction, Pvec &particles)`.

This function is implemented by the `Metropolis` and `MetropolisHastings` subclasses, and it performs one VMC iteration (step).

Metropolis Class

The `Metropolis` class is designed to implement the `step` function, which performs one VMC iteration using the Metropolis sampling algorithm. In Algorithm 15, we schematically explain how this function works.

Essentially, this function represents our implementation of the general Algorithm 1 with proposal densities $q(X|Y)$ and $q(Y|X)$ both equal to one. To compute the new-state wave function, we first move all particles. If the step is rejected, we move all particles back to their previous positions. To restore the particles exactly, we save all uniform random numbers in the `rand` vector and then use those values to reverse the moves. If the step is accepted, this function returns `true`; if rejected, it returns `false`. The acceptance ratio is a very helpful metric, as it allows us to determine whether the chosen step size and learning rate (for GD) are appropriate. The choice is poor if the acceptance rate is close to zero or one. We will see that, as the systems become larger, both the learning rate and the step size need to be chosen smaller.

Algorithm 15 Metropolis Class Step Function

```
1: procedure STEP(sl, &waveFunction, &particles)
2:     int N = particles.size();
3:     int d = particles[0].getNumberOfDimensions();
4:     Initialize double PsiOld = 0.0 and double PsiNew = 0.0;
5:     Initialize 2D vector rand of size N × d;
6:     Compute the old-state ansatz: PsiOld = waveFunction.evaluate(particles);
7:     for int i = 0; i < N; i++ do
8:         for int j = i; j < d; j++ do
9:             rand[i][j] = m_rng.nextDouble();
10:            particles[i].adjustPosition(sl·(rand[i][j] - 0.5), j);
11:        end for
12:    end for
13:    Compute the new-state ansatz: PsiNew = waveFunction.evaluate(particles);
14:    Compute the acceptance ratio: alpha = min(1, PsiNew^2/PsiOld^2);
15:    if m_rng.nextDouble() > alpha then
16:        for int i = 0; i < N; i++ do
17:            for int j = i; j < d; j++ do
18:                particles[i].adjustPosition(-sl·(rand[i][j] - 0.5), j);
19:            end for
20:        end for
21:        return false;
22:    else
23:        return true;
24:    end if
25: end procedure
```

Metropolis Hastings Class

The `MetropolisHastings` class is similar to the `Metropolis` class in that it implements the same function. However, it is designed according to the Metropolis–Hastings algorithm. Algorithm 16 schematically describes the Metropolis–Hastings algorithm. Here, the approach is slightly different. We move one particle at a time, then compute the quantum force and the new wave function for the updated positions. Afterwards, we compute the Green's function ratio and accept or reject the step for that particle before repeating for all particles. In this case, the overall acceptance rate is not very informative, since it will be almost always close to one. However, it can still be used to assess the step size and learning rate.

3.10.8 Sampler Class

This class is responsible for computing and storing all observables. Because of this, the `Sampler` class has many private variables, some of which are:

```
int m_nPairs, int m_energy, int m_cumulativeEnergy, Dvec m_O1Jastow
```

and many others. The constructor of this class has the following structure:

```
Sampler(int N, int d, double sl, int n)
{
  m_N = N, m_n = n, m_d = d, m_sl = sl;
  m_nPairs = m_N(m_N - 1)/2;
}.
```

All other variables are initialized to zero or zero vectors of the appropriate size.

This class has three functions:

- `sample(bool acceptedStep, System* system)` is responsible for computing all sampled values. All sampled values are added to the respective `m_cumulative` private variables;
- `printOutputToTerminal(System& system)` is a helper function that outputs information about the system to the terminal;
- `computeAverages()` computes mean values of the `m_cumulative` variables.

In Algorithm 17, we outline the structure of the `sample` function. Basically, it just samples all observables: local energy, O_1, and O_2 for α, β, and β_{ij} optimization. In this algorithm, we use the shorthand `m_c` for `m_cumulative` for all cumulative variables.

The function `computeAverages()` simply divides all cumulative variables by the number of VMC iterations, n.

Additionally, the `Sampler` class implements simple functions that return observables. For example, the function `getEnergy()` returns `m_energy`, and `getO1Jastrow()` returns the 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 the VMC algorithm. This structure is very flexible. We developed six subclasses of `WaveFunction`, two subclasses of `MonteCarlo`, and `Hamiltonian` has two modes.

The constructor of this class has the following structure:

```
System(Hamiltonian *H, WaveFunction *WF, MonteCarlo *MC, *Pvec parts)
{
  m_N = parts.size();
  m_d = parts[0].getNumberOfDimensions();
  m_H = move(H);
  m_WF = move(WF);
  m_MC = move(MC);
  m_particles = move(parts);
}
```

All arguments of the constructor have type `unique_ptr<class >`. The method `move()` copies the object from memory to the member variable and then deletes the original. As before, variables that start with `m_` are private members of the class.

The `System` class implements the `runEquilibrationSteps` function and the `runMetropolisSteps` function. The entire VMC calculation is hidden inside these simple functions. In Algorithm 18, we present these functions. Essentially, the `runEquilibrationSteps` function runs the first m VMC iterations without sampling and returns the number of accepted steps. The `runMetropolisSteps` function runs n VMC iterations and samples all observables. It creates and then returns a `sampler`, which can be used in the main function to retrieve all needed observables.

We have presented the VMC solver's class structure and discussed all of its components. In the following sections, we will show both the serial and parallel main programs that perform the VMC and GD calculations.

3.10.10 Non-Interacting Serial VMC

Now we are ready to discuss how to perform a VMC calculation using the presented class structure. We begin our discussion with non-interacting systems, i.e. systems with one variational parameter α. The first step is to initialize the 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 number of parameters needed to run the simplest VMC calculation. In Algorithm 19, 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 an object of the respective class. Afterward, we move all objects using the `move` method. In this way, we avoid copying objects and simply transfer their memory locations to set up the `System` object. Then we start the timer and run the burn-in and VMC iterations. Finally, we output information about the system and the ground state energy to the terminal. The class structure is complicated, but now it pays off: we can change the ansatz, solver, and Hamiltonian just by changing the `H`, `WF`, and `solver` objects. In the presented algorithm, we chose a non-interacting Hamiltonian, a bosonic ansatz, and the Metropolis solver. If we want to switch to an interacting Hamiltonian, we set `H` $\leftarrow$ `make_unique<HarmonicOscillator>(1,1)`. Recall that the second argument of the `HarmonicOscillator` constructor is `bool Coulomb`. To change the wave function, we set `WF` $\leftarrow$ `make_unique<Class>(`α`,` N`)`, where `Class` represents one of six `WaveFunction` subclasses. However, we need to be careful, because interacting ansätze require two additional arguments. Finally, we can change the solver by setting `solver` $\leftarrow$ `make_unique<MetropolisHastings>(move(rng))`.

The ability to change a "block" of the VMC so easily is the reason we implemented this class structure. If we want to add an additional ansatz, solver, or Hamiltonian, we just need to implement the respective subclass and use it to create the `System`, and perform VMC using it.

It is very easy to parallelize this VMC algorithm but quite tricky to parallelize the GD VMC algorithm. Therefore, we will first present the serial GD VMC algorithm and then discuss how to parallelize it.

3.10.11 Non-Interacting Serial GD VMC

The VMC algorithm computes the variational energy for a given variational parameter α. We aim to find the optimal variational parameter that minimizes the variational energy, so that it represents the ground-state energy. To optimize α, we use the GD algorithm shown in Algorithm 20. This algorithm has two additional arguments: the stopping criterion B and the constant learning rate η. As discussed earlier, the GD algorithm is quite simple: sample O_1, O_2, and E, and use them to find the optimal variational parameter. In this algorithm, we describe the optimization of α. Similar algorithms for Jastrow and Pade-Jastrow optimization will be discussed in the following section.

3.10.12 Interacting Serial GD VMC

The plain VMC algorithm for interacting systems is just Algorithm 19 with the `Fermion` or `FermionNumerical` class. In addition, we need to specify a `double` β for the Pade-Jastrow ansatz or a `Dvec` β_{ij} for the Jastrow ansatz. We also need to set `bool mode = 0` to use the Jastrow factor or `bool mode = 1` to use the Pade-Jastrow factor:

- `Fermion WF = make_unique<Fermion>(alpha, betaij, mode = 0, N)` sets up the interacting Jastrow fermionic ansatz;
- `Fermion WF = make_unique<Fermion>(alpha, beta, mode = 1, N)` sets up the interacting Pade-Jastrow fermionic ansatz;
- `FermionNumerical WF = make_unique<FN>(alpha, betaij, mode = 0, N)` sets up the numerical interacting Jastrow fermionic ansatz;
- `FermionNumerical WF = make_unique<FN>(alpha, beta, mode = 1, N)` sets up the numerical interacting Pade-Jastrow fermionic ansatz;
- `HarmonicOscillator H = make_unique<HarmonicOscillator>(omega, 1)` sets up the interacting HO Hamiltonian with oscillator frequency ω.

`VMC(N, d, n, m, s, sl, beta, omega, mode)` will denote Algorithm 19 with one of the ansätze described above and the interacting Hamiltonian. We assume that, for such a system, the optimal parameter α has already been found by solving the non-interacting GD VMC. The goal of the interacting GD VMC is to optimize the variational parameters β or β_{ij}. The GD VMC algorithm for the Jastrow ansatz is presented in Algorithm 21. As we can see, it is similar to the non-interacting GD VMC algorithm. The main difference is that we now update $p = N(N-1)/2$ variational parameters at a time, instead of just one. Another difference is that the L2 norm determines when to stop the GD algorithm. The Pade-Jastrow GD algorithm is similar to the non-interacting GD algorithm (Algorithm 20), but uses the Pade-Jastrow ansatz and is designed to minimize the single variational parameter β.

3.10.13 Interacting OMP GD VMC

In this section, we discuss how the GD VMC algorithm can be parallelized using OpenMP (OMP). OMP can be used only on shared-memory systems, such as typical desktops and laptops. OMP provides an easy way to parallelize code. Loops whose iterations are independent are the easiest to parallelize. This is exactly our case: we create `n_threads` copies of the system with different RNGs and let them run VMC in parallel. The benefit of this approach is that each system runs only `n/n_threads` VMC iterations instead of `n` iterations. Parallelized code must satisfy two requirements: it should run faster than the serial version and produce correct results. In Algorithm 22, we present the OMP-parallelized version of the GD VMC algorithm for the Jastrow ansatz. Inside the **while** loop, we create a parallel region using `#pragma omp parallel`. Inside this region, each thread executes the code in parallel. To ensure that each thread creates a different system, each thread gets its own seed `ts`. Threads use their seeds to create independent RNGs. Each thread then generates its own initial particle positions and constructs its own solver using its RNG instance. By doing so, we ensure that each thread generates different random numbers and that its system evolves independently. After threads complete the VMC computation, each thread collects its observables.

To obtain the mean value of observables over all threads, we compute

$$\langle E \rangle = \frac{\sum_{\texttt{tid}=0}^{\texttt{Nt}-1} E^{(\texttt{tid})}}{\texttt{Nt}}.$$

Within the parallel region, we use `#pragma omp atomic` to correctly accumulate each thread's contributions without data races. We apply the same procedure for each element of `O1` and `O2`. After the parallel region, we divide the accumulated sums by `Nt` to obtain the mean values of all observables. Finally, we perform the GD update as before.

3.10.14 Interacting MPI GD VMC

In this final section describing computational implementation, we will focus on MPI GD VMC algorithm. MPI is designed for distributed memory systems, meaning that all communications between processes need to be coded directly. For example, if first thread has computed value a and thread two needs this value for its computation, thread one needs to send it directly to thread two. We did not see such phenomenon in 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 parallelized, but it is much more difficult. 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. Afterwards, we create unique seed for each rank and use it to create unique systems. Each thread evolves its own system using its own RNG, as in the OMP case. When all threads computed, each thread collects its observables. To send these observables from each thread to the master thread we use `MPI_Reduce` and `MPI_Allreduce` functions. After calling `MPI_Reduce`, each process sends its local energy `E` to the root process, which stores the summed value in `TE` variable. The function `MPI_Allreduce` performs

the same operation on a vector and returns the result to all processes. Therefore, we use this function to fill vector `TO1` in the following way:

$$\texttt{TO1} = \left(\sum_{i=\texttt{rank}}^{\texttt{size}} \texttt{O1}_0^{(\texttt{rank})}, \cdots, \sum_{i=\texttt{rank}}^{\texttt{size}} \texttt{O1}_{\texttt{np}}^{(\texttt{rank})} \right).$$

The vector `TO2` is filled in the same way. After we send observables from each thread to the master thread, it performs GD update. It is very important so send updated vector of variational parameters β_{ij} from master thread to all other 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. 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.

3.11 Methods

To perform this study 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 perform VMC simulations and generate results;
- `OMP` [2] and `MPI` [3] APIs were used to parallelize VMC code;
- `Autodiff` [1] 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 analyze 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`.

All program implementations and high-quality figures are available on GitHub: https://github.com/YVol322/VMC.

Algorithm 16 Metropolis Hastings Step Function

```
1: procedure STEP(sl, &waveFunction, &particles))
2:     int N = particles.size();
3:     int d = particles[0].getNumberOfDimensions();
4:     Initialize double PsiOld = 0.0 and double PsiNew = 0.0;
5:     Initialize 2D vector Fold of size N × d with zeros;
6:     double dt = sl;
7:     double dtSqrt = sqrt(dt);
8:     Compute the old-state ansatz: PsiOld = waveFunction.evaluate(particles);
9:     for int i = 0; i < N; i++ do
10:        Fold[i] = waveFunction.quantumForce(particles, i);
11:    end for
12:    bool accept = 0;
13:    for int i = 0; i < N; i++ do
14:        Dvec Rold = particles[i].getPosition();
15:        Initialize Dvec Rnew of size d with zeros;
16:        for int j = 0; j < d; j++ do
17:            double drift = D · Fold[i][j] · dt;
18:            double gauss = nextGaussian(0, 1);
19:            Rnew[j] = Rold[j] + drift·gauss· dtSqrt;
20:        end for
21:        Compute the new-state ansatz: PsiNew =
   waveFunction.evaluate(particles);
22:        Dvec Fnew = waveFunction.quantumForce(particles, i);
23:        G = GreensFunctionRatio(Rnew, Rold, dt, Fold[i], Fnew);
24:        Compute the acceptance ratio: alpha = min(1, G · PsiNew^2/PsiOld^2);
25:        if m_rng.nextDouble()≤ alpha then
26:            PsiOld = PsiNew;
27:            Fold[i] = Fnew;
28:            accept = 1;
29:        else
30:            for int j = 0; j < d; j++ do
31:                particles[i].adjustPosition(Rold[d] - RNew[d], d);
32:            end for
33:        end if
34:    end for
35:    return accept;
36: end procedure
```

Algorithm 17 Sample Function

```
1: procedure SAMPLE(bool acceptedStep, System* system)
2:     Compute local energy: E = system.computeLocalEnergy();
3:     Accumulate local energy: m_cEnergy += E;
4:     Compute sum of r_i^2 of all particles: r2 = system -> computer2();
5:     Accumulate O_1 for α optimization: m_cO1alpha -= r2;
6:     Accumulate O_2 for α optimization: m_cO2alpha -= r2 · E;
7:     int N = m_N;
8:     int p = 0;
9:     double O = 0;
10:    for int i = 0; i < N-1; i++ do
11:        for int j = i; j < N; j++ do
12:            Compute r_ij: double rij = system.computerij(i, j);
13:            Accumulate O_1^(J) for β_ij optimization: m_cO1Jastrow[p] += rij;
14:            Accumulate O_2^(J) for β_ij optimization: m_cO2Jastrow[p] += rij · E;
15:            double a = 0;
16:            if i < N/2 && j > N/2 then
17:                a = 1.0 / 3.0;
18:            else
19:                a = 1.0;
20:            end if
21:            double beta = system.getWaveFunctionParameters()[0];
22:            Compute O_1^(P): O -= a · rij^2 / (1 + beta · rij)^2;
23:            p++;
24:        end for
25:    end for
26:    Accumulate O_1^(P): m_O1cPade += O;
27:    Accumulate O_2^(P): m_O2cPade += O · E;
28:    m_stepnumber++;
29:    m_numberOfAcceptedSteps += acceptedStep.
30: end procedure
```

Algorithm 18 System Class functions

```
1: procedure runEquilibrationSteps(double sl, int m)
2:     int acceptedSteps = 0;
3:     for int i = 0; i < m; i++ do
4:         acceptedSteps += 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 = Sampler(m_N, m_d, sl, n);

10:    for int i = 0; i < n; i++ do
11:        bool acceptedStep = 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
```

Algorithm 19 Non-Interacting VMC solver

```
1: procedure VMC(N, d, n, m, alpha, s, sl)
2:     Construct RNG: Random rng = make_unique<Random>(s);
3:     Pvec particles = setupRandomUniformInitialState(d, N, *rng);
4:     Construct Hamiltonian: HarmonicOscillator H = make_unique<HO>(1,0);
5:     Construct wave function: Boson WF ← make_unique<Boson>(alpha, N);
6:     Construct solver: Metropolis solver = make_unique<M>(move(rng));
7:     Construct system:

         System syst = make_unique<System>(H, WF, solver, particles);

8:     Start timer: high_resolution_clock start;
9:     Burn-in VMC: accsteps = system.runEquilibrationSteps(sl, m);
10:    VMC: Sampler sampler = system.runMetropolisSteps(sl, n);
11:    Stop timer: high_resolution_clock stop;
12:    Compute elapsed time: dur = duration_cast(start - stop);
13:    Fill sampler's m_time variable: sampler.set_time(dur);
14:    Print output to the terminal: sampler.printOutputToTerminal(*syst).
15: end procedure
```

Algorithm 20 Non-Interacting GD VMC solver

procedure GDVMC(`N, d, n, m, alpha, s, sl, B, eta`)
    Initialize the L2 norm with a number larger than B: `double L2 = 100.0`;
    Initialize the iteration counter: `int i = 0`, and choose the maximum number of iterations `max`;
    while `grad > B && i < max` **do**
        Perform VMC: `VMC(N, d, n, m, alpha, s, sl)`;
        Collect the local energy from the sampler: `double E = sampler.getEnergy()`;
        Collect O_1 from the sampler: `double O1 = sampler.getO1alpha()`;
        Collect O_2 from the sampler: `double O2 = sampler.getO2alpha()`;
        Compute the gradient: `double grad = 2*(O2 - E · O1)`;
        Update the variational parameter: `alpha -= eta · grad`;
        Increment the iteration counter: `i++`;
        if `grad` $\leq$ `B` **then**
            Print `E` to the terminal.
        end if
    end while
end procedure

Algorithm 21 Interacting Jastrow GD VMC Solver

procedure JGD(`N, d, n, m, s, sl, beta, omega, mode, B, eta`)
    Initialize the L2 norm with a number larger than B: `double L2 = 100.0`;
    Initialize the iteration counter: `int i = 0`, and choose the maximum number of iterations `max`;
    Compute the number of pairs: `int np = N(N-1)/2`;
    while `L2 > B && i < max` **do**
        Initialize `Dvec grad` of size `np` with zeros;
        Perform VMC: `VMC(N, d, n, m, s, sl, beta, omega, mode)`;
        Collect the local energy from the sampler: `double E = sampler.getEnergy()`;
        Collect O_1 from the sampler: `Dvec O1 = sampler.getO1Jastow()`;
        Collect O_2 from the sampler: `Dvec O2 = sampler.getO2Jastow()`;
        Reset `L2 = 0`;
        for `int k = 0; k < np; k++` **do**
            Compute the gradient: `grad[k] = 2*(O2[k] - E · O1[k])`;
            Update the variational parameters: `beta[k] -= eta · grad[k]`;
            Accumulate L2 norm squared: `L2 += grad[k]`^{2};
        end for
        Compute the L2 norm: `L2 = sqrt(L2)`;
        Increment the iteration counter: `i++`;
        if `L2` $\leq$ `B` **then**
            Print the local energy `E` to the terminal.
        end if
    end while
end procedure

Algorithm 22 Interacting Jastrow OMP GD VMC Solver

1: **procedure** OMPJGD(`N, d, n, m, s, sl, beta, omega, mode, B, eta`)
2: Initialize the L2 norm with a number larger than B: `double L2 = 100.0;`
3: Initialize the iteration counter: `int i = 0`, and choose the maximum iterations `max`;
4: Compute the number of pairs: `int np = N(N-1)/2;`
5: `#pragma omp parallel`
6: `int Nt = omp_get_num_threads();`
7: `#pragma omp end parallel`
8: **while** `L2 > B && i < max` **do**
9: Initialize `Dvec grad` of size `np` with zeros;
10: Initialize `Dvec O1, O2, TO1, TO2` of size `np` with zeros;
11: Initialize `double E = 0, TE = 0;`
12: `#pragma omp parallel`
13: `int tid = omp_get_thread_num();`
14: `int ts = s + tid;`
15: Perform VMC on each thread:

```
VMC(N, d, n/Nt, m/Nt, alpha, ts, sl, beta, omega, mode);
```

16: Collect local energy on this thread: `E = sampler.getEnergy();`
17: Collect O_1 on this thread: `O1 = sampler.getO1Jastow();`
18: Collect O_2 on this thread: `O2 = sampler.getO2Jastow();`
19: `#pragma omp atomic`
20: `TE += E;`
21: `#pragma omp end atomic`
22: **for** `int j = 0; j < np; j++` **do**
23: `#pragma omp atomic`
24: `TO1[j] += O1[j];`
25: `TO2[j] += O2[j];`
26: `#pragma omp end atomic`
27: **end for**
28: `#pragma omp end parallel`
29: Reset `L2 = 0;`
30: **for** `int k = 0; k < np; k++` **do**
31: Compute the gradient: `grad[k] = 2*(TO2[k] - TE/Nt * TO1[k]) / Nt;`
32: Update the variational parameters: `beta[k] -= eta * grad[k];`
33: Accumulate L2 norm squared: `L2 += grad[k]`^{2};
34: **end for**
35: Compute the L2 norm: `L2 = sqrt(L2);`
36: Increment the iteration counter: `i++;`
37: **end while**
38: **end procedure**

Algorithm 23 Interacting Jastrow OMP GD VMC Solver

1: **procedure** MPIJGD(`N, d, n, m, s, sl, beta, omega, mode, B, eta`)
2: Initialize L2 norm with a number bigger than B: `double L2 = 100.0`;
3: Initialize iteration counter: `int i = 0`, and choose max iterations number `max`;
4: Compute number of pairs: `np = N(N-1)/2`;
5: Initialize variables `int size, rank`;
6: Set up parallel environment: `MPI_Init(&argc, &argv)`;
7: Assign number of threads to `size` variable,

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

8: Assign thread number of each thread to `rank` variable,

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

9: Create seeds for each thread, `s *= rank`;
10: **while** `L2 > B && i < max` **do**
11: Initialize `Dvec grad` of size `np` with zeros;
12: Initialize `Dvec O1, O2, TO1, TO2` of size `np` with zeros;
13: Initialize `double` E = 0, TE = 0;
14: Perform VMC on each thread:

```
VMC(N, d, n/size, m/size, s, sl, beta, omega, mode);
```

15: Collect local energy on all threads: `E = sampler.getEnergy()`;
16: Collect O_1 on all threads: `O1 = sampler.getO1Jastow()`;
17: Collect O_2 on all threads: `O2 = sampler.getO2Jastow()`;
18: Save sum of `E` from each thread to `TE`: `MPI_Reduce(&E, &TE, MPI_SUM)`;
19: Save sum of each `O1` element to `TO1`: `MPI_Allreduce(&O1, &TO1, MPI_SUM)`;
20: Save sum of each `O2` element to `TO2`: `MPI_Allreduce(&O2, &TO2, MPI_SUM)`;
21: **if** `rank == 0` **then**
22: Reset `L2 = 0`;
23: **for** `int k = 0; k < np; k++` **do**
24: Compute gradient: `grad[k] = 2(TO2[k]-E · TO1[k]) / size`;
25: Update variational parameters: `beta[k]-= eta · grad[k]`;
26: Compute L2 norm squared: `L2 += grad[k]`^{2};
27: **end for**
28: Broadcast `beta` to all threads: `MPI_Bcast(beta.data())`;
29: Compute L2 norm: `L2 = sqrt(L2)`;
30: Add one iteration to counter: `i++`;
31: Send L2 norm to all threads using `MPI_Bcast(&L2)`;
32: **end if**
33: **end while**
34: Finalize parallel environment: `MPI_Finalize()`;
35: **end procedure**

Chapter 4

Results and Discussion

4.1 Non-Interacting Bosons

The first and simplest system we consider is N non-interacting bosons. In Appendix 6.4, we derive an analytical solution for the three-dimensional case, which formula (6.32) then generalizes to d dimensions (3.25). In N dimensions, the variational energy as a function of the parameter α is

$$E_T = \frac{Nd}{2}\left(\alpha + \frac{1}{4\alpha}\right).$$

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

$$E_0(\tilde{\alpha}) = \frac{Nd}{2}.$$

Omega dependency can be recovered as:

$$E_0(\omega) = \frac{Nd}{2}\omega.$$

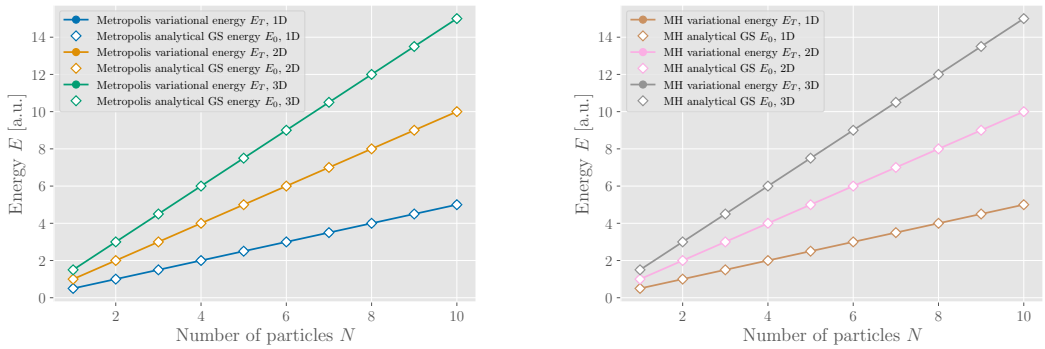
We chose this system as a starting point of the research, because it has analytical solution that we can compare numerical results with. If these results align, then developed VMC simulation works as intended.

4.1.1 VMC algorithm

As a first step, we want to see if Metropolis and Metropolis-Hastings algorithms estimate GS energy correctly. In Figure 4.1 we present results of Metropolis and Metropolis-Hastings VMC simulations for $1 - 10$ particles in 1, 2 and 3 dimensions.

For all simulations in this section, 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 in Figure 4.1, numerical results align perfectly with analytical solutions. 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 results. In Figure 4.2, we show difference between variational energy computed using analytical derivatives and numerical derivatives $\Delta E = E_T^{\mathrm{AD}} - E_T^{\mathrm{ND}}$ in $1 - 3$ dimensions. To



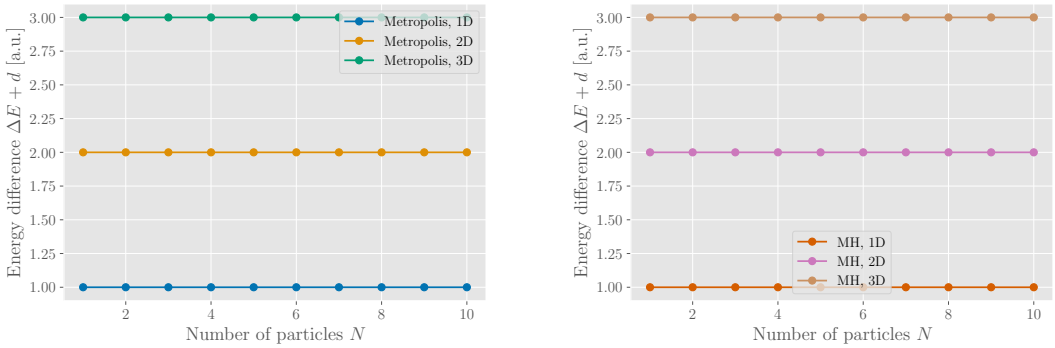
(a) Metropolis energy.

(b) Metropolis-Hastings energy.

Figure 4.1: Metropolis (a) and Metropolis-Hastings (b) variational energy E_T compared with analytical GS energy E_0 as a function of number of particles N in 1, 2 and 3 dimensions.

distinguish dependencies we add number of dimensions d to energy difference ΔE. Data in the figure show that we obtain the same variational 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 aligns, it is still not a 100% guaranty that implementation is correct, so we need to be careful with the results 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.

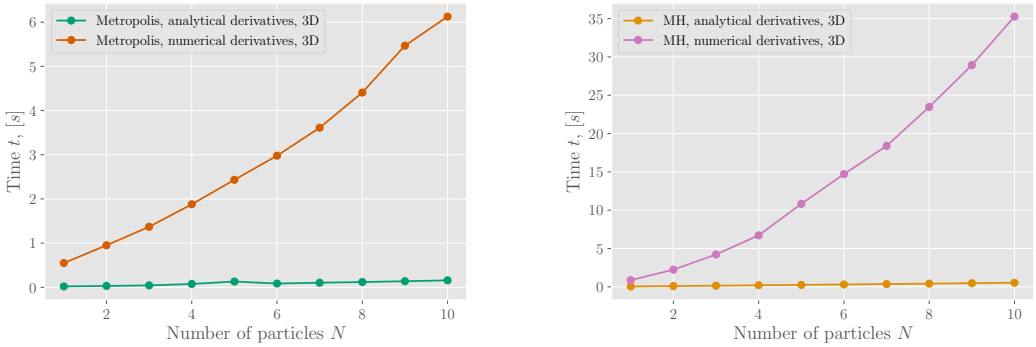


(a) Metropolis energy difference.

(b) Metropolis-Hastings energy difference.

Figure 4.2: Metropolis (a) and Metropolis-Hastings (b) variational energy difference $\Delta E = E_T^{\mathrm{AD}} - E_T^{\mathrm{ND}}$ plus number of dimensions d as a function of number of particles N in 1, 2 and 3 dimensions.

We also want to study analytical and numerical algorithms execution time. Algorithms runtime for 3D systems is presented in Figure 4.3. Although both 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 the execution time for numerical algorithms is enormous. Bosonic system is a system with the simplest ansatz, every system we will study later requires 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.

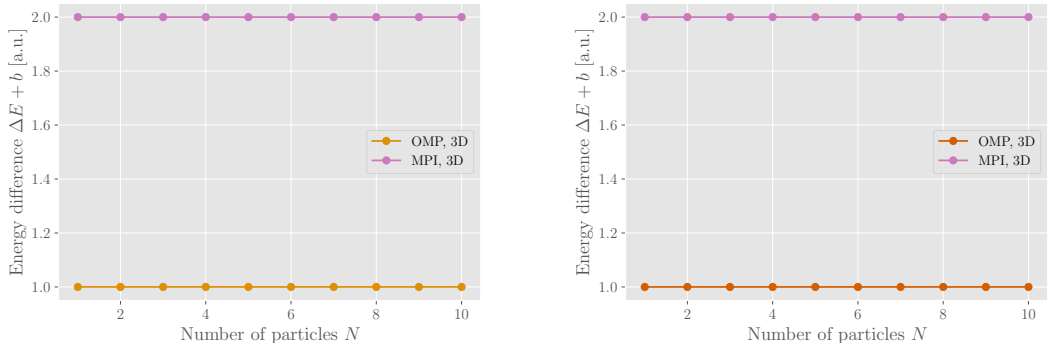


(a) Metropolis execution time.

(b) Metropolis-Hastings execution time.

Figure 4.3: Metropolis (a) and Metropolis-Hastings (b) execution time t as a function of number of particles N in 3D.

Next, we want to focus on MPI and OMP parallel VMC algorithms. We would like to see what speedup they provide and make sure that they still generate correct results. Figure 4.4 shows energy difference ΔE between analytical and parallel algorithms. To



(a) Metropolis energy difference.

(b) Metropolis-Hastings energy difference.

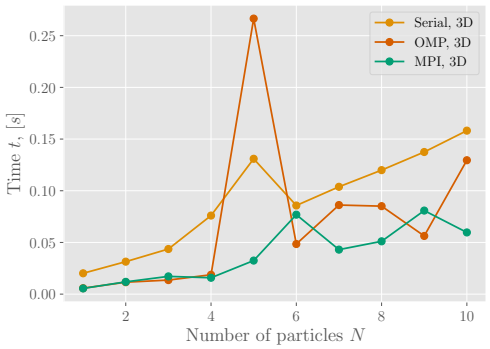
Figure 4.4: Metropolis (a) and Metropolis-Hastings (b) variational energy difference $\Delta E = E_T^{\text{an}} - E_T^{\text{parallel}}$ plus parallel identifier b in 3D. $b = 1$ for OMP and $b = 2$ for MPI.

distinguish data we added $b = 1$ to ΔE for OMP and $b = 2$ for MPI. We can clearly see that parallel algorithms compute the same variational energy as analytical algorithms, meaning that they generate correct results.

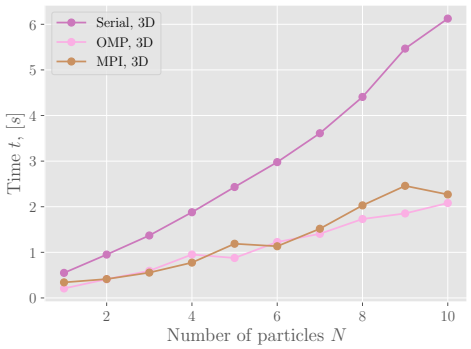
Now, when we made sure that parallel algorithms generate correct results, we want to study algorithm runtime for all algorithms and compare them. We have seen already that numerical algorithms take more time to estimate GS energy. Parallel algorithms must speed computations up.

In 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 from 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 0.15 seconds, there is no need in parallelization. Numerical Metropolis algorithm



(a) Analytical Metropolis execution time.

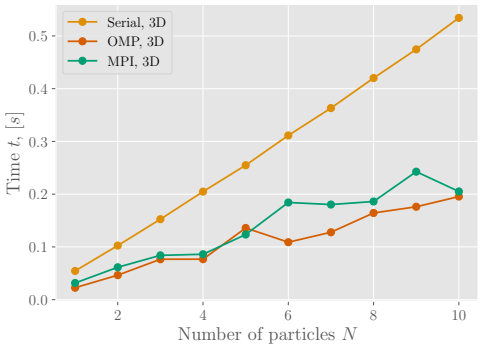


(b) Numerical Metropolis execution time.

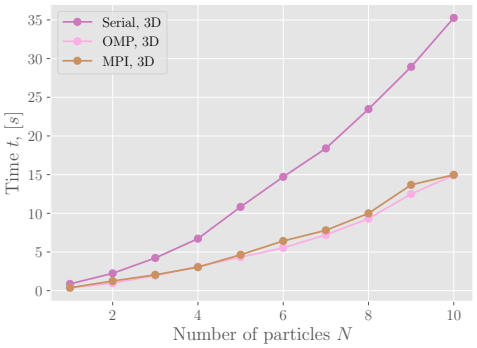
Figure 4.5: Parallel and serial Metropolis algorithms with analytical derivatives (a) and numerical derivatives (b) execution time t as a function of number of particles N in 3D.

is more computationally expensive, and from Figure 4.5b we see that for all N parallel runtime it approximately three times faster then for serial algorithm. We chose such small number of VMC iterations on purpose, to show that parallelization only makes sense when algorithm is computationally expensive.

Metropolis-Hasting algorithm runtime is displayed on Figure 4.6. In this case, even for analytical algorithm parallel versions give sufficient speedup. When we consider numerical algorithm, reducing runtime from 35 seconds to 15 is an improvement. 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 a good speedup.



(a) Analytical MH execution time.

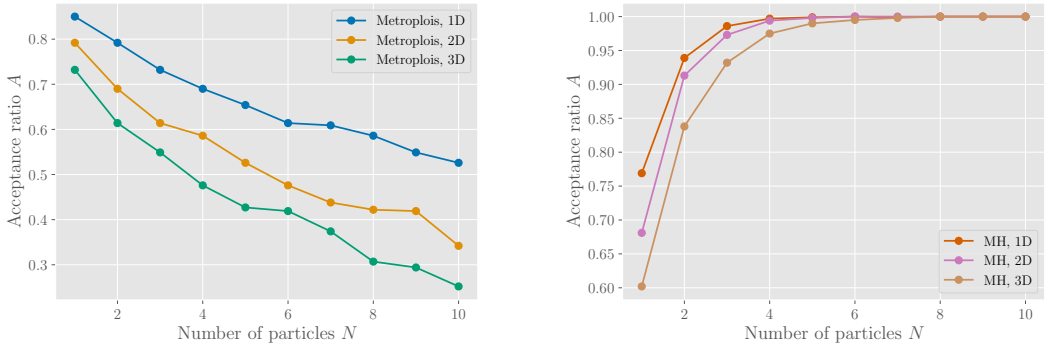


(b) Numerical MH execution time.

Figure 4.6: Parallel and serial Metropolis-Hastings algorithms with analytical derivatives (a) and numerical derivatives (b) execution time t as a function of number of particles N in 3D.

Next point of interest is how acceptance ratio A and number of particles N are related to each other. This information will help us to develop intuition of what values of h should be chosen in order to obtain good results. In Figure 4.7, we see two completely different dependencies. For Metropolis algorithm, acceptance rate decreases as number of particles increase, while in Metropolis-Hastings, acceptance ratio converges to one when N increases.

There is nothing mysterious in this, because we compute acceptance ratio differently for these algorithms. Metropolis algorithm moves all particles and then accepts or rejects the step. On the other hand, Metropolis algorithm moves one particle, accepts or rejects



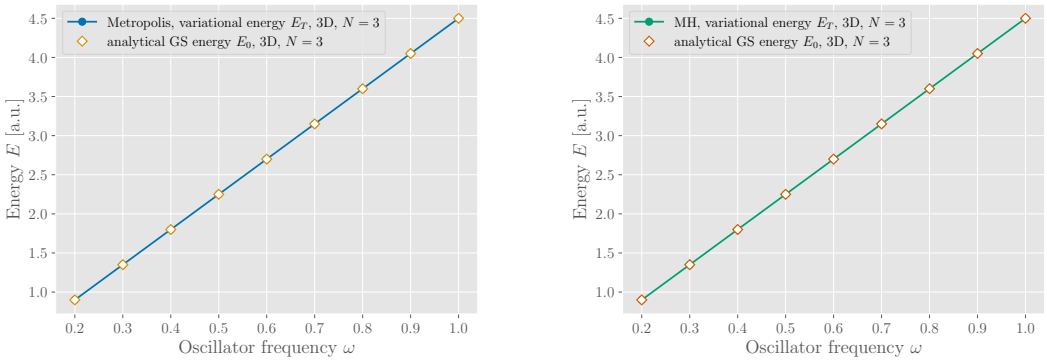
(a) Metropolis acceptance ratio.

(b) Metropolis-Hastings Acceptance ratio.

Figure 4.7: Metropolis (a) and Metropolis-Hastings (b) algorithms acceptance ratio A as a function of number of particles N in 1, 2 and 3 dimensions.

the step and repeats 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 a choice. Unfortunately, these data do not give us a hint on how to choose step size for Metropolis-Hastings algorithm.

Finally, we study how our VMC simulation estimates the GS energy for $\omega \neq 1$. We ran simulations at several values of ω and compared the variational energy $E_T(\omega)$ to the analytic result $E_0(\omega)$. For each ω, we used analytical Metropolis and Metropolis-Hastings algorithms. The results are shown in Figure 4.8: in every case, the analytic energy $E_0(\omega)$ lies on top of the simulated curve $E_T(\omega)$, demonstrating that our VMC implementation reproduces the true ground-state energy exactly for all tested ω.



(a) Metropolis energy.

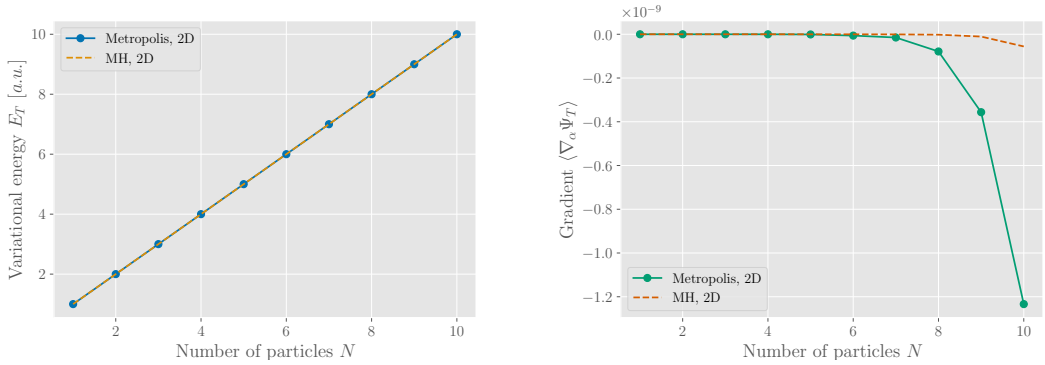
(b) Metropolis-Hastings energy.

Figure 4.8: Metropolis (a) and Metropolis-Hastings (b) variational energy energy E_T compared with analytical GS energy E_0 as a function of oscillator frequency ω. Three particles in 3D.

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 is able to find optimal variational parameter that minimizes variational 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 4.9 contains data of such simulation.



(a) Variational energy.

(b) Gradient.

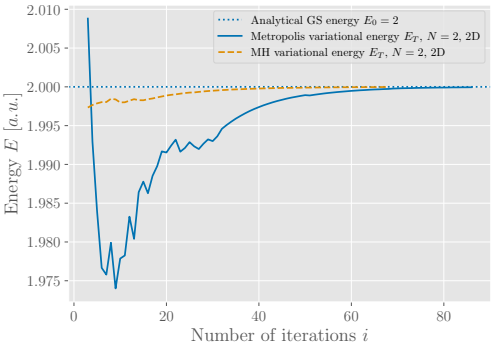
Figure 4.9: Variational energy E_T (a) and gradient $\langle \nabla_\alpha \Psi_T \rangle$ (b) as a function of number of particles N in 2D with optimal variational parameter $\alpha = \tilde{\alpha} = 1/2$.

For all N particles, GD algorithm converged after one iteration, and variational energy aligns with analytical solution. Moreover, gradients, computed for GD update are values proportional to 10^{-9}, which is a great result.

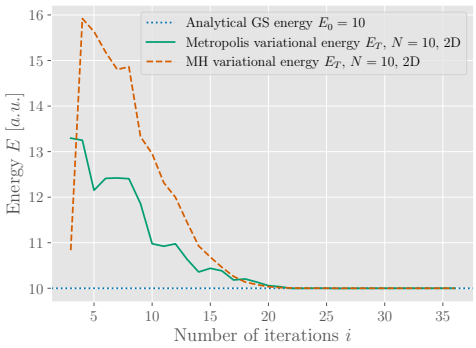
Now when we know that GD algorithm converges at optimal variational parameter, we are ready to perform real GD simulations, when initial variational parameter is not optimal. If we did not have any information about the system, the natural choice of initial variational parameter would be a small value close to zero or one. Figures 4.10, 4.11 and 4.12 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 $n = 10^3$ and umber 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 at first iterations, and then slowly begin converging their true values. This is a consequence of choosing such small number of iterations. If we were to choose, for example $n = 10^4$ VMC iterations, fluctuations would be smaller. With such parameters, Metropolis and Metropolis-Hastings algorithms need approximately the same number of iteration to converge. When Metropolis-Hastings algorithm does not converge in less iterations than

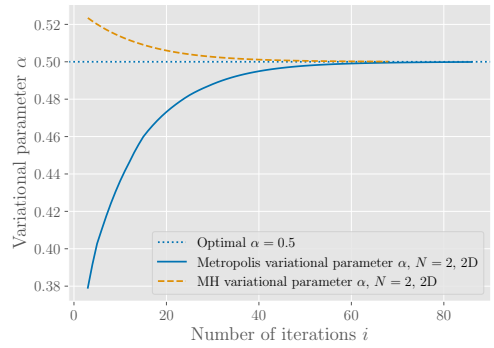


(a) Energy, $N = 2$.

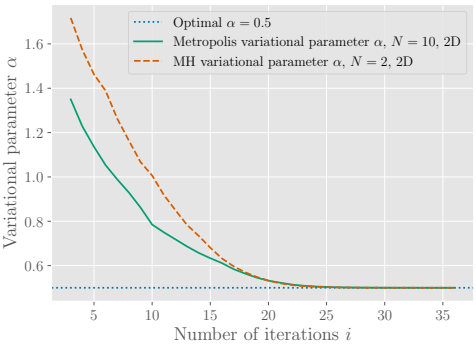


(b) Energy, $N = 10$.

Figure 4.10: Variational energy E_T for (a) $N = 2$ and (b) $N = 10$ particles compared to analytical GS energy E_0 as a function of number of iterations i in 2D.



(a) Variational parameter, $N = 2$.



(b) Variational parameter, $N = 10$.

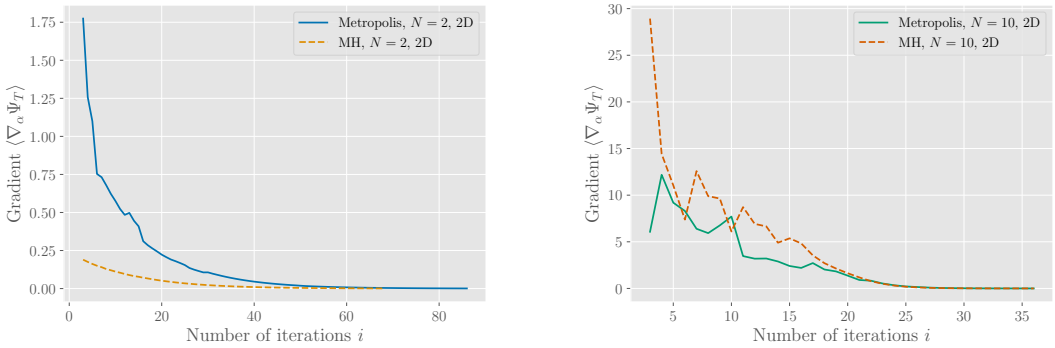
Figure 4.11: Variational parameter α for (a) $N = 2$ and (b) $N = 10$ particles compared to optimal variational parameter $\alpha = 0.5$ as a function of number of iterations i in 2D.

Metropolis, it is better to use Metropolis algorithm, because it is less computationally expensive.

During first iterations, gradient was big, and therefore variational parameter and variational 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 variational energy with these parameters is very close to analytical GS energy with $\Delta E_0 = E_0 - E_T = 10^{-4}$.

We have two parameters that we can tune: step size h and learning rate η. This is the next topic of interest. We want to gain intuition for coming systems about what values we should 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 diverge. Safe choice it to initialize parameters with small values, because then GD will likely converge. But using small parameters can take thousands of iteration 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 not allowing GD to diverge while achieving reasonable convergence time.

We study GD convergence speed with different learning rates and step sizes for a

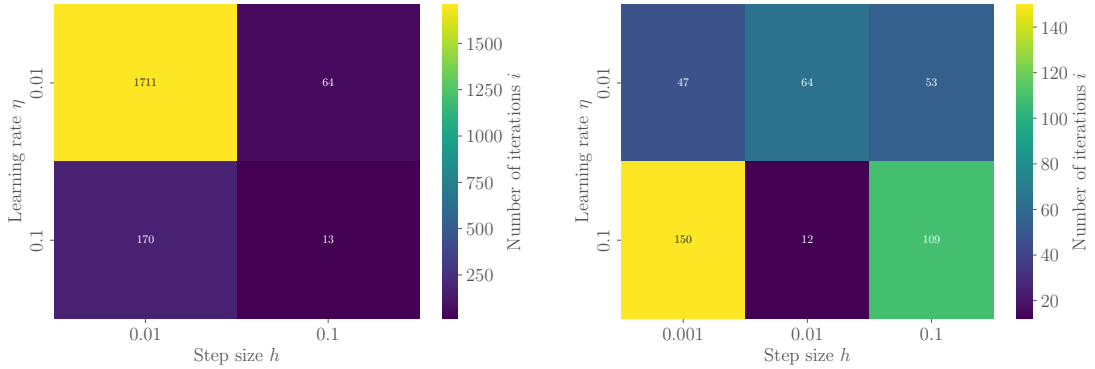


(a) Gradient, $N = 2$.

(b) Gradient, $N = 10$

Figure 4.12: Gradient $\langle \nabla_\alpha \Psi_T \rangle$ for (a) $N = 2$ and (b) $N = 10$ particles as a function of number of iterations i in 2D.

system of five 2-dimensional particles with $n = 10^4$, $m = 10^3$, $\alpha_0 = 1$, and $B = 10^{-3}$. Heatmaps on Figure 4.13 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 $h = \eta = 0.01$.



(a) Metropolis heatmap.

(b) Metropolis-Hasting heatmap.

Figure 4.13: GD iterations until convergence heatmaps as a function of step size h and learning rate η for (a) Metropolis and (b) Metropolis-Hastings algorithms for five particles in two dimensions.

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 $h = 0.001$ and $\eta = 0.01$, MH does not converge, because we sat maximal number of iterations $i_{\max}$ to 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 are "unlucky" parameters.

From results of this simulation, we can conclude that for Metropolis algorithm, one needs to choose the biggest η and h that still do not "blow up" variational parameter. In contrast, Metropolis-Hastings method works better with smaller values of h and η in the range of $(0.001, 0.1)$.

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 estimates GS energy of $E_0 = 12$ correctly. Numerical algorithm 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 closed shells of two-dimensional non-interacting fermions in atomic units with $\omega = 1$. We derived analytical solutions for the first closed shell, and they are given by formulae (6.37), (6.38) and (6.39):

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

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 energy of harmonic orbitals is given by formula (2.20):

$$E_{n_x,n_y} = (n_x + n_y + 1).$$

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

$$E_0^{(N=12)} = 2 \cdot 1 + 4 \cdot 2 + 6 \cdot 3 = 28.$$

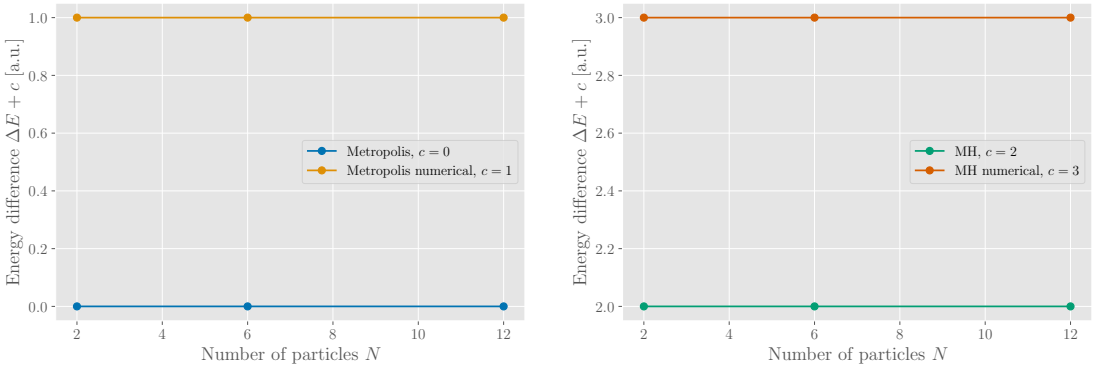
Omega dependence can be restored as:

$$E_0^{(N=2)}(\omega) = 2\omega, \quad E_0^{(N=6)}(\omega) = 10\omega, \quad E_0^{(N=12)}(\omega) = 28\omega.$$

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 an 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 for the first closed shell $\alpha = 1/2$. Energy difference $\Delta E = E_0 - E_T$ is plotted in the Figure 4.13. We ran this simulation with parameters $n = 10^4$, $m = 10^3$, and $h = 0.1$. Variational energy converges to its analytical value even in 10^4 VMC iterations. Data from this figure shows that both Metropolis and Metropolis-Hastings algorithms are implemented correctly and they generate correct results. During every VMC iterations, program

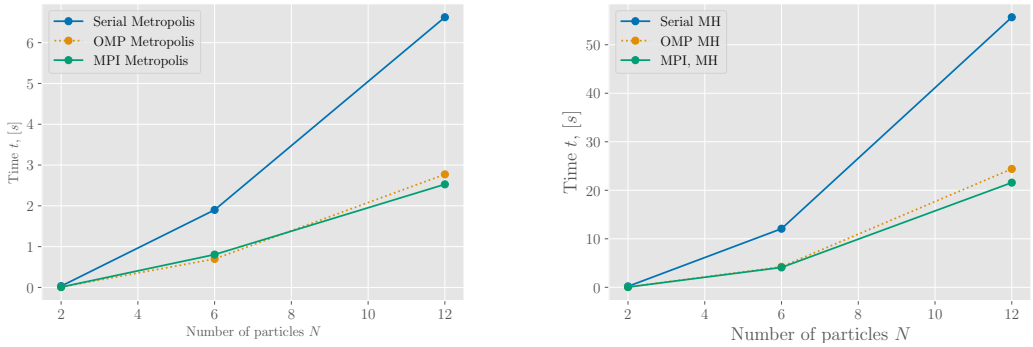


(a) Metropolis energy difference.

(b) Metropolis-Hastings energy difference.

Figure 4.14: Metropolis (a) and Metropolis-Hastings (b) variational energy difference $\Delta E = E_T^{\text{an}} - E_T^{\text{parallel}}$ plus algorithm identifier c.

computes determinants, which takes much more time, compared to bosonic ansatz. In Figures 4.15 and 4.16, we show execution time of analytical and numerical algorithms for the system with the same parameters. Data from these figures show that MH algorithm

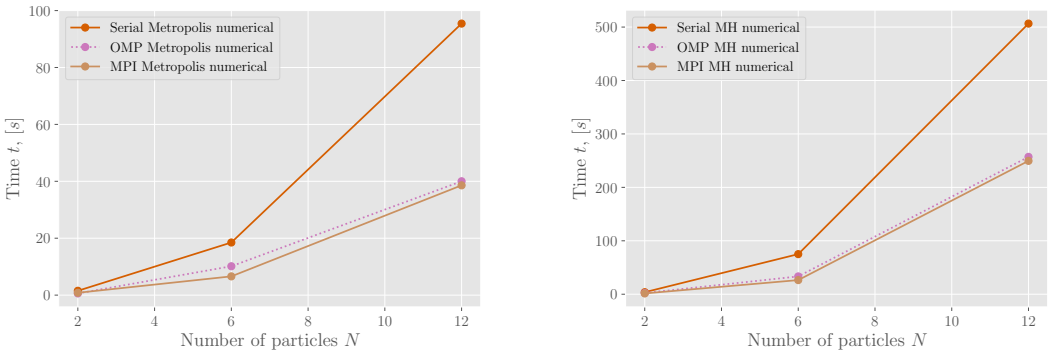


(a) Analytical Metropolis execution time.

(b) Analytical MH execution time.

Figure 4.15: Parallel and serial analytical (a) Metropolis and (b) MH algorithms execution time t as a function of number of particles N.

is much more costly – its runtime is approximately ten times higher that of 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 algorithms have even bigger runtime. Numerical serial VMC computes 10^4 VMC iterations in 500 seconds, which is huge, compared with parallel analytical of 20 seconds. Obviously, we want the algorithm to be executed as fast as possible, therefore parallel analytical algorithms is our choice. Once we make sure that numerical, analytical



(a) Numerical Metropolis execution time.

(b) Numerical MH execution time.

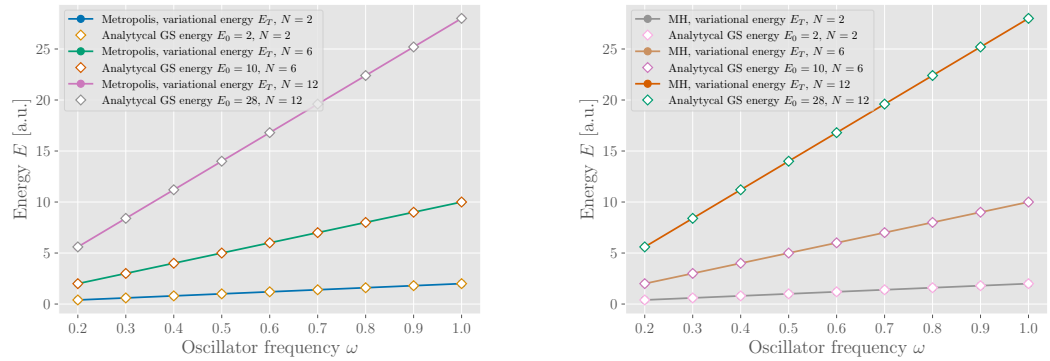
Figure 4.16: Parallel and serial numerical (a) Metropolis and (b) MH algorithms execution time t as a function of number of particles N.

and parallel analytical results are correct, we will use parallel analytical algorithms for heavy computations.

We did not know optimal α for second and third closed shell. We chose $\alpha = 1/2$ because it is optimal for the first closed shell. Since variational energy for all three systems aligns with analytical GS energy, we conclude that for all systems optimal variational parameter is $\tilde{\alpha} = 1/2$.

Another observation is that for non-interacting systems, both Metropolis and MH algorithms generate **exact** results. In situation like this, it is more convenient to choose algorithm which runs faster. In this case, Metropolis shows better performance.

In Figure 4.17, we present variational energy as a function of oscillator frequency ω for $N = 2$, 6 and 12 particles. For this simulation, we used serial Metropolis and Metropolis-Hastings algorithms. As for the bosonic case, the variational energy E_T aligns with the GS E_0 perfectly for each value of ω, demonstrating the accuracy of our implementation.



(a) Metropolis energy.

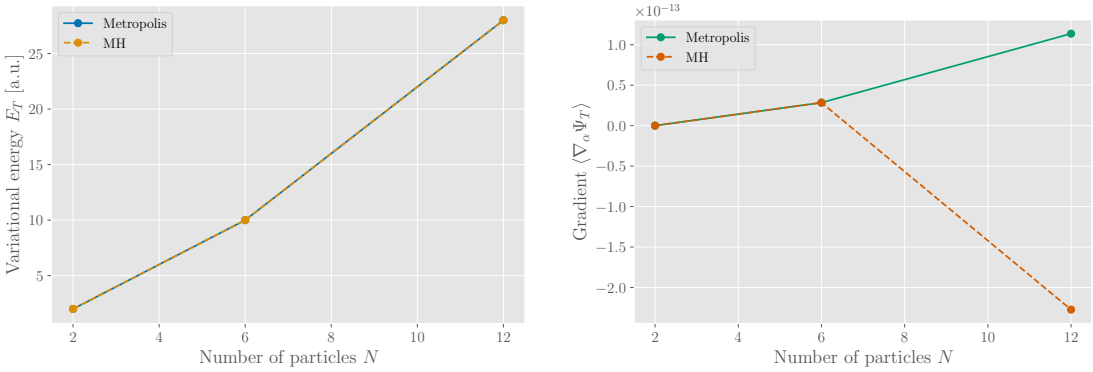
(b) Metropolis-Hastings energy.

Figure 4.17: Metropolis (a) and Metropolis-Hastings (b) variational energy energy E_T compared with analytical GS energy E_0 as a function of oscillator frequency ω.

4.2.2 GD VMC algorithm

We start start with GD VMC algorithm by making sure it converges when $\alpha = \tilde{\alpha} = 1/2$.

Figure 4.18 demonstrates data generated with parameters: $h = 0.1$, $\eta = 0.01$, $n = 10^3$, and $m = 10^2$. Variational energy is accurate using only 110 VMC iterations, it converges to the correct GS value. Gradient has values on the order of 10^{-13}.

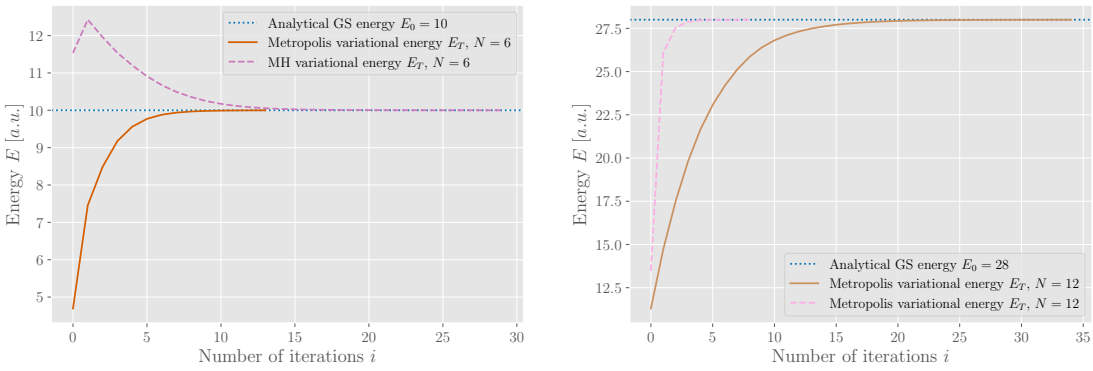


(a) Variational energy. (b) Gradient.

Figure 4.18: (a) Variational energy E_T and (b) 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 algorithms. We will focus on second and third closed shells.

Figure 4.19 shows how GS energy changes during GD iterations. System parameters are specified in the figure captions. 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 progresses, particles reach equilibrium state and energy becomes accurate.

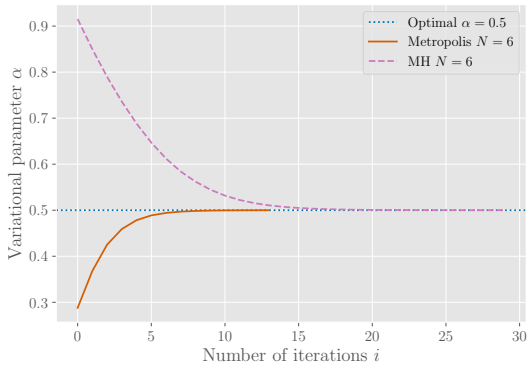


(a) Energy for $N = 6$, $h_\mathrm{M} = 0.1$, and $h_{MH} = 0.01$. (b) Energy for $N = 12$, $h_\mathrm{M} = 0.1$, and $h_{MH} = 0.001$.

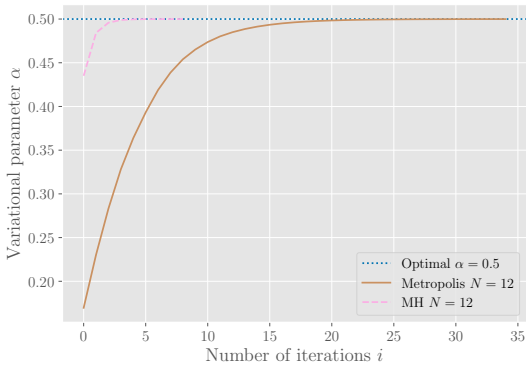
Figure 4.19: Variational energy E_T for (a) $N = 6$ and (b) $N = 12$ particles compared to analytical GS energy E_0 as a function of number of iterations i. System parameters are $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$, and $B = 10^{-3}$

Figures 4.20 and 4.21 demonstrate how variational parameter and sample mean

gradients change during GD iterations. Essentially, we observe the same trend as for energy – far from optimal value during first iterations and starting to converge after approximately 10 iterations. It is also important to note, that after fist iterations convergence is smooth: we do not see fluctuations after some q number of iterations. It means that after q-th iteration, gradient becomes smaller with every iteration and variational parameter with variational energy become close to their true values.



(a) Variational parameter for $N = 6$, $h_{\mathrm{M}} = 0.1$, and $h_{MH} = 0.01$.

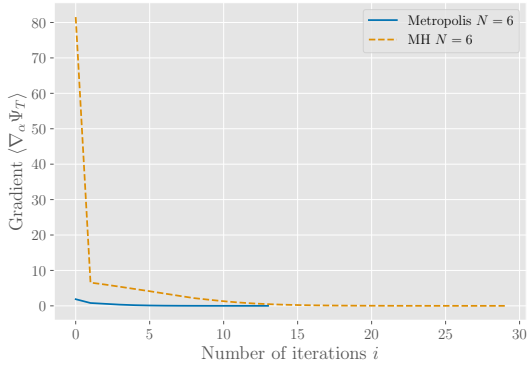


(b) Variational parameter for $N = 12$, $h_{\mathrm{M}} = 0.1$, and $h_{MH} = 0.001$.

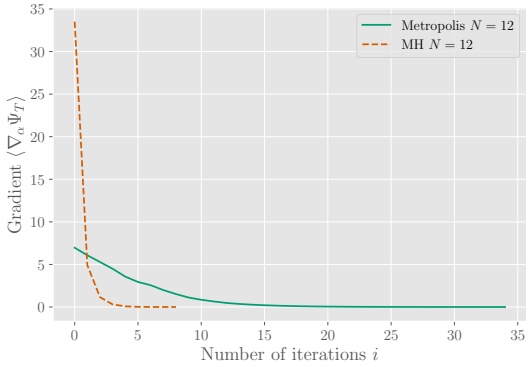
Figure 4.20: Variational parameter α for (a) $N = 6$ and (b) $N = 12$ particles compared to optimal variational parameter $\alpha = 0.5$ as a function of number of iterations i. System parameters are $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$, and $B = 10^{-3}$.

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 diverges if parameters are chosen too big for a given system.



(a) Gradient for $N = 6$, $h_{\mathrm{M}} = 0.1$, and $h_{MH} = 0.01$.

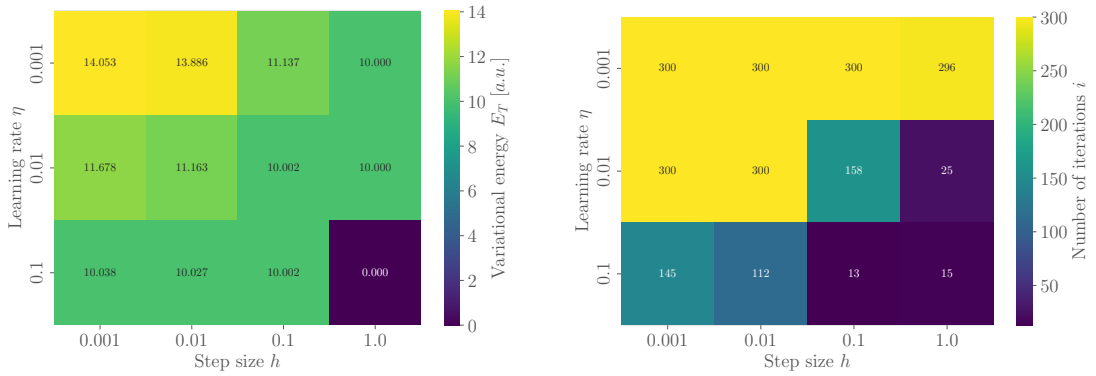


(b) Gradient for $N = 12$, $h_{\mathrm{M}} = 0.1$, $h_{MH} = 0.001$.

Figure 4.21: Gradient $\langle \nabla_\alpha \Psi_T \rangle$ for (a) $N = 6$ and (b) $N = 12$ particles as a function of number of iterations i. System parameters are $\alpha_0 = 0.1$, $n = 10^3$, $m = 10^2$, $\eta = 0.1$, and $B = 10^{-3}$.

Therefore, now we will run simulations with fixed number of iterations, varying learning rate and step size and studying how gradient, variational 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 as shown by heatmaps in Figure 4.22. If energy heatmap contains 0, it means that GD diverged. If iteration heatmap shows 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) Metropolis variational energy heatmap, $N = 6$.

(b) Metropolis iterations heatmap, $N = 6$.

Figure 4.22: Variational energy (a) and GD iterations until convergence (b) heatmaps as a functions of step size h and learning rate η for Metropolis algorithm, $N = 6$.

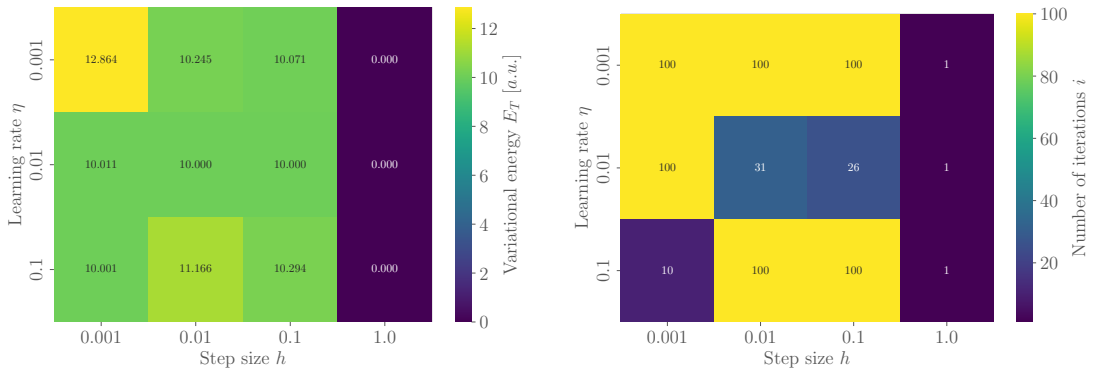
Heatmaps analysis is rather simple. Fist, we discard $(h,\ \eta)$ for with GD diverged. In this case, it is $(1,\ 0.1)$ pair. Then we discard pairs for which GD did not converge and energy is far from it's optimal value. In this case it is all $(h,\ \eta)$ pairs for which $i = 300$. Now only "green" values in energy heatmaps are left. All pairs $(h,\ \eta)$ are acceptable, they generate correct results. Data from iterations heatmaps can help us to choose such $(h,\ \eta)$ pairs that generate correct GS energy using fewest iterations. In this case, two pairs that converge fast and generate correct results are $(h = 0.1,\ \eta = 0.1)$ and $(h = 1,\ \eta = 0.01)$.

The same heatmaps for $N = 6$ Metropolis-Hastings algorithm are presented in Figure 4.23. For this simulation, we set $i = 100$ 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 $(h = 0.001,\ \eta = 0.1)$.

Now we continue with $N = 12$ Metropolis algorithm, heatmaps for which are in the Figure 4.24. For this system, we have thee optimal pairs: $(h = 1,\ \eta = 0.01)$, $(h = 0.01,\ \eta = 0.1)$, and $(h = 0.001,\ \eta = 0.1)$.

And finally we show $N = 12$ Metropolis-Hastings heatmaps in Figure 4.25. 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.

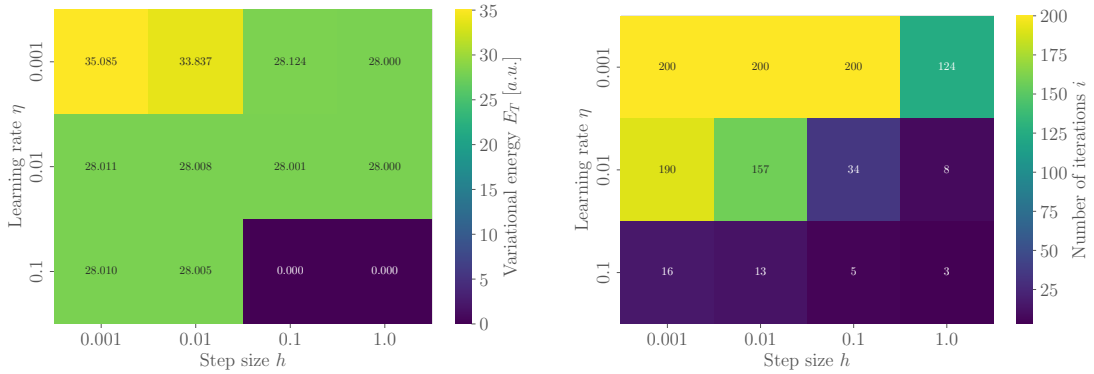
GD for non-interacting system can converge in less than ten iterations, if step size and learning rate are optimally chosen. Both Metropolis and Metropolis-Hastings algorithms generate variational energies exactly equal to analytical GS energies. Therefore, we can conclude that both perform equally well and better algorithm to use for non-interacting systems is Metropolis, simply because it is less computationally expensive.



(a) MH variational energy heatmap, $N = 6$.

(b) MH iterations heatmap, $N = 6$.

Figure 4.23: Variational energy (a) and GD iterations until convergence (b) heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm, $N = 6$.



(a) Metropolis variational energy heatmap, $N = 12$.

(b) Metropolis iterations heatmap, $N = 12$.

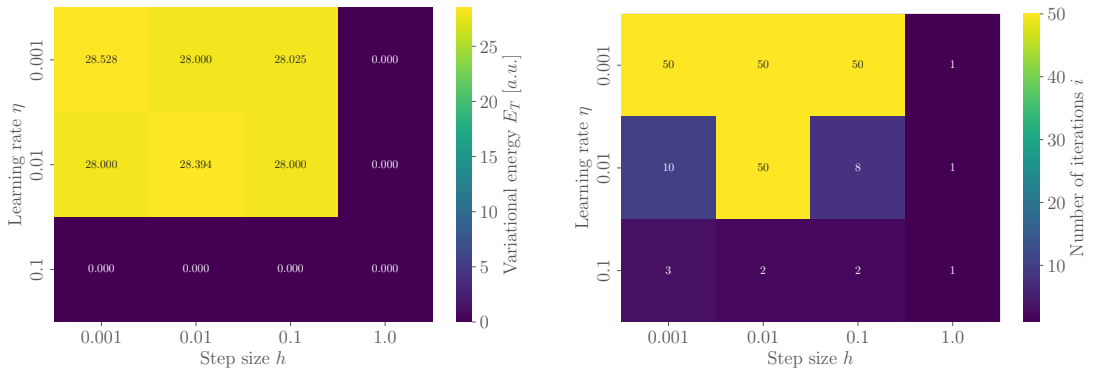
Figure 4.24: Variational energy (a) and GD iterations until convergence (b) heatmaps as a functions of step size h and learning rate η for Metropolis algorithm, $N = 12$.

Finally, we present Table 4.2 that contains number of iterations and runtime for all algorithms. All of them work correctly and converge in less than 11 iterations. In some cases, even after $3-5$ iterations. Numerical and serial algorithms generate identical results, providing that they were implemented correctly. Algorithm execution times even for the largest system is very small, 0.17 seconds. Once again, we see that the parallel algorithm produces the correct results and delivers a significant speedup—approximately a factor of 3.

Now, when we have studied non-interacting systems, we are ready to move on to interacting ones. So far, both algorithms 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 Jastrow 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 to be calculated. Jastrow factors introduce two new terms to local energy, and,



(a) MH variational energy heatmap, $N = 12$.

(b) MH iterations heatmap, $N = 12$.

Figure 4.25: Variational energy (a) and GD iterations until convergence (b) heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm, $N = 12$.

$N = 2$	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 for parameters $n = 10^3$, $m = 10^2$, and $B = 10^{-3}$.

therefore, computation becomes more expensive. Additionally, configuration space where variational parameters belong to 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.

However, there is a feature that can help us to analyze information. Interaction term adds a positive value to the Hamiltonian. If we see that variational energy of interacting systems is smaller than its non-interacting counterpart, it means that simulation generates incorrect result. This is a mathematical explanation. From physics perspective, every system wants 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 particles, 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 their mean relative distance as $\langle r_{12} \rangle = 1/2$. Under these assumptions, GS energy of interacting system is $E_0 = 2 + 1 = 3$. However, this value is only an estimation, and it can be far from true GS energy.

Taut [4] obtains an exact solution for two electrons in a harmonic-oscillator plus Coulomb potential using center-of-mass coordinates. When converted to standard atomic units, this gives $E_0 = 3$. Likewise, Lohne et al. [5] report benchmark GS energies for circular quantum dots (via Coupled Cluster and diffusion Monte Carlo), finding $E_0 = 3$ at $\omega = 1$ and $E_0 = 1.65975$ at $\omega = 0.5$. Therefore, we expect to find variational energy E_T close to $E_0 = 3$.

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 (although having benchmarks). Let us proceed as if the GS energy is not known and see what we can infer from the data we have collected. One strategy is to run short simulations for different step size, 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 energy value. Moreover, we run four simulations: Metropolis and Metropolis-Hastings algorithms for Jastrow and Pade-Jastrow ansatzes. Having four different simulation, we can find correlations that can help to determine GS energy.

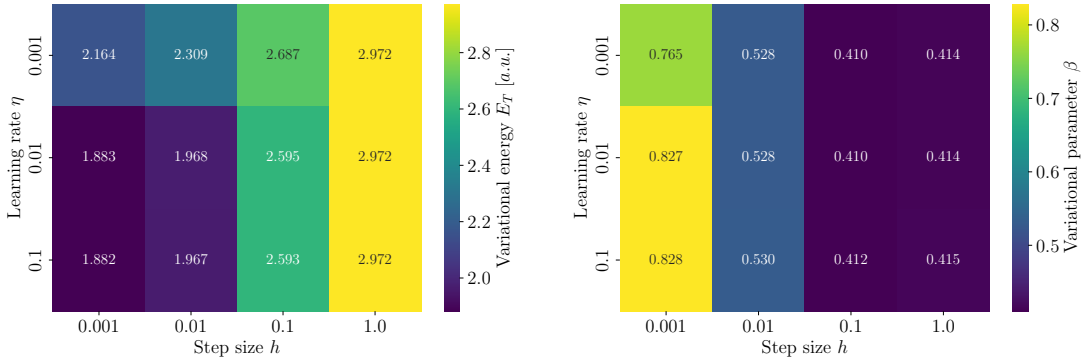
For the first simulation, initial parameters are: $m = 10^2$, $n = 10^3$ and $B = 10^{-2}$. Additionally, we set maximal number of GD iterations to $j = 3000$. OMP algorithms are used to generate results.

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 $\langle \beta(h, \eta) \rangle$. In this case, sample mean values of betas and energy are even more important than number of iterations until convergence.

Let us begin with Jastrow ansatz and Metropolis algorithm. Simulation results are depicted in Figure 4.26. Purple 2×2 box in (a) 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 with GS energy to be $E_0 \approx 3$.

We continue with Metropolis-Hastings algorithm with Jastrow ansatz, see Figure 4.27. Once again we see energy approximately equal to 3, which is a good sign. Simulations with $h = 1$ generate the same variational parameter, therefore, we mark the same pairs of h and η for MH algorithm.

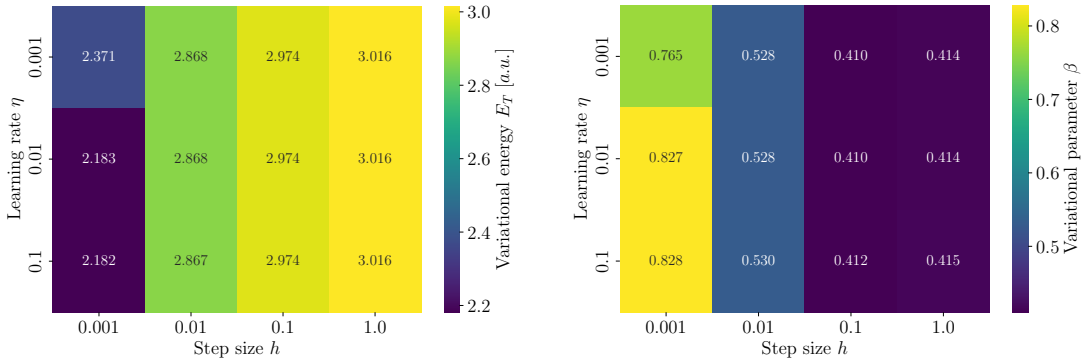
Finally, we consider Metropolis and Metropolis-Hastings with Pade-Jastrow ansatzes as depicted in Figures 4.28 and 4.29. Once again, we see that variational energy values are close to 3, and variational parameters are $\beta \approx 0.38$. Jastrow and Pade-Jastrow factors



(a) Metropolis variational energy heatmap.

(b) Metropolis variational parameter heatmap.

Figure 4.26: Variational energy (a) and variational parameter (b) heatmaps as a functions of step size h and learning rate η for Metropolis algorithm with Jastrow ansatz.



(a) MH variational energy heatmap.

(b) MH variational parameter heatmap.

Figure 4.27: Variational energy (a) and variational parameter (b) heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm with Jastrow ansatz.

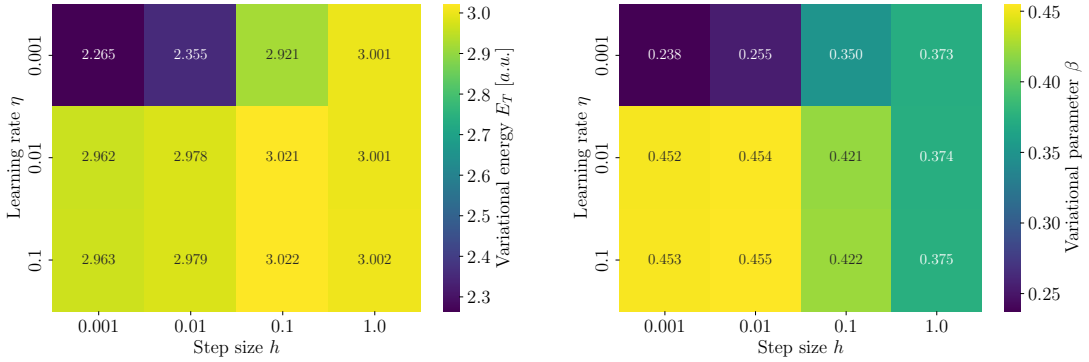
include variational parameters differently, therefore it is natural that their optimal values are different. The most promising parameters is still $h = 1$ independent of η.

Next step is to determine with what leaning rate algorithms converge fastest. To keep it short, we will just state the result: for all algorithms GD with $(h = 1,\ \eta = 0.1)$ converge with fastest rate (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 and 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:

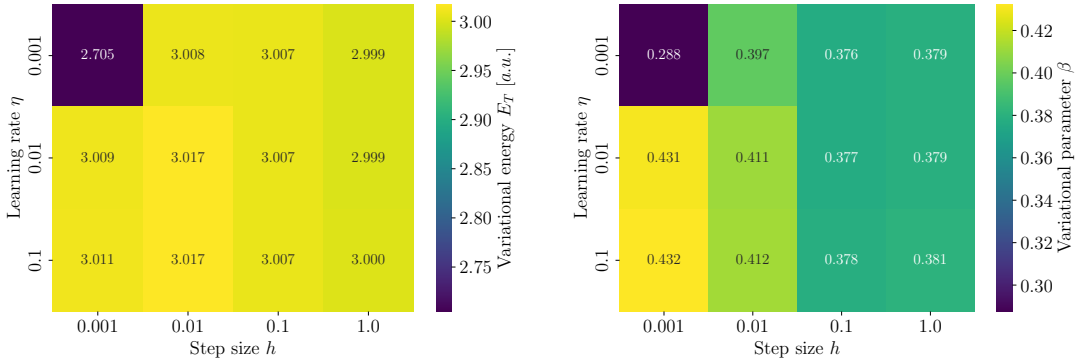
- Jastrow Metropolis GD simulation did not converge in 50 iterations, $\beta_{12} = 0.41398$, $E_0 = 3.01644$, $\langle \nabla_{\beta_{12}} \Psi_T \rangle = 0.0111455$;
- Jastrow MH GD simulation converged in 31 iterations, $\beta_{12} = 0.416592$, $E_0 = 3.01408$, $\langle \nabla_{\beta_{12}} \Psi_T \rangle = 9.7 \cdot 10^{-6}$;
- Pade-Jastrow Metropolis GD simulation did not converge in 50 iterations, $\beta_{12} = 0.341$, $E_0 = 3.00033$, $\langle \nabla_{\beta} \Psi_T \rangle = 1.6 \cdot 10^{-4}$;



(a) Metropolis variational energy heatmap.

(b) Metropolis variational parameter heatmap.

Figure 4.28: Variational energy (a) and variational parameter (b) heatmaps as a functions of step size h and learning rate η for Metropolis algorithm with Pade-Jastrow ansatz.



(a) MH variational energy heatmap.

(b) MH variational parameter heatmap.

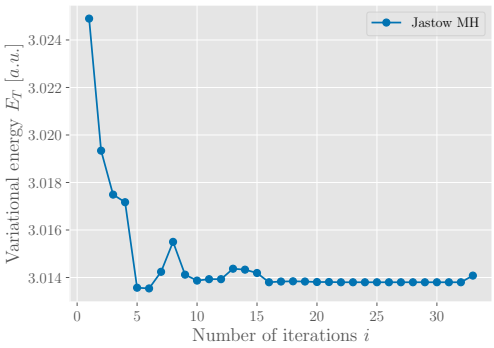
Figure 4.29: Variational energy (a) and variational parameter (b) heatmaps as a functions of step size h and learning rate η for Metropolis-Hastings algorithm with Pade-Jastrow ansatz.

- Pade-Jastrow MH GD simulation did not converge in 50 iterations, $\beta_{12} = 0.341$, $E_0 = 3.00110$, $\langle \nabla_\beta \Psi_T \rangle = 7 \cdot 10^{-5}$.

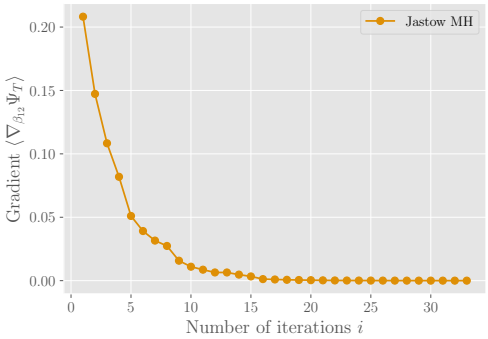
In Figure 4.30, we show how variational energy and gradient change during GD VMC procedure for Jastrow MH algorithm. Gradient smoothly decrees, while energy has some small peaks. The energy rises slightly in the final iteration because we used a longer burn-in period and more VMC samples there, allowing the variational energy to shift a bit.

We can conclude that GS energy for this system is $E_0 \approx 3$, and MH algorithm shows better performance – observable smoothly converges to this value. During computation with Metropolis algorithm, gradient decreases to some relatively small value and jumps back and forth. Also, our estimation of GS energy turned out to be correct.

Now that we have carried out our own analysis, we can compare our results against established benchmarks. In particular, Taut's analytical solution [4] and Lohne's numerical benchmarks [5] both give a ground-state energy of $E_0 = 3$ As the Table 4.3 below shows, each of the algorithms we have implemented reproduces this value—within statistical uncertainty—demonstrating that our variational methods accurately recover the true E_0.



(a) MH Jastrow variational energy.



(b) MH Jastrow Gradient.

Figure 4.30: Variational energy (a) and gradient $\langle \nabla_\alpha \Psi_T \rangle$ (b) as a function of number of iterations for Metropolis-Hastings with Jastrow ansatz.

Method	E_T	$\delta E = \frac{\lvert E_T - E_0 \rvert}{E_0}$
Analytic benchmark (Taut [4])	3.00000	0.00000
DMC benchmark (Lohne [5])	3.00000(1)	0.00000
Jastrow Metropolis GD	3.01644	0.00548
Jastrow MH GD	3.01408	0.00469
Pade–Jastrow Metropolis GD	3.00033	0.00011
Pade–Jastrow MH GD	3.00110	0.00036

Table 4.3: Ground-state energy estimates and relative errors for two interacting fermions in 2D.

Finally, we will consider system with omega different from one. Lohne [5] has a benchmark for $\omega = 0.5$, and, therefore, we will study the system with $\omega = 0.5$, using optimal parameters we found earlier and compare with the benchmark GS energy value.

In Figure 4.31 we show how variational energy changes during MH GD procedure with Pade-Jastrow ansatz. This simulation produced

$$E_T = 1.66016.$$

The relative error is

$$\delta E = \frac{|E_T - E_0|}{E_0} = 0.00024,$$

demonstrating that the result is highly accurate.

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;

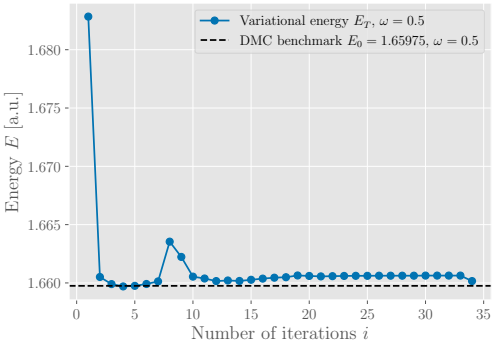


Figure 4.31: MH Pade-Jastrow variational energy E_T as a function of number of iterations for $\omega = 0.5$.

- Run long GD simulation with algorithms that showed best performance to find GS energy.

For the second closed shell, $n_{\text{pairs}} = 15$, and therefore Jastrow factor contains fifteen variational parameters β_{ij}. We update each variational parameter β_{ij} after every GD iteration.

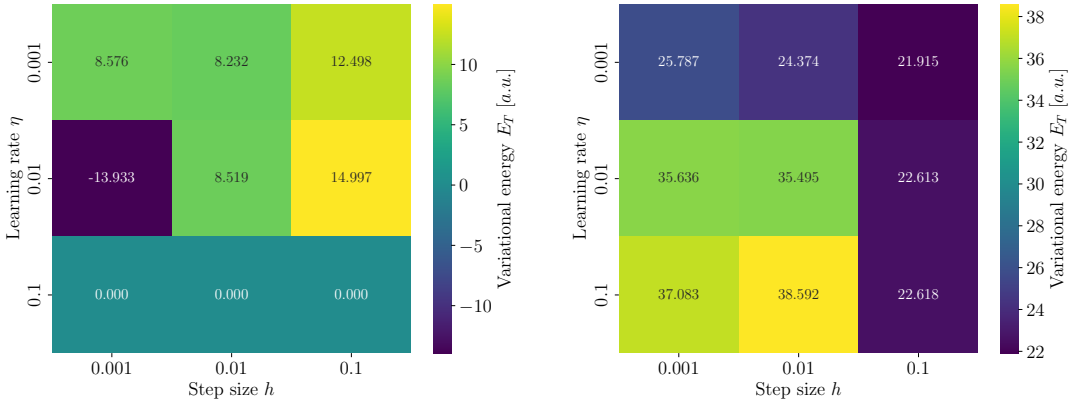
As a first step, we performed a short VMC simulation for all OMP parallelized algorithms with parameters $m = 10^2$, $n = 10^2$, and $\alpha = 1/2$. As initial Pade-Jastrow variational parameter, we chose optimal parameter for $N = 2$ interacting system, namely $\beta = 0.4$. All Jastrow factor variational 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 by Metropolis simulations only. On heatmaps depicted in Figure 4.32, we can see that Jastrow estimation of GS energy is not good. Pade-Jastrow simulation, on the other 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 wrong ansatz, it just says that 110 VMC iterations is not enough for it to generate good result. 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 Metropolis-Hastings algorithm. Energy heatmaps for the same number of VMC iterations as in Figure 4.29 are depicted in Figure 4.30. Metropolis-Hastings method 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 Jastrow and Pade-Jastrow 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.

By analyzing similar heatmaps for L2 norm and number of iterations until convergence, we conclude the following:

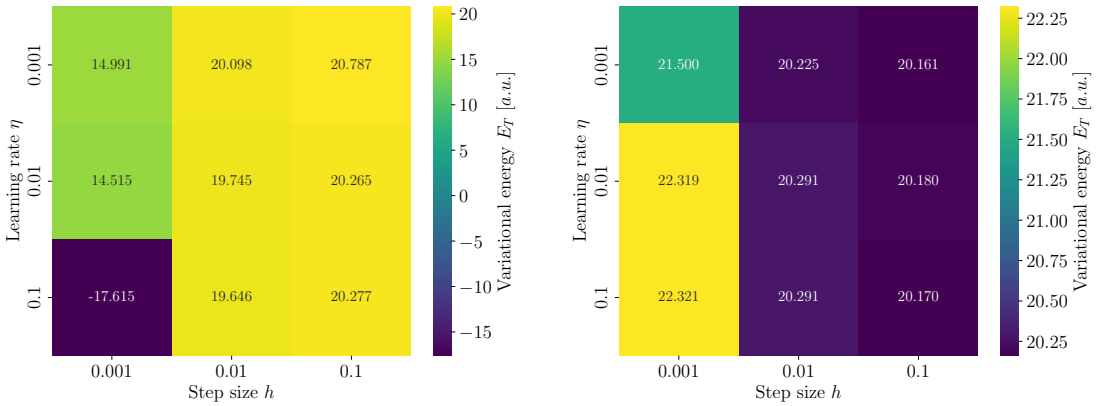
- Metropolis algorithm with Jastrow factor ansatz shows worse performance than Metropolis algorithm with Pade-Jastrow ansatz. Therefore, we will study only Pade-Jastrow Metropolis algorithm. Optimal parameters are ($h = 0.1$, $\eta = 0.1$);



(a) Jastrow Metropolis energy heatmap.

(b) Pade-Jastrow Metropolis energy heatmap.

Figure 4.32: Variational energy heatmap with Jastrow ansatz (a) and Pade-Jastrow ansatz (b) 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 4.33: Variational energy heatmap with Jastrow ansatz (a) and Pade-Jastrow ansatz (b) as a functions of step size h and learning rate η for Metropolis-Hastings algorithm.

- MH algorithm with Jastrow ansatz shows good performance. It converges to $B = 10^{-1}$, 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 $B = 10^{-2}$ 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 of $t \approx 10$ $[s]$. Variational energy is $E_T = 20.3582$. 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 of $t \approx 25$ $[s]$. 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 of $t \approx 35$ $[s]$. Estimated GS energy is $E_0 = 20.1958$. Optimal variational parameter is $\tilde{\beta} = 0.46676$.

From the obtained data we can conclude that GS energy of the 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 results than Metropolis one when number of VMC iterations is small. When number of VMC cycles is relatively big, both algorithm estimate GS energy well. Algorithms were able to converge fast due to optimal choice of parameters, which was based on analysis of short GD VMC computations.

Lohne's [5] DMC benchmark for the second closed shell GS energy with $\omega = 1$ is $E_0 = 20.1597$. If we compare our results with the benchmark, we can conclude that Metropolis algorithm with Pade-Jastrow factor is the most close to the benchmark, showing $E_T = 20.178$.

In Table 4.4, we compare our long GD VMC estimates with the benchmark value and report the corresponding relative errors. At first glance the absolute errors ($|E_T - E_0|$) may appear large, but the relative errors are of the same order as in the two-particle case. This is simply because the GS energy for $N = 6$ is $E_0 \approx 20$, rather than $E_0 = 3$ for $N = 2$. It is inherently more difficult to pinpoint a larger energy with the same fractional accuracy.

Method	E_T	$\delta E = \frac{\lvert E_T - E_0 \rvert}{E_0}$
DMC benchmark (Lohne [5])	20.1597	0.00000
Jastrow MH GD	20.3582	0.00984
Pade–Jastrow Metropolis GD	20.178	0.000090
Pade–Jastrow MH GD	20.1958	0.00179

Table 4.4: Ground-state energy estimates and relative errors for six interacting fermions in 2D.

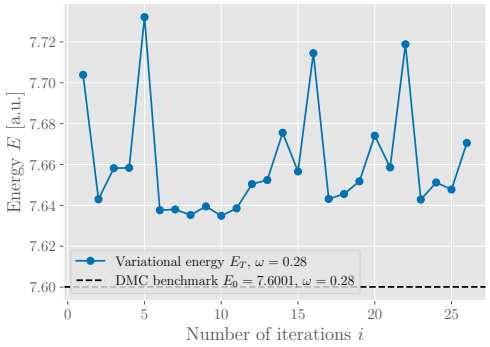
Finally, we perform long GD VMC simulation with optimal parameters determined earlier to study if our simulation can recreate GS energy for $\omega = 0.28$ and $\omega = 0.5$. Lohne [5] has benchmark exactly for these values of frequency. We show how variational energy changes during GS iterations for there values of ω in the Figure 4.34. For $\omega = 0.28$, GD did not converge in 25 iterations, while for $\omega = 0.5$, GD did converge in 11 iterations to $B = 7 \cdot 10^{-5}$.

For this set of simulations, we used Metropolis–Hastings with the Pade–Jastrow trial wavefunction. As shown in the figures, the variational energy does not fully converge to the benchmark values. However, our chosen initial parameters already yield a reasonably accurate GS estimate, hence the apparent discrepancy over this limited range. In practice, changing the oscillator frequency ω also changes the ideal step size and learning rate, so we must re-tune those parameters for each new ω.

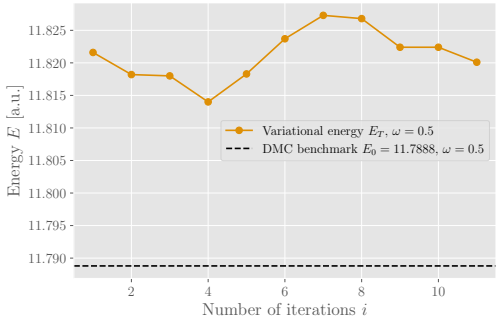
Even without perfect convergence, the specific results

$$E_T(0.28) = 7.67055, \; E_T(0.5) = 11.8201, \; \delta E(0.28) = 0.0092, \; \delta E(0.5) = 0.0026,$$

show that the relative errors remain of the same order. In other words, once the step size and learning rate are optimized, even a relatively short VMC run can produce a good estimate of the GS energy. These runs are straightforward to set up but can be computationally expensive to tune for each ω.



(a) MH Pade-Jastrow variational energy E_0 for $\omega = 0.28$.



(b) MH Pade-Jastrow variational energy E_0 for $\omega = 0.5$.

Figure 4.34: Metropolis-Hastings with Pade-Jastrow ansatz variational energy for $\omega = 0.28$ (a) and $\omega = 0.5$ (b) compared with benchmarks from Lohne [5].

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, analysis of heatmaps, long VMC.

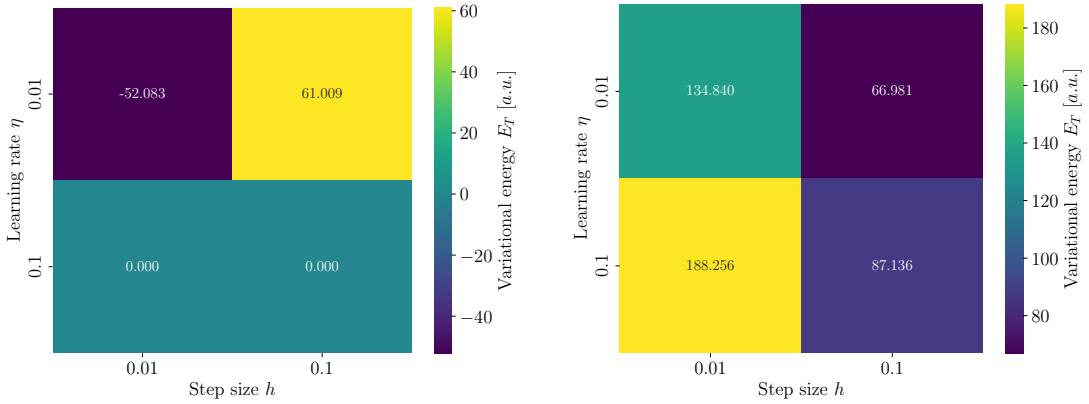
For this system, we need to choose step size and learning rate carefully, because each VMC GD can take hours. We have seen that when system becomes bigger, solving algorithms work better with smaller parameters. Therefore, we will test algorithms only for $h = \eta = 0.1, 0.01$.

For the following simulation, we initialized parameters as $m = 10^3$, $n = 5 \cdot 10^3$, and $\alpha = 1/2$ doing 25 GD iterations. We initialized starting variational parameters as $\beta = 0.385$ and $\beta_{ij} = 0.4$. This time we decided not to choose optimal variational parameters for Jastrow factor to see if distinct values would generate better results.

Energy heatmaps for this simulation are presented in Figures 4.35 and 4.36. In contrast to the second closed-shell case, the Metropolis variational energies here give us little insight into the actual GS energy. However, by analyzing L2 norm, number of iterations until convergence and energy together, we concluded 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.

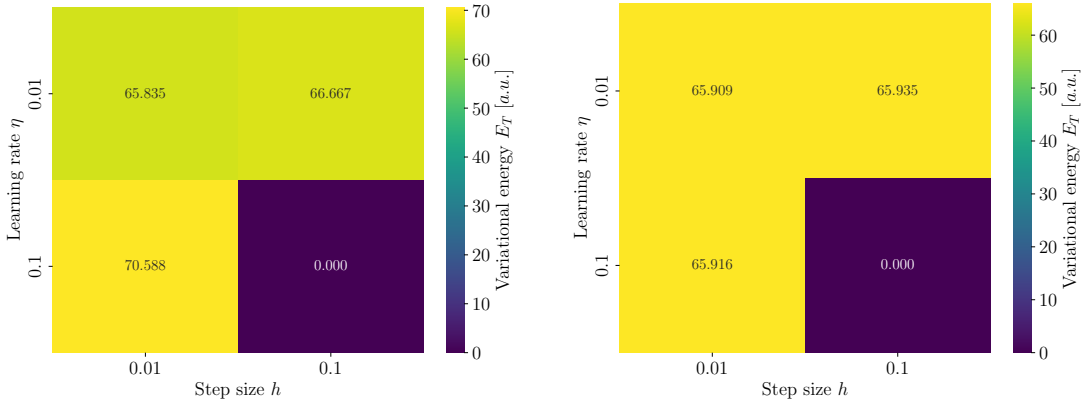
This time we will perform long GD VMC simulations only with Metropolis-Hastings algorithm. Reason for that is that Metropolis needs approximately $n = 10^5$ GD VMC iterations to generate decent results. Even with parallel algorithms, it would take days to simulate it on a normal computer. And after all, we aim to estimate GS energy accurately and fast. Therefore, MH is our choice for this system.



(a) Jastrow Metropolis energy heatmap.

(b) Pade-Jastrow Metropolis energy heatmap.

Figure 4.35: Variational energy heatmap with Jastrow ansatz (a) and Pade-Jastrow ansatz (b) 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 4.36: Variational energy heatmap with Jastrow ansatz (a) and Pade-Jastrow ansatz (b) as a functions of step size h and learning rate η for Metropolis-Hastings algorithm.

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 $m_0 = 5 \cdot 10^5$ with step size and 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 variational energy is $E_T = 66.3778$. 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 big 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 variational energy is $E_0 = 65.83$. Once again, variational energy is consistent with other methods. The fact that energy obtained by different methods is approximately equal indicates 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 twelve iterations only.

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!

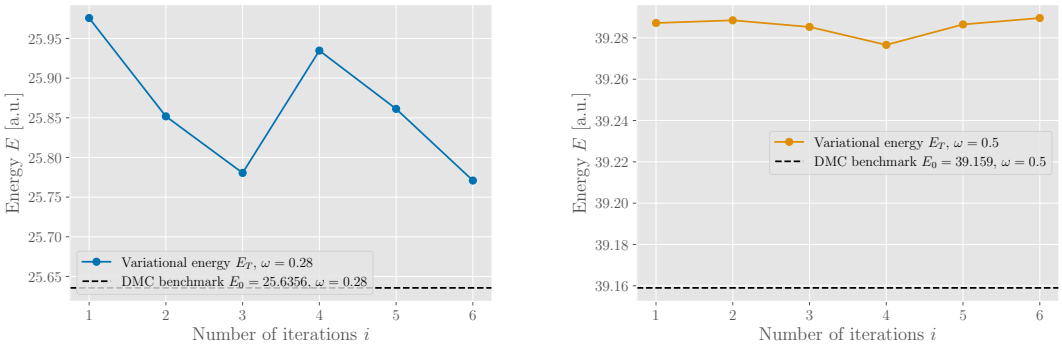
Now we want to compare our results with the benchmark from [5]. DMC simulation with optimized ansatz for this system yield $E_0 = 65.700$. In Table 4.5, we compare our results with the benchmark and present relative errors. Again, even though the absolute error appears large, the relative error remains similarly small—which is excellent. Do not be misled by the absolute error!

Method	E_T	$\delta E = \dfrac{\lvert E_T - E_0 \rvert}{E_0}$
DMC benchmark (Lohne [5])	65.700	0.00000
Jastrow MH GD	66.3778	0.0103
Pade–Jastrow MH GD	65.83	0.0019

Table 4.5: Ground-state energy estimates and relative errors for six interacting fermions in 2D.

As the final element of this research, we study omega-dependence for $N = 12$ system and compare them with the benchmarks from [5]: $E_0(0.28) = 25.6356$, and $E_0(0.5) = 39.159$.

In the Figure 4.37, we show Metropolis-Hastings variational energy as a function of number of GD iterations. For this we used: $h = 0.1$, $\eta = 0.01$, $m_0 = 5 \cdot 10^3$, $n_0 = 5 \cdot 10^4$, $m_1 = 5 \cdot 10^4$, and $m_1 = 5 \cdot 10^5$. Both algorithms converged to L2= 10^{-2} in 5 iterations. Once again we observe that initial estimation of GS energy is pretty accurate, because we use optimal h and η determined for $\omega = 1$. Variational energies are close to the benchmark values.



(a) MH Pade-Jastrow variational energy E_T for $\omega = 0.28$.

(b) MH Pade-Jastrow variational energy E_T for $\omega = 0.5$.

Figure 4.37: Metropolis-Hastings with Pade-Jastrow ansatz variational energy for $\omega = 0.28$ (a) and $\omega = 0.5$ (b) compared with benchmarks from Lohne [5].

Finally, we state the result and relative errors:

$$E_T(0.28) = 25.7709,\ E_T(0.5) = 39.2896,\ \delta E(0.28) = 0.0052,\ \delta E(0.5) = 0.0033.$$

Relative errors are of the same order as for the systems we previously considered.

In summary, our VMC implementations consistently reproduce benchmark energies across all tested systems. These results validate the accuracy and robustness of our implementation, providing confidence in its application to more complex many-body problems.

Chapter 5

Conclusion

At the outset of this research, we established two primary objectives: to estimate the ground-state (GS) energy of selected systems and to develop a flexible, object-oriented Variational Monte Carlo (VMC) algorithm.

On the computational side, we developed a `C++` implementation of the VMC algorithm. We succeeded in creating a flexible VMC solver, in the sense that the Hamiltonian, trial wave function, and sampling method can each be swapped out easily. Moreover, our framework can be extended to study additional systems simply by implementing the appropriate subclasses. We employed the `autodiff` API [1] for automatic differentiation to verify our analytic derivatives. Although this introduces some overhead, it confirmed the correctness of our hand-coded expressions. Finally, we successfully parallelized the VMC solver using OpenMP (OMP) [2] and MPI [3]. By providing an MPI implementation, our code can now run on high-performance clusters, enabling studies of much larger systems. Both parallelization strategies achieved a 2-3x speedup of four cores. Thus, we conclude that we have met out original goal of developing a versatile, high-performance VMC implementation.

On the physical side, we estimated GS energies for all systems with high precision. For the non-interacting bosons and the non-interacting first closed shell, we derived analytical solutions and confirmed that our VMC results agree with them. For the second and third closed shells, the ground-state energy is a sum of single-particle energies, and our computational results align with them. In the interacting fermion case, we augmented the Slater determinant ansatz with a Jastrow or a Pade-Jastrow correlation factor to capture electron correlations. For interacting closed shells, we compared our VMC energies to Taut's [4] analytical solution for the first closed shell and to Lohne et al., [5] for larger shells. In every case, our estimates lie within a relative error of $10^{-2} - 10^{-4}$, demonstrating that each algorithm and correlation factor performed extremely well. Below are the key results:

- For the non-interacting bosonic systems of N particles in d dimensions, both the Metropolis and the Metropolis-Hastings sampling algorithms estimated the ground-state energy $E_0 = \frac{Nd}{2}\omega$ exactly in 10^4 VMC iterations. Starting from $\alpha_0 = 0.1$ they converged in less than 100 Gradient Descent (GD) iterations. Additionally, we were able to estimate GS energy for different values of frequency ω. Since the Metropolis-Hasting algorithm is more computationally demanding, we conclude that for this system the Metropolis algorithm is the optimal algorithm to use. For this system, both OMP and MPI showed a speedup of three times.

- For the non-interacting fermionic system of $N = 2, 6$ and 12 particles, both

sampling algorithms showed equally good performance. They were able to recreate exact GS energies $E_0 = 2$, $E_0 = 10$, and $E_0 = 28$ for $N = 2$, $N = 6$ and $N = 12$ respectively using only 10^4 VMC iterations. Starting from $\alpha_0 = 0.1$, the algorithms converged in less than 30 GD iterations. For this system, we restored GS energies for different values of omega correctly. Again, the parallel strategies showed approximately a speedup of three times and the Metropolis method is the most optimal.

- For the first closed shell, the Metropolis algorithm with the Pade-Jastrow ansatz generated results close to Taut's [4] analytical solution $E_0 = 3$ – $E_T = 3.00033$ with relative error $\delta E(1) = 0.00036$. When studying the system with $\omega = 0.5$, the Metropolis-Hastings algorithm with the Pade-Jastrow ansatz generated $E_T = 1.66016$, while the benchmark value of [5] is $E_0 = 16.9574$, giving a relative error $\delta E(0.5) = 0.00024$.

- For the second closed shell, the Metropolis-Hastings algorithm with the Pade-Jastrow ansatz showed the best performance: $E_T = 20.178$. The benchmark from [5] for $\omega = 1$ is $E_0 = 20.1597$, so that $\delta E(1) = 0.00009$. We also compared our results for $\omega = 0.28$ and $\omega = 0.5$ with benchmark energies for these values, and obtained results with relative errors $\Delta E(0.28) = 0.0092$ and $\delta E(0.5) = 0.0026$.

- For the third closed shell, once again, the Metropolis-Hastings algorithm with the Pade-Jastrow ansatz were closest to the benchmark values of [5]: $E_T = 65.83$ and $E_0 = 65.7$ with a relative error $\delta E(1) = 0.0019$. As for the previous shell, we compared our estimations with benchmarks for $\omega = 0.28$ and $\omega = 0.5$, and our results have relative errors $\delta E(0.0052)$ and $\delta E(0.5) = 0.0033$.

Thus, we conclude that for the non-interacting systems, the Metropolis algorithm is the optimal algorithm to use, while for the interacting fermionic systems, the Metropolis-Hastings algorithm with the Pade-Jastrow ansatz gives the best result. The Jastrow ansatz showed worse convergence than the Pade-Jastrow ansatz, but only before we used a small number of GD iterations. It is able to estimate the GS energy with a good accuracy when bigger numbers of VMC iterations are used.

There are many directions for extending this work:

- Statistical-error estimation: Implement blocking or bootstrap methods to compute errors of VMC energies and gradients.

- Enhanced correlation factors: Explore alternative Jastrow factors and/or include three-body terms.

- Neural-network wavefunctions: Replace or augment analytic ansatzes with neural-network based trial functions to improve energy estimates in strongly correlated regimes.

- Larger and non-closed shells: Apply the framework to higher closed shells: ($N >$ 12) and to open-shell configurations.

- Interacting bosons: Extend to bosonic systems with repulsive or attractive interactions (e.g. dipolar) and study phenomena such as condensation or vortex formation.

- New Hamiltonians: Use the same object-oriented infrastructure to investigate other potentials (e.g. anharmonic traps, double wells) or lattice Hamiltonians.

- Alternative samplers: Substitute or compare with other Monte Carlo schemes (e.g. hybrid Monte Carlo) to improve sampling efficiency.
- Single-particle moves: Change the update strategy to move one particle at a time (rather than all particles simultaneously) in order to reduce recomputation in the Slater determinant and accelerate the code.

By pursuing any of these avenues—and combining several at once—we can further broaden the scope and power of our VMC implementation.

Chapter 6

Appendix

6.1 Harmonic Oscillator Units and Atomic Units

In this appendix, we derive dimensionless Hamiltonian for the systems we study. If we consider non-interacting harmonic-oscillator (HO) problem, Hamiltonian (2.18) 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 (2.14) 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 represents gradient with respect to ρ coordinates. Putting this back to equation:

$$\begin{aligned} &\sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m}\frac{1}{a^2}\nabla_{\rho_i}^2\psi(\rho) + \frac{m\omega^2}{2}a^2\rho^2\psi(\rho) \right) - E\psi(\rho) = 0 \Big| \times \frac{ma^2}{\hbar^2}; \\ &\sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_{\rho_i}^2\psi(\rho) + \frac{1}{2}\frac{m^2\omega^2}{\hbar^2}a^4\rho^2\psi(\rho) \right) - \frac{ma^2}{\hbar^2}E\psi(\rho) = 0. \end{aligned}$$

Choose constant a so that:

$$\frac{m^2\omega^2}{\hbar^2}a^4 = 1 \implies a = a_{\text{HO}} = \sqrt{\frac{\hbar}{m\omega}}.$$

Constant 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

$$\hat{H} = \sum_{i=1}^{N} -\frac{1}{2}\nabla^2_{\rho_i} + \frac{1}{2}\rho^2. \tag{6.1}$$

Therefore, if we just set $\hbar = \omega = m = 1$ in the original Hamiltonian, we obtain Hamiltonian (6.1). We will refer to these units as "HO units".

Now, consider Hamiltonian with only electrostatic Coulomb's potential (2.22) without the second term, related to nuclei:

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\nabla^2_i + \sum_{j>i}^{N}\frac{e^2}{4\pi\varepsilon_0}\frac{1}{r_{ij}}\right).$$

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:

$$\begin{aligned} &\sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\frac{1}{a^2}\nabla^2_{\rho_i}\psi(\rho) + \sum_{j>i}^{N}\frac{e^2}{4\pi\varepsilon_0}\frac{1}{a}\frac{1}{\rho_{ij}}\psi(\rho)\right) - E\psi(\rho) = 0 \;\Big|\times\frac{ma^2}{\hbar^2}; \\ &\sum_{i=1}^{N}\left(\nabla^2_{\rho_i}\psi(\rho) + \sum_{j>i}^{N}\frac{e^2ma}{4\pi\varepsilon_0\hbar^2}\frac{1}{\rho_{ij}}\psi(\rho)\right) - \frac{ma^2}{\hbar^2}E\psi(\rho) = 0. \end{aligned}$$

Choose constant a so that

$$\frac{e^2ma}{4\pi\varepsilon_0\hbar^2} = 1 \implies a = a_o = \frac{4\pi\varepsilon_0\hbar^2}{me^2}.$$

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

$$\hat{H} = \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_{\rho_i}^2 + \sum_{j>i}^{N}\frac{1}{\rho_{ij}}\right) \tag{6.2}$$

This Hamiltonian can be obtained by setting $\hbar = 4\pi\epsilon_0 = e = m = 1$. We will refer to this choice of constants as atomic units.

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 variables:

$$\rho = \frac{\mathbf{r}}{a}, \quad \mathbf{r} = a\rho.$$

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, $a_{\text{HO}} = \sqrt{\frac{\hbar}{m\omega}}$. 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 [19]. It's numerical value is

$$A(\omega) \approx \frac{(1.602\cdot10^{-19})^2}{4\cdot3.141\cdot8.854\cdot10^{-12}}\sqrt{\frac{9.109\cdot10^{-31}}{(1.054\cdot10^{-34})^3\omega}} = 2.034\cdot10^8\omega^{-1/2}.$$

However, this choice is not optimal for our problem. In these units we can tune interaction strength by varying oscillator frequency. More natural choice is the 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 [19], numerical value of Bohr's radius is $a_o \approx 5.291\cdot10^{-11}$. Numerical value of $B(\omega)$ is

$$B(\omega) \approx \frac{(9.109\cdot10^{-31})^2}{2(1.054\cdot10^{-34})^2}(5.291\cdot10^{-11})^4\omega^2 = 2.926\cdot10^{-34}\omega^2.$$

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 and 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 the 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

$$\Delta\psi_1(x,y) = \frac{\partial}{\partial x}\left(-2\alpha x e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left(-2\alpha y e^{-\alpha r^2}\right) = -2\alpha e^{-\alpha r^2} + 4\alpha^2 x^2 e^{-\alpha r^2} - 2\alpha e^{-\alpha r^2} + 4\alpha^2 y^2 e^{-\alpha r^2}.$$

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 only 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

$$\begin{aligned}\nabla\psi_2(x,y) &= \left(\frac{\partial}{\partial x}\left[xe^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[xe^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right) \\ &= \left((1-2\alpha x^2)e^{-\alpha r^2}\cdot\mathbf{e}_x - 2\alpha xy e^{-\alpha r^2}\cdot\mathbf{e}_y\right),\end{aligned}$$

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:

$$\begin{aligned}\nabla\psi_3(x,y) &= \left(\frac{\partial}{\partial x}\left[ye^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[ye^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right) \\ &= \left(-2\alpha xy e^{-\alpha r^2}\cdot\mathbf{e}_x + (1-2\alpha y^2)e^{-\alpha r^2}\cdot\mathbf{e}_y\right),\end{aligned}$$

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

$$\begin{aligned}&\Delta\psi_2(x,y) = \frac{\partial}{\partial x}\left((1-2\alpha x^2)e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left(-2\alpha xy e^{-\alpha r^2}\right) \\ &= \frac{\partial}{\partial x}\left(e^{-\alpha r^2} - 2\alpha x^2 e^{-\alpha r^2}\right) - \frac{\partial}{\partial y}\left(2\alpha xy e^{-\alpha r^2}\right) \\ &= -2\alpha x e^{-\alpha r^2} - 4\alpha x e^{-\alpha r^2} + 4\alpha^2 x^3 e^{-\alpha r^2} - 2\alpha x e^{-\alpha r^2} + 4\alpha^2 xy^2 e^{-\alpha r^2} \\ &= \left(-2\alpha x - 4\alpha x + 4\alpha^2 x^3 - 2\alpha x + 4\alpha^2 xy^2\right)e^{-\alpha r^2} \\ &= \left(-8\alpha x + 4\alpha^2 x(x^2+y^2)\right)e^{-\alpha r^2}.\end{aligned}$$

Final expression is

$$\Delta\psi_2(x,y) = x\left(-8\alpha + 4\alpha^2 r^2\right)e^{-\alpha r^2}. \tag{6.16}$$

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

$$\Delta\psi_3(x,y) = y\left(-8\alpha + 4\alpha^2 r^2\right)e^{-\alpha r^2}. \tag{6.17}$$

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 $\psi_4(x,y)$:

$$\begin{aligned}\nabla\psi_4(x,y) &= \left(\frac{\partial}{\partial x}\left[(x^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[(x^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(2xe^{-\alpha r^2} - 2\alpha x^3 e^{-\alpha r^2} + 2\alpha x e^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(2\alpha y(1-x^2)e^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right).\end{aligned}$$

So that

$$\nabla\psi_4(x,y) = \left(\mathbf{e}_x\left(2x - 2\alpha x^3 + 2\alpha x\right) + \mathbf{e}_y\left(2\alpha y - 2\alpha x^2 y\right)\right)e^{-\alpha r^2}. \tag{6.21}$$

Continue with ψ_5:

$$\begin{aligned}\nabla\psi_5(x,y) &= \left(\frac{\partial}{\partial x}\left[(y^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[(y^2-1)e^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(2\alpha x(1-y^2)e^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(2ye^{-\alpha r^2} - 2\alpha y^3 e^{-\alpha r^2} + 2\alpha y e^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right).\end{aligned}$$

Final form is

$$\nabla\psi_5(x,y) = \left(\left(2\alpha x(1-y^2)\right)\cdot\mathbf{e}_x + \left(2y - 2\alpha y^3 + 2\alpha y\right)\cdot\mathbf{e}_y\right)e^{-\alpha r^2}. \tag{6.22}$$

The final gradient of ψ_6 is:

$$\begin{aligned}\nabla\psi_6(x,y) = \nabla xye^{-\alpha r^2} &= \left(\frac{\partial}{\partial x}\left[xye^{-\alpha r^2}\right]\cdot\mathbf{e}_x + \frac{\partial}{\partial y}\left[xye^{-\alpha r^2}\right]\cdot\mathbf{e}_y\right)\\ &= \left(\left(ye^{-\alpha r^2} - 2\alpha x^2 y e^{-\alpha r^2}\right)\cdot\mathbf{e}_x + \left(xe^{-\alpha r^2} - 2\alpha x y^2 e^{-\alpha r^2}\right)\cdot\mathbf{e}_y\right),\end{aligned}$$

so that

$$\nabla\psi_6(x,y) = \left(\mathbf{e}_x\left(y - 2\alpha x^2 y\right) + \mathbf{e}_y\left(x - 2\alpha x y^2\right)\right)e^{-\alpha r^2}. \tag{6.23}$$

Laplacian vectors can be derived as

$$\begin{aligned}\Delta\psi_4(x,y) &= \left(\frac{\partial}{\partial x}\left(2xe^{-\alpha r^2} - 2\alpha x^3 e^{-\alpha r^2} + 2\alpha x e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left((1-x^2)2\alpha y e^{-\alpha r^2}\right)\right)\\ &= \Big(2e^{-\alpha r^2} - 4\alpha x^2 e^{-\alpha r^2} - 6\alpha x^2 e^{-\alpha r^2} + 4\alpha^2 x^4 e^{-\alpha r^2} + 2\alpha e^{-\alpha r^2} - 4\alpha^2 x^2 e^{-\alpha r^2}\\ &\quad + (1-x^2)2\alpha e^{-\alpha r^2} + (x^2-1)4\alpha^2 y^2 e^{-\alpha r^2}\Big)\\ &= \Big(2 - 4\alpha x^2 - 6\alpha x^2 + 4\alpha^2 x^4 + 2\alpha - 4\alpha^2 x^2 + (1-x^2)2\alpha + (x^2-1)4\alpha^2 y^2\Big)e^{-\alpha r^2}\\ &= \Big(2 - 4\alpha x^2 - 6\alpha x^2 + 4\alpha^2 x^4 + 2\alpha - 4\alpha^2 x^2 + 2\alpha - 2\alpha x^2 + 4\alpha^2 x^2 y^2 - 4\alpha^2 y^2\Big)e^{-\alpha r^2}.\end{aligned}$$

Finally,

$$\Delta\psi_4(x,y) = \Big(2 - 12\alpha x^2 + 4\alpha^2 x^2 r^2 + 4\alpha - 4\alpha^2 r^2\Big)e^{-\alpha r^2} \tag{6.24}$$

Applying the same trick we used to obtain $\Delta\psi_3$ yields:

$$\Delta\psi_5(x,y) = \Big(2 - 12\alpha y^2 + 4\alpha^2 y^2 r^2 + 4\alpha - 4\alpha^2 r^2\Big)e^{-\alpha r^2}. \tag{6.25}$$

And the final Laplacian can be found as

$$\begin{aligned}
\Delta\psi_6(x,y) &= \left(\frac{\partial}{\partial x}\left(ye^{-\alpha r^2} - 2\alpha x^2 y e^{-\alpha r^2}\right) + \frac{\partial}{\partial y}\left(xe^{-\alpha r^2} - 2\alpha x y^2 e^{-\alpha r^2}\right)\right) \\
&= \left(4\alpha^2 x^3 y e^{-\alpha r^2} - 2\alpha x y e^{-\alpha r^2} - 4\alpha x y e^{-\alpha r^2} + 4\alpha^2 x y^3 e^{-\alpha r^2} - 2\alpha x y e^{-\alpha r^2} - 4\alpha x y e^{-\alpha r^2}\right) \\
&= \Big(4\alpha^2 x^3 y - 2\alpha x y - 4\alpha x y + 4\alpha^2 x y^3 - 2\alpha x y - 4\alpha x y\Big)e^{-\alpha r^2}.
\end{aligned}$$

The final form of $\Delta\psi_6$ is

$$\Delta\psi_6(x,y) = \Big(-12\alpha x y + 4\alpha^2 x y r^2\Big)e^{-\alpha r^2}. \tag{6.26}$$

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 (2.28):

$$\langle E_L\rangle = \frac{\langle\Psi|\,\hat{H}\,|\Psi\rangle}{\langle\Psi|\Psi\rangle},$$

we derive 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:

$$\begin{aligned}
\frac{\partial\langle E_L\rangle}{\partial\alpha} &= \frac{\partial}{\partial\alpha}\left(\frac{\langle\Psi|\,\hat{H}\,|\Psi\rangle}{\langle\Psi|\Psi\rangle}\right) \\
&= \frac{\frac{\partial}{\partial\alpha}\left(\langle\Psi|\,\hat{H}\,|\Psi\rangle\right)\langle\Psi|\Psi\rangle - \langle\Psi|\,\hat{H}\,|\Psi\rangle\,\frac{d}{d\alpha}\left(\langle\Psi|\Psi\rangle\right)}{\langle\Psi|\Psi\rangle^2}.
\end{aligned}$$

Compute terms with derivatives separately, starting with

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\left(\langle\Psi|\,\hat{H}\,|\Psi\rangle\right) &= \frac{\partial}{\partial\alpha}\left(\int\hat{H}|\Psi|^2 dV\right) = \int\left(\frac{\partial}{\partial\alpha}(\Psi)\,\hat{H}\Psi + \hat{H}\Psi\frac{\partial}{\partial\alpha}(\Psi)\right)dV \\
&= 2\int\hat{H}\Psi\frac{\partial\Psi}{\partial\alpha}dV = 2\int\Psi^2\frac{1}{\Psi^2}\hat{H}\Psi\frac{\partial\Psi}{\partial\alpha}dV \\
&= 2\int\Psi^2\frac{\hat{H}\Psi}{\Psi}\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV = 2\int\Psi^2 E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV \\
&= \frac{2\int\Psi^2 E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}dV}{\langle\Psi|\Psi\rangle}\langle\Psi|\Psi\rangle = 2\left\langle E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}\right\rangle\langle\Psi|\Psi\rangle.
\end{aligned}$$

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

$$\frac{\partial}{\partial \alpha}\left(\langle \Psi | \hat{H} | \Psi \rangle\right) = 2\left\langle E_L \frac{\partial \ln(\Psi)}{\partial \alpha}\right\rangle \langle \Psi | \Psi \rangle .$$

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 can be simplified as

$$\begin{aligned}\frac{\partial}{\partial \alpha}\langle \Psi_T | \Psi_T \rangle = \frac{\partial}{\partial \alpha}\int |\Psi|^2 dV = \int \frac{\partial}{\partial \alpha}(\Psi \cdot \Psi)\, dV = 2\int \Psi \frac{\partial \Psi}{\partial \alpha} dV \\ = 2\int \Psi^2 \frac{1}{\Psi}\frac{\partial \Psi}{\partial \alpha} dV = \frac{2\int \Psi^2 \frac{1}{\Psi}\frac{\partial \Psi}{\partial \alpha} dV}{\langle \Psi | \Psi \rangle}\langle \Psi | \Psi \rangle .\end{aligned}$$

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, expression for the expectation value derivative:

$$\begin{aligned}\frac{\partial \langle E_L \rangle}{\partial \alpha} &= \frac{\frac{\partial}{\partial \alpha}\left(\langle \Psi | E_L | \Psi \rangle\right)\langle \Psi | \Psi \rangle - \langle \Psi | E_L | \Psi \rangle \frac{\partial}{\partial \alpha}\left(\langle \Psi | \Psi \rangle\right)}{\langle \Psi | \Psi \rangle^2} \\ &= \frac{2\left\langle E_L \frac{\bar{\Psi}}{\Psi}\right\rangle \langle \Psi | \Psi \rangle - 2\langle E_L \rangle \left\langle \frac{\bar{\Psi}}{\Psi}\right\rangle \langle \Psi | \Psi \rangle}{\langle \Psi | \Psi \rangle},\end{aligned}$$

which simplifies to

$$\frac{\partial \langle E_L \rangle}{\partial \alpha} = 2\left\langle E_L \frac{\bar{\Psi}}{\Psi}\right\rangle - 2\langle E_L \rangle \left\langle \frac{\bar{\Psi}}{\Psi}\right\rangle . \tag{6.28}$$

6.4 Bosonic System: Analytical Solution and Observables

We wish to derive analytical ground state energy E_0 by computing variational integral directly for N bosons in 3D with the trial wave function(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 (with $\omega = 1$) is given by (2.34):

$$\hat{H} = \sum_{i=1}^{N} \left[-\frac{1}{2} \left(\frac{d^2}{dx_i^2} + \frac{d^2}{dy_i^2} + \frac{d^2}{dz_i^2} \right) + \frac{1}{2} \left(x_i^2 + y_i^2 + z_i^2 \right) \right].$$

We find variational energy by solving equation (2.31):

$$E_T(\alpha) = \int P E_L dV,$$

where probability density function (2.29) is

$$P = \frac{|\Psi|^2}{\int |\Psi|^2 dV}$$

and local energy (2.30) 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 can be computed as

$$\begin{aligned} |\Psi|^2 = &\int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha x_N^2} dx_N \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha y_N^2} dy_N \\ &\times \int_{-\infty}^{\infty} e^{-2\alpha z_1^2} dz_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N = \mathcal{I}_1^{3N}. \end{aligned}$$

Here, we defined $\mathcal{I}_1$ as

$$\mathcal{I}_1(x) = \int_{-\infty}^{\infty} e^{-2\alpha x^2} dx.$$

Gaussian integral [44] is given by

$$G = \int_{-\infty}^{\infty} e^{-ax^2} dx = \sqrt{\frac{\pi}{a}},$$

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 squared becomes

$$|\Psi|^2 = \mathcal{I}_1^{3N} = \left(\frac{\pi}{2\alpha}\right)^{3N/2}.$$

Therefore probability density function is

$$P = \left(\frac{2\alpha}{\pi}\right)^{3N/2} |\Psi|^2 = \left(\frac{2\alpha}{\pi}\right)^{3N/2} \prod_{i=1}^{N} e^{-2\alpha x_i^2} e^{-2\alpha y_i^2} e^{-2\alpha z_i^2}.$$

Variational energy can be simplified:

$$\begin{aligned}
E_T &= \left(\frac{2\alpha}{\pi}\right)^{3N/2} \\
&\times \int_{-\infty}^{\infty} \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} \sum_{i=1}^{N} \left[3\alpha - 2\alpha^2(x_i^2 + y_i^2 + z_i^2) + \frac{1}{2}\left(x_i^2 + y_i^2 + z_i^2\right)\right] dV \\
&= \left(\frac{2\alpha}{\pi}\right)^{3N/2} \sum_{i=1}^{N} \left[3\alpha \int_{-\infty}^{\infty} \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV \right. \\
&\left. + \left(-2\alpha^2 + \frac{1}{2}\right) \int_{-\infty}^{\infty} \left(x_i^2 + y_i^2 + z_i^2\right) \prod_{j=1}^{N} e^{-2\alpha x_j^2} e^{-2\alpha y_j^2} e^{-2\alpha z_j^2} dV \right] \\
&= \left(\frac{2\alpha}{\pi}\right)^{3N/2} (\mathcal{T}_1 + \mathcal{T}_2).
\end{aligned}$$

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:

$$\begin{aligned}
\mathcal{T}_1 = 3N\alpha \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha x_N^2} dx_N \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha y_N^2} dy_N \\
\int_{-\infty}^{\infty} e^{-2\alpha z_1^2} dz_1 \cdots \int_{-\infty}^{\infty} e^{-2\alpha z_N^2} dz_N \; = 3N\alpha \mathcal{I}_1^N \cdot \mathcal{I}_1^N \cdot \mathcal{I}_1^N = 3N\alpha \left(\frac{\pi}{2\alpha}\right)^{3N/2}.
\end{aligned}$$

Proceed with the 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 for the variational energy:

$$\begin{aligned}
E_T &= \left(\frac{2\alpha}{\pi}\right)^{3N/2} \left[3N\alpha \left(\frac{\pi}{2\alpha}\right)^{3N/2} - 3\alpha^2 N \cdot \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2} \right. \\
&\left. + \frac{3}{4}N \cdot \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2}\right] = -3\alpha^2 N \left(\frac{2\alpha}{\pi}\right)^{3N/2} \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2} \\
&3N\alpha \left(\frac{2\alpha}{\pi}\right)^{3N/2} \left(\frac{\pi}{2\alpha}\right)^{3N/2} + \frac{3}{4}N \left(\frac{2\alpha}{\pi}\right)^{3N/2} \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \left(\frac{\pi}{2\alpha}\right)^{(3N-1)/2} \\
&= 3N\alpha - 3\alpha^2 N \left(\frac{2\alpha}{\pi}\right)^{1/2} \left(\frac{\pi}{8\alpha^3}\right)^{1/2} + \frac{3}{4}N \left(\frac{2\alpha}{\pi}\right)^{1/2} \left(\frac{\pi}{8\alpha^3}\right)^{1/2} \\
&= 3N\alpha - 3\alpha^2 N \frac{1}{2\alpha} + \frac{3}{4}N\frac{1}{2\alpha} = 3N\alpha - \frac{3}{2}N\alpha + \frac{3}{8}\frac{N}{\alpha} = \frac{3}{2}N\alpha + \frac{3}{8}\frac{N}{\alpha}.
\end{aligned}$$

Variational energy for 3D system as a function of α is

$$\boxed{E_T = \frac{3N}{2}\left(\alpha + \frac{1}{4\alpha}\right)}. \tag{6.32}$$

We can find optimal variational parameter $\tilde{\alpha}$ by setting derivative of E_T with respect to α:

$$\frac{dE_T}{d\alpha} = \frac{3}{2}N - \frac{3}{8\tilde{\alpha}^2}N = 0 \implies \tilde{\alpha}^2 = \frac{1}{4} \implies \tilde{\alpha} = \pm\frac{1}{2}.$$

We reject $\alpha = -1/2$, because Gaussian integral requires parameter α to be positive, and, therefore,

$$\boxed{\tilde{\alpha} = \frac{1}{2}}. \tag{6.33}$$

Ground-state energy is equal to

$$\boxed{E_0(\tilde{\alpha}) = N\left(\frac{3}{4} + \frac{6}{8}\right) = \frac{3N}{2}}. \tag{6.34}$$

Finally, score O_1 is given by (3.16):

$$O_1 = \frac{\tilde{\Psi}}{\Psi},$$

where $\tilde{\Psi}$ (3.15) is

$$\tilde{\Psi} = \frac{\partial \Psi}{\partial \alpha}.$$

It can be computed 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 aim to find ground state energy E_0 and expression for the local energy E_L. Trial wave function for the system is given by (3.7):

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = D_\uparrow(\mathbf{r}_1) D_\downarrow(\mathbf{r}_2).$$

In this case, Slater determinants (3.8), (3.9) are 1×1, meaning that

$$D_\uparrow(\mathbf{r}_1) = \psi_1(\mathbf{r}_1) = e^{-\alpha r_1^2} \quad \text{and} \quad D_\downarrow(\mathbf{r}_2) = \psi_1(\mathbf{r}_2) = e^{-\alpha r_2^2},$$

where ψ_1 is given by (6.7):

$$\psi_1(\mathbf{r}) = e^{-\alpha r^2}.$$

Let us begin with deriving expression for the local energy using SD formulation (3.20):

$$\begin{aligned} E_L = &-\frac{1}{2}\sum_{i=1}^{N/2} \frac{\Delta_i D_\uparrow}{D_\uparrow} - \frac{1}{2}\sum_{i=N/2+1}^{N} \frac{\Delta_i D_\downarrow}{D_\downarrow} + \frac{1}{2}\sum_{i=1}^{N} r_i^2 \\ &- \frac{1}{2}\frac{\Delta_1 D_\uparrow}{D_\uparrow} - \frac{1}{2}\frac{\Delta_1 D_\downarrow}{D_\downarrow} + \frac{1}{2}(r_1^2 + r_2^2) \\ &- \frac{1}{2}\frac{\Delta_1 \psi_1(\mathbf{r}_1)}{\psi_1(\mathbf{r}_1)} - \frac{1}{2}\frac{\Delta_2 \psi_1(\mathbf{r}_2)}{\psi_1(\mathbf{r}_2)} + \frac{1}{2}(r_1^2 + r_2^2). \end{aligned}$$

Two-dimensional 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

$$\boxed{E_L = 4\alpha + (r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right)}. \tag{6.36}$$

Now, let us derive analytical solution. It is very similar to the one obtained in Appendix 6.4. Probability density function (2.29) 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:

$$\begin{aligned}\int e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} dV &= \int_{-\infty}^{\infty} e^{-2\alpha x_1^2} dx_1 \int_{-\infty}^{\infty} e^{-2\alpha y_1^2} dy_1 \int_{-\infty}^{\infty} e^{-2\alpha x_2^2} dx_2 \int_{-\infty}^{\infty} e^{-2\alpha y_2^2} dy_2 \\ &= \mathcal{I}_1^4 = \left(\sqrt{\frac{\pi}{2\alpha}}\right)^4 = \frac{\pi^2}{4\alpha^2}.\end{aligned}$$

Here, we used the same integral (6.30):

$$\mathcal{I}_1 = \sqrt{\frac{\pi}{2\alpha}}.$$

Probability distribution function is then:

$$P = \frac{4\alpha^2}{\pi^2} e^{-2\alpha r_1^2} e^{-2\alpha r_2^2}.$$

Variational energy (2.31) can be simplified as:

$$\begin{aligned}E_T &= \int P E_L dV = \frac{4\alpha^2}{\pi^2} \int \left[e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} \left((r_1^2 + r_2^2)\left(\frac{1}{2} - 2\alpha^2\right) + 4\alpha \right) \right] dV \\ &= \frac{4\alpha^2}{\pi^2} \left[4\alpha \int_{\infty}^{\infty} e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} dV + \left(\frac{1}{2} - 2\alpha^2\right) \int_{\infty}^{\infty} e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} (r_1^2 + r_2^2) dV \right] \\ &= \frac{4\alpha^2}{\pi^2} \left(\mathcal{T}_1 + \mathcal{T}_2 \right).\end{aligned}$$

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:

$$\begin{aligned}
\mathcal{T}_2 &= \left(\frac{1}{2} - 2\alpha^2\right) \int_\infty^\infty e^{-2\alpha r_1^2} e^{-2\alpha r_2^2} (r_1^2 + r_2^2) dV \\
&= \left(\frac{1}{2} - 2\alpha^2\right) \left[\int_\infty^\infty e^{-2\alpha(x_1^2+y_1^2)} e^{-2\alpha(x_2^2+y_2^2)} (x_1^2 + y_1^2) dx_1 dy_1 dx_2 dy_2 \right. \\
&\left. + \int_\infty^\infty e^{-2\alpha(x_1^2+y_1^2)} e^{-2\alpha(x_2^2+y_2^2)} (x_2^2 + y_2^2) dx_1 dy_1 dx_2 dy_2 \right] \\
&= \left(\frac{1}{2} - 2\alpha^2\right) \left[\int_\infty^\infty x_1^2 e^{-2\alpha x_1^2} dx_1 \int_\infty^\infty e^{-2\alpha y_1^2} dy_1 \int_\infty^\infty e^{-2\alpha x_2^2} dx_2 \int_\infty^\infty e^{-2\alpha y_2^2} dy_2 \right. \\
&+ \int_\infty^\infty e^{-2\alpha x_1^2} dx_1 \int_\infty^\infty y_1^2 e^{-2\alpha y_1^2} dy_1 \int_\infty^\infty e^{-2\alpha x_2^2} dx_2 \int_\infty^\infty e^{-2\alpha y_2^2} dy_2 \\
&+ \int_\infty^\infty e^{-2\alpha x_1^2} dx_1 \int_\infty^\infty e^{-2\alpha y_1^2} dy_1 \int_\infty^\infty x_2^2 e^{-2\alpha x_2^2} dx_2 \int_\infty^\infty e^{-2\alpha y_2^2} dy_2 \\
&\left. + \int_\infty^\infty e^{-2\alpha x_1^2} dx_1 \int_\infty^\infty e^{-2\alpha y_1^2} dy_1 \int_\infty^\infty e^{-2\alpha x_2^2} dx_2 \int_\infty^\infty y_2^2 e^{-2\alpha y_2^2} dy_2 \right] \\
&= \left(\frac{1}{2} - 2\alpha^2\right) \cdot 4\mathcal{I}_1^3 \mathcal{I}_2.
\end{aligned}$$

Here we have used definition of $\mathcal{I}_2$ (6.31):

$$\mathcal{I}_2 = \int_\infty^\infty x^2 e^{-2\alpha x^2} dx = \frac{1}{2}\sqrt{\frac{\pi}{8\alpha^3}}.$$

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 variational state energy as a function of α is

$$E_T = \frac{4\alpha^2}{\pi^2} \left(\frac{\pi^2}{\alpha} + \frac{\pi^2}{8\alpha^3} - \frac{\pi^2}{2\alpha} \right) = 4\alpha + \frac{1}{2\alpha} - 2\alpha = 2\alpha + \frac{1}{2\alpha}.$$

Therefore,

$$\boxed{E_T = 2\alpha + \frac{1}{2\alpha}}. \tag{6.37}$$

Optimal variational parameter $\tilde{\alpha}$ is then:

$$\frac{dE_T}{d\alpha} = 2 - \frac{1}{2\alpha} = 0 \implies \alpha^2 = \frac{1}{4} \implies \tilde{\alpha} = \pm\frac{1}{2}.$$

As before, we discard solution with minus sign due to Gaussian integrals restriction of its parameters and

$$\boxed{\tilde{\alpha} = \frac{1}{2}}. \tag{6.38}$$

Ground-state energy is equal to

$$\boxed{E_0(\tilde{\alpha}) = 2}. \tag{6.39}$$

6.6 Interacting Fermions: Analytical Expressions and Observables

In atomic units (with $\omega = 1$), Hamiltonian is given by (2.35):

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

Trial wave functions can be written as

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha, \beta) = \Phi(\mathbf{r}_1, ..., \mathbf{r}_N; \alpha) I(\mathbf{r}_1, ..., \mathbf{r}_N; \beta),$$

where I represents Jastrow factors in the form J (2.38) or in the form P (2.39):

$$J(\mathbf{r}_1, ..., \mathbf{r}_N; \beta_1, ..., \beta_p) = \exp\left(\sum_{i=1}^{N} \sum_{j>i}^{N} \beta_{ij} r_{ij} \right) = \exp\left(\sum_{k=1}^{p} \beta_k r_k \right);$$

$$P(\mathbf{r}_1, ..., \mathbf{r}_N; \beta) = \exp\left(\sum_{i=1}^{N} \sum_{j>i}^{N} \frac{a r_{ij}}{1 + \beta r_{ij}} \right) = \exp\left(\sum_{k=1}^{p} \frac{a r_k}{1 + \beta r_k} \right).$$

Here, we have defined number of pairs as $p = N(N-1)/2$ and used transformation (3.18) of 2D index (i, j) into 1D index k. Φ represents non-interacting wave function (3.7):

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

Begin with the scores O_1 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 p variational parameters β_{ij}, and O_1, 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 it is

$$\mathbf{O}_1^{(J)} = (r_0, r_1, \cdots, r_{N-1}). \tag{6.40}$$

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

$$O_1^{(P)} = \sum_{k=0}^{N-1} -\frac{a r_k^2}{(1 + \beta r_k)^2}. \tag{6.41}$$

Proceed with deriving analytical expression for the 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

$$\frac{\hat{H}_0\Psi_T}{\Psi_T} = -\frac{1}{2}\cdot\frac{\sum_{i=1}^{N}\Delta_i\left(\Phi I\right)}{\Phi I}. \tag{6.42}$$

Laplacian can be expanded using the chain rule:

$$\begin{aligned}\Delta\left(\Phi I\right) &= \nabla\left(\nabla\left(\Phi I\right)\right) = \nabla\left(I\nabla\Phi + \Phi\nabla I\right) = \nabla\left(I\nabla\Phi\right) + \nabla\left(\Phi\nabla I\right)\\ &= I\Delta\Phi + \nabla I\nabla\Phi + \nabla\Phi\nabla I + \Phi\Delta I = I\Delta\Phi + 2\nabla I\nabla\Phi + \Phi\Delta I.\end{aligned}$$

Inserting it back to expression to (6.42) gives

$$\begin{aligned}\frac{\hat{H}_0\Psi_T}{\Psi_T} &= -\frac{1}{2}\cdot\frac{\sum_{i=1}^{N}\left(I\Delta_i\Phi + 2\nabla_i I\nabla_i\Phi + \Phi\Delta_i I\right)}{\Phi I}\\ &= -\frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i\Phi}{\Phi} - \frac{\sum_{i=1}^{N}\nabla_i\Phi\nabla_i I}{\Phi I} - \frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i I}{I}.\end{aligned}$$

In the expression above, first term represents local energy of non-interacting wave function Φ. Additional two terms appear because of more complex form of the ansatz. Fist two terms in the expression above can be rewritten as

$$\begin{aligned}&-\frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i\left(D_\uparrow D_\downarrow\right)}{D_\uparrow D_\downarrow} - \frac{\sum_{i=1}^{N}\nabla_i\left(D_\uparrow D_\downarrow\right)\nabla_i I}{D_\uparrow D_\downarrow I}\\ &= -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\Delta_i D_\downarrow}{D_\downarrow}\right] - \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right].\end{aligned}$$

Putting in back to equation (6.42):

$$\begin{aligned}\frac{\hat{H}_0\Psi_T}{\Psi_T} &= -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\Delta_i D_\downarrow}{D_\downarrow}\right]\\ &\quad - \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right] - \frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i I}{I}\\ &= \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3.\end{aligned}$$

Here, we have defined

$$\mathcal{E}_1 = -\frac{1}{2}\left[\frac{\sum_{i=1}^{N/2}\Delta_i D_\uparrow}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\Delta_i D_\downarrow}{D_\downarrow}\right], \tag{6.43}$$

$$\mathcal{E}_2 = \left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow\nabla_i I}{D_\uparrow I} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow\nabla_i I}{D_\downarrow I}\right], \tag{6.44}$$

and

$$\mathcal{E}_3 = -\frac{1}{2}\frac{\sum_{i=1}^{N}\Delta_i I}{I}. \tag{6.45}$$

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:

$$\frac{\hat{V}\Psi_T}{\Psi_T} = \frac{B(\omega)}{2}\cdot\frac{\sum_{i=1}^{N}r_i^2\Psi_T}{\Psi_T} = \frac{B(\omega)}{2}\sum_{i=1}^{N}r_i^2 = \hat{V}.$$

Therefore, local energy can be computed as

$$E_L = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3 + \hat{H}_I + \hat{V}. \tag{6.46}$$

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 the expression (6.44) is

$$\mathcal{E}_2 = -\left[\frac{\sum_{i=1}^{N/2}\nabla_i D_\uparrow \nabla_i \ln I}{D_\uparrow} + \frac{\sum_{i=N/2+1}^{N}\nabla_i D_\downarrow \nabla_i \ln I}{D_\downarrow}\right]. \tag{6.47}$$

Likewise, alternative form of the equation (6.45) is

$$\mathcal{E}_3 = -\frac{1}{2}\sum_{i=1}^{N}\left(\Delta \ln I - (\nabla \ln I)^2\right). \tag{6.48}$$

Continue with the 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 of r_{ij} 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, therefore,

$$\nabla_i (\ln J) = \sum_{j=1\neq i}^{N}\beta_{ij}\frac{(\mathbf{r}_i-\mathbf{r}_j)}{r_{ij}}. \tag{6.49}$$

Proceed with the 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 by 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

$$\Delta_i \ln J = \sum_{j=1\neq i}^{N} \frac{\beta_{ij}}{r_{ij}} = \sum_{m=1\neq i}^{p} \frac{\beta_m}{r_m}. \tag{6.50}$$

Continue with the 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, the gradient is computed by using

$$\nabla_i \ln P = \sum_{j=1\neq i}^{N} \frac{a (\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}(1+\beta r_{ij})^2}. \tag{6.51}$$

Finally, the 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 the gradient in expression above separately:

$$\begin{aligned}\nabla_i \left[r_{ij}(1+\beta r_{ij})^2 \right] &= (\mathbf{r}_i - \mathbf{r}_j)\, r_{ij}^{-1}(1+\beta r_{ij})^2 + r_{ij} \cdot 2(1+\beta r_{ij}) \cdot \beta\, (\mathbf{r}_i - \mathbf{r}_j)\, r_{ij}^{-1} \\ &= (\mathbf{r}_i - \mathbf{r}_j)\, r_{ij}^{-1}(1+\beta r_{ij}) \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right],\end{aligned}$$

so that

$$\begin{aligned}(\mathbf{r}_i - \mathbf{r}_j)\, \nabla_i \left[\Phi(\mathbf{r}_i) \right] &= -a\, (\mathbf{r}_i - \mathbf{r}_j)\, \frac{(\mathbf{r}_i - \mathbf{r}_j)\, r_{ij}^{-1}(1+\beta r_{ij}) \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right]}{r_{ij}^2 (1+\beta r_{ij})^4} \\ &= -\frac{a r_{ij}(1+\beta r_{ij}) \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right]}{r_{ij}^2 (1+\beta r_{ij})^4}.\end{aligned}$$

Collecting everything back:

$$\begin{aligned}\nabla_i \left[(\mathbf{r}_i - \mathbf{r}_j)\, \Phi(\mathbf{r}_i) \right] &= \frac{2a}{r_{ij}(1+\beta r_{ij})^2} - \frac{r_{ij}(1+\beta r_{ij}) \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right]}{r_{ij}^2 (1+\beta r_{ij})^4} \\ &= \frac{2a r_{ij}(1+\beta r_{ij})^2 - a r_{ij}(1+\beta r_{ij}) \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right]}{r_{ij}^2 (1+\beta r_{ij})^4} \\ &= a \frac{2(1+\beta r_{ij}) - \left[(1+\beta r_{ij}) + 2 r_{ij}\beta \right]}{r_{ij}(1+\beta r_{ij})^3} = \frac{a(1-\beta r_{ij})}{r_{ij}(1+\beta r_{ij})^3}.\end{aligned}$$

The final expression for the Laplacian is

$$\Delta_i \left(\ln P \right) = \sum_{j=1 \neq i}^{N} \frac{a(1-\beta r_{ij})}{r_{ij}(1+\beta r_{ij})^3} = \sum_{m=1 \neq i}^{p} \frac{a(1-\beta r_m)}{r_m (1+\beta r_m)^3}. \tag{6.52}$$

Bibliography

[1] A. M. M. Leal. autodiff: a modern, fast and expressive C++ library for automatic differentiation, 2018. URL https://autodiff.github.io.

[2] OpenMP Architecture Review Board. *OpenMP Application Programming Interface Version 5.2*, 2023. URL https://www.openmp.org/specifications/.

[3] MPI Forum. *MPI: A Message-Passing Interface Standard Version 4.1*, 2021. URL https://www.mpi-forum.org/docs/.

[4] M. Taut. Two electrons in an external oscillator potential: Particular analytic solutions of a coulomb correlation problem. *Physical Review A*, 48:3561–3566, 1993. doi: 10.1103/PhysRevA.48.3561.

[5] M. P. Lohne, G. Hagen, M. Hjorth-Jensen, S. Kvaal, and F. Pederiva. Ab initio computation of the energies of circular quantum dots. *Physical Review B*, 84:115302, 2011. doi: 10.1103/PhysRevB.84.115302.

[6] M. Hjorth-Jensen. Project 1: Bec variational monte carlo. FYS4411, 2024. URL https://compphysics.github.io/ComputationalPhysics2/doc/Projects/2024/Project1/pdf/Project1.pdf. Accessed: 11 May 2025.

[7] F. Dalfovo, S. Giorgini, L. P. Pitaevskii, and S. Stringari. Theory of bose–einstein condensation in trapped gases. *Reviews of Modern Physics*, 71:463–512, 1999. doi: 10.1103/RevModPhys.71.463.

[8] C. Yannouleas and U. Landman. Symmetry breaking and wigner molecules in few-electron quantum dots. *Physica Status Solidi (a)*, 203:1160, 2006. doi: 10.1002/pssa.200566197.

[9] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum monte carlo simulations of solids. *Reviews of Modern Physics*, 73:33–83, 2001. doi: 10.1103/RevModPhys.73.33.

[10] I. Bloch, J. Dalibard, and W. Zwerger. Many-body physics with ultracold gases. *Reviews of Modern Physics*, 80:885–964, 2008. doi: 10.1103/RevModPhys.80.885.

[11] A. Szabo and N. S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Publications, 1996.

[12] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7: 1–49, 1998. doi: 10.1017/S0962492900002038.

[13] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970. doi: 10.1093/biomet/57.1.97.

[14] P. L. Bowers. *Lectures on Quantum Mechanics: A Primer for Mathematicians*. Cambridge University Press, 2020.

[15] D. F. Styer. *The Strange World of Quantum Mechanics*. Cambridge University Press, 2000.

[16] D. C. Chang. Physical interpretation of planck's constant based on the maxwell theory. *Chinese Physics B*, 26:040301, 2017. doi: 10.1088/1674-1056/26/4/040301.

[17] J. 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.

[18] D. J. Griffiths. *Introduction to Quantum Mechanics*. Prentice Hall, 1994.

[19] P. J. Mohr, D. B. Newell, and B. N. Taylor. Codata recommended values of the fundamental physical constants: 2018. *Reviews of Modern Physics*, 88:035009, 2016. doi: 10.1103/RevModPhys.88.035009.

[20] J. A. Rock. A lecture on integration by parts, 2016. URL https://arxiv.org/abs/1606.04141.

[21] J. Binney and D. Skinner. *The Physics of Quantum Mechanics*. Cappella Archive, 2008.

[22] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. National Bureau of Standards, Applied Mathematics Series 55, 1964.

[23] I. Shavitt and R. J. Bartlett. *Many-Body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*. Cambridge University Press, 2009.

[24] E. Süli and D. F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[25] C. J. Pethick and H. Smith. *Bose–Einstein Condensation in Dilute Gases*. Cambridge University Press, 2002.

[26] 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.

[27] T. Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Communications on Pure and Applied Mathematics*, 10:151–177, 1957. doi: 10.1002/cpa.3160100201.

[28] B. L. Hammond, W. A. Jr. Lester, and P. J. Reynolds. *Monte Carlo Methods in Ab Initio Quantum Chemistry*. World Scientific, 1994.

[29] M. Hjorth-Jensen. Project 2: Variational monte carlo. FYS4411, 2024. URL https://compphysics.github.io/ComputationalPhysics2/doc/Projects/2024/Project2/Project2VMC/pdf/Project2VMC.pdf. Accessed: 11 May 2025.

[30] C.-J. Huang, C. Filippi, and C. J. Umrigar. Spin contamination in quantum monte carlo wave functions. *The Journal of Chemical Physics*, 108:8838–8847, 1998. doi: 10.1063/1.476330.

[31] Morten Hjorth-Jensen. Week 1: Introduction and numerical methods. Lecture notes, FYS4411, 2024. URL http://compphysics.github.io/ComputationalPhysics2/doc/pub/week1/pdf/week1.pdf. Accessed: 11 May 2025.

[32] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1996.

[33] U. S. Fjordholm, N. H. Risebro, and H. Hoel. 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: 14 May 2025.

[34] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer, 2003.

[35] F. Becca and S. Sorella. *Quantum Monte Carlo Approaches for Correlated Systems*. Cambridge University Press, 2017.

[36] J. E. Gubernatis, N. Kawashima, and P. Werner. *Quantum Monte Carlo Methods: Algorithms for Lattice Models*. Cambridge University Press, 2016.

[37] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[38] 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*, 20:1107–1119, 1981. doi: 10.1002/qua.560200508.

[39] A. Lüchow. Optimized quantum monte carlo wave functions. In *Many-Body Methods for Real Materials: Modeling and Simulation, Vol. 9*. Forschungszentrum Jülich, 2019.

[40] M. Hjorth-Jensen. Metropolis algorithm and monte carlo methods. Lecture notes, FYS4411, University of Oslo, 2025. URL https://compphysics.github.io/ComputationalPhysics2/doc/pub/week3/pdf/week3.pdf. Accessed: 15 May 2025.

[41] K. B. Petersen and M. S. Pedersen. The matrix cookbook. Tech report, Technical University of Denmark, 2012. URL http://www2.imm.dtu.dk/pubdb/p.php?3274.

[42] G. Argentini. A generalization of amdahl's law and relative conditions of parallelism, 2002. URL https://arxiv.org/abs/cs/0209029.

[43] M. Ledum. Variational monte carlo for fys4411. GitHub repository, 2025. URL https://github.com/mortele/variational-monte-carlo-fys4411.

[44] L. L. de Sales, J. A. Silva, E. P. B. de Souza, H. T. C. M. de Souza, A. D. S. Farias, and O. P. Lavor. Gaussian integral by taylor series and applications. *REMAT: Revista Eletrônica da Matemática*, 7:e3001, 2021. doi: 10.35819/remat2021v7i2id4330.