Resource Efficient Non-Gradient Optimization Methods

for Variational Quantum Eigensolvers

MINIMAL THESIS

YURY CHERNYAK

SUPERVISOR: DOC. RNDR. MARTIN PLESCH, PHD.

Mathematics, Physics and Informatics Department
COMENIUS UNIVERSITY
&
Department of Complex Systems
SLOVAK ACADEMY OF SCIENCES

submitted to and in accordance with the requirements of the degree of DOCTOR IN PHILOSOPHY in the Faculty of Mathematics, Physics and Informatics.

DATE: JUNE 31ST, 2024

Abstract

In this thesis proposal we examine non-gradient optimization methods and how they apply and improve the class of hybrid quantum-classical algorithms known as Variational Quantum Algorithms, in which we divert particular attention to the specific subclass of these algorithms known as variational quantum eigensolvers. We will use such optimization methods like the particle swarm optimization (PSO) method and analyze how they improve on what the VQE is attempting to do – search for the lowest eigenvalue of a complex chemical-system. Furthermore, we introduce a new optimization algorithm that is physics-based and resolves the issues of explosions and parameter-sensitive tuning that exists in the PSO method. We apply these optimization methods to a set of standard optimization test functions and compare them to other non-gradient optimization algorithms. We discover that our algorithm performs much better on test functions and therefore have reason to apply it to more complex systems. Therefore, we then proceed to apply our newly-invented algorithm to quantum circuits. We begin by applying it to the hydrogen Hamiltonian but continue to more complex Hamiltonian's, such as Lithium Hydrite, that are yet to be intricately studied in the field.

Dedication and acknowledgements

I dedicate this work first and foremost to my family: my parents – Aleksandr and Elena Chernyak – for continuously pushing me to be the best version of myself; and my sisters – Katherine and Elizabeth – for their continuous encouragement and support as well. I also sincerely acknowledge and dedicate this thesis to my research group: my supervisor, Martin Plesch, who provided invaluable help, wisdom and insight, showed immense support; and my friend and colleague, Ijaz Mohammad, whose presence was very important to this work and my PhD studies, both professionally and personally. I am extremely grateful for these people.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

TABLE OF CONTENTS

List of Tables

List of Figures

Figure **Page**

CHAPTER 1

FUNDAMENTALS OF LINEAR ALGEBRA AND QUANTUM MECHANICS

Linear Algebra is the language of Quantum Mechanics. It is a description of the transformation of space–which is our current best model of how nature works at the most fundamental level. Operators, which represent quantities like momentum, position, and energy, are represented by matrices, which act on the state of a particle represented as a vector. This action can be abstractly visualized as the transformation of a space (bending, rotating etc.). The result, as dictated by quantum theory, is that the vector representing the state of the particle gets scalarly multiplied by discrete numbers. This is represented in what is called the eigenvalue-equation, in which the discrete scalar numbers are known as eigenvalues. Practically, when scientists measure a quantity like energy, the results of these measurements are the eigenvalues–implying that quantities in nature take on only discrete, not continuous, values. For example, in the Hydrogen atom, the energy levels are specified by a principle quantum number in the equation:

$$E_n = -\frac{m_e e^4}{8h^2\epsilon_0^2}\frac{1}{n^2} = -\frac{13.6\,\text{eV}}{n^2}, \quad \text{for} \quad n = 1,2,3,\ldots \tag{1.1}$$

where:

- m_e is the electron mass,
- e is the elementary charge,
- h is Planck's constant,
- ϵ_0 is the permittivity of free space,
- n is the principal quantum number which takes on positive integer values.

The ground state of Hydrogen has an energy of -13.6 electron-volts. That is, if you want to tear the electron off of the Hydrogen atom (a process known as ionization), you need to supply at least 13.6 eV of energy. This amount of energy is necessary to enable the electron to escape to very large distances away from the proton. Higher energy levels are less tightly bound, meaning it takes less energy to ionize the atom and free the electron. Additionally, these energy levels get closer and closer together as they increase. We say that: "the allowed energy levels are quantized"–from which the name ***quanta*** (plural for quantum) arises. An explicit definition of quantum is that a quantum is a discrete quantity of energy proportional in magnitude to the frequency of the radiation it represents, and it is the minimum amount of any physical entity involved in an interaction. And hence, the study of these discrete energies is ***Quantum Mechanics***. This thesis focuses on determining the ground state energy of various molecules (a process that can be extended to other systems) through an optimization process via quantum computation.

1.1 Fundamentals of Linear Algebra

1.1.1 Vector Space and Hilbert Space

A vector space is an assembly of entities known as vectors that can undergo addition and scalar multiplication to form new vectors within the same vector space [1]. When the scalars are complex numbers, the space is termed a "complex vector space". The principles governing addition and scalar multiplication adhere to certain axioms, which create and structure the vector space.

Below are the axioms necessary for a set V to be termed a vector space for all elements $\mathbf{u},\mathbf{v},\mathbf{w} \in V$ and all scalars $a,b \in \mathbb{F}$, where $\mathbb{F}$ is the field over which V is defined (commonly $\mathbb{R}$ or $\mathbb{C}$):

1. **Associativity of Addition:** $\mathbf{u}+(\mathbf{v}+\mathbf{w}) = (\mathbf{u}+\mathbf{v})+\mathbf{w}$.
2. **Commutativity of Addition:** $\mathbf{u}+\mathbf{v} = \mathbf{v}+\mathbf{u}$.
3. **Identity Element of Addition:** There exists an element $\mathbf{0} \in V$, called the zero vector, such that $\mathbf{u}+\mathbf{0} = \mathbf{u}$ for all $\mathbf{u} \in V$.
4. **Inverse Elements of Addition:** For every $\mathbf{u} \in V$, there exists an element $-\mathbf{u} \in V$ such that $\mathbf{u}+(-\mathbf{u}) = \mathbf{0}$.
5. **Associativity of Scalar Multiplication:** $a(b\mathbf{u}) = (ab)\mathbf{u}$.
6. **Identity Element of Scalar Multiplication:** $1\mathbf{u} = \mathbf{u}$, where 1 is the multiplicative identity in $\mathbb{F}$.
7. **Distributivity of Scalar Multiplication with respect to Field (Scalar) Addition:** $(a+b)\mathbf{u} = a\mathbf{u}+b\mathbf{u}$.

8. **Distributivity of Scalar Multiplication with respect to Vector Addition:** $a(\mathbf{u}+\mathbf{v}) = a\mathbf{u}+a\mathbf{v}$.

The vector space *dimensionality* states the size of the vector space. it is the number of vectors (in any basis) for that space, which is the same as the maximum number of linearly independent vectors it can contain. By linear independence, it is meant that no vector in the set can be written as a linear combination of the other vectors in the set.

Euclidean Space:

Take the three-dimensional *Euclidean space*, denoted as $\mathbb{R}^3$, as an example. All possible vectors are of the form $\mathbf{v} = (x, y, z)$ where $x, y, z \in \mathbb{R}$. Consider the following vectors:

$$\mathbf{v}_1 = (1,0,0), \quad \mathbf{v}_2 = (0,1,0), \quad \mathbf{v}_3 = (0,0,1).$$

These vectors are linearly independent because there are no scalars a, b, c (other than $a = b = c = 0$) that can satisfy the equation:

$$a\mathbf{v}_1 + b\mathbf{v}_2 + c\mathbf{v}_3 = \mathbf{0},$$

where $\mathbf{0} = (0,0,0)$ is the zero vector in $\mathbb{R}^3$.

The set $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ also *spans* the entire vector space of $\mathbb{R}^3$ because any vector $\mathbf{v} = (x, y, z)$ can be expressed as a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$:

$$\mathbf{v} = x\mathbf{v}_1 + y\mathbf{v}_2 + z\mathbf{v}_3.$$

Since these vectors satisfy two conditions— ***linear independence*** and ***spanning*** the vector space—these vectors form a ***basis*** for $\mathbb{R}^3$. Consequently, the dimensionality of the vector space is the number of vectors in the basis, which is three in this case.

Hilbert Space

The Hilbert space is the space under discussion in quantum theory. The Hilbert space is a specific linear complex vector space which satisfies the following axioms [6]:

1. It has an inner product operation, denoted as $\langle u|v\rangle \in \mathbb{C}$, which in turn satisfies a few conditions:
   (a) Conjugate symmetry: $\langle u|v\rangle = \langle v|u\rangle^*$, where $*$ stands for complex conjugate operation.
   (b) Linearity with respect to another vector: $\langle u|av + bw\rangle = a\langle u|v\rangle + b\langle u|w\rangle$
   (c) Anti-linearity with respect to first vector: $\langle au + bv|w\rangle = a^*\langle u|w\rangle + b^*\langle v|w\rangle$
   (d) Positive definiteness: $\langle u|u\rangle = |u|^2 \geq 0$

An inner product of a vector is always positive and only zero when the vector itself is a zero vector. Here $|\cdot|$ denotes the Euclidean norm or the length of the vector.

2. The Hilbert space is complete with respect to its norm–meaning that there are no gaps in the sequence of its elements. That is, for a given sequence in which two elements get arbitrarily close as we move further down the sequence, $\lim_{m,n\to\infty} |u_m - u_n| = 0$, every Cauchy sequence converges to an element ϕ in the Hilbert space, $\lim_{n\to\infty} |\phi - u_n| = 0$.

   "Complete" refers to the property that every Cauchy sequence in the space converges to a limit that is also within the space. A Cauchy sequence is a sequence whose elements become arbitrarily close to each other as the sequence progresses. That is, for any small distance, there is a point in the sequence after which all subsequent elements of the sequence are within that distance of each other. This concept defines what it means for elements of the sequence to be getting closer together without referencing a specific limit. The completeness idea is an important concept because, in a complete space, it is guaranteed that these Cauchy sequences have a limit in the same space. This is not a trivial property as there are spaces that are not complete (where you can have Cauchy sequences that do not converge within the space) which means that the space has "holes" or is missing points.

1.1.2 Operators, Eigenvectors and Eigenvalues

Operators and Hermicity

Linear operators are mappings that preserve vector space operations, while eigenvectors and eigenvalues emerge from the action of operators applied on vectors. "Mapping" means transforming elements of one space into elements of another space. Since quantum mechanics is linear in nature, the subject deals with linear mappings. A map $\mathbf{A} : \mathbf{V} \to \mathbf{U}$ is linear if it satisfies the following property for all $\mathbf{u}, \mathbf{v} \in \mathbf{V}$ and scalars a, b:

$$\mathbf{A}(a\mathbf{u} + b\mathbf{v}) = a\mathbf{A}\mathbf{u} + b\mathbf{A}\mathbf{v}. \tag{1.2}$$

Operators in quantum mechanics are represented by mathematical quantities known as matrices [9]. Moreover, another requirement for operators in quantum mechanics is the *hermitian* property.

An operator $\hat{\mathbf{A}}$ is said to be Hermitian if it satisfies:

$$\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\dagger}, \tag{1.3}$$

in which the dagger symbol (†) is the "adjoint" or "conjugate transpose" of the operator [20]. This condition ensures that the eigenvalues are real and the eigenstates are orthogonal to each other (the latter term can be thought of as being "perpendicular in higher dimensions").

Lastly, a unitary condition must be satisfied. An operator $\mathbf{U}$ is unitary if:

$$\mathbf{U}\mathbf{U}^{\dagger} = \mathbf{U}^{\dagger}\mathbf{U} = \mathbf{I}, \tag{1.4}$$

where **I** is the identity matrix–a square matrix with values of 1 along the diagonal and zero everywhere else. Unitary operators have orthogonal eigenvectors, though their eigenvalues may be complex. An operator whose inverse is its self-adjoint is a unitary operator.

Eigenvectors

An eigenvector, **u**, is a vector that changes in magnitude by being multiplied by a scalar factor, λ, after being transformed by some given square matrix **A**. The scalar value λ is known as an eigenvalue. Only if the eigenvalue is negative does the direction of the vector become reversed. By definition, eigenvectors are never zero.

This description above is describing what is called the *eigenvalue equation* and it is mathematically stated as:

$$\mathbf{Au} = \lambda\mathbf{u}. \tag{1.5}$$

The eigenvalues of matrix **A** can be determined by solving the "characteristic equation" which is given by:

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = 0, \tag{1.6}$$

and is solved by finding the determinant which must equal to zero.

$$\Rightarrow \det(\mathbf{A} - \lambda\mathbf{I}) = 0, \tag{1.7}$$

where **I** is the identity matrix. The eigenvectors of the matrix correspond to the non-zero solutions of the equation derived from the above eigenvalue equation. In quantum mechanics, these eigenvectors are the possible states of the system, with their associated eigenvalues being the possible measured values obtained from experiments. This concept will be elaborated in the next section.

1.2 Fundamentals of Quantum Mechanics

Postulates of Quantum mechanics

The following are the general set of axioms that formulate Quantum Mechanics along with an explanation of the axiom:

1. The state of a physical system is represented by a ket $|\psi\rangle$ in the state space, where $|\psi\rangle$ is a vector in the vector space (state space).

   One of the founders of quantum theory, theoretical physicist Paul Dirac (1902 – 1984), invented a common and convenient notation for representing quantum states called Dirac notation (also called Bra-ket notation) [7]. A "ket" is symbolized as $|\psi\rangle$ and represents a vector. A “bra” is a linear functional that maps each vector in a Hilbert space to a complex

number and is represented as $\langle\phi|$. When a "bra" $\langle\phi|$ is multiplied by a "ket" $|\psi\rangle$, they produce a scalar number. A mathematical example is provided below:

Considering a quantum state $|\psi\rangle$ represented as a column vector with complex entries c_1 and c_2. The corresponding bra $\langle\phi|$ will be a row vector containing the complex conjugates of these entries, denoted as c_1^* and c_2^*. The inner product (bra-ket) is then:

$$\langle\phi|\psi\rangle = \begin{pmatrix} c_1^* & c_2^* \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = c_1^* c_1 + c_2^* c_2$$

2. Any observable characteristic of a physical system is described by an operator $\hat{\mathbf{A}}$, and the outcome of measuring the observable is one of the operator's eigenvalues, λ.

   Observables in quantum physics manifest as linear operators on a Hilbert space that represent the vector space of quantum states [2]. The eigenvalues of observables are real numbers that correspond to the possible values of the dynamical variable represented by the observable. In other words, observables in quantum mechanics assign real numbers to the results of specific measurements, corresponding to the operator's eigenvalue with respect to the system's measured quantum state.

   A key distinction between classical and quantum mechanical observables is that, in quantum mechanics, observables cannot always be measured simultaneously. This property is known as complementarity. This is mathematically expressed by the non-commutativity property of the corresponding operators, meaning that the commutator of two operators $\hat{A}$ and $\hat{B}$ is non-zero:

   $$[\hat{A}, \hat{B}] := \hat{A}\hat{B} - \hat{B}\hat{A} \neq \hat{0} \tag{1.8}$$

   This inequality expresses how measurement results are affected by the order in which observables $\hat{A}$ and $\hat{B}$ are measured. Incompatible observables correspond to non-commuting operators. A complete set of common eigenstates cannot exist for incompatible observables.

3. Upon measurement of an observable $\hat{\mathbf{A}}$, the only possible outcomes are the eigenvalues of the observable. Since we can perceive only real values, the eigenvalues should be real, which is a defining characteristic of a Hermitian operator. The Hermitian operators form an orthonormal basis, as it is a property of theirs that their eigenstates are orthogonal to each other and span the entire state space, forming an orthonormal basis.

4. When an observable $\hat{\mathbf{A}}$ is measured on a quantum state $|\psi\rangle$, the probability of observing an eigenvalue a_n is given by Born's rule:

   $$\text{Prob}(a_n) = |\langle\langle a_n|\psi\rangle\rangle|^2 \tag{1.9}$$

This scalar product results in a scalar quantity, which represents the probability amplitude when the states are the same or a transition amplitude between different states.

One of the key phenomena in quantum mechanics is the principle of superposition. This principle states that a quantum state may reside in any linear combination of different and distinct eigenstates/basis-states. This principle also reflects the probabilistic nature of quantum mechanics.

Expressing a quantum state $|\psi\rangle$ as a linear combination (i.e. sum) of basis states $|\phi_i\rangle$, we write:

$$\psi = \sum_i c_i \phi_i = c_1\phi_1 + c_2\phi_2 + \cdots + c_n\phi_n \tag{1.10}$$

where each coefficient c_i signifies the probability amplitude associated with the state $|\phi_i\rangle$ within the superposition, encapsulating the state's probabilistic nature. Specifically, the probability is calculated via the *Born Rule*–that is, the probability P_i of finding the system in the state ϕ_i is given by:

$$P_i = |c_i|^2 \tag{1.11}$$

where $|c_i|^2$ is the modulus squared of the complex amplitude c_i.

5. If the measurement leads to an eigenvalue a_n, the quantum state $|\psi\rangle$ collapses to the corresponding eigenstate $|a_n\rangle$.

6. The time evolution of a quantum state is given by:

$$|\psi(t)\rangle = \hat{\mathbf{U}}(\mathbf{t}, \mathbf{t_0})|\psi(t_0)\rangle \tag{1.12}$$

where $\hat{\mathbf{U}}$ is a unitary operator [5]. This evolution conserves the probability distribution over time due to the unitarity of $\hat{\mathbf{U}}$.

Hamiltonian Operator

The Hamiltonian is a quantity that can be measured and hence it is described as an operator in quantum mechanics [26]. It corresponds to the total energy of the system which is equal to the sum of two other energy quantities: kinetic energy and potential energy (both are also represented as operators). Based on the eigenvalue equation, the eigenvalues corresponding to the Hamiltonian are therefore the allowed energy levels of the system.

Displayed below is the energy eigenvalue equation where $|e_i\rangle$ and e_i represent the eigenstate and eigenvalues of the Hamiltonian, respectively:

$$H|e_i\rangle = e_i|e_i\rangle \tag{1.13}$$

1.3 Fundamentals of Quantum Computation

Qubits and Computational Basis

A qubit, or quantum bit, serves as the quantum analog of the classical bit used in information processing [3]. However, contrary to a classical bit that is restricted to binary values of either 0 or 1, a qubit exists as any coherent *superposition* of the basis quantum states $|0\rangle$ and $|1\rangle$. These foundational states $|0\rangle$ and $|1\rangle$ establish a complete orthonormal basis for the two-dimensional quantum state space, and together are frequently referred to as the "computational basis". A qubit's general state is expressed as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.14}$$

where α and β are complex coefficients encoding the probability amplitudes. For the state to be physically realizable, it must be *normalized*, enforcing the sum of the probability to equal 1 (i.e. $|\alpha|^2 + |\beta|^2 = 1$). Based on the principles of quantum mechanics, upon measurement, the qubit collapses to either one of the eigenstates $|0\rangle$ or $|1\rangle$ with respective probabilities $|\alpha|^2$ and $|\beta|^2$. Since it has been stated that quantum states are represented as vectors, the computational basis is therefore presented (in matrix notation) as the following vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.15}$$

Bloch Sphere: A Representation of Quantum States (Qubits)

An excellent visual scheme developed for quantum computing is the concept of a Bloch Sphere, named in honor of physicist Felix Bloch, which is presented in Figures 1.1 and 1.2.

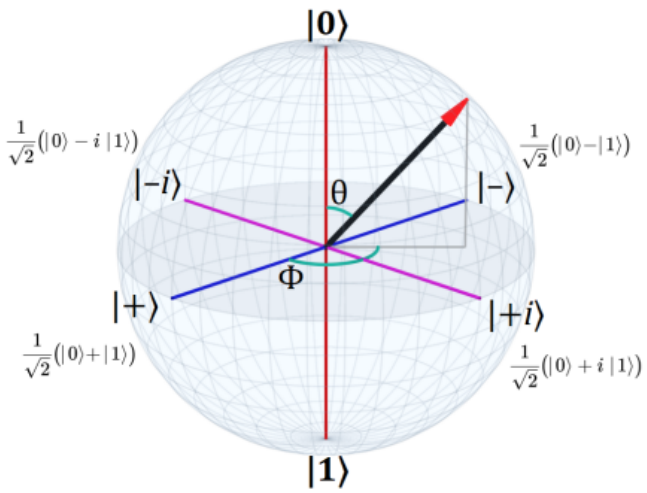


Figure 1.1

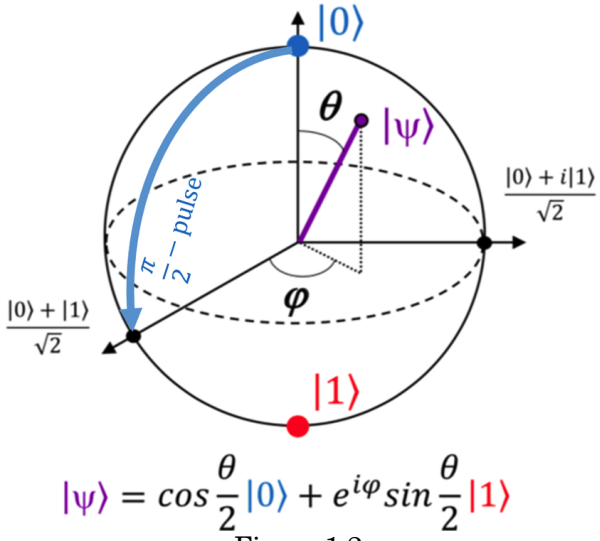


Figure 1.2

Quantum states of a single qubit can be depicted within this three-dimensional Bloch sphere. States contained within this sphere are termed Bloch vectors (or Bloch states). More specifically,

a single qubit *pure state* are states that exist on the surface of this sphere, and are described mathematically as the superposition state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers that span two dimensions. (These coefficients are subject to the normalization condition, where the sum of the probabilities must equal one). A vector that does not lie on the surface of the sphere but rather exists inside it is referred to as a *mixed state*.

The general form of a qubit state on the Bloch Sphere is:

$$|\psi\rangle = e^{i\gamma}\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right) \tag{1.16}$$

where θ and ϕ are real numbers representing the polar and azimuthal angles, respectively, defining the orientation of the Bloch vector. Since the global phase factor $e^{i\gamma}$ does not affect measurements, it can be omitted in most cases, simplifying the state to:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{1.17}$$

The computational basis states $|0\rangle$ and $|1\rangle$ correspond to the north and south poles of the Bloch Sphere, respectively. The state $|\psi\rangle$ is a point on the surface of the sphere, with the angle θ measured from the positive z-axis and the angle ϕ from the positive x-axis.

1.3.1 Quantum Gates

In quantum computing, quantum logic gates are operations that act on qubits thereby transforming the quantum states [15]. Since they are quantum operators, they are therefore represented as matrices and hold all the properties of traditional quantum operators (unitary and whose inverses are self-adjoint (Hemitian)). The action that these quantum logic gates do is some rotation operation.

Single Qubit Gates

Among the most commonly used single qubit gates are the Pauli gates: X (or σ_x), Y (or σ_y), and Z (or σ_z). These gates are represented in matrix form as follows:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.18}$$

It should be noted that the Pauli matrices do not commute, and their square yields the identity matrix, $\sigma_i^2 = I$, for $i = x, y, z$.

Another fundamental single qubit gate is the Hadamard gate (H), which transforms the computational basis states $|0\rangle$ and $|1\rangle$ into superposition states:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad H|0\rangle = |+\rangle, \qquad H|1\rangle = |-\rangle. \tag{1.19}$$

In Dirac notation, the action of these gates on the computational basis states is given by:

$$
\begin{aligned}
X|0\rangle &= |1\rangle, & X|1\rangle &= |0\rangle, \\
Y|0\rangle &= i|1\rangle, & Y|1\rangle &= -i|0\rangle, \\
Z|0\rangle &= |0\rangle, & Z|1\rangle &= -|1\rangle, \\
H|0\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}}, & H|1\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}}.
\end{aligned}
$$

Parameter-Dependent Gates

Rotation gates, which depend on angle-parameters, are also crucial in quantum computing. These gates, denoted by $R_x(\theta)$, $R_y(\theta)$, and $R_z(\phi)$, rotate the state vector on their Bloch sphere about the respective axes:

$$R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{1.20}$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{1.21}$$

$$R_z(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}. \tag{1.22}$$

For a parameter-dependent gate such as a rotation around the Z-axis, denoted by $R_z(\phi)$, the action we have is:

$$
\begin{aligned}
R_z(\phi)|0\rangle &= e^{-i\phi/2}|0\rangle, \\
R_z(\phi)|1\rangle &= e^{i\phi/2}|1\rangle.
\end{aligned}
$$

Multiple Qubit Gates

Multi-qubit gates, such as CNOT, CZ, and the SWAP gate, are operations that affect composite quantum systems:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{1.23}$$

$$\text{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \tag{1.24}$$

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.25}$$

The CNOT gate acts as a conditional NOT, with the first qubit as the control and the second qubit as the target. The CZ gate adds a phase shift to the target qubit conditionally, and the SWAP gate exchanges the states of two qubits.

With this foundational knowledge of qubits and quantum gates, we can proceed to explore variational algorithms and their application in quantum computing. Before doing so however, two concepts crucial to quantum theory will be briefly described. These are *Basis Transformation* and *Entanglement*.

1.3.2 Change of Basis (or Basis Transformation)

One critical and fundamental aspect found in many quantum mechanics problems and a crucial process in performing quantum computational operations, is the idea of changing basis. A vector can be described in various ways, depending on the description of the space–that is, the basis that is relative to the vector-space under consideration. Hence, a vector may be characterized by an alternate set of basis vectors.

To elaborate, the mathematical explanation of changing basis in Dirac notation is represented below.

Vectors can be transformed from one basis to another via:

$$|\psi'\rangle = U\,|\psi\rangle\,, \tag{1.26}$$

where $|\psi'\rangle$ is the vector in the new basis and U is the unitary operator that transforms the vector from the original basis to the new basis.

Operators in the original basis can be transformed to the new basis using the same unitary transformation:

$$O' = UOU^{\dagger}, \tag{1.27}$$

where O is the operator in the original basis, and O' is the operator in the new basis.

For instance, some common operators transform as follows when changing basis:

$$\text{In the standard basis:} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$
$$\text{Transformed by } U: \quad X' = UXU^\dagger, \quad Y' = UYU^\dagger, \quad Z' = UZU^\dagger,$$

These transformed operators X', Y', and Z' represent the Pauli matrices in the basis defined by the eigenvectors of X operator.

The basis transformation of vectors and the corresponding transformation of operators can be seen with this example:

$$\text{In the standard basis:} \quad |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

$$\text{Unitary transformation (Hadamard gate):} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

In the eigenbasis of X :

$$|+\rangle = H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

From the above, the Hadamard gate H is the unitary operator that maps the standard basis to the eigenbasis of the Pauli X operator. Applying H to $|0\rangle$ and $|1\rangle$ yields the states $|+\rangle$ and $|-\rangle$ respectively. These resulting states are, in fact, superposition states.

The entire procedure is described below using the Pauli-X operator as an example:

Given a vector $|\psi\rangle$ in the computational basis $\{|0\rangle, |1\rangle\}$, the change of basis to the eigenbasis of the Pauli X operator $\{|+\rangle, |-\rangle\}$ is performed through the unitary transformation U that diagonalizes the Pauli X operator.

1. Define the computational basis:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

2. Define the eigenbasis of the Pauli X operator:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

3. Construct the unitary transformation matrix U using the eigenvectors $|+\rangle$ and $|-\rangle$:

$$U = \begin{pmatrix} \langle 0|+\rangle & \langle 0|-\rangle \\ \langle 1|+\rangle & \langle 1|-\rangle \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

   The resulting matrix is the Hadamard matrix H.

4. Express a general vector $|\psi\rangle = a|0\rangle + b|1\rangle$ in the new basis using the transformation matrix U:

$$|\psi'\rangle = U|\psi\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} a+b \\ a-b \end{pmatrix}.$$

5. The vector $|\psi'\rangle$ in the eigenbasis of the Pauli X operator is thus given by:

$$|\psi'\rangle = \frac{a+b}{\sqrt{2}}|+\rangle + \frac{a-b}{\sqrt{2}}|-\rangle.$$

Now that it has been shown how to transform a *vector* from one basis to another, let us consider the explicit transformation for the Pauli X operator as an example of transforming *operators* from one basis to another:

$$X' = HXH^\dagger, \tag{1.28}$$

where H is the Hadamard transform that switches between the computational basis and the $\{|+\rangle, |-\rangle\}$ basis.

The computation of this transformed operator can be expanded from (1.28) as follows:

$$X' = \begin{pmatrix} \langle +| \\ \langle -| \end{pmatrix} X \begin{pmatrix} |+\rangle & |-\rangle \end{pmatrix}$$

Continuing to focus specifically on the Pauli-X operator as the example, we further expand the calculation of the transformed Pauli-X operator:

$$X' = \begin{pmatrix} \langle +|X|+\rangle & \langle +|X|-\rangle \\ \langle -|X|+\rangle & \langle -|X|-\rangle \end{pmatrix}$$

Recall that the Pauli-X operator is $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, we can calculate each matrix element of the above equation as:

$$\langle +|X|+\rangle = \langle +| \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |+\rangle = \langle +|0\rangle\langle 1|+\rangle + \langle +|1\rangle\langle 0|+\rangle = \frac{1}{2}(1+1) = 1$$

$$\langle +|X|-\rangle = \langle +| \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |-\rangle = \langle +|0\rangle\langle 1|-\rangle + \langle +|1\rangle\langle 0|-\rangle = \frac{1}{2}(1-1) = 0$$

$$\langle -|X|+\rangle = \langle -| \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |+\rangle = \langle -|0\rangle\langle 1|+\rangle + \langle -|1\rangle\langle 0|+\rangle = \frac{1}{2}(1-1) = 0$$

$$\langle -|X|-\rangle = \langle -| \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |-\rangle = \langle -|0\rangle\langle 1|-\rangle + \langle -|1\rangle\langle 0|-\rangle = \frac{1}{2}(1+1) = 1$$

Thus,

$$X' = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

This example succinctly encapsulates the transition of operators across different bases, a concept that is vital in the analysis and manipulation of quantum systems.

1.3.3 Entanglement

Quantum entanglement is a remarkable quantum mechanical phenomenon where the quantum states of two or more particles become interlinked so that the state of each particle cannot be described independently of the state of the others, even when the particles are separated by extremely large distances.

A quantum state of a multi-partite system (i.e. a composite quantum system consisting of multiple, distinguishable subsystems, with each subsystem considered as a quantum system in its own right, with its own Hilbert space) is said to be separable if it can be factored into a *product of states* of its individual subsystems. If a state is not separable, it is considered to be entangled.

Consider two parties, Alice and Bob, who possess parts of a bipartite system described by the state $|\psi_{AB}\rangle$. This state is separable if it can be written as (a product state):

$$|\psi_{AB}\rangle = |\phi_A\rangle \otimes |\chi_B\rangle\,. \tag{1.29}$$

If such a decomposition into a product of individual states is not possible, we say that the state is entangled:

$$|\psi_{AB}\rangle \neq |\phi_A\rangle \otimes |\chi_B\rangle\,. \tag{1.30}$$

The prototypical example of an entangled state is the *Bell state*, which cannot be factored into a product of individual states of subsystems A and B:

$$\frac{1}{\sqrt{2}}(|0_A\rangle|0_B\rangle + |1_A\rangle|1_B\rangle) \neq |\phi_A\rangle \otimes |\chi_B\rangle\,. \tag{1.31}$$

Methods by which one can determine whether the state is entangled or not relate closer to quantum information theory and will not be further discussed here.

CHAPTER 2

VARIATIONAL QUANTUM EIGENSOLVER

Variational Quantum Eigensolver (VQE) is a hybrid quantum-classical algorithm designed to find approximations to the ground state (lowest) energy of a quantum system. The basis of this method lies in the Variational principle:

Given a Hamiltonian $\hat{H}$ of a system, the true ground state energy E_0 is the lowest possible energy eigenvalue. The variational principle states that for any trial wavefunction ψ_{trial} that is normalized (i.e., $\langle\psi_{\text{trial}}|\psi_{\text{trial}}\rangle = 1$), the expectation value of the Hamiltonian with respect to this trial wavefunction will be greater than or equal to the true ground state energy:

$$E_0 \leq \langle\psi_{\text{trial}}|\hat{H}|\psi_{\text{trial}}\rangle$$

As an overview, the procedure of this method involves choosing a "trial wavefunction" depending on one or more parameters, and finding the values of these parameters for which the expectation value of the Hamiltonian operator (representing energy) is the lowest possible [14]. The wavefunction obtained by fixing the parameters to such values is then an approximation to the ground state wavefunction, and the expectation value of the Hamiltonian in that state is an upper bound to the ground state energy. In other words, the VQE operates by preparing a parameterized quantum state on a quantum computer, measuring the expectation value of the Hamiltonian, and then through the use of a classical optimization algorithm it adjusts the parameters iteratively to minimize the attained expectation value.

2.1 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE) algorithm falls under the umbrella of Variational Quantum Algorithms (VQA)–which are based on the variational method of quantum mechanics

described above. This class of quantum algorithms, first introduced in 2013 by Alberto Peruzzo, Alán Aspuru-Guzik and Jeremy O'Brien [17], offers a promising outlook for the use of quantum computers in the near-term NISQ era (noisy-intermediate quantum algorithms era) as they are hybrid algorithms meaning that such algorithms use both classical computers and quantum computers to achieve the goal in finding the optimal solution (i.e. the global maximum or minimum) to a problem. In the specific case of the VQE algorithm, this optimal solution is the ground state energy of a given physical system – the minimum quantity of energy that can be available in that system.

2.2 The General VQE Pipeline

The general VQE outline is as follows: given an ansatz–that is, a "guess" of an initial trial function to start the algorithm–the quantum computer then calculates the expectation value of the system with respect to an observable–the Hamiltonian–using the ansatz [11]. The process is followed by a classical optimizer to improve the initial guess of the wavefunction.

The overall framework is presented below:

1. Hamiltonian Representation and Construction The Hamiltonian represents the physical system. Hence, many different forms of Hamiltonians exist in the physical sciences in order to model the various complex systems that exist in nature. Examples include models like lattice models and vibrational spectroscopy, with each making specific assumptions when defining the Hamiltonian to model the system. While those assumptions hold merit for certain purposes, here will be presented and discussed the electronic structure Hamiltonian, whose construction is generally developed for the primary purpose of reducing the complexity of the eigenvalue problem.

   The initial stage of the Variational Quantum Eigensolver (VQE) procedure is to define the Hamiltonian, that is, the system for which we want to find the ground state energy. The Hamiltonian construction is done by identifying the set of operators and their associated weights between basis functions that span the space representing the physical system. These basis functions symbolize the individual particle's degrees of freedom. This first step in the VQE process is crucial as setting up the Hamiltonian – with their operators, their associated wavefunctions, and the choice of basis – will have a profound impact on the computational resources, accuracy, and outcome of the calculation.

   The different depictions of the electronic structure include mean-field calculations, local-atomic functions, or plane-wave methods. These are all depictions of the spatial distribution–"orbitals"– of the single-particle Fock states that form the basis of the many-body system. There is further complexity when considering electrons as the Pauli Exclusion Principle

dictates that the wavefunction must be anti-symmetric with respect to the exchange of two electrons (i.e., fermionic particles cannot be distinguished from one another). A decision must be made as to whether the anti-symmetry must be imposed via the definition of the wavefunction or through the definition of the operators. This has been historically called "first-" and "second-quantization", respectively.

In second quantization, the Hamiltonian is expressed in terms of fermionic operators known as the creation and annihilation operators (denoted a_j, and $a_j^\dagger$, respectively). These *fermionic* operators indicate/dictate to either add or subtract an electron from a specifically indexed (labeled j) basis function (representing an orbital or lattice position). Furthermore, these operators inherit the fermionic anti-symmetry condition for the exchange of two particles. Second quantization is intended as a simplified procedure to ensure anti-symmetry.

2. Operator Encoding

   Operator encoding is a necessary process in quantum computing as quantum computers are limited to measuring observables only in the Pauli basis (because spins of particles are binary – either spin up or spin down). This Pauli basis is expressed as: $\{I,X,Y,Z\}^{\otimes N}$ for N qubits. In first quantization, operators can be translated into spin-operators directly since they are not used for the anti-symmetry condition. However, in the second quantization (for which this thesis is concerned with), the Hamiltonian must be represented as a linear combination of the fermionic (i.e. creation and annihlation) operators to enforce the anti-symmetry. Therefore, the Hamiltonian in this form must be transformed into a form that is made up of a string of spin – or "Pauli"– operators. The encoding has an effect on gate depth and other quantum computing resources, especially when considering the choice of ansatz (which will be discussed later).

3. Measurement Strategy

   When the resultant Hamiltonian has been established as a string of Pauli-operators, it is necessary to appropriately group the Hamiltonian terms for the measurement of their expectation values. The purpose of this is to reduce the number of quantum circuit completions–that is, measurements–also known as *shots*. This is important as the greater number of shots implies a greater number of computational resources. The primary method by which to do this is for one to exploit the commutative property between the groups of operators in the Hamiltonian which (because of this property) can be measured together, thereby reducing the number of necessary shots. Considering this implies that the quantum circuit must be designed such that it will process each group appropriately and perform the joint measurement.

4. Ansatz Implementation and State Preparation

The procedure following the Hamiltonian setup in the VQE process is to appropriately prepare the trial wavefunction. This is known as "state preparation" and it first requires one to choose the structure of the parametrized quantum circuit (called the *ansatz*). This ansatz generates the trial state with which the Hamiltonian may then be measured. This trial state becomes the model for the ground state wavefunction of the system. Various ansatz have been proposed, with each offering benefits while carrying certain limitations in the context of the problem at hand. Therefore the selection of the ansatz is crucial. The main question regarding ansatz selection is the ansatz's capability of encapsulating and reaching the various states in the Hilbert space representing the problem. This ability of reaching a large set of states in the Hilbert space is defined as the *expressibility*. Another critical question is regarding the ansatz practical ability on the quantum hardware, considering factors like the number of parameters and their linear dependence, the optimization landscape, and the phenomenon of barren plateaus. This is known as *trainability*. An effective ansatz balances the expressivity to ensure accurate ground state representation without becoming overly complex that it makes the target state search unfeasible and daunting. Lastly, it is important to mention that the choice of ansatz will also be crucial as it heavily influences the VQE's tolerance to noise.

5. Parameter Optimization

   The next step in the VQE pipeline is to iteratively optimize the parameters of the ansatz until a convergent solution is achieved. To do this, one must compute the expectation value of the Hamiltonian multiple times to develop an update rule for the parameters. Selecting an optimization is important as it affects the shot-number, and the number of iterations necessary for reaching the converging solution. Moreover, different optimizers have different trade-offs.

6. Error Mitigation

   The attention now shifts to the last part of the VQE process: error mitigation. Noise is one of the main challenges in the evolution of quantum technology. Hence, error mitigation is the process of reducing the effect of quantum noise through post-processing techniques on the measurement data. More on this topic will be discussed later.

2.2.1 The (electronic structure) Hamiltonian

The construction of a Hamiltonian ought to be the first step in any quantum computing scenario. This is true because the Hamiltonian represents the system which is being studied. In order to construct the Hamiltonian that models the physical system, one must have a suitable understanding of the physical processes (physics and chemistry) behind the system, represent this mathematically, and then translate this mathematical representation in the language of quantum computing–namely, in terms of qubits and gates.

In quantum chemistry, the Hamiltonian is the mathematical object known as an "operator" that represents the total energy of a system. Mathematically, the operator is represented by a matrix.

The molecular Hamiltonian is the representation of the total energy (hence, an operator) of a molecular system defined by its atomic configuration. After attaining this molecular Hamiltonian–which also holds information on the geometry of the system– the electronic *wavefunction* must be obtained as this contains the lowest (i.e. ground state) energy. This wavefunction presents the correlated probability amplitudes of the electrons in the space surrounding the nuclei.

Assuming non-relativistic settings and ignoring the motion of the much-heavier nuclei (i.e. Born-Oppenheimer approximation), the electronic Hamiltonian can be written as:

$$\hat{H} = \hat{T}_e + \hat{V}_{ne} + \hat{V}_{ee}, \tag{2.1}$$

where

$$\hat{T}_e = -\sum_i \frac{\hbar^2}{2M_i} \nabla_i^2, \tag{2.2}$$

$$\hat{V}_{ne} = -\sum_{i,k} \frac{e^2}{4\pi\epsilon_0} \frac{Z_k}{|\mathbf{r}_i - \mathbf{R}_k|}, \tag{2.3}$$

in which R_k are the nuclear positions, and

$$\hat{V}_{ee} = \sum_{i<j} \frac{e^2}{4\pi\epsilon_0} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}. \tag{2.4}$$

Equations (2.3) and (2.4) depict the potential energy due to electron-nucleus and electron-electron interactions, respectively. Also, $\mathbf{r}_i$ signifies the position of electron i, M_i its mass, Z_k the atomic number of nucleus k, e is the elementary charge, $\hbar$ is the reduced Planck constant, and ∇_i^2 is the Laplace operator acting on electron i [25].

Second Quantization

In quantum computing, second quantization is important in order to formulate problems in a way that can be processed by quantum algorithms [24]. In the first quantization, each particle in a system is described by its own wave function, and the overall state of the system is a product of these individual wave functions. This approach becomes cumbersome for systems with a large number of particles. But second quantization shifts the focus from individual particles to the states that particles can occupy.

The central idea behind second quantization is the principle that particles in quantum mechanics are indistinguishable. That is, in classical mechanics, different arrangements of the

position vectors represent distinct many-body states. However, in the quantum realm, particles are identical, and swapping any two particles does not result in a new many-body quantum state. This means that the quantum many-body wave function remains consistent (except for a potential phase change–i.e. a sign change) when two particles are exchanged. As a result, the two types of particles–bosons and fermions– will have a many-body wave function that is either symmetric or antisymmetric, respectively, upon an exchange of particles.

The wave function for Bosons and Fermions:

$$\Psi_{\mathrm{B}}(\ldots,\mathbf{r}_i,\ldots,\mathbf{r}_j,\ldots) = +\Psi_{\mathrm{B}}(\ldots,\mathbf{r}_j,\ldots,\mathbf{r}_i,\ldots)$$

$$\Psi_{\mathrm{F}}(\ldots,\mathbf{r}_i,\ldots,\mathbf{r}_j,\ldots) = -\Psi_{\mathrm{F}}(\ldots,\mathbf{r}_j,\ldots,\mathbf{r}_i,\ldots)$$

The first step is to then understand the total energy of the molecule being studied. In this thesis, we present the case of Lithium Hydrite (LiH) as the molecule under study. To do this, one must first define the **electronic Hamiltonian** of the molecule in question, which describes the behavior of electrons under the influence of the nuclei.

$$\hat{H} = \sum_{ij} t_{ij} a_i^\dagger a_j + \frac{1}{2} \sum_{ijkl} V_{ijkl} a_i^\dagger a_j^\dagger a_l a_k$$

in which:

- t_{ij} are the one-electron integrals, representing the kinetic energy of the electrons and their interaction with the nuclei.
- V_{ijkl} are the two-electron integrals, representing the repulsion between pairs of electrons.
- $a_i^\dagger$ and a_j are the creation and annihilation operators, which add or remove an electron in the i-th or j-th quantum state, respectively.

The above Hamiltonian describes the total energy of the system, accounting for both the individual movements of electrons and their interactions.

2.2.2 Ansatz

An ansatz consists of a quantum circuit or a quantum state represented in a parameterized form. This means that the ansatz includes adjustable parameters (angles) that can be tuned during the optimization process to explore different quantum states.

Ansatz may be classified into two different types: the *fixed structure ansatz* which are those ansatz that are set at the beginning of the optimization process and do not change throughout, and the *adaptive structure ansatz* which are made to iteratively optimize. Another important aspect to note regarding ansatz is the metrics used to define and measure their performance objectively, making them easy to compare against one another. These include:

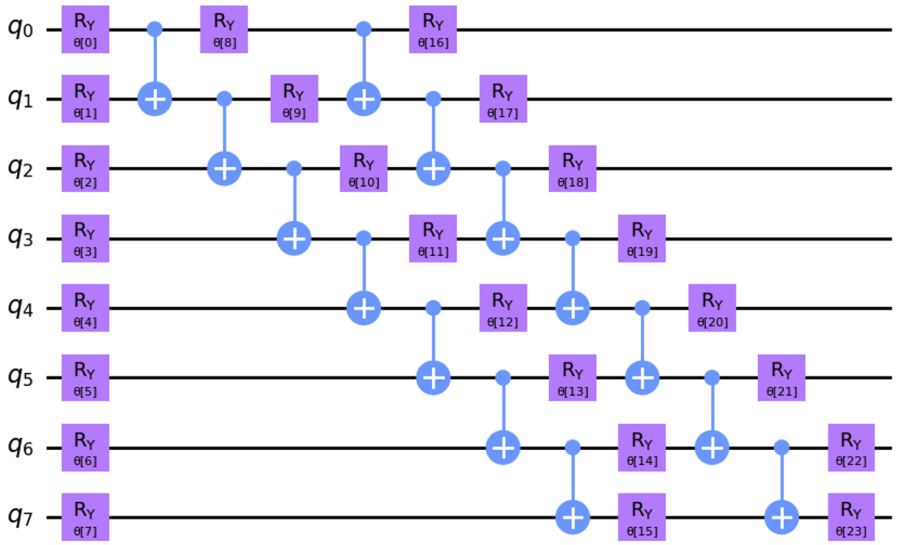


Figure 2.1: HEA Structure

1. The depth of the ansatz (number of sequential operations required for the implementation), which impacts the overall runtime of the method, its resilience to noise, and to barren plateaus.

2. The number of parameters, which significantly influence the overall runtime of the implementation (although this cost can, in theory, be entirely parallelized) and the complexity of the optimization process.

3. The number of entangling gates, which is generally the main source of noise resulting from execution of a quantum circuit.

2.2.3 Hardware-Efficient Ansatz (HEA)

The Hardware-Efficient Ansatz (HEA) is currently a widely used approach in quantum computing. This is likely due to the limitations of the current state-of-the-art quantum-computing hardware which thereby give reason to use an ansatz that is particularly adjusted to the quantum device being used. Nonetheless, the HEA ansatz has a common structure: repeated blocks/"layers" consisting of single-rotation gates followed by connecting entangling gates.

A general example of a HEA-structure can be visualized in 2.1

The mathematical representation of the HEA ansatz is as follows:

$$|\psi(\Theta)\rangle = \left[\prod_{i=1}^{d} U_{\text{rota}}(\theta_i) \times U_{\text{ent}}\right] \times U_{\text{rota}}(\theta_{d+1})|\psi_{\text{init}}\rangle\,,$$

where:

- $U_{\text{rota}}(\theta_i) = \prod_{q,p} R_p^q(\theta_p^q)$, with q representing qubit and $p \in \{X, Y, Z\}$ (Parametrized Pauli rotations)

Main Advantages of HEA include:

- flexible to adjust on the appropriate device with different rotation- and entangling-gates
- simplicity in construction

Main Disadvantages of HEA include:

- does not guarantee the spanning of the entire Hilbert space
- problems and limitations by barren plateaus
- typically requires more parameters with more qubits involved when compared to other ansatz
- theorized to be less efficient for larger scale problems

2.2.4 Unitary Coupled Cluster (UCC) Ansatz and Variations

The Unitary Coupled Cluster (UCC) Ansatz is perhaps the most applicable ansatz for the VQE and has garnered significant attention in recent years. The "coupled cluster method" is a well known method originating in traditional (classical) chemistry computations. It works by taking a simple guess of a molecule's state and then systematically improving it by considering how electrons in the molecule interact with each other. However, this ansatz was not used in this research.

2.2.5 Barren Plateaus

One of the major obstacles presented against the variational quantum algorithms is the "barren plateau" phenomenon [13]. This is where optimization gradients vanish thereby giving a flat optimization landscape and making it challenging to identify the direction of optimization. In other words, the cost function essentially becomes flat thereby making it extremely difficult to find a minima or maxima. This leads to a search that is a stochastic (i.e. random) gradient estimation. Despite mentioning gradients, gradient-free optimizers (like the particle swarm optimization method that will be discussed later) progress iterative by sampling the cost landscape of specific parameters, and so if the variance across the landscape is too small, then it becomes impossible to accurately progress through the optimization step. The barren plateau problem implies that the expectation value of an observable with respect to a random state concentrates exponentially around the mean value of that observable [13], rendering the optimization process intractable away from the mean [25]. This problem is dependent on the number of qubits at play, the expressibility of the ansatz, the locality of the cost function, the level of entanglement of the trial wavefunction, and/or the amount and presence of quantum noise. For example, it has been

shown that the more expressive an ansatz is, the more likely it is to encounter the barren-plateau problem [10]. It seems that the barren plateau phenomenon is an inherent property that emerges as a result of entanglement, and since entanglement is a key feature of quantum computation, the phenomenon of barren plateaus is an unavoidable situation.

CHAPTER 3

PARTICLE SWARM OPTIMIZATION

This thesis is about applying *non-gradient optimization methods* to the VQE procedure as the classical method of optimizing the parameters in search for the lowest energy levels of molecular systems. The classical optimization method that is discuss here is known as the Particle Swarm Optimization algorithm (PSO).

3.1 Introduction to Particle Swarm Optimization

The principle idea behind the original Particle Swarm Optimization method is that the collective and self-organized system can identify the best solution of some existing search space. The collective entity – known as a swarm – is made up of n individual particles [12].

Each particle follows the following update rule in each iteration:

$$\text{Velocity update: } v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (p_{\text{best}}^{(t)} - x_i^{(t)}) + c_2 \cdot r_2 \cdot (g_{\text{best}}^{(t)} - x_i^{(t)}),$$
$$\text{Position update: } x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}.$$

where v_i and x_i are the velocity and position of the i-th particle at the t-th iteration, respectively. p_{best} denotes the best position that the particle has encountered thus far, and g_{best} represents the best position found by the entire swarm. The coefficients w, c_1, and c_2 are the inertia weight, cognitive coefficient, and social coefficient, respectively, which balance the exploration and exploitation abilities of the swarm. Random variables r_1 and r_2 are introduced as to provide a stochastic feature to the update process, enhancing the algorithm's searching ability.

As the swarm iterates through the search space, the particles adjust their trajectories based on both their individual experiences and the collective knowledge of the swarm. The balance between these two functions is crucial for effectively converging to the optimal solution without over-exploring or over-exploiting the search space.

Reasons that motivate for the study and implementation of the PSO algorithm include the inherent ability in the PSO algorithm to efficiently solve high-dimensional and complex optimization problems, and its ease and simplicity in implementation.

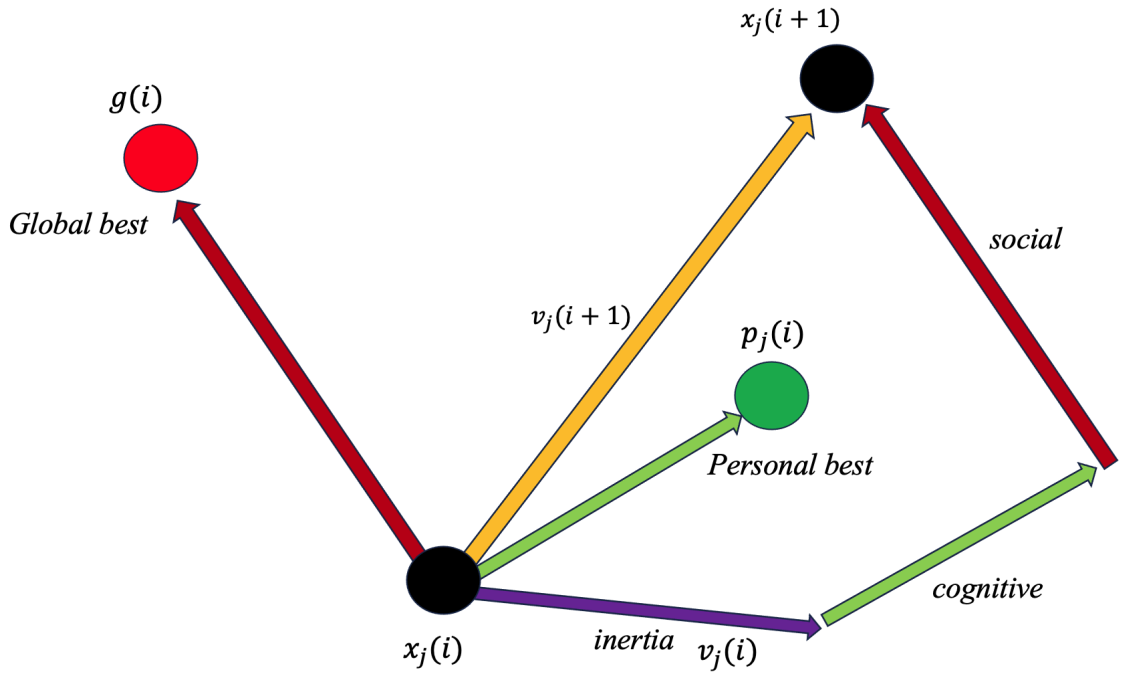


Figure 3.1: An example of the movement of a particle in a two-dimensional space based on the PSO algorithm in a single iteration. An inertia term given by velocity that drives the particle in some direction, a memory term (p_j) that influences the particle's trajectory based on its history, and a global-best cooperation term (g) that reflects the best result amongst the entire swarm which also influences the particle's projected movement. Beware the different notation used in this diagram: i indicates iteration, while j indicates particle number unlike in the equations presented above.

3.2 Constriction Factor PSO

The original PSO was developed in 1995 by Eberhart and Kennedy having been inspired by social behavior patterns of birds flocking or fish schooling [8]. Since then the PSO algorithm has evolved various variants and undergone crucial analysis. For example, the original PSO did not include the w "inertia" term. An analysis of this original PSO done by the authors, and another study done independently by Ozcan and Mohan (1998) was done without an inertial weight until it was analyzed to be an extremely helpful requirement [16]. The particular PSO that we have chosen to study and use as a comparison to other optimization methods is known as the "Constriction Factor PSO". Introduced by Clerc and Kennedy in 2002, the constriction factor PSO variation incorporates a constrictor coefficient to control the particle's velocity, thereby preventing the particles from exploding to infinity and helping to converge more rapidly to an optimal solution

[4]. This model was a significant advancement over the original PSO, giving better stability and convergence properties.
This PSO-algorithm velocity update equation is modified as follows:

$$\text{Velocity update: } v_i^{(t+1)} = \chi \left[v_i^{(t)} + c_1 \cdot r_1 \cdot (p_{\text{best}}^{(t)} - x_i^{(t)}) + c_2 \cdot r_2 \cdot (g_{\text{best}}^{(t)} - x_i^{(t)}) \right],$$
$$\text{where } \chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|},$$
$$\varphi = c_1 + c_2, \quad \varphi > 4.$$

As previously mentioned, r_1 and r_2 are random vectors, with each element drawn from a uniform distribution in the range [0, 1]. These vectors introduce stochasticity into the particle updates and ensure that the search space is explored thoroughly.

The coefficients c_1 and c_2 are the cognitive and social coefficients, respectively. The cognitive coefficient c_1 influences how much a particle is attracted to its own best-known position, representing the particle's memory or "cognition" of its own experience. The social coefficient c_2 influences how much a particle is attracted to the best-known position found by the swarm, representing the "social" interaction between particles.

The mathematical restrictions on c_1 and c_2 in the standard PSO (from here on will be shortened to SPSO) are typically related to ensure that the swarm converges to a solution without becoming unstable or exhibiting excessively random behavior. This is the motivation for the constriction factor χ seen in the constriction factor PSO. As previously stated, χ's role is to control the velocities and ensure a stable proper convergence. In fact, it behaves as a damping term to reduce explosions in the system. The requirement that $\phi > 4$ was shown in the PSO analysis [4] to be an important implementation for the constriction factor to be meaningful. This term, which is the sum of the cognitive and social coefficients, is called the “acceleration constant”. Its behavior is such that if it is set too small, the trajectory of a particle falls and rises quite slowly; but as its value is increased, the frequency of the particle oscillating around the weighted average of its personal best and global best is also increased.

In practice, c_1 and c_2 are often set to values that allow the swarm to balance exploration and exploitation effectively. A common choice is to set $c_1 = c_2 = 2.05$, which, using the above formula for χ, gives a constriction factor of approximately 0.729.

It is important to note that the choice of these parameters can greatly affect the performance of the PSO algorithm, and they may require tuning based on the specific problem being solved.

3.3 Pseudocode of Basic PSO

Provided here is the pseudocode for the basic original PSO algorithm, followed by a description of each variable in the table:

Algorithm 1 Basic PSO Algorithm

Randomly generate an initial population of particles with positions x_i and velocities v_i.
while termination criterion is not met **do**
    for each particle i **do**
        if $f(x_i) < f(p_i)$ **then**
            $p_i \leftarrow x_i$
        end if
        $p_g \leftarrow \min(p\_neighbours)$
        for each dimension d **do**
            VELOCITY_UPDATE($v_i[d], p_i[d], p_g[d], x_i[d]$)
            POSITION_UPDATE($x_i[d], v_i[d]$)
        end for
    end for
end while
function VELOCITY_UPDATE($v_i[d], p_i[d], p_g[d], x_i[d]$)
    $v_i[d] \leftarrow w \cdot v_i[d] + c1 \cdot \texttt{rand()} \cdot (p_i[d] - x_i[d]) + c2 \cdot \texttt{rand()} \cdot (p_g[d] - x_i[d])$
end function
function POSITION_UPDATE($x_i[d], v_i[d]$)
    $x_i[d] \leftarrow x_i[d] + v_i[d]$
end function

Table 3.1: Explanation of Symbols Used in PSO Algorithm

Symbol	Description
x_i	Represents the position of the i-th particle, in d-dimension.
v_i	Represents the velocity of the i-th particle, in d-dimension.
p_i	The best-known position of the i-th particle (personal best), in d-dimension.
p_g	The best-known position among the particle's neighbours (global best), in d-dimension.
f	The fitness function to be minimized.
w	The inertia weight.
$c1$ and $c2$	Cognitive and social coefficients.
`rand()`	A random number between 0 and 1.

3.4 Problems with PSO

While the PSO algorithm boasts several advantages such as robustness, simplicity, ease of implementation, and a potential for parallelization, it is not without significant drawbacks. These drawbacks can hinder its effectiveness and efficiency, especially in complex or dynamic optimization problems [18].

One of the primary issues with PSO is its tendency to prematurely converge to local optima, especially in complex and high-dimensional search spaces. This issue with premature convergence occurs because the particles in the swarm can become overly influenced by a few individuals that have found relatively good solutions early on, leading the entire swarm to cluster around suboptimal areas of the search space, also known as "local minima". This is a limitation of the algorithm's exploratory ability.

Another significant problem with PSO is its sensitivity to the choice of parameters. While there are only three tunable parameters: inertia weight, cognitive coefficient, and social coefficient, selecting inappropriate parameter values can either cause the swarm to converge too quickly, missing the global optimum, or prevent convergence altogether, leading to erratic and inefficient search behavior. Finding the right parameter settings to balance between exploitation of the area or exploration of new area is therefore crucial for the algorithm to navigate the search space effectively and successfully. This is particularly challenging when the search space is highly complex and/or dynamic, and also since different problems have different solution landscapes (objective functions). Moreover, analysis done on the PSO indicates a specific required settings for the parameters (which we have stated them to be $c_1 = c_2 = 2.05$ and an inertia weight of 0.729). Any adjustments to these settings will cause significant changes and most likely result in an unstable and poor PSO performance.

Lastly, PSO is generally computationally intensive, particularly for large-scale optimization problems. The algorithm requires the evaluation of the objective function for each particle in the swarm at every iteration, leading to high computational costs. This can be particularly problematic for problems with expensive objective functions or when the number of particles and iterations required for convergence is large. The motivation for studying the PSO and applying it to VQE problems was largely due to the PSO's parallelization abilities. Nonetheless, we have developed a new optimization algorithm that we are applying to VQE problems which address the optimization impediments of the particle swarm optimization method. This new algorithm will be discussed in the next chapter.

CHAPTER 4

HOPSO: HARMONIC OSCILLATOR PARTICLE SWARM OPTIMIZATION

4.1 Introduction and Motivation

In the previous chapter it has been shown how the Particle Swarm Optimization algorithm operates, and what advantages and disadvantages it has. Despite having many benefits, the PSO algorithm has some issues which can be improved upon. To reiterate what has been stated in the last chapter, the primary issue with the PSO is that the few parameters involved require careful tuning to achieve optimal performance. The number of particles in the swarm, the inertia weight, and the cognitive and social components of the algorithm all dramatically influence the evolution of the search, and incorrect parameter settings can lead to poor results including premature convergence or explosions. Tuning these parameters requires extensive experimentation and domain knowledge, which can be time-consuming and may not always yield satisfactory results, especially when applying PSO to various problems. In general, analysis on the PSO performance indicates a few specific settings of the inertia, cognitive and social coefficients, which lead to proper performance. But practically any adjustment to these settings will result in explosions or premature convergence.

To retaliate and improve on the drawbacks of the PSO, an optimization algorithm is presented here that takes the general idea of the particle swarm optimization, but modifies it such that the algorithm is based on some physically-inspired system, namely the harmonic oscillator. As a result, not only is it easier to keep track of the motion (since we know the equations and movement of harmonic oscillations), but we are also able to fine-tune the algorithm such that it will avoid explosions as seen in the PSO algorithm, and will also converge in a more controlled manner.

Particularly, suppose that each particle searches the space by oscillating about a term which represents the balance between the personal- and global-best positions. This motion is represented as simple harmonic motion. Specifically, the particle acts as a damped oscillator.

The motivation behind this idea is that searching the space with this physical model will ensure a convergence in a controlled manner, and avoid explosions entirely as the physical model implies that the energy cannot increase in the search. This model therefore allows not only for a more finely-tuned search overtime but also guanrantees a convergence. It's controllability should also result in a stronger resilience against getting trapped in a local minima, something that the PSO often struggles with.

4.2 HOPSO Algorithm: a Modified PSO

4.2.1 Damped Oscillation Introduction and Review

Undamped Mass-Spring System:

Imagine a mass m attached to a spring with spring constant k. The spring exerts a force on the mass proportional to the displacement x from equilibrium according to Hooke's law: $F_s = -kx$. The negative sign indicates the force is always opposite the displacement, acting as a restoring force. Applying Newton's 2nd law $F = ma$ gives the equation of motion: $m\frac{d^2x}{dt^2} = -kx$ Rearranging: $\frac{d^2x}{dt^2} + \frac{k}{m}x = 0$ Let $\omega_0^2 = \frac{k}{m}$, then: $\frac{d^2x}{dt^2} + \omega_0^2 x = 0$ This is the differential equation for simple harmonic motion. The general solution is: $x(t) = A\cos(\omega_0 t + \phi)$ where A is the amplitude, $\omega_0 = \sqrt{\frac{k}{m}}$ is the natural frequency, and ϕ is the phase determined by initial conditions.

Damped Mass-Spring System: 2 Now suppose a damping force is added, which is proportional to the velocity, $F_d = -b\frac{dx}{dt}$, where b is the damping coefficient. This could represent air resistance or friction in the system.
The equation of motion becomes:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = 0 \tag{4.1}$$

Dividing through by m:

$$\frac{d^2x}{dt^2} + \frac{b}{m}\frac{dx}{dt} + \frac{k}{m}x = 0 \tag{4.2}$$

Let $2\gamma = \frac{b}{m}$ and $\omega_0^2 = \frac{k}{m}$, then:

$$\frac{d^2x}{dt^2} + 2\gamma\frac{dx}{dt} + \omega_0^2 x = 0 \tag{4.3}$$

This is the damped harmonic oscillator equation.
The solution depends on the relative values of γ and ω_0:

Overdamped ($\gamma > \omega_0$):

$$x(t) = c_1 e^{-(\gamma - \sqrt{\gamma^2 - \omega_0^2})t} + c_2 e^{-(\gamma + \sqrt{\gamma^2 - \omega_0^2})t} \tag{4.4}$$

Critically damped ($\gamma = \omega_0$):

$$x(t) = (c_1 + c_2 t)e^{-\gamma t} \tag{4.5}$$

Underdamped ($\gamma < \omega_0$):

$$x(t) = e^{-\gamma t}[c_1 \cos(\omega t) + c_2 \sin(\omega t)] \tag{4.6}$$

where $\omega = \sqrt{\omega_0^2 - \gamma^2}$.

The constants c_1 and c_2 are determined by initial conditions.

For the underdamped case, the oscillations decay exponentially with a time constant $\tau = \frac{1}{\gamma}$. The "frequency" is reduced to $\omega = \sqrt{\omega_0^2 - \gamma^2}$. Many real systems are underdamped and exhibit decaying oscillations, such as a mass bouncing on a spring. The critical damping coefficient b_c occurs when $\gamma = \omega_0$, or $\frac{b_c}{2m} = \sqrt{\frac{k}{m}}$. Solving for b_c: $b_c = 2\sqrt{mk}$ This represents the boundary between oscillatory and non-oscillatory motion.

In summary, the key equations for damped mass-spring systems are: the Equation of motion: $\frac{d^2x}{dt^2} + 2\gamma\frac{dx}{dt} + \omega_0^2 x = 0$, the Natural frequency: $\omega_0 = \sqrt{\frac{k}{m}}$, the Damping ratio: $\zeta = \frac{\gamma}{\omega_0} = \frac{b}{2\sqrt{mk}}$, the Pseudo-frequency (underdamped): $\omega = \omega_0\sqrt{1-\zeta^2}$, the Critical damping: $b_c = 2\sqrt{mk}$. The type of motion (overdamped, critically damped, underdamped) depends on the relative values of the damping coefficient b compared to the critical damping b_c.

4.2.2 HOPSO Algorithm Explained

In the above section that served as a refresher on damped systems, we see that the equation of motion for the damped harmonic oscillator (re-written):

$$x(t) = A_0 e^{-\lambda t}\cos(\omega t + \theta) \tag{4.7}$$

where, x(t), A_0, λ, ω and θ represent position of the particle at time t, initial amplitude, damping factor, angular frequency and initial phase of the oscillation respectively.

For this harmonic-oscillator inspired PSO-based algorithm, we consider this solution to the damped-harmonic oscillator to represent each particle's position (in each dimension of the position vector) along with a damping term as a control mechanism to vanquish the issue of

swarm-explosion as found in the standard PSO algorithm.

Since we now have the position equation for the algorithm, we know the velocity equation as being simply the derivative of the position variable (4.7) at any time t:

$$v(t) = -\omega(A_0 e^{-\lambda t}\sin(\omega t + \theta)) - \lambda(x(t)) \tag{4.8}$$

The time parameter t is generally taken to be equal to one in the standard PSO and reflects the change in iteration. Here in HOPSO however, the time parameter does not relate with the change in iteration but rather the positions are sampled randomly in time. This is the source of randomness in the HOPSO algorithm unlike r_1 and r_2 in PSO. The time is chosen randomly within the interval $[0,t_{ul}]$ for each particle and in each dimension, where t_{ul} is the upper limit of the sampling range. The iterative change in parameter t is shown in (4.9).

$$t(i+1) = t(i) + rand[0, t_{ul}], \tag{4.9}$$

where i is the iteration. We generally set this upper limit lower than 2π, which is the period of the oscillation. As a rule of thumb, one should choose a lower t_{ul} for sampling positions close to each other and higher t_{ul} to increase the chances of sampling positions far away from each other.

The HOPSO optimization proceeds as follows:

Initialize each particle j in the swarm with random positions x_j and velocities v_j uniformly distributed over a D-dimensional parameter search space bounded by $[-\pi, \pi]$.

Note the initial personal best position p_j to be the initial position for each particle, and define the global best position g as the best position among all particles.

After knowing the global and personal best position, calculate the position of the attractors for each particle as

$$a_{j,d} = \frac{|c_1 p_{j,d} + c_2 g_d|}{c_1 + c_2} \tag{4.10}$$

where a, p, g represent the position of the attractor, personal best position of the particle and global best position. The c_1 and c_2 represent the weights of attraction towards personal-best and global-best positions, respectively. Typically, the values c_1 and c_2 are set equal, thereby the attractor lies equidistant between the personal best and global best position. The subscript j represents the index of the particle and subscript d represents the dimension.

The initial amplitude for each particle is determined with initial conditions when setting t as zero. We solve (4.8) to obtain the A_0 for each particle:

$$A_0 = \sqrt{(x(0) - a)^2 + \frac{(v(0) + \lambda(x(0) - a))^2}{\omega^2}} \tag{4.11}$$

Once the amplitude has been determined, the initial phase of the oscillation can be

calculated as

$$\theta = \arccos \frac{x(0) - a_{j,d}}{A_0} \tag{4.12}$$

We then let the particles oscillate in time. For every iteration, we stop the clock at a random time, calculate the values of the cost function for all the particle positions. If there is a change in p_j, then the attractor for the j^{th} particle is recalculated using (4.10) while the amplitudes are recalculated by resetting the time for that particular particle as zero using (4.11). If instead there is a change in g, then all the attractors are changed accordingly using (4.10). The amplitudes are also recalculated in this case.

4.2.3 Amplitude-Threshold Equations

When the amplitudes are first calculated, it is possible that the amplitude of oscillation is calculated to be not large enough to explore the landspace properly between the personal best position and the global best position about its attractor. As a solution to this, we provide the option for the amplitude to take on either the initially calculated amplitude, or an added lower bound in which the amplitude is at least half the absolute difference between the personal-best and global-best position times a multiplier m. The choice depends on which one is more advantageous for the search (i.e. a greater amplitude). This is reflected mathematically as:

$$A_0 = \max(A_0, \frac{|p_{j,d} - g_d|}{2} * m) \tag{4.13}$$

The multiplier term (m) is applied to the amplitude to account for the magnitude of the c_1 and c_2 (since (4.10) only accounts for ratio between c_1 and c_2 and not the magnitude). This is taken as inspiration from the PSO.

We also make a 'cut' on the amplitude of the oscillation of each particle. That is, irrespective of whether a global or personal best position is found, the particle's oscillation will decay until it reaches a threshold and then continue to oscillate with a constant amplitude, as seen below:

$$A = \max(A_0 e^{-\lambda t}, \frac{|p_{j,d} - g_d|}{2} * m) \tag{4.14}$$

This amplitude-cut is done so that the particle continues searching in a reasonable-sized space.

Figure (4.1) is the general representation of the HOPSO algorithm. Specifically, this figure is a one-dimensional visualization that demonstrates a single particle's oscillation in one-dimension, about the attractor a_i which is set half-way between its personal best position (p_j) and the swarm's global best position (g) according to the weighted average equation (4.10) when $c_1 = c_2 = 1$.

In the movement of the HOPSO algorithm we find three situations when we must recalculate the amplitudes:

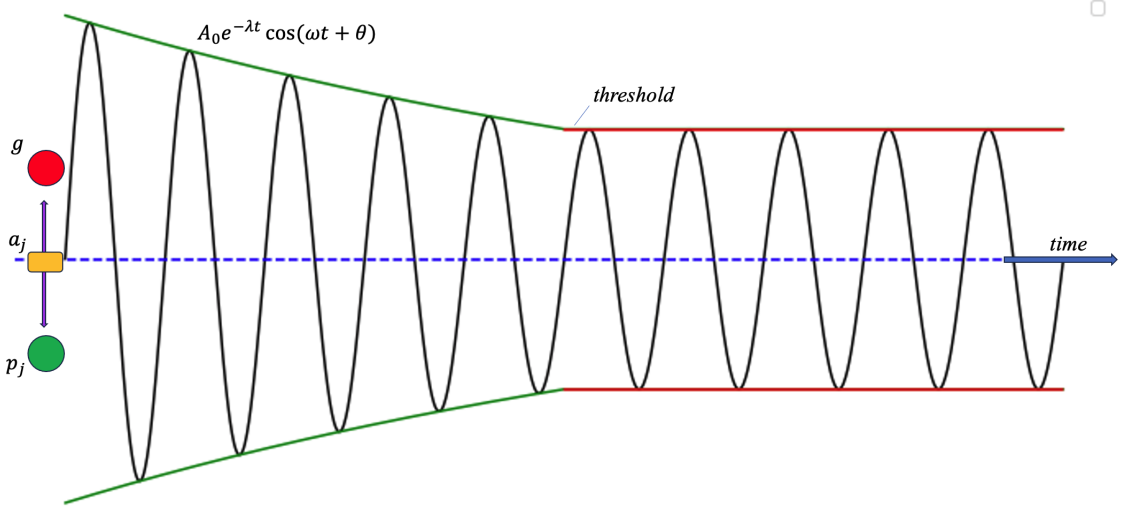


Figure 4.1: HOPSO Visualization: In one-dimension, the particle oscillates about the attractor a_j which is set half-way between its personal best (p_j) and the swarm's global best (g) based on the weighted average equation (4.10). j represents the particle number.

- $x_j \rightarrow p_j$: when the position is found to be the particle's new best position.
- $x_j \rightarrow g$: when the position is found to be the particle's new best position but it is also the global best position of the swarm.
- $x_j \rightarrow x_j$: when the position is neither the personal nor the global best position, but another particle has found a global best position hence the amplitudes are recalculated for all particles in the swarm.

In summary, amplitudes are recalculated when a new personal or global best position is found. This is because the attractor term will change since it depends on personal and global positions (4.10). As a result, the initial amplitude equation (4.11) will be different.

As previously mentioned, when the amplitude is recalculated, it is possible for the amplitude to become smaller despite a better solution found by the particle. Since this is a physically-inspired system, we reckon that the energy of a system should not be lost in the event that it finds a better position. In other words, the system should not be penalized when a better position is found. Since the energy in a system is represented in the amplitude, we deemed it appropriate to provide options for the amplitude as depicted in the following equation (4.15).

$$(A_0)_{i+1} = \max((A_0)_i, (A_0)_{i+1}, \frac{|p_{j,d,i+1} - g_{d,i+1}|}{2} * m) \tag{4.15}$$

This condition will ensure that the energy will remain at least as much as it had previously hence allowing for proper exploration.

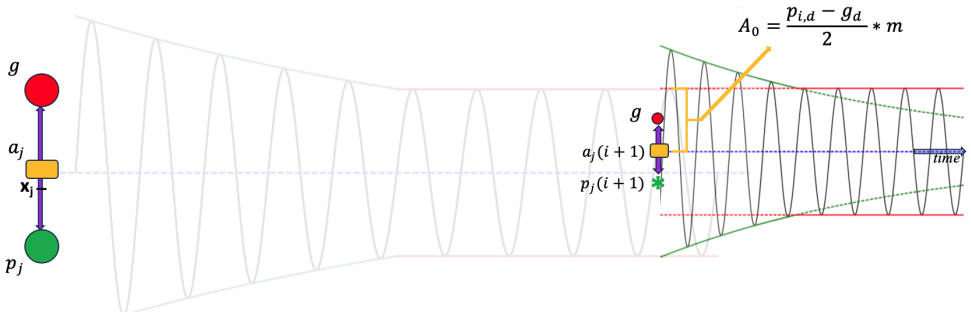


Figure 4.2: HOPSO Movement Example: When the amplitude reaches a threshold, we make sure it does not go below the equation seen in the figure, where 'm' is a multiplier term set to 2.05 which is the same as the cognitive and social coefficient in the standard PSO.

The figure 4.2 depicts a specific case where x_j is found to be a new personal best position, thereby changing the attractor. If there occurs a small velocity that was calculated in the next iteration. In this case, the second term in (4.11) can be neglected (since velocity is near zero and squaring the already-small lambda term also results in an overall term that is negligible). This implies that equation (4.11) will change to:

$$A_0 = \sqrt{(x(0) - a)^2} = x(0) - a \tag{4.16}$$

Recall (4.10) for the a_j term. Also recall that if a personal best was found then $x(0) = p_j$ Plugging this into the above equation we have:

$$A_0 = x(0) - \frac{|c_1 p_{j,d} + c_2 g_d|}{c_1 + c_2} \tag{4.17}$$

Since in our case the attractor places equal weights to the social and personal best positions, $c_1 = c_2$, we can rewrite the above equation simply as:

$$A_0 = \frac{|p_{j,d} - g_d|}{2} \tag{4.18}$$

4.2.4 Parameters of HOPSO

The amplitude A_j and phase angle θ_j are calculated 'parameters' that control the magnitude and direction of each particle's oscillation around its attractor point, and are responsible for enhancing the thoroughness of the search in the surrounding area. They are the primary drivers of finding better positions, and are calculated based on the current particle positions, velocities, and attractors.

Apart from the amplitudes and phase angles, there are actually four main parameters in the HOPSO optimization stratergy. These are ω (which is related to time, and set to equal one), the cognitive and social coefficients (which are also set to equal one), and the damping factor, λ.

The damping factor λ, however, is the most crucial of all the parameters. It is this parameter that allows for the control over the systems convergence. Specifically, if you have a specific budget (that is, how many iterations to run), you can adjust λ appropriately in the

HOPSO algorithm, which is entirely impossible for the PSO. The PSO has the χ parameter which acts as a damping factor. However, a slight re-adjustment of the χ factor to reduce damping can cause explosion in the system. There is not a clear understanding or transition in the damping of the system. But in HOPSO, this λ factor allows for a more finely-tuned system and no explosions involved.

It is crucial to experimentally verify which setting of these coefficients produce optimal results, and understand what the effects are on the system when varying these settings.

4.2.5 Pseudocode of HOPSO

The pseudo-code for the algorithm is shown in 2.

In summary, the provided code implements a damped particle swarm optimization algorithm inspired by the mass-on-a-spring system. It incorporates equations of motion, damping, and attraction forces to guide the particles towards optimal solutions. While it deviates from the physical system in some aspects, such as discrete-time updates and additional optimization features, it is effective in its application to solve optimization problems.

The code also includes various data structures and variables to store and track the optimization progress, such as the particle positions, velocities, personal best positions, global best position, and cost function values at each iteration. These data structures are used for analysis and visualization of the optimization process.

Algorithm 2 HOPSO

Set constants c_1, c_2 for attraction weights
Set constants λ
Initialize particles with random positions $x_{j,d}$ and velocities $v_{i,d}$
Set initial personal best positions p_j for each particle
Set initial global best position g
Calculate position of attractors for each particle via $a_i = \frac{c_1 p_i + c_2 g}{c_1 + c_2}$
Calculate initial amplitude: $A_0 = \sqrt{(x(0) - a)^2 + \left(\frac{v(0) + \lambda(x(0) - a)}{\omega}\right)^2}$
Calculate initial phase: $\theta = \arccos\left(\frac{x(0) - a}{A_0}\right)$
while iteration < max_iterations **do**
    for each particle **do**
        for each dimension **do**
            Amplitude is decreasing: $A = A_0 \cdot \exp(-\lambda \cdot t)$
            Select the larger amplitude: $A = \max\left(A_0 e^{-\lambda t}, \frac{|p_j - g|}{2} \cdot m\right)$
            Calculate position: $x(t) = A_0 e^{-\lambda t} \cos(\omega t + \theta)$
            Calculate velocity: $v(t) = -\omega(A_0 e^{-\lambda t} \sin(\omega t + \theta)) - \lambda x(t)$
        end for
    end for
    for each particle **do**
        Calculate Cost function from positions
        if $Cost\_function(x_{j,d}(t)) < Cost\_function(p_j)$ **then**
            $p_j = x_{j,d}$
            new best value = $Cost\_function(x_{j,d}(t))$
            Reset time = 0
            Recalculate attractors
            Recalculate amplitude
            Recalculate phase
        end if
    end for
    if personal best value < global best energy **then**
        new best value = $Cost\_function(x_{j,d}(t))$
        $g = p_j$
        for each particle **do**
            for each dimension **do**
                Reset time = 0 for each particle, all dimensions
                Recalculate attractors for each particle, all dimensions
                Recalculate amplitude for each particle, all dimensions
                Recalculate phase for each particle, all dimensions
            end for
        end for
    end if
    iteration += 1
end while

CHAPTER 5

PROGRESS AND RESULTS

5.1 Test Functions and Optimization Method Comparison

Before applying the HOPSO optimization algorithm to the VQE algorithm, we present its performance on more simple, standard benchmark functions. This test on smaller functions will demonstrate the reason for pursuing the development of this algorithm with the goal of applying it to the complicated hamiltonian of Lithium-Hydrite. The functions that will be used were chosen for their different properties [22]. For example, Ackley and Rastrigin are functions with many local minima's, the Sphere function is a bowl-shaped function while the Beale function is multimodal, with sharp peaks at the corners of the input domain.

The performance of HOPSO will be compared to that of PSO, COBYLA, and DE optimization methods. COBYLA and Differential evolution are optimizers existing within the scipy package that was used, while PSO was used with the pyswarm package. It is difficult to compare optimization methods as each one has a unique set of parameters that require fine-tuning and a particular configuration setting that will result in an optimum solution to the particular problem. However, there are commonly-known"standard settings" which result in a typically-good performance by the optimization method. For the PSO these standard settings are χ is 0.729, and with c_1, c_2 being each 2.05. The HOPSO settings were $c_1 = c_2 = \omega = 1$, while λ and t_{ul} were often adjusted for various cases. These two latter terms may be considered the main tuning parameters to achieve optimal results.

In comparing these optimizers, the same number of function evaluations was used as the budget for each optimizer. The PSO's function evaluation is a product of the number of particles used and the number of iterations which is the same for the HOPSO. We used scipy's optimize module to implement COBYLA and DE [27]. For COBYLA, its maxiter parameter equals the number of

Table 5.1: Commonly used test functions for optimization methods

Name	functional form
Sphere	$\sum_{i=1}^{d} x_i^2$
Beale	$(1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2) + (2.625 - x_1 + x_1x_2^3)$
Goldstein-price	$[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)][30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
Ackley	$-a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$
Rastrigin	$10d + \sum_{i=1}^{d}[x_i^2 - 10\cos 2\pi x_i]$
Schwefel	$\sum_{i=1}^{d}\left[-x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)\right]$
Griewank	$\frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Rosenbrock	$\sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Levy	$\sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_d - 1)^2[1 + \sin^2(2\pi w_d)]$
Drop-Wave	$-\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$
Cross-in-Tray	$-0.0001\left[\left\lvert\sin(x_1)\sin(x_2)\exp\left(\left\lvert 100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert + 1\right]^{0.1}$
Michalewicz	$-\sum_{i=1}^{d}\sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m}$

function evaluations. Meanwhile, differential evolution's function evaluation is calculated as the product of population size, maximum iterations and dimensions of the cost function. The default population size for DE is taken as the dimension of the cost function. The popsize parameter in scipy's implementation for DE is actually a multiplier for the population size and not the population size itself [21]. It is important to note that since COBYLA and DE are scipy optimizers, they run up to (but may be less) the max-iterations which is set by their own convergence criteria. However, the tolerance for the convergence was changed so that most of the budget is utilised. Lastly, 100 runs of the optimization process are represented in each box-plot graph comparing optimization methods below. Refer to figures 5.1-5.12 on page 44-45 for visualization of the commonly-used test function for optimization methods [23].

5.1.1 A Refresher on Other Non-Gradient Optimization Methods

Constrained Optimization by Linear Approximation (COBYLA)

COBYLA (Constrained Optimization BY Linear Approximations) is an optimization method that is specifically designed to handle constraints. It is a derivative-free algorithm, making it suitable for problems where the objective function's derivatives are not readily available or are difficult to compute. COBYLA operates on the principles of a trust-region approach to manage both the variables and constraints of the optimization problem. The process starts with an initial guess at the solution, around which a simplex is formed—a polytope with $n+1$ vertices in n-dimensional space. COBYLA uses linear approximations of the objective function and the constraints within

this simplex to guide its search for the optimum. During each iteration, it attempts to improve upon the current solution by exploring within a trust region, which adjusts adaptively based on the success of previous steps. The algorithm incrementally refines this region and the simplex's configuration to navigate towards the optimal solution while satisfying the constraints. This approach is particularly useful for practical engineering problems where derivatives are either unavailable or unreliable, providing a robust method for finding near-optimal solutions under complex constraints. [19].

Differential Evolution (DE)

Differential Evolution (DE) is a gradient-free optimization strategy that evolves candidate solutions through mutation and recombination. This optimization algorithm belongs to the family of evolutionary algorithms. The mechanism of DE begins with the initialization of a randomly generated population of candidate solutions. Each candidate, known as an individual or vector, is represented by a set of parameters. Through the mutation step, DE generates new candidate solutions by adding the weighted difference between two population vectors to a third vector. This helps in exploring the solution space broadly. The crossover (or recombination) step enhances the diversity of the population by combining the mutated vector with another predetermined vector from the population, typically the best vector. Finally, the selection process determines whether the newly generated vector or the existing vector in the population provides a better solution based on the objective function value, with the superior one surviving into the next generation.This iterative process continues until a stopping criterion is met, such as reaching a maximum number of generations or achieving sufficient convergence of the population towards the optimum. DE is particularly valued as it does not require functions to be continuous or differentiable and is therefore well-suited for noisy functions and offers enhanced explorability. The DE algorithm is rather effective in finding global optima in a wide range of challenging optimization problems [21].

Visualization of Commonly-used Test Functions

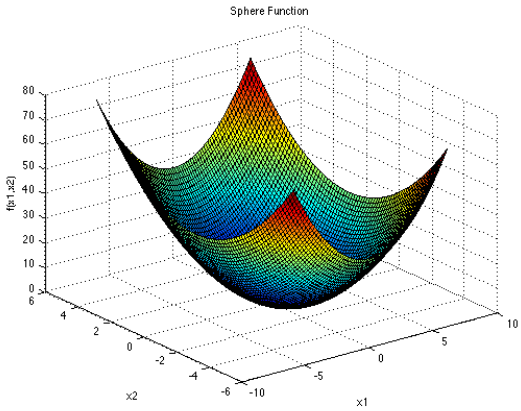


Figure 5.1: Sphere Function

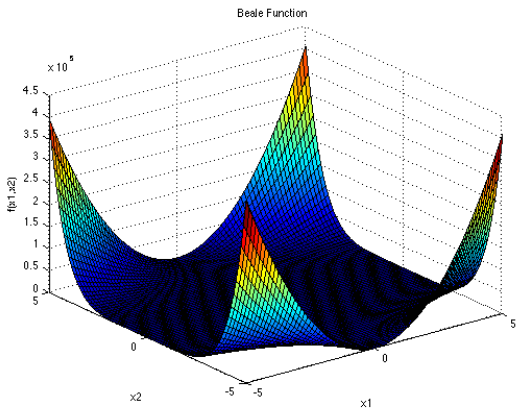


Figure 5.2: Beale Function

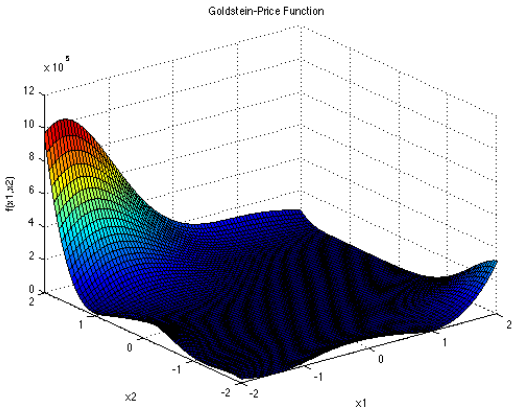


Figure 5.3: Goldstein-Price Function

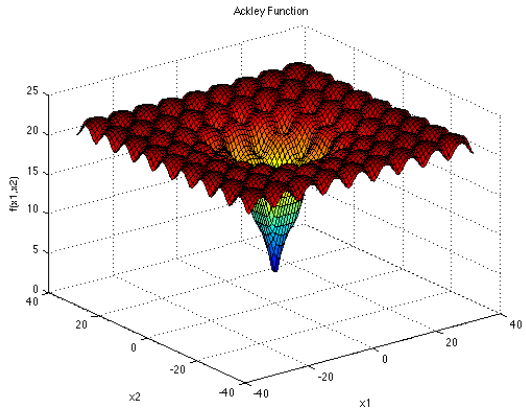


Figure 5.4: Ackley Function

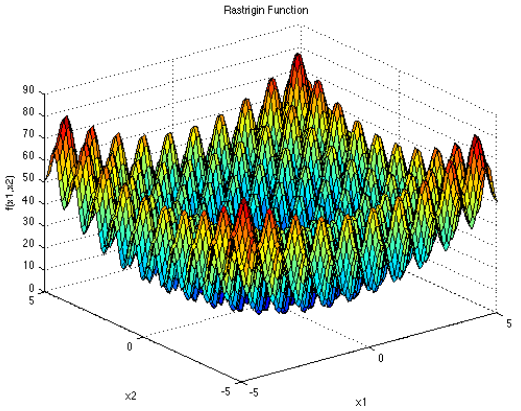


Figure 5.5: Rastrigin Function

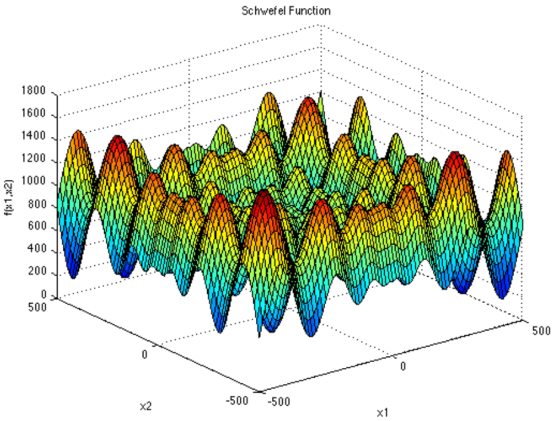


Figure 5.6: Schwefel Function

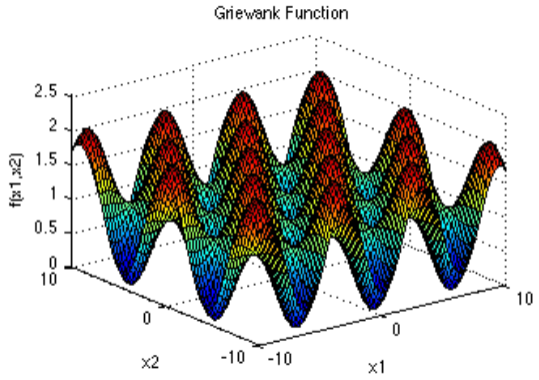


Figure 5.7: Griewank Function

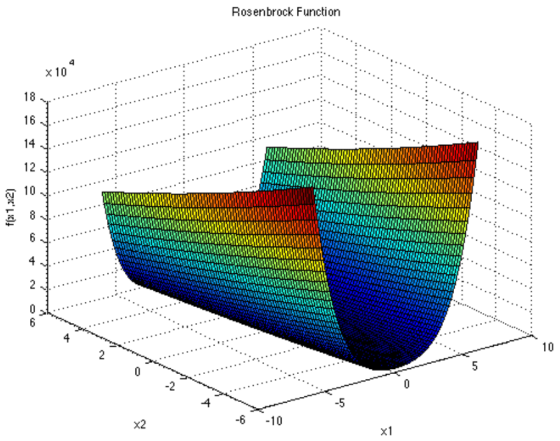


Figure 5.8: Rosenbrock Function

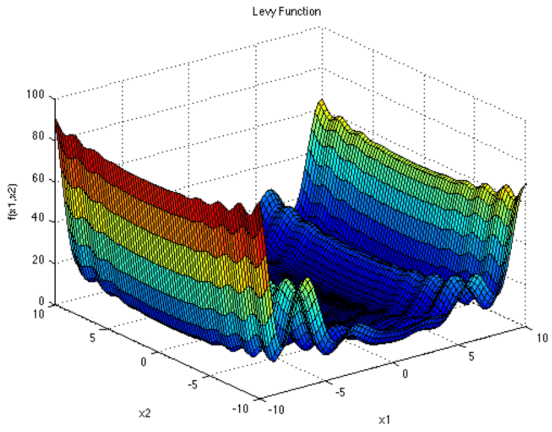


Figure 5.9: Levy Function

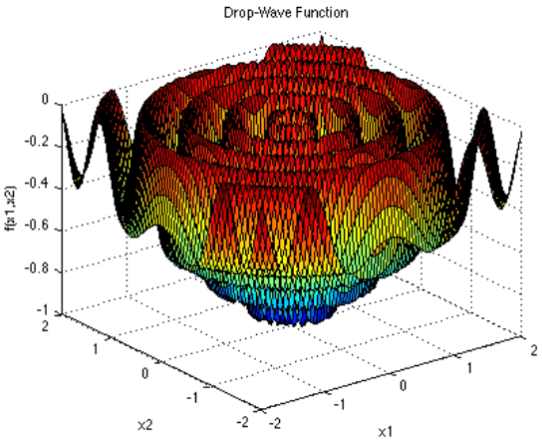


Figure 5.10: Drop-Wave Function

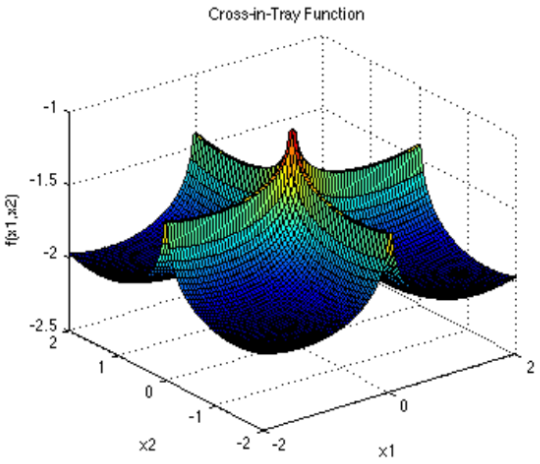


Figure 5.11: Cross-Tray Function

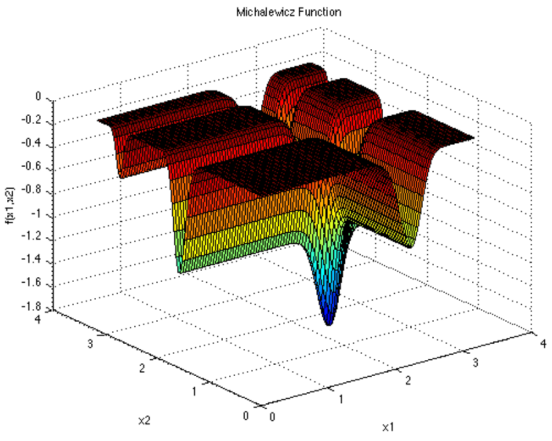
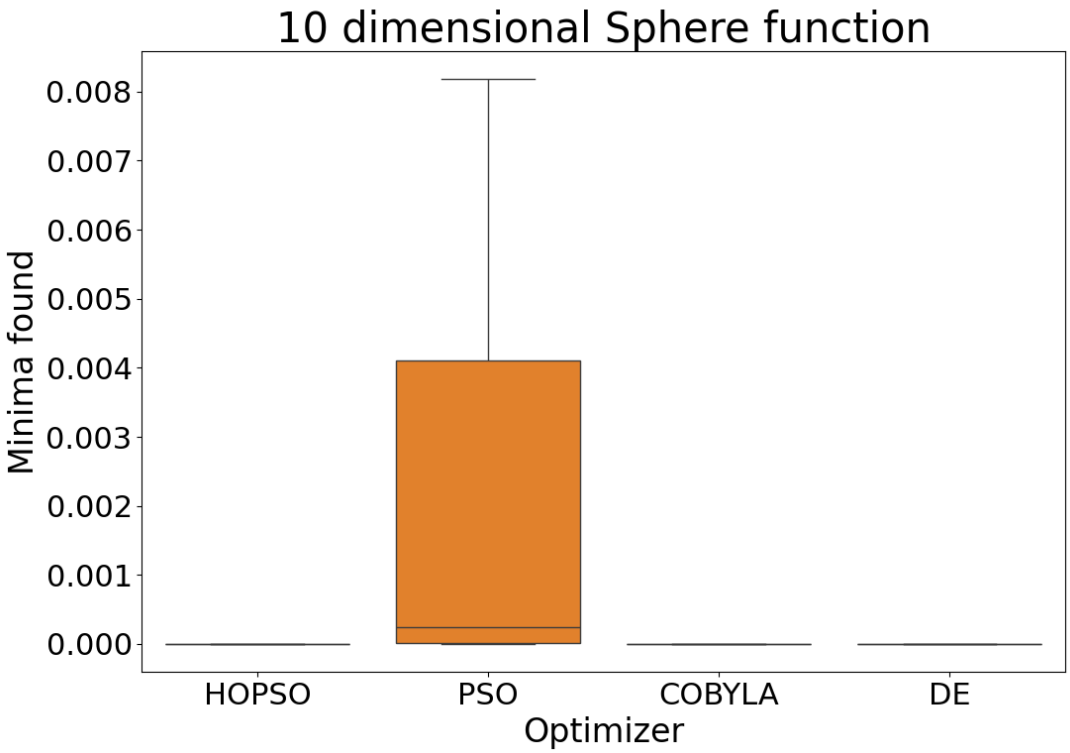


Figure 5.12: Michalewicz Function

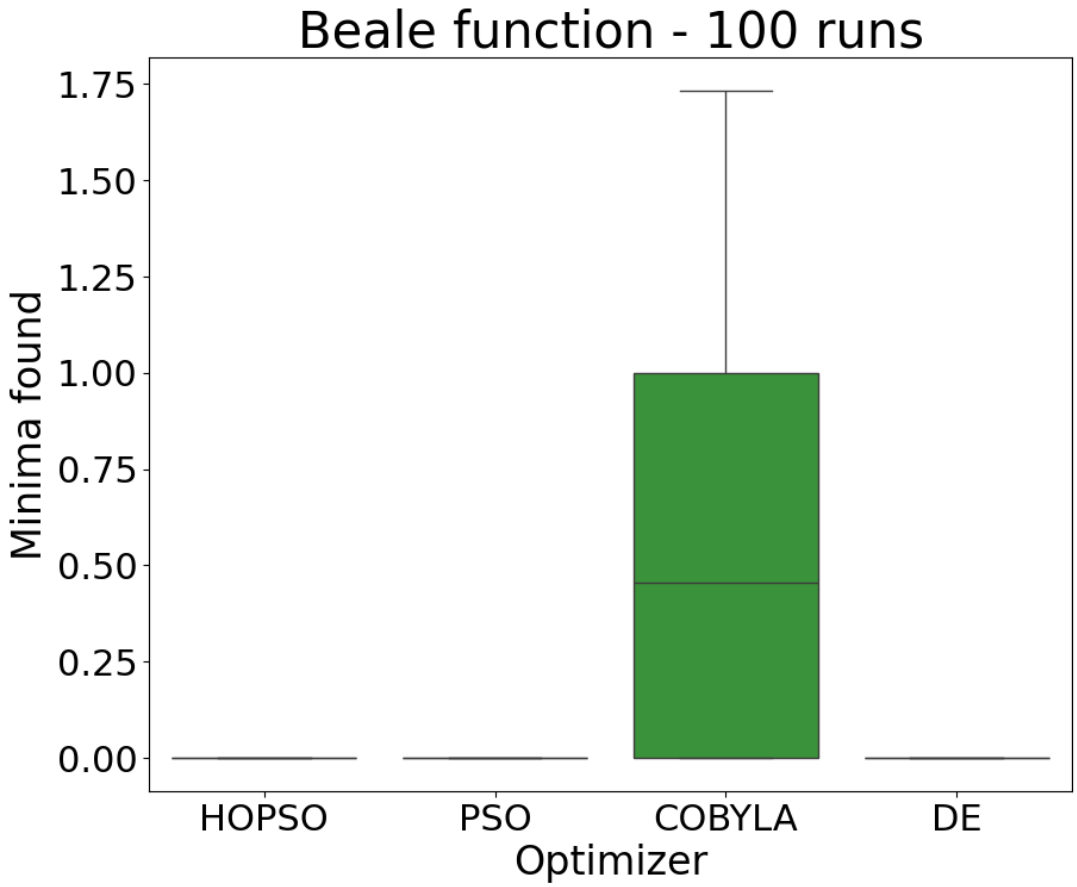
5.1.2 Graphical Results

Function 1: Sphere



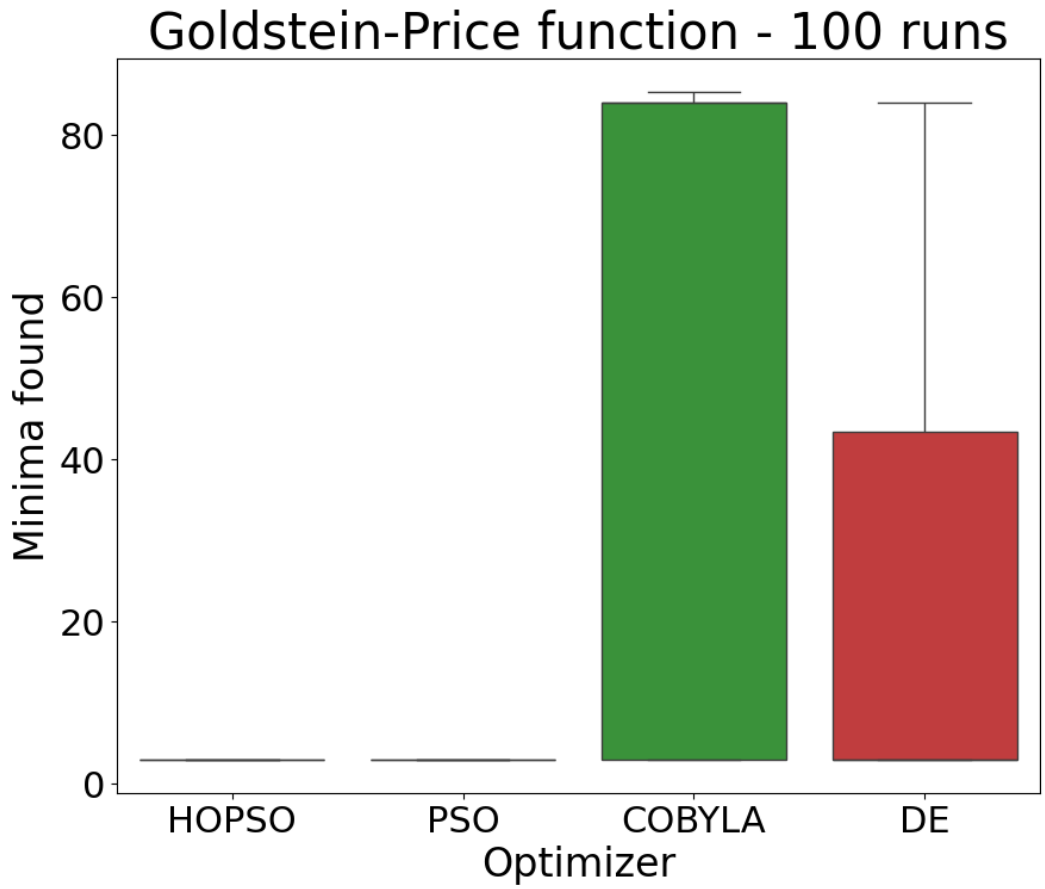
This graph above represents the comparison of boxplots of different optimizers with a 10 dimensional Sphere function as the cost function. Each one of the boxplots represents 100 different runs of the optimizer. Each run has a budget for function evaluations set to 5000. To achieve this we used 5 particles for PSO and HOPSO with 1000 iterations. The parameters λ and t_{ul} for HOPSO was set to 0.01 and 1 respectively. For COBYLA the parameter maxiter was set to 5000. For DE the parameter popsize was set to 1 and maxiter to 100. All of these optimizers start with a uniform random parameters within the range [-5,5]. Other than PSO, rest of the optimizers reach the minimum value which is 0.

Function 2: Beale



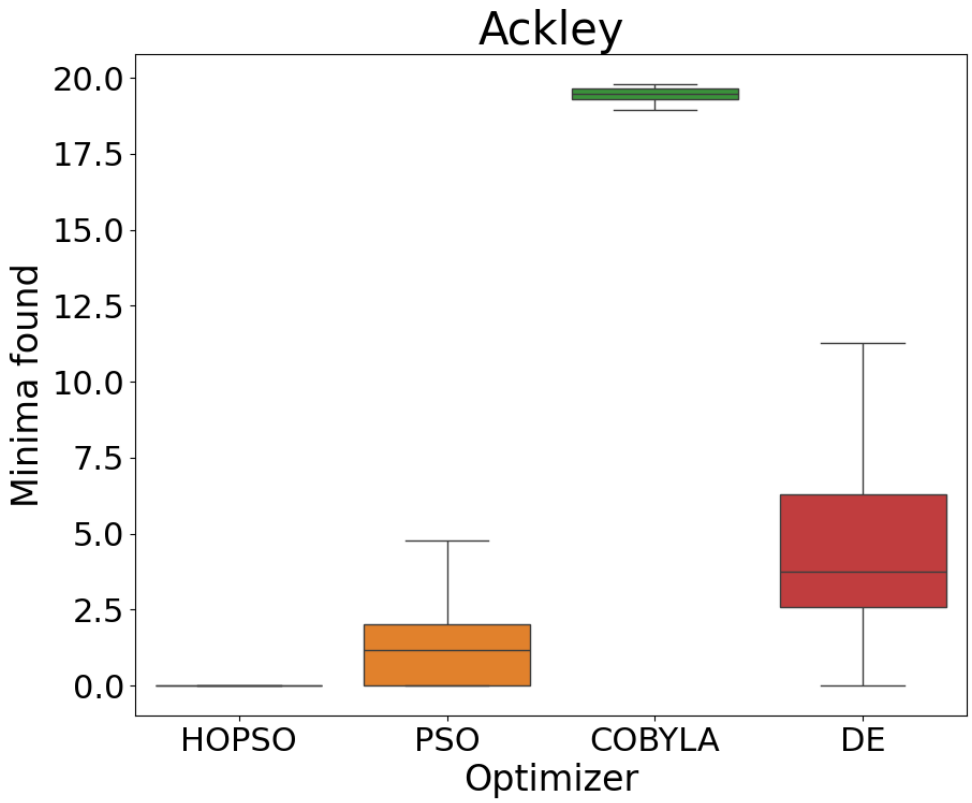
This graph represents the comparison of boxplots of different optimizers with the Beale function as the cost function. Each one of the boxplots represents 100 different runs of the optimizer. Each run has a budget for function evaluations set to 1500. To achieve this we used 10 particles for PSO and HOPSO with 150 iterations. The parameters λ and t_{ul} for HOPSO was set to 0.1 and 1, respectively. For COBYLA the parameter maxiter was set to 1500. For DE, the parameter popsize was set to 10 and maxiter to 75. All of these optimizers start with a uniform random parameters within the range [-5,5]. Other than COBYLA, the rest of the optimizers reach the minimum value which is 0.

Function 3: Goldstein-Price



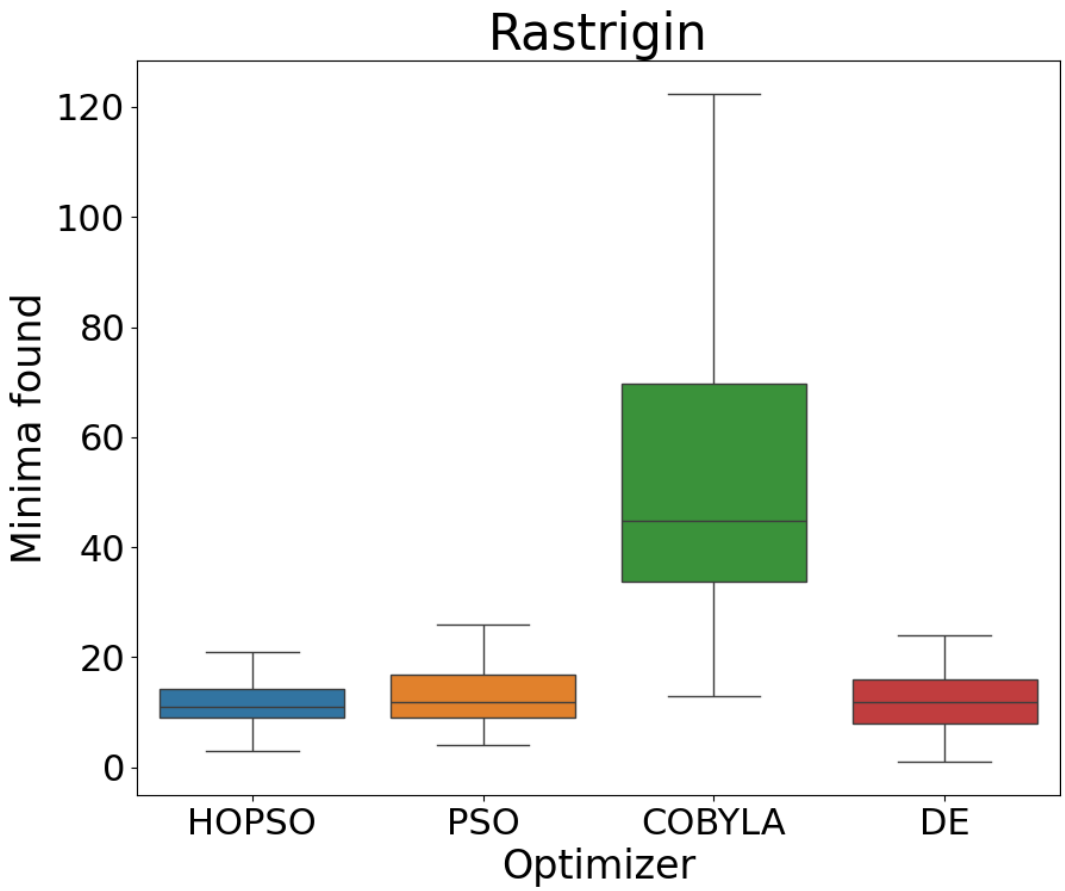
This graph represents the comparison of boxplots of different optimizers with the Goldstein Price function as the cost function. Each one of the boxplots represents 100 different runs of the optimizer. Each run has a budget for function evaluations set to 1500. To achieve this we used 5 particles for PSO and HOPSO with 200 iterations. The parameters λ and t_{ul} for HOPSO was set to 0.01 and 1 respectively. For COBYLA the parameter maxiter was set to 1000. For DE the parameter popsize was set to 3 and maxiter to 250. All of these optimizers start with a uniform random parameters within the range [-5,5]. Other than COBYLA and DE, rest of the optimizers reach the minimum value which is 3.

Function 4: Ackley



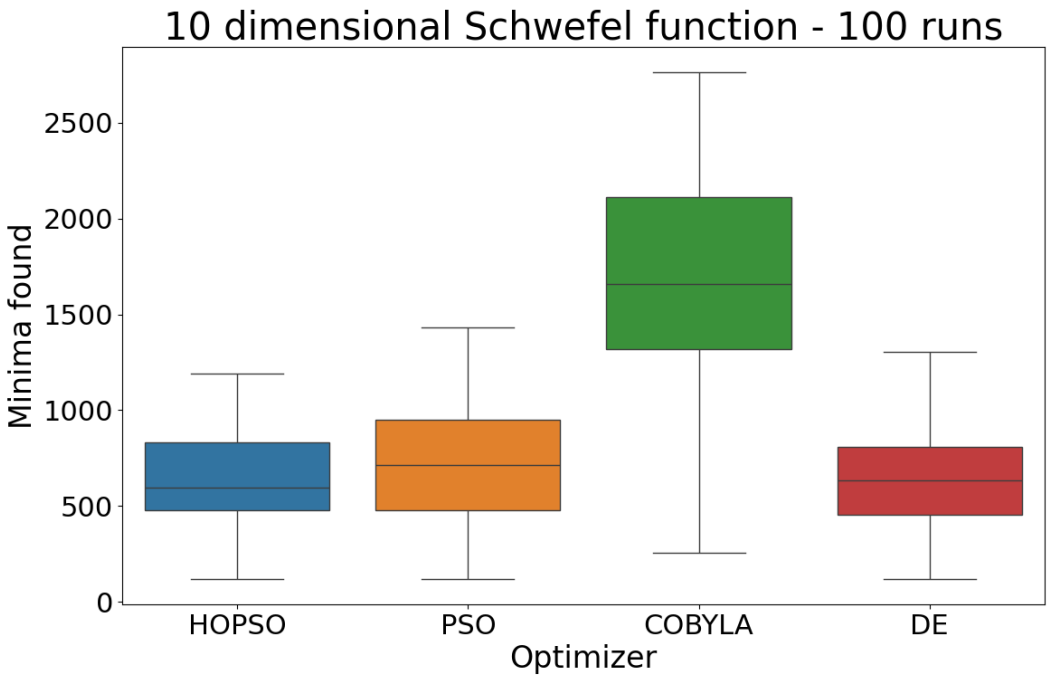
This graph represents the comparison of boxplots of different optimizers with a 10 dimensional Ackley function as the cost function. Each one of the boxplots represents 100 different runs of optimizer. Each run has a budget for function evaluations set to 5000. To achieve this we used 10 particles for PSO and HOPSO with 500 iterations. The parameters λ and t_{ul} for HOPSO was set to 0.02 and 1, respectively. For COBYLA the parameter maxiter was set to 5000. For DE, the parameter popsize was set to 1 and maxiter to 500. All of these optimizers start with a uniform random parameters within the range [-32.768,32.768]. Other than COBYLA and PSO, rest of the optimizers reach the minimum value which is 0.

Function 5: Rastrigin



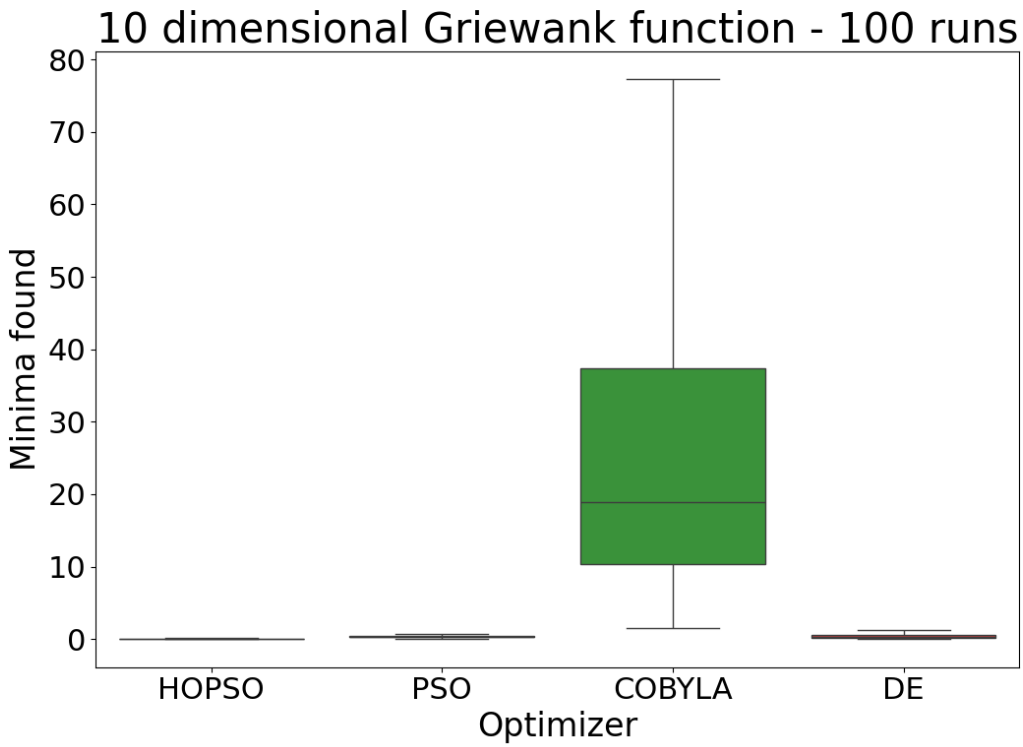
This graph represents the comparison of boxplots of different optimizers with 10 dimensional Rastrigin function as the cost function. Each one of the boxplots represents 100 different runs of optimizer. Each run has a budget for function evaluations set to 5000. To achieve this we used 10 particles for PSO and HOPSO with 500 iterations. The parameters λ and t_{ul} for HOPSO was set to 0.027 and 1 respectively. For COBYLA the parameter maxiter was set to 5000. For DE the parameter popsize was set to 1 and maxiter to 500. All of these optimizers start with a uniform random parameters within the range [-5.12,5.12]. Other than COBYLA and PSO, rest of the optimizers reach the minimum value which is 0.

Function 6: Schwefel



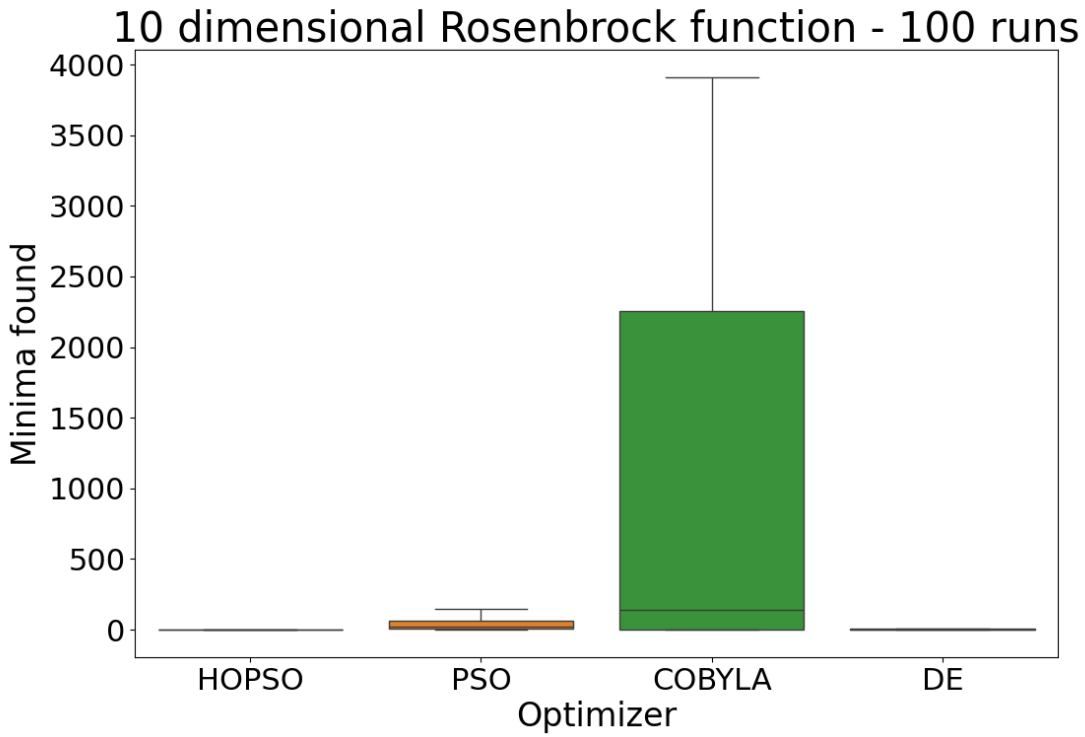
This graph represents the comparison of boxplots of different optimizers with the 10-dimensional Schwefel function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For the damped PSO with amplitude (HOPSO), we used hyperparameters [1, 1, 2π, 0.05], 100 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-500, 500]$ and $[-10, 10]$, respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 100 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 100000 with a tolerance of 10^{-8}, starting with initial positions uniformly random within the range $[-500, 500]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-500, 500]$, with a population size of 1, maximum iterations of 1000, bounds of $[-500, 500]$ for each dimension, and a tolerance of 10^{-8}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Schwefel function more consistently compared to COBYLA and PSO.

Function 7: Griewank



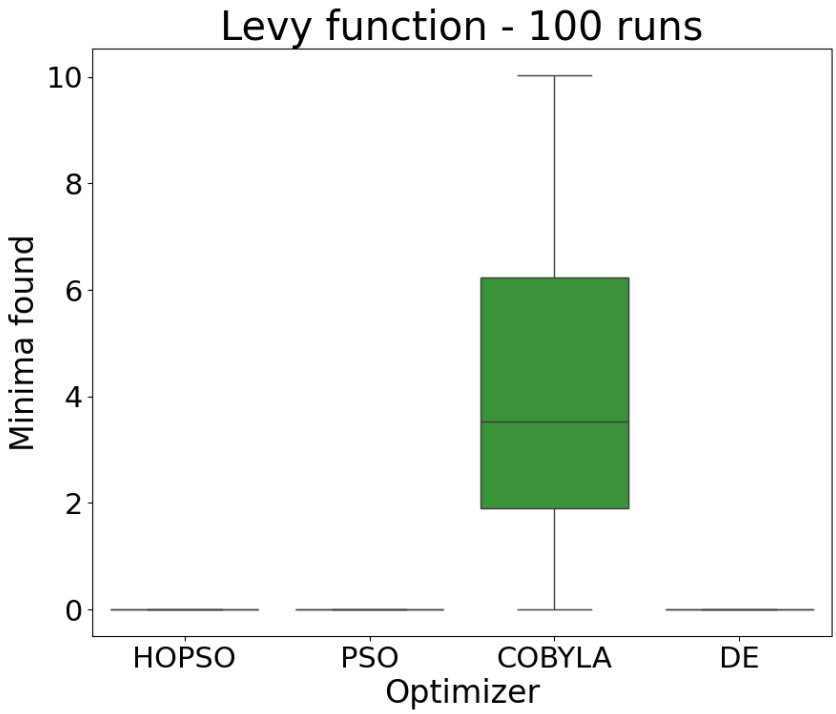
This graph represents the comparison of boxplots of different optimizers with the 10-dimensional Griewank function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For HOPSO, we used hyperparameters [1, 1, 2π, 0.02], 100 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-600, 600]$ and $[-10, 10]$ respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 100 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 100000 with a tolerance of 10^{-8}, starting with initial positions uniformly random within the range $[-500, 500]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-500, 500]$, with a population size of 1, maximum iterations of 1000, bounds of $[-500, 500]$ for each dimension, and a tolerance of 10^{-8}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Griewank function more consistently compared to COBYLA and PSO.

Function 8: Rosenbrock



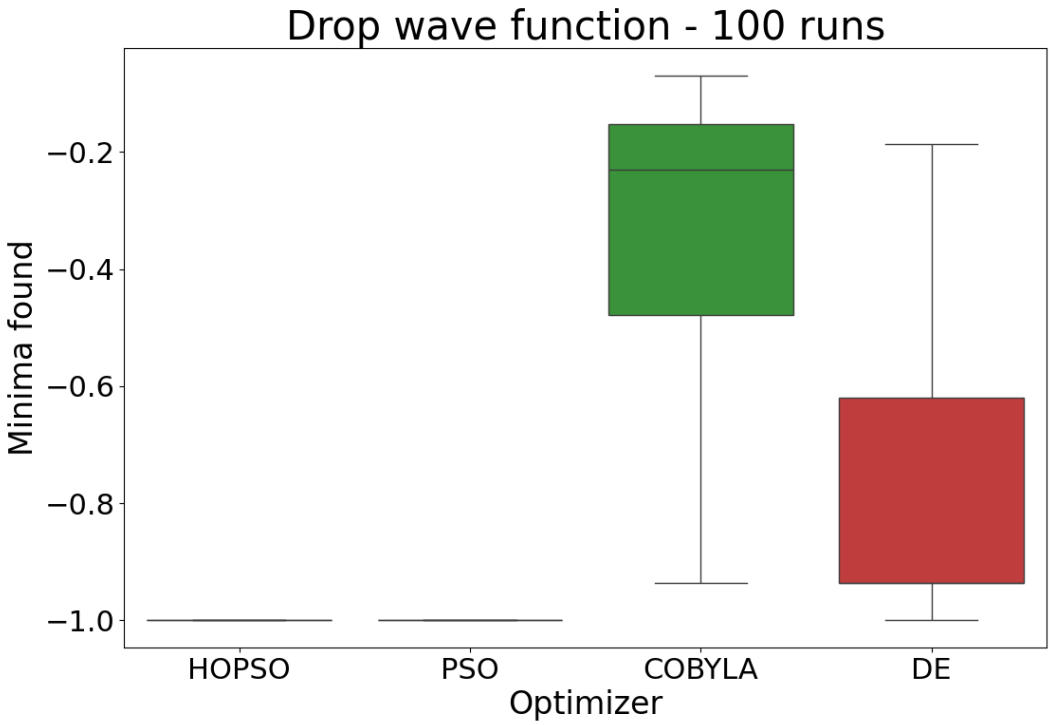
This graph represents the comparison of boxplots of different optimizers with the 10-dimensional Rosenbrock function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For HOPSO, we used hyperparameters [1, 1, 2π, 0.02], 100 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-30,30]$ and $[-10,10]$, respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 100 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 100000 with a tolerance of 10^{-8}, starting with initial positions uniformly random within the range $[-500,500]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-500,500]$, with a population size of 1, maximum iterations of 1000, bounds of $[-500,500]$ for each dimension, and a tolerance of 10^{-8}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Rosenbrock function more consistently compared to COBYLA and PSO.

Function 9: Levy



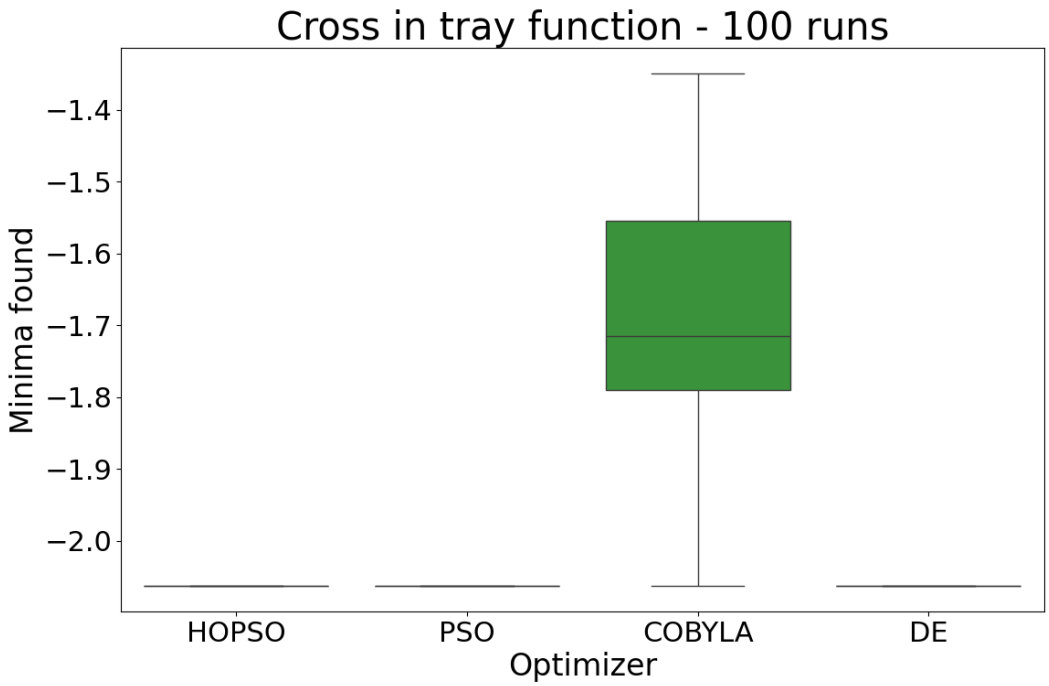
This graph represents the comparison of boxplots of different optimizers with the Levy function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For the HOPSO algorithm, we used hyperparameters [1, 1, 2π, 0.01], 10 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-10, 10]$ and $[-2, 2]$, respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 10 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 10000 with a tolerance of 10^{-16}, starting with initial positions uniformly random within the range $[-10, 10]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-10, 10]$, with a population size of 1, maximum iterations of 1000, bounds of $[-10, 10]$ for each dimension, and a tolerance of 10^{-16}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Levy function more consistently compared to COBYLA and PSO.

Function 10: Drop-Wave



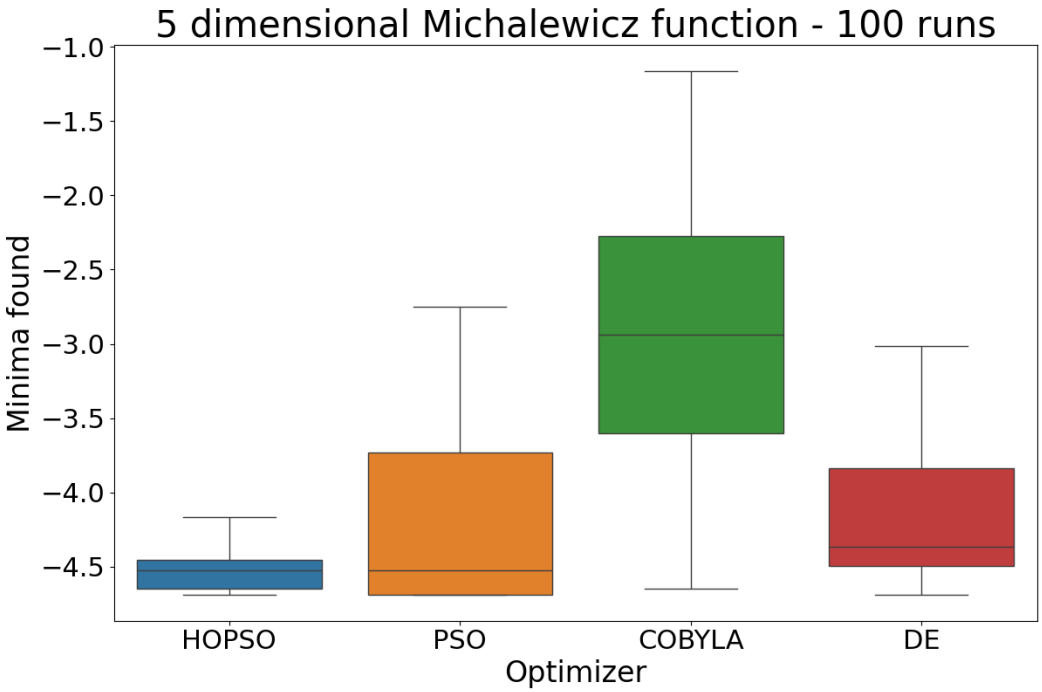
This graph represents the comparison of boxplots of different optimizers with the Drop Wave function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For HOPSO, we used hyperparameters [1, 1, 2π, 0.0001], 10 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-5.12, 5.12]$ and $[-2, 2]$ respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 10 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 10000 with a tolerance of 10^{-16}, starting with initial positions uniformly random within the range $[-5.12, 5.12]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-5.12, 5.12]$, with a population size of 1, maximum iterations of 1000, bounds of $[-5.12, 5.12]$ for each dimension, and a tolerance of 10^{-16}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Drop Wave function more consistently compared to COBYLA and PSO.

Function 11: Cross-in-Tray



This graph represents the comparison of boxplots of different optimizers with the Cross-in-Tray function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For the HOPSO algorithm, we used hyperparameters [1, 1, 2π, 0.02], 10 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[-10, 10]$ and $[-5, 5]$, respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 10 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 10000 with a tolerance of 10^{-16}, starting with initial positions uniformly random within the range $[-10, 10]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[-10, 10]$, with a population size of 1, maximum iterations of 1000, bounds of $[-10, 10]$ for each dimension, and a tolerance of 10^{-16}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Cross-in-Tray function more consistently compared to COBYLA and PSO.

Function 12: Michalewicz



This graph represents the comparison of boxplots of different optimizers with the 5-dimensional Michalewicz function as the cost function. Each of the boxplots represents 100 different runs of the optimizer, with each run having a budget for function evaluations set to 1000. For HOPSO, we used hyperparameters [1, 1, 2π, 0.01], 10 particles, and a max cut of 2.05. The threshold was set to 10^{-4}, with initial positions and velocities uniformly random within the range $[0, \pi]$ and $[-1, 1]$, respectively. For the Standard PSO, the hyperparameters were [0.7298, 2.05, 2.05], with 10 particles and max iterations of 1000, starting with positions and velocities uniformly random within the same range. For COBYLA, the maximum iterations were set to 10000 with a tolerance of 10^{-16}, starting with initial positions uniformly random within the range $[0, \pi]$. For Differential Evolution (DE), the initial positions were uniformly random within the range $[0, \pi]$, with a population size of 1, maximum iterations of 1000, bounds of $[0, \pi]$ for each dimension, and a tolerance of 10^{-16}. The comparison showcases the minima found by each optimizer across the 100 runs, with HOPSO and DE optimizers reaching values close to the global minimum of the Michalewicz function more consistently compared to COBYLA and PSO.

Periodic Function in 4D

Since the Lithium Hydrite hamiltonian is periodic, it is reasonable to test the HOPSO algorithm on simpler periodic functions. The HOPSO algorithm was applied to a few periodic functions, but here we will be presenting the results of one specific 4-dimensional function which has a global minima at -9.25999:

$$f = -8.6 + 0.34 \cdot \sin(1.5x_1 - \frac{\pi}{2}) + 0.17 \cdot \cos(1.5x_2) - 0.1 \cdot \sin(1.5x_3 + \frac{\pi}{4}) + 0.05 \cdot \cos(1.5x_4 - \frac{\pi}{3}) \tag{5.1}$$

Below is the result of the performance of HOPSO on this function with a comparison to the standard pso and a standard SciPy optimizer. This is done with a setting of 10 particles, and 100 000 iterations for 20 runs.

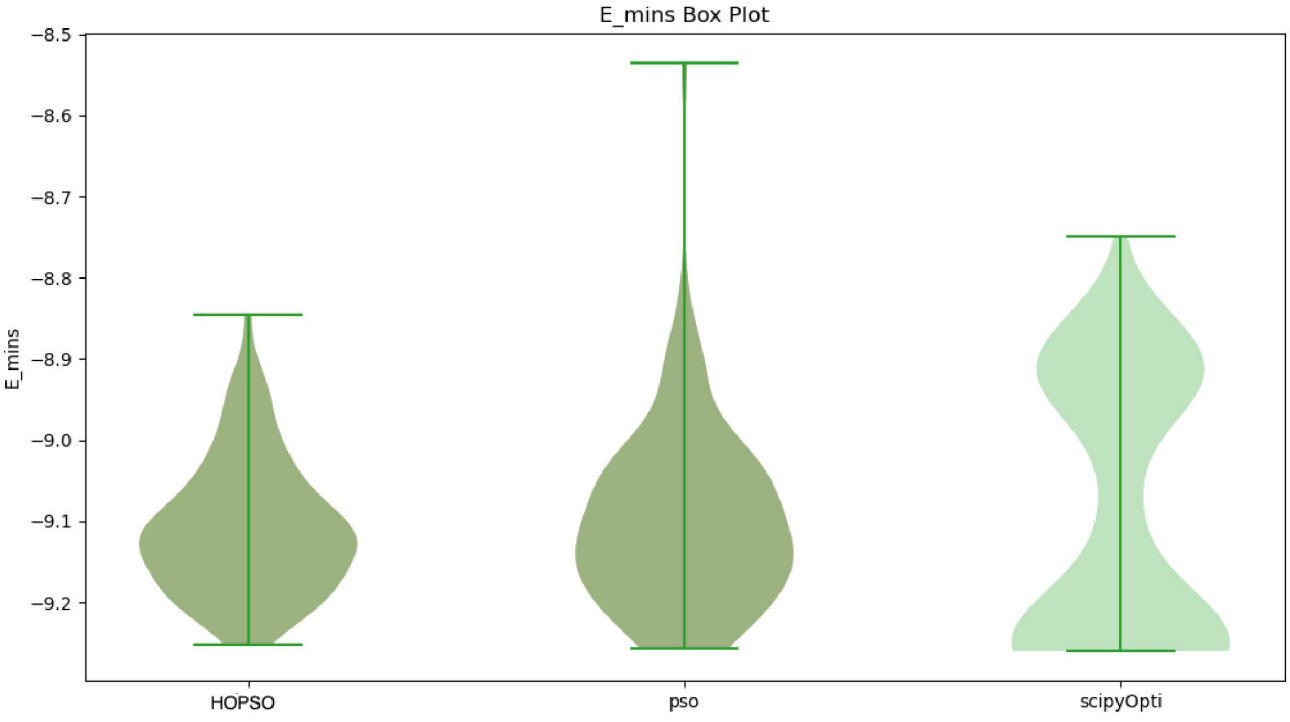


Figure 5.13: Optimizers comparison

5.2 Application to Quantum Circuits

Modulo

A crucial feature to mention is the modulo operation. Specifically, since the cost functions that are dealt with on quantum circuits are naturally periodic, we figure that it is necessary to apply a modulo operation to ensure that all evaluations of the cost function remain within a specific bound and avoid potential problems. Particularly, if we imagine a grid made up of squares from 0 to 2π, we do not restrict any particle and allow each particle to fly around beyond any single square thereby taking full advantage of its searching capabilities. However, if the particle finds a personal (or global) best position then a modulo operation is applied in order to ensure a proper position of the attractor term (since the attractor (and amplitude) are reliant on the position between personal- and global best positions). This description can be pictured as follows:
Imagine a line that goes from 0 to 2π followed by another continuing line also from 0 to 2π. If we suppose a global best position was found at π and suddenly a personal best position was found on the *second* line at $\pi/2$ (i.e. a quarter past the 2π marking), then the difference between the personal and global best would be $3\pi/2$ without a mod application. However, with a mod applied then the distance between the personal and global best would really be $\pi - \pi/2$ which is equal to $\pi/2$. It is important to note that had we applied a modulo on each position for each particle irrespective of whether or not it has found a personal or global best position, we would be limiting the particles search.

This is an important result because it solves the issue on how to deal with periodicity, specifically solving the issue of misusing distances by using too large of a distance. This avoids explosions. We have observed this fact that the modulo operation is vital for periodic functions for these reasons stated above.

Hydrogen Hamiltonian

To test the performance of our optimizer in VQE, we first chose the quantum system as the Hydrogen molecule. We obtained two different Hamiltonians: one corresponding to a two qubit system and the other corresponding to a four qubit system. Using hardware efficient ansatzs as shown in 5.14, we compared the performance of the different optimizers in VQE.

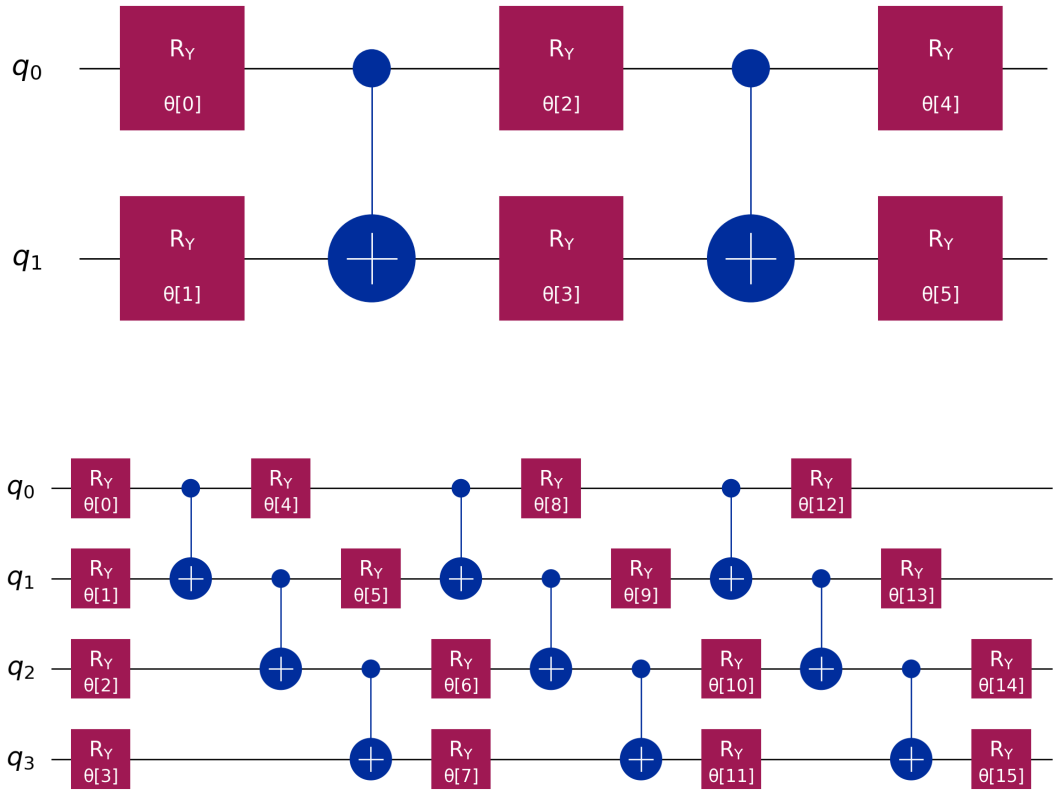


Figure 5.14: Ansatz for VQE for 2-qubit Hydrogen and 4-qubit Hydrogen, respectively.

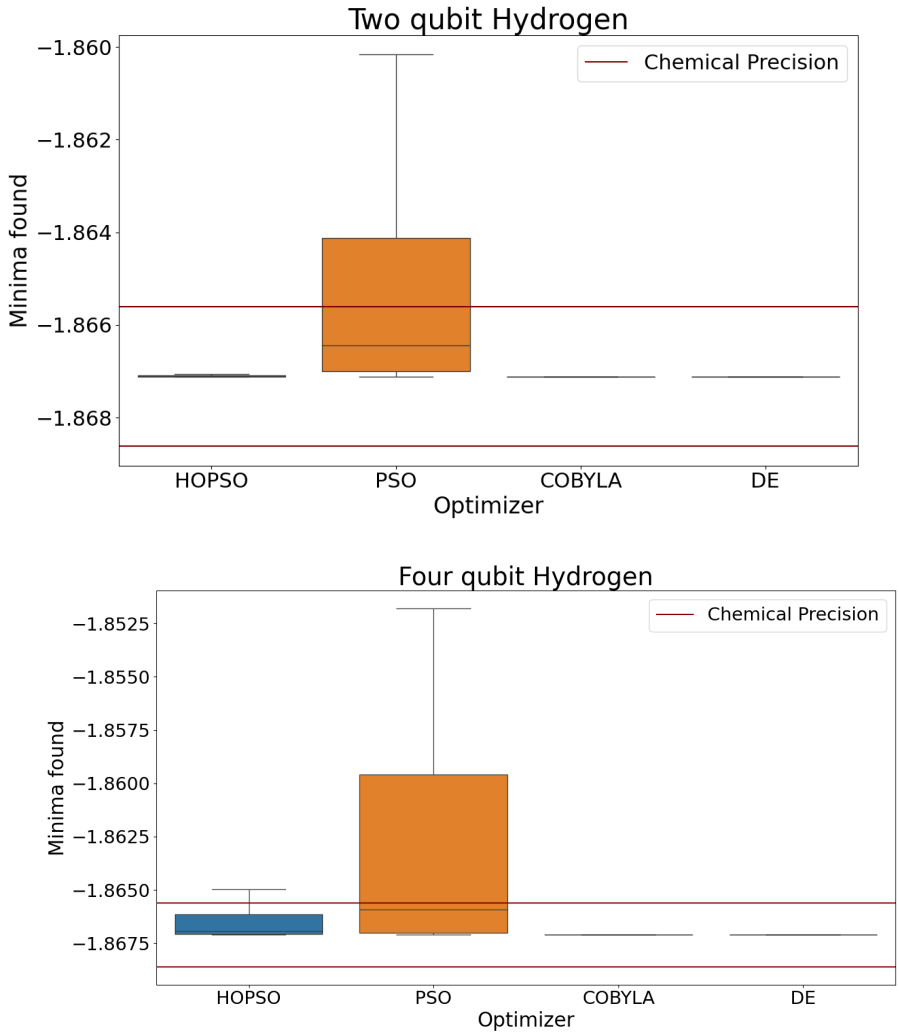


Figure 5.15: **(a)** This graph represents the comparison of boxplots of different optimizers with the expectation value of the Hamiltonian corresponding to two qubit Hydrogen molecule as the cost function. Each one of the boxplots represents 100 different runs of optimizer. Each run has a budget for function evaluations set to 500. To achieve this we used 10 particles with 50 iterations for PSO and HOPSO. The parameters λ and t_u for HOPSO was set to 0.1 and 4, respectively. For COBYLA the parameter maxiter was set to 500. For DE the parameter popsize was set to 1 and maxiter to 80. All of these optimizers start with a uniform random parameters within the range [-π,π]. All the optimizers reach the chemical precision($\pm$ 0.0015) of the real value -1.86712. However, PSO performs poorly when compared to the rest of the optimizers. **(b)** This graph represents the comparison of boxplots of different optimizers with the expectation value of the Hamiltonian corresponding to four qubit Hydrogen molecule as the cost function. Each one of the boxplots represents 100 different runs of optimizer. Each run has a budget for function evaluations set to 10000. To achieve this we used 10 particles and 1000 iterations for PSO and HOPSO. The parameters λ and t_u for HOPSO was set to 0.1 and 4, respectively. For COBYLA, the parameter maxiter was set to 10000. For DE, the parameter popsize was set to 1 and maxiter to 625. All of these optimizers start with uniform random parameters within the range [-π,π]. All the optimizers reach the chemical precision($\pm$ 0.0015) of the real value of -1.86712. However, PSO performs poorly when compared to the rest of the optimizers.

From the above graphs it can be noticed that the HOPSO algorithm is being outperformed by the COBYLA optimizer, which is unlike what occurred when comparing performance on the test-functions. We assume this is the case because the Hydrogen hamiltonian is too simple of a system. That is, we currently believe that the budget is too low for this kind of optimization method and hence the poor performance of HOPSO in comparison to COBYLA. For example, if the system is too simple, one may only need a single particle in the PSO or HOPSO algorithm, otherwise over-parameterization is taking place as the other particles (which are not needed) will be incorrectly influencing the search in the system.

Lithium Hydrite Hamiltonian

We used the qiskit-nature package and PySCF to obtain the Hamiltonian for the LiH molecule. The distance between the atoms was considered to be 1.5474 angstroms and we used the commonly used sto3g as the basis set. We then applied the Jordan Wigner mapping to map the electronic Hamiltonian to a qubit Hamiltonian. Lastly, we applied some reduction methods to taper the qubits and obtained an 8-qubit Hamiltonian for LiH. The full hamiltonian can be found in the appendix. For the ansatz, we chose to use a hardware efficient ansatz made up of 4 layers and with 40 parameters. The circuit is also linearly entangled. Below is a figure of this ansatz:

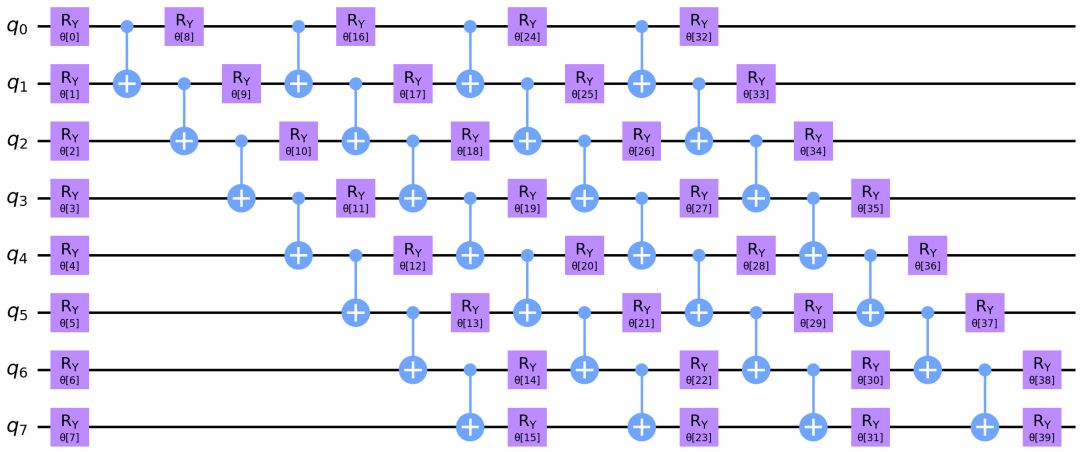


Figure 5.16: 8 qubit, 4-layer Quantum Circuit

0-Layer Circuit

Firstly, the HOPSO algorithm was applied to Lithium Hydrite Hamiltonian but without any CNOT gates in the circuit (that is, it was a 0-layer circuit with no entanglement involved). This was done to observe the performance of HOPSO when applied to the simplest case of application to LiH.

Below are the graphical results. These graphics are rather technical and differ from the previous graphs as this is currently in a 'work-in-progress' stage. It is suitable to place these here however, since these are results that are currently being developed.

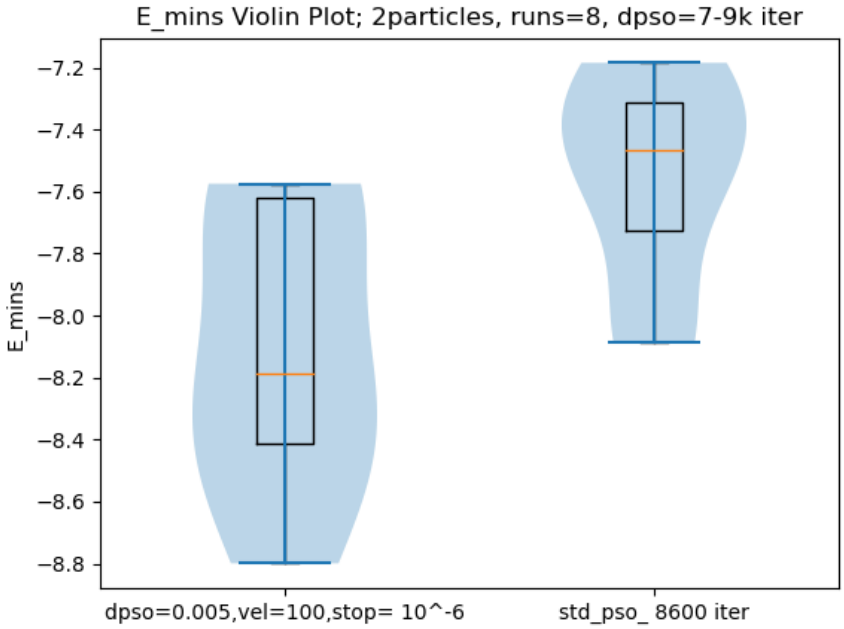


Figure 5.17: Box plots inside violin plots representing HOPSO's performance against the standard PSO. The minimum is at -8.92. The graph represents 8 runs for each box-plot in which each run is done with 2 particles and without a strictly set number of iterations. Instead, there was a stopping criteria in which reaching 10e-6 would cease the run. HOPSO's settings include a lambda of 0.005 and velocity was adjusted to be 100.

.

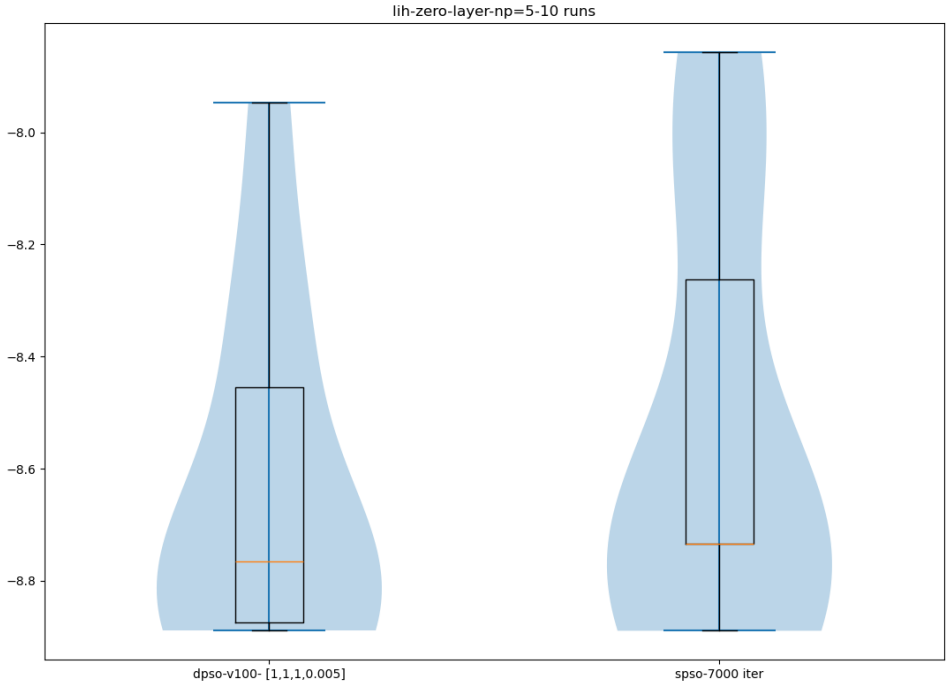


Figure 5.18: Another comparison of HOPSO to standard PSO with 7000 iterations.

.

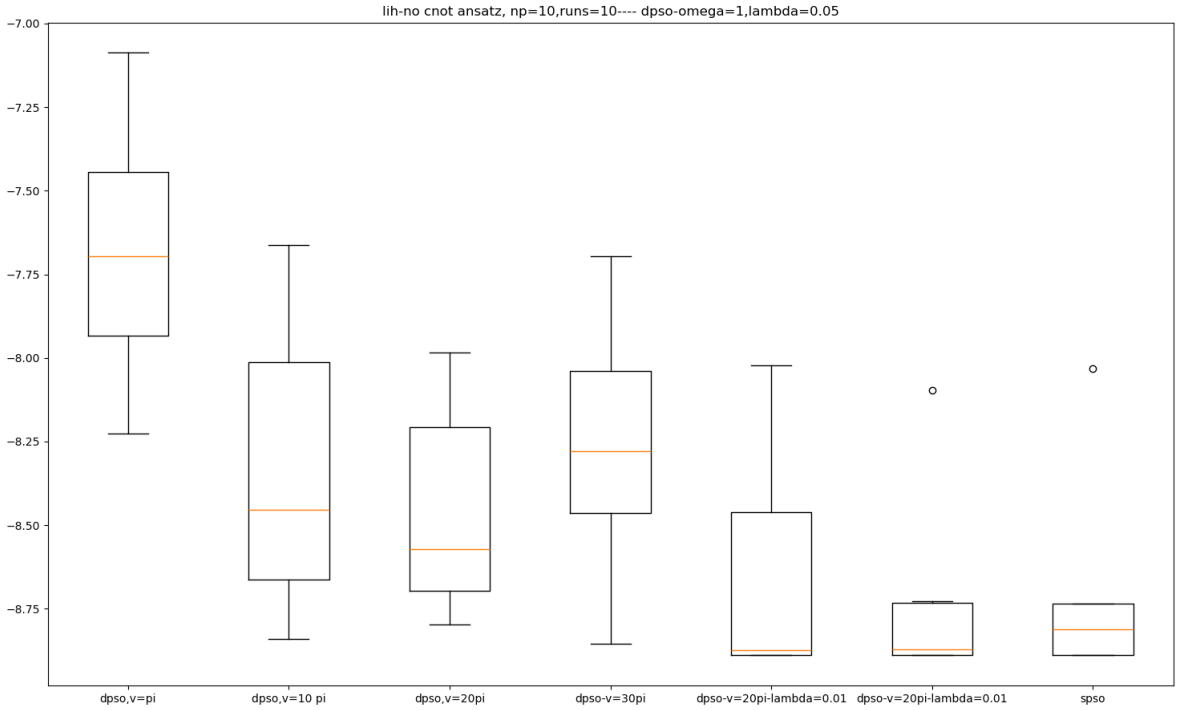


Figure 5.19: Box plot of HOPSO algorithm being run with various settings including initial velocity-ranges and damping coefficients. We can see that a smaller lambda value is the parameter that significantly improves the optimization. Note that *dpso* is the labelling used for HOPSO in this graph.

CHAPTER 6

CONCLUSION AND OUTLOOK

This thesis has presented a new non-gradient optimization algorithm that is based on the popular particle swarm optimization algorithm. This new algorithm – coined as 'HOPSO'– offers a strong advantage against the PSO in the fact that it prevents explosions and has better tuning features. That is, it is less sensitive to the tuning parameters and offers more control over them hence, unlike the PSO, converging to a value is done in a more controlled manner with more flexibility with less likeliness in premature convergence (getting stuck in a local minima). This was shown to be true when applied to a set of standard benchmark test optimization functions and compared to other non-gradient methods.

Since the algorithm that has been developed works well as a computational method, we now seek to apply this to quantum computation. As stated at the end of the previus chapter, some preliminary data on applying HOPSO to Hydrogen and LiH Hamiltonian's has been obtained. However this data, in the present stage, provides only an optimistic outlook without an in-depth analysis.

This is largely due to two reasons: over-parameterization and limiting computational resources. Firstly, by definition, every hardware-efficient ansatz is over-parameterized. To tackle this issue, we will take inspiration the from machine learning community on how to deal with over-parameterized systems. Secondly, it is difficult to study HOPSO on a hamiltonian such as Lithium Hydrite as it takes a significant amount of time to run the computer and collect data for such a large and complex function. Fortunately however, we have recently gained access to a supercomputer and anticipate in its use during this summer and fall period.

Eventually, our plan is to work with quantum computers. We plan to witness the performance of HOPSO applied onto VQE in the presence of (simulated) noise, and to use a finite number of shots as opposed to simply statevector simulation.

In order to achieve our main goal in applying this newly-developed algorithm on real quantum computers we need access to a quantum computer. IBM's quantum computers have been our primary source for such access however due to recent events, access to IBM's quantum

computer has been strictly limiting. Fortunately (again), we will be granted access to a quantum computer by the end of this year as we have a collaboration with researchers at Masaryk University in Brno, Czechia.

In my final concluding remarks, it is not yet clear on how to make HOPSO a powerful algorithm when applied to VQE quantum circuits due to the more complex landscape and the requirement of more computational power and this will be the focus for furthering our work in this field. But we are optimistic and motivated in continuing this endeavour with this newly-developed non-gradient optimization method, HOPSO.

Thank you for reading this thesis.

APPENDIX A

LITHIUM HYDRITE HAMILTONIAN

Lithium Hydrite Hamiltonian

+ 0.0017100589060386672 * IIZIIXII
- 0.0017100589060386672 * IZIIIXZZ
+ 0.19570240962718122 * IIZIIZZZ
- 0.004121355295189862 * IIZIIZZX
+ 0.004121355295189862 * IZIIIIIX
- 0.005041332803018909 * IIZIIZXI
+ 0.005041332803018909 * IZIIIIXZ
+ 0.0017100589060386688 * IZIIIXII
- 0.0017100589060386688 * IIZIIXZZ
+ 0.0884684221257211 * IZZIIIZZ
+ 0.0008858660045390596 * IZZIIIZX
- 0.0008858660045390596 * IIIIIZIX
- 0.0039011347589671453 * IZZIIIXI
+ 0.0039011347589671453 * IIIIIZXZ
+ 0.4145960467361261 * ZIIZZZZZ
- 0.02856906194286898 * ZIIZZZZX
+ 0.02856906194286898 * ZZZZZIIX
- 0.03452608676060419 * ZIIZZZXI
+ 0.03452608676060419 * ZZZZZIXZ
+ 0.012096681162908688 * ZZZZZXII
- 0.012096681162908688 * ZIIZZXZZ
- 0.028569061942868984 * ZIIZXZZZ
+ 0.028569061942868984 * IZZIXZZZ
+ 0.003501247382918566 * ZIIZXZZX
- 0.003501247382918566 * IZZIXZZX
- 0.003501247382918566 * ZZZZXIIX
+ 0.003501247382918566 * IIIIXIIX
+ 0.002844684118188048 * ZIIZXZXI
- 0.002844684118188048 * IZZIXZXI
- 0.002844684118188048 * ZZZZXIXZ
+ 0.002844684118188048 * IIIIXIXZ
- 0.0021397473591264747 * ZZZZXXII
+ 0.0021397473591264747 * IIIIXXII
+ 0.0021397473591264747 * ZIIZXXZZ
- 0.0021397473591264747 * IZZIXXZZ
- 0.03452608676060419 * ZIIXIZZZ
+ 0.03452608676060419 * IZZXZZZZ
+ 0.002844684118188048 * ZIIXIZZX
- 0.002844684118188048 * IZZXZZZX
- 0.002844684118188048 * ZZZXIIIX
+ 0.002844684118188048 * IIIXZIIX
+ 0.0053971349650047615 * ZIIXIZXI
- 0.0053971349650047615 * IZZXZZXI
- 0.0053971349650047615 * ZZZXIIXZ
+ 0.0053971349650047615 * IIIXZIXZ
- 0.000455489048037509 2 * ZZZXIXII
+ 0.000455489048037509 2 * IIIXZXII
+ 0.000455489048037509 2 * ZIIXIXZZ
- 0.000455489048037509 2 * IZZXZXZZ
+ 0.002454631350440893 * ZIXIIZII
- 0.002454631350440893 * IZXZZZII
- 0.002454631350440893 * ZZXIIIZZ
+ 0.002454631350440893 * IIXZZIZZ
+ 0.002454631350440893 * ZXIIIZII
- 0.002454631350440893 * IXZZZZII
- 0.002454631350440893 * ZXZIIIZZ
+ 0.002454631350440893 * IXIZZIZZ
+ 0.012096681162908688 * XZZIIZZZ
- 0.012096681162908688 * XIIZZZZZ
- 0.0021397473591264742 * XZZIIZZX
+ 0.0021397473591264742 * XIIZZZZX
+ 0.0021397473591264742 * XIIIIIIX
- 0.0021397473591264742 * XZZZZIIX
- 0.0004554890480375091 4 * XZZIIZXI
+ 0.0004554890480375091 4 * XIIZZZXI
+ 0.0004554890480375091 4 * XIIIIIXZ
- 0.0004554890480375091 4 * XZZZZIXZ
+ 0.0019750445457964406 * XIIIIXII
- 0.0019750445457964406 * XZZZZXII
- 0.0019750445457964406 * XZZIIXZZ
+ 0.0019750445457964406 * XIIZZXZZ
+ 0.09331458087068703 * IZZIZZZZ
+ 0.001684277431668603 * IZZIZZZX
- 0.001684277431668603 * IIIIZIIX
- 0.004123040337830278 * IZZIZZXI
+ 0.004123040337830278 * IIIIZIXZ
- 0.0016120261914684597 * IIIIZXII
+ 0.0016120261914684597 * IZZIZXZZ
- 0.0030917706989499424 * IZZYYZZZ
- 0.0030917706989499424 * IZZXXZZZ
+ 0.0008758435998611308 * IZZYYZZX
+ 0.0008758435998611308 * IZZXXZZX
- 0.0008758435998611308 * IIIYYIIX
- 0.0008758435998611308 * IIIXXIIX
- 5.172669061580785e-05 * IZZYYZXI
- 5.172669061580785e-05 * IZZXXZXI
+ 5.172669061580785e-05 * IIIYYIXZ
+ 5.172669061580785e-05 * IIIXXIXZ
- 0.0003673688232102457 * IIIYYXII
- 0.0003673688232102457 * IIIXXXII
+ 0.0003673688232102457 * IZZYYXZZ
+ 0.0003673688232102457 * IZZXXXZZ
- 0.0018832437601555981 * IZYZYZII
- 0.0018832437601555981 * IZXZXZII
+ 0.0018832437601555981 * IIYZYIZZ
+ 0.0018832437601555981 * IIXZXIZZ
- 0.0018832437601555983 * IYZZYZII
- 0.0018832437601555983 * IXZZXZII
+ 0.0018832437601555983 * IYIZYIZZ
+ 0.0018832437601555983 * IXIZXIZZ
+ 0.008763876122974601 * YIIZYZZZ
+ 0.008763876122974601 * XIIZXZZZ
- 0.0013069184709863311 * YIIZYZZX
- 0.0013069184709863311 * XIIZXZZX
+ 0.0013069184709863311 * YZZZYIIX
+ 0.0013069184709863311 * XZZZXIIX
+ 2.1449569520233115e-05 * YIIZYZXI
+ 2.1449569520233115e-05 * XIIZXZXI
- 2.1449569520233115e-05 * YZZZYIXZ
- 2.1449569520233115e-05 * XZZZXIXZ
- 6.0105529646887575e-05 * YZZZYXII
- 6.0105529646887575e-05 * XZZZXXII
+ 6.0105529646887575e-05 * YIIZYXZZ
+ 6.0105529646887575e-05 * XIIZXXZZ
+ 0.09895669236867029 * IZZZIZZZ
- 0.0028396620743700207 * IZZZIZZX
+ 0.0028396620743700207 * IIIZIIIX
+ 0.00047905267283199714 * IZZZIZXI
- 0.00047905267283199714 * IIIZIIXZ
+ 0.0025090192231687755 * IIIZIXII

- 0.0025090192231687755 * IZZZIXZZ
- 0.0025623903948187787 * IZYYIZII
- 0.0025623903948187787 * IZXXIZII
+ 0.0025623903948187787 * IIYYIIZZ
+ 0.0025623903948187787 * IIXXIIZZ
- 0.0025623903948187796 * IYZYIZII
- 0.0025623903948187796 * IXZXIZII
+ 0.0025623903948187796 * IYIYIIZZ
+ 0.0025623903948187796 * IXIXIIZZ
- 0.00437298071633473 * YIIYIZZZ
- 0.00437298071633473 * XIIXIZZZ
+ 0.0009903574427628007 * YIIYIZZX
+ 0.0009903574427628007 * XIIXIZZX
- 0.0009903574427628007 * YZZYIIIX
- 0.0009903574427628007 * XZZXIIIX
- 0.0011129841175505399 * YIIYIZXI
- 0.0011129841175505399 * XIIXIZXI
+ 0.0011129841175505399 * YZZYIIXZ
+ 0.0011129841175505399 * XZZXIIXZ
- 0.0010639354560186233 * YZZYIXII
- 0.0010639354560186233 * XZZXIXII
+ 0.0010639354560186233 * YIIYIXZZ
+ 0.0010639354560186233 * XIIXIXZZ
+ 0.0015148814402507484 * YIYIIZII
+ 0.0015148814402507484 * XIXIIZII
- 0.0015148814402507484 * YZYIIIZZ
- 0.0015148814402507484 * XZXIIIZZ
+ 0.0015148814402507484 * YYIIIZII
+ 0.0015148814402507484 * XXIIIZII
- 0.0015148814402507484 * YYZIIIZZ
- 0.0015148814402507484 * XXZIIIZZ
+ 0.09044346667151755 * ZZZIIZZZ
+ 0.0009459715341859472 * ZZZIIZZX
- 0.0009459715341859472 * ZIIIIIIX
- 0.0028371993029485216 * ZZZIIZXI
+ 0.0028371993029485216 * ZIIIIIXZ
- 0.0006523490210767924 * ZIIIIXII
+ 0.0006523490210767924 * ZZZIIXZZ
+ 0.053149299296670505 * IIIIIIZZ
+ 0.0042801745632079355 * IZZIIYYZ
+ 0.0042801745632079355 * IZZIIXXZ
+ 0.004958861420478259 * IZZIIYIY
+ 0.004958861420478259 * IZZIIXIX
+ 0.13045931889894347 * IIZIIIIZ
+ 0.0022054582814017175 * IIZIIIYY
+ 0.0022054582814017175 * IIZIIIXX
+ 0.001877634506430728 * IZIIIYZY
+ 0.001877634506430728 * IZIIIXZX
+ 0.1304593188989435 * IZIIIIIZ
+ 0.0022054582814017175 * IZIIIIYY
+ 0.0022054582814017175 * IZIIIIXX
+ 0.0018776345064307207 * IIZIIYZY
+ 0.0018776345064307207 * IIZIIXZX
+ 0.08310642186719826 * IIIIIZIZ
+ 0.002822853017034635 * IIIIIZYY
+ 0.002822853017034635 * IIIIIZXX
+ 0.09331458087068703 * ZZZZZIIZ
- 0.0030917706989499424 * ZZZZZIYY
- 0.0030917706989499424 * ZZZZZIXX
+ 0.0087638761229746 * ZIIZZYZY
+ 0.0087638761229746 * ZIIZZXZX
+ 0.001684277431668603 * ZZZZXIIZ
- 0.001684277431668603 * IIIIXIIZ
+ 0.0008758435998611308 * ZZZZXIYY
- 0.0008758435998611308 * IIIIXIYY
+ 0.0008758435998611308 * ZZZZXIXX
- 0.0008758435998611308 * IIIIXIXX
- 0.0013069184709863311 * ZIIZXYZY
+ 0.0013069184709863311 * IZZIXYZY
- 0.0013069184709863311 * ZIIZXXZX
+ 0.0013069184709863311 * IZZIXXZX
+ 0.004123040337830278 * ZZZXIIIZ
+ 0.004123040337830278 * IIIXZIIZ
- 5.172669061580785e-05 * ZZZXIIYY
+ 5.172669061580785e-05 * IIIXZIYY
- 5.172669061580785e-05 * ZZZXIIXX
+ 5.172669061580785e-05 * IIIXZIXX
+ 2.1449569520233115e-05 * ZIIXIYZY
- 2.1449569520233115e-05 * IZZXZYZY
+ 2.1449569520233115e-05 * ZIIXIXZX
- 2.1449569520233115e-05 * IZZXZXZX
- 0.0018832437601555981 * ZZYIIIZY
+ 0.0018832437601555981 * IIYZZIZY
- 0.0018832437601555981 * ZZXIIIZX
+ 0.0018832437601555981 * IIXZZIZX
- 0.0018832437601555983 * ZYZIIIZY
+ 0.0018832437601555983 * IYIZZIZY
- 0.0018832437601555983 * ZXZIIIZX
+ 0.0018832437601555983 * IXIZZIZX
- 0.0016120261914684597 * XIIIIIIZ
+ 0.0016120261914684597 * XZZZZIIZ
+ 0.0003673688232102457 * XIIIIIYY
+ 0.0003673688232102457 * XZZZZIYY
- 0.0003673688232102457 * XIIIIIXX
+ 0.0003673688232102457 * XZZZZIXX
- 6.010552964688759e-05 * XZZIIYZY
+ 6.010552964688759e-05 * XIIZZYZY
- 6.010552964688759e-05 * XZZIIXZX
+ 6.010552964688759e-05 * XIIZZXZX
+ 0.12274244052545143 * IIIIZIIZ
+ 0.011925529284481389 * IIIIZIYY
+ 0.011925529284481389 * IIIIZIXX
- 0.03239529731986452 * IZZIZYZY
- 0.03239529731986452 * IZZIZXZX
+ 0.011925529284481389 * IIIYYIIZ
+ 0.011925529284481389 * IIIXXIIZ
+ 0.003139482375497464 * IIIYYIYY
+ 0.003139482375497464 * IIIXXIYY
+ 0.003139482375497464 * IIIYYIXX
+ 0.003139482375497464 * IIIXXIXX
- 0.008499158469806097 * IZZYYYZY
- 0.008499158469806097 * IZZXXYZY
- 0.008499158469806097 * IZZYYXZX
- 0.008499158469806097 * IZZXXXZX
- 0.005928916533102496 * IIXZYIZY
+ 0.005928916533102496 * IIYZXIZY
+ 0.005928916533102496 * IIYZYIZX
+ 0.005928916533102496 * IIXZXIZX
- 0.005928916533102498 * IXIZYIZY
+ 0.005928916533102498 * IYIZXIZY
+ 0.005928916533102498 * IYIZYIZX
+ 0.005928916533102498 * IXIZXIZX
- 0.03239529731986452 * YZZZYIIZ
- 0.03239529731986452 * XZZZXIIZ
- 0.008499158469806097 * YZZZYIYY
- 0.008499158469806097 * XZZZXIYY
- 0.008499158469806097 * YZZZYIXX
- 0.008499158469806097 * XZZZXIXX
+ 0.030846096963274373 * YIIZYYZY
+ 0.030846096963274373 * XIIZXYZY
+ 0.030846096963274373 * YIIZYXZX
+ 0.030846096963274373 * XIIZXXZX
+ 0.05628878167216797 * IIIZIIIZ
- 0.0016974649623867296 * IIIZIIYY
- 0.0016974649623867296 * IIIZIIXX
+ 0.0027372506123350326 * IZZZIYZY
+ 0.0027372506123350326 * IZZZIXZX
- 0.004809206450331084 * IIXYIIZY
+ 0.004809206450331084 * IIYXIIZY
+ 0.004809206450331084 * IIYYIIZX
+ 0.004809206450331084 * IIXXIIZX
- 0.004809206450331085 * IXIYIIZY
+ 0.004809206450331085 * IYIXIIZY
+ 0.004809206450331085 * IYIYIIZX
+ 0.004809206450331085 * IXIXIIZX
+ 0.012779333033014033 * YZZYIIIZ
+ 0.012779333033014033 * XZZXIIIZ
+ 0.002221610808143227 * YZZYIIYY
+ 0.002221610808143227 * XZZXIIYY
+ 0.002221610808143227 * YZZYIIXX
+ 0.002221610808143227 * XZZXIIXX
- 0.007859003265895824 * YIIYIYZY
- 0.007859003265895824 * XIIXIYZY
- 0.007859003265895824 * YIIYIXZX
- 0.007859003265895824 * XIIXIXZX
+ 0.004890562019499969 * YZXIIIZY
- 0.004890562019499969 * XZYIIIZY
- 0.004890562019499969 * YZYIIIZX
- 0.004890562019499969 * XZXIIIZX
+ 0.004890562019499969 * YXZIIIZY
- 0.004890562019499969 * XYZIIIZY
- 0.004890562019499969 * YYZIIIZX
- 0.004890562019499969 * XXZIIIZX
+ 0.11395251883047261 * ZIIIIIIZ
+ 0.010681856282930459 * ZIIIIIYY

+ 0.010681856282930459 * ZIIIIIXX
- 0.03438974814048047 * ZZZIIYZY
- 0.03438974814048047 * ZZZIIXZX
+ 0.13073488182192297 * IIZIIIZI
- 0.004347376649778053 * IZIIIYYI
- 0.004347376649778053 * IZIIIXXI
+ 0.13073488182192297 * IZIIIIZI
- 0.004347376649778064 * IIZIIYYI
- 0.004347376649778064 * IIZIIXXI
+ 0.0538525443830966 * IIIIIZZI
+ 0.09895669236867029 * ZZZZZIZI
- 0.00437298071633473 * ZIIZZYYI
- 0.00437298071633473 * ZIIZZXXI
- 0.002839662074370021 * ZZZZXIZI
+ 0.002839662074370021 * IIIIXIZI
+ 0.0009903574427628007 * ZIIZXYYI
- 0.0009903574427628007 * IZZIXYYI
+ 0.0009903574427628007 * ZIIZXXXI
- 0.0009903574427628007 * IZZIXXXI
+ 0.00047905267283199714 * ZZZXIIZI
- 0.00047905267283199714 * IIIXZIZI
- 0.0011129841175505399 * ZIIXIYYI
+ 0.0011129841175505399 * IZZXZYYI
- 0.0011129841175505399 * ZIIXIXXI
+ 0.0011129841175505399 * IZZXZXXI
- 0.002562390394818779 * ZZYIIIYI
+ 0.002562390394818779 * IIYZZIYI
- 0.002562390394818779 * ZZXIIIXI
+ 0.002562390394818779 * IIXZZIXI
- 0.002562390394187796 * ZYZIIIYI
+ 0.002562390394187796 * IYIZZIYI
- 0.002562390394187796 * ZXZIIIXI
+ 0.002562390394187796 * IXIZZIXI
+ 0.002509019223168775 * XIIIIIZI
- 0.002509019223168775 * XZZZZIZI
- 0.0010639354560186233 * XZZIIYYI
+ 0.0010639354560186233 * XIIZZYYI
- 0.0010639354560186233 * XZZIIXXI
+ 0.0010639354560186233 * XIIZZXXI
+ 0.05628878167216797 * IIIIZIZI
+ 0.012779333033014033 * IZZIZYYI
+ 0.012779333033014033 * IZZIZXXI
- 0.00169746496238673 * IIIYYIZI
- 0.00169746496238673 * IIIXXIZI
+ 0.0022216108081432265 * IZZYYYYI
+ 0.0022216108081432265 * IZZXXYYI
+ 0.0022216108081432265 * IZZYYXXI
+ 0.0022216108081432265 * IZZXXXXI
- 0.004809206450331084 * IIXZYIYI
+ 0.004809206450331084 * IIYZXIYI
+ 0.004809206450331084 * IIYZYIXI
+ 0.004809206450331084 * IIXZXIXI
- 0.004809206450331085 * IXIZYIYI
+ 0.004809206450331085 * IYIZXIYI
+ 0.004809206450331085 * IYIZYIXI
+ 0.004809206450331085 * IXIZXIXI
+ 0.0027372506123350326 * YZZZYIZI
+ 0.0027372506123350326 * XZZZXIZI
- 0.007859003265895824 * YIIZYYYI
- 0.007859003265895824 * XIIZXYYI
- 0.007859003265895824 * YIIZYXXI
- 0.007859003265895824 * XIIZXXXI
+ 0.08460131391824206 * IIIZIIZI
- 0.009002501243838557 * IZZZIYYI
- 0.009002501243838557 * IZZZIXXI
- 0.010323101079295216 * IIXYIIYI
+ 0.010323101079295216 * IIYXIIYI
+ 0.010323101079295216 * IIYYIIXI
+ 0.010323101079295216 * IIXXIIXI
- 0.010323101079295216 * IXIYIIYI
+ 0.010323101079295216 * IYIXIIYI
+ 0.010323101079295216 * IYIYIIXI
+ 0.010323101079295216 * IXIXIIXI
- 0.009002501243838557 * YZZYIIZI
- 0.009002501243838557 * XZZXIIZI
+ 0.006587584190054828 * YIIYIYYI
+ 0.006587584190054828 * XIIXIYYI
+ 0.006587584190054828 * YIIYIXXI
+ 0.006587584190054828 * XIIXIXXI
+ 0.0034522696762380027 * YZXIIIYI
- 0.0034522696762380027 * XZYIIIYI
- 0.0034522696762380027 * YZYIIIXI
- 0.0034522696762380027 * XZXIIIXI
+ 0.0034522696762380044 * YXZIIIYI
- 0.0034522696762380044 * XYZIIIYI
- 0.0034522696762380044 * YYZIIIXI
- 0.0034522696762380044 * XXZIIIXI
+ 0.06044012857315143 * ZIIIIIZI
+ 0.01095277357379615 * ZZZIIYYI
+ 0.01095277357379615 * ZZZIIXXI
+ 0.2707726623751816 * IZZIIIII
+ 0.1294881663056809 * IIZIIZII
+ 0.19570240962718116 * ZZIZZIII
- 0.004121355295189859 * ZZIZXIII
+ 0.004121355295189859 * IIZIXIII
- 0.005041332803018909 * ZZIXIIII
+ 0.005041332803018909 * IIZXZIII
+ 0.0015148814402507484 * ZIYIIYII
- 0.0015148814402507484 * IZYZZYII
+ 0.0015148814402507484 * ZIXIIXII
- 0.0015148814402507484 * IZXZZXII
+ 0.0017100589060386672 * XIZIIIII
- 0.0017100589060386672 * XZIZZIII
+ 0.13045931889894347 * IIZIZIII
+ 0.0022054582814017175 * IIZYYIII
+ 0.0022054582814017175 * IIZXXIII
+ 0.004890562019499968 * IZXZYYII
- 0.004890562019499968 * IZYZXYII
- 0.004890562019499968 * IZYZYXII
- 0.004890562019499968 * IZXZXXII
+ 0.0018776345064307276 * YZIZYIII
+ 0.0018776345064307276 * XZIZXIII
+ 0.13073488182192297 * IIZZIIII
+ 0.003452269676238003 * IZXYIYII
- 0.003452269676238003 * IZYXIYII
- 0.003452269676238003 * IZYYIXII
- 0.003452269676238003 * IZXXIXII
- 0.004347376649778053 * YZIYIIII
- 0.004347376649778053 * XZIXIIII
+ 0.0084345697568455 * IXXIIIII
+ 0.0084345697568455 * IYYIIIII
- 0.004902761294508434 * YIXIIYII
+ 0.004902761294508434 * XIYIIYII
+ 0.004902761294508434 * YIYIIXII
+ 0.004902761294508434 * XIXIIXII
+ 0.1294881663056809 * ZIZIIIII
+ 0.12948816630568094 * IZIIIZII
+ 0.19570240962718122 * ZIZZZIII
- 0.004121355295189862 * ZIZZXIII
+ 0.004121355295189862 * IZIIXIII
- 0.005041332803018909 * ZIZXIIII
+ 0.005041332803018909 * IZIXZIII
+ 0.0015148814402507487 * ZYIIIYII
- 0.0015148814402507487 * IYZZZYII
+ 0.0015148814402507487 * ZXIIIXII
- 0.0015148814402507487 * IXZZZXII
+ 0.0017100589060386688 * XZIIIIII
- 0.0017100589060386688 * XIZZZIII
+ 0.1304593188989435 * IZIIZIII
+ 0.002205458281401719 * IZIYYIII
+ 0.002205458281401719 * IZIXXIII
+ 0.004890562019499969 * IXZZYYII
- 0.004890562019499969 * IYZZXYII
- 0.004890562019499969 * IYZZYXII
- 0.004890562019499969 * IXZZXXII
+ 0.0018776345064307207 * YIZZYIII
+ 0.0018776345064307207 * XIZZXIII
+ 0.13073488182192297 * IZIZIIII
+ 0.003452269676238004 * IXZYIYII
- 0.003452269676238004 * IYZXIYII
- 0.003452269676238004 * IYZYIXII
- 0.003452269676238004 * IXZXIXII
- 0.004347376649778064 * YIZYIIII
- 0.004347376649778064 * XIZXIIII
- 0.004902761294508436 * YXIIIYII
+ 0.004902761294508436 * XYIIIYII
+ 0.004902761294508436 * YYIIIXII
+ 0.004902761294508436 * XXIIIXII
+ 0.12948816630568094 * ZZIIIIII
+ 0.09044346667151755 * ZZZZZZII
+ 0.0009459715341859472 * ZZZZXZII
- 0.0009459715341859472 * IIIIXZII
- 0.0028371993029485216 * ZZZXIZII
+ 0.0028371993029485216 * IIIXZZII
- 0.0006523490210767925 * XIIIIZII

+ 0.0006523490210767925 * XZZZZZII
+ 0.11395251883047261 * IIIIZZII
+ 0.01068185628293046 * IIIYYZII
+ 0.01068185628293046 * IIIXXZII
- 0.03438974814048047 * YZZZYZII
- 0.03438974814048047 * XZZZXZII
+ 0.06044012857315143 * IIIZIZII
+ 0.01095277357379615 * YZZYIZII
+ 0.01095277357379615 * XZZXIZII
+ 0.11384335176465168 * ZIIIIZII
+ 0.08981333348776849 * ZZZZIIII
+ 0.0002470865807618955 * ZZZXXIII
+ 0.0002470865807618955 * ZZZYYIII
- 0.006624128763848125 * XIIIXIII
- 0.006624128763848125 * YIIIYIII
- 0.003247196737969148 * ZZZXZIII
+ 0.003247196737969148 * IIIXIIII
- 0.0029189446624547895 * XIIIZIII
+ 0.0029189446624547895 * XZZZIIII
- 0.0003888183927304 7865 * XIIYYIII
- 0.0013577262659730466 * XIIXXIII
- 0.0009689078732425674 * YIIYXIII
- 0.0009689078732425674 * YZZYXIII
- 0.0013577262659730466 * XZZXXIII
- 0.0003888183927304 7865 * XZZYYIII
+ 0.09355955740366553 * ZZZIZIII
+ 0.004828469764372238 * XIIXZIII
+ 0.004828469764372238 * YIIYZIII
- 0.0028913887649858303 * ZZZIXIII
+ 0.0028913887649858303 * IIIZXIII
+ 0.001396035105618235 * XIIZIIII
- 0.001396035105618235 * XZZIZIII
+ 0.0884684221257211 * IZZZZIII
+ 0.0008858660045390596 * IZZZXIII
- 0.0008858660045390596 * ZIIIXIII
- 0.0039011347589671453 * IZZXIIII
+ 0.0039011347589671453 * ZIIXZIII
+ 0.053149299296670505 * IIIZZIII
+ 0.0042801745632079355 * YZZYZIII
+ 0.0042801745632079355 * XZZXZIII
+ 0.004958861420478259 * YZZIYIII
+ 0.004958861420478259 * XZZIXIII
+ 0.08310642186719826 * ZIIIZIII
+ 0.002822853017034635 * ZIIYYIII
+ 0.002822853017034635 * ZIIXXIII
+ 0.0538525443830966 * ZIIZIIII

BIBLIOGRAPHY

[1] S. AXLER, *Linear Algebra Done Right*, Springer, 1996.

[2] L. E. BALLENTINE, *Quantum Mechanics: A Modern Development*, World Scientific, 1998.

[3] P. BENIOFF, *The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines*, Springer, 1980.

[4] M. CLERC AND J. KENNEDY, *The particle swarm-explosion, stability, and convergence in a multidimensional complex space*, IEEE Transactions on Evolutionary Computation, 6 (2002), pp. 58–73.

[5] C. COHEN-TANNOUDJI, B. DIU, AND F. LALOË, *Quantum Mechanics, Volume 1*, Wiley, 1977.

[6] J. B. CONWAY, *A Course in Functional Analysis*, Springer, 1990.

[7] P. DIRAC, *The Principles of Quantum Mechanics*, Oxford University Press, 1981.

[8] R. C. EBERHART AND J. KENNEDY, *Particle swarm optimization*, Proceedings of IEEE International Conference on Neural Networks, 4 (1995), pp. 1942–1948.

[9] D. J. GRIFFITHS, *Introduction to Quantum Mechanics*, Pearson Prentice Hall, 2004.

[10] Z. HOLMES, K. SHARMA, M. CEREZO, AND P. J. COLES, *Connecting ansatz expressibility to gradient magnitudes and barren plateaus*, PRX Quantum, 3 (2022), p. 010313.

[11] A. KANDALA, A. MEZZACAPO, K. TEMME, M. TAKITA, M. BRINK, J. M. CHOW, AND J. M. GAMBETTA, *Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets*, Nature, 549 (2017), pp. 242–246.

[12] J. KENNEDY AND R. EBERHART, *Particle swarm optimization*, in Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, IEEE, 1995, pp. 1942–1948.

[13] J. MCCLEAN, S. BOIXO, V. SMELYANSKIY, AND ET AL., *Barren plateaus in quantum neural network training landscapes*, Nature Communications, 9 (2018), p. 4812.

[14] J. R. MCCLEAN, J. ROMERO, R. BABBUSH, AND A. ASPURU-GUZIK, *The theory of variational hybrid quantum-classical algorithms*, New Journal of Physics, 18 (2016), p. 023023.

[15] M. A. NIELSEN AND I. L. CHUANG, *Quantum Computation and Quantum Information*, Cambridge University Press, 2010.

[16] E. OZCAN AND C. MOHAN, *Particle swarm optimization: surfing the waves*, Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (IEEE World Congress on Computational Intelligence), (1998), pp. 1939–1944.

[17] A. PERUZZO, J. MCCLEAN, P. SHADBOLT, M.-H. YUNG, X.-Q. ZHOU, P. J. LOVE, A. ASPURU-GUZIK, AND J. L. O'BRIEN, *A variational eigenvalue solver on a photonic quantum processor*, Nature Communications, 5 (2014), p. 4213.

[18] R. POLI, J. KENNEDY, AND T. BLACKWELL, *Particle swarm optimization*, Swarm Intelligence, 1 (2007), pp. 33–57.

[19] M. J. POWELL, *A direct search optimization method that models the objective and constraint functions by linear interpolation*, Advances in Optimization and Numerical Analysis, (1994), pp. 51–67.

[20] J. SAKURAI, *Modern Quantum Mechanics*, Addison-Wesley, 2011.

[21] R. STORN AND K. PRICE, *Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, 11 (1997), pp. 341–359.

[22] S. SURJANOVIC AND D. BINGHAM, *Virtual library of simulation experiments: Test functions and datasets*.
`https://www.sfu.ca/~ssurjano/optimization.html`, 2023.
Accessed: 2024-06-18.

[23] S. SURJANOVIC AND D. BINGHAM, *Virtual library of simulation experiments: Test functions and datasets*, 2024.

[24] A. SZABO AND N. S. OSTLUND, *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*, Dover Publications, 2012.

[25] J. TILLY, H. CHEN, S. CAO, D. PICOZZI, K. SETIA, Y. LI, E. GRANT, L. WOSSNIG, I. RUNGGER, G. H. BOOTH, AND J. TENNYSON, *The variational quantum eigensolver: A review of methods and best practices*, Physics Reports, 986 (2022), p. 1–128.

[26] J. S. TOWNSEND, *A Modern Approach to Quantum Mechanics*, University Science Books, 2012.

[27] P. VIRTANEN, R. GOMMERS, T. E. OLIPHANT, M. HABERLAND, T. REDDY, D. COURNAPEAU, E. BUROVSKI, P. PETERSON, W. WECKESSER, J. BRIGHT, S. J. VAN DER WALT, M. BRETT, J. WILSON, K. J. MILLMAN, N. MAYOROV, A. R. J. NELSON, E. JONES, R. KERN, E. LARSON, C. J. CAREY, İ. POLAT, Y. FENG, E. W. MOORE, J. VANDERPLAS, D. LAXALDE, J. PERKTOLD, R. CIMRMAN, I. HENRIKSEN, E. A. QUINTERO, C. R. HARRIS, A. M. ARCHIBALD, A. H. RIBEIRO, F. PEDREGOSA, P. VAN MULBREGT, AND SCIPY 1.0 CONTRIBUTORS, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272.