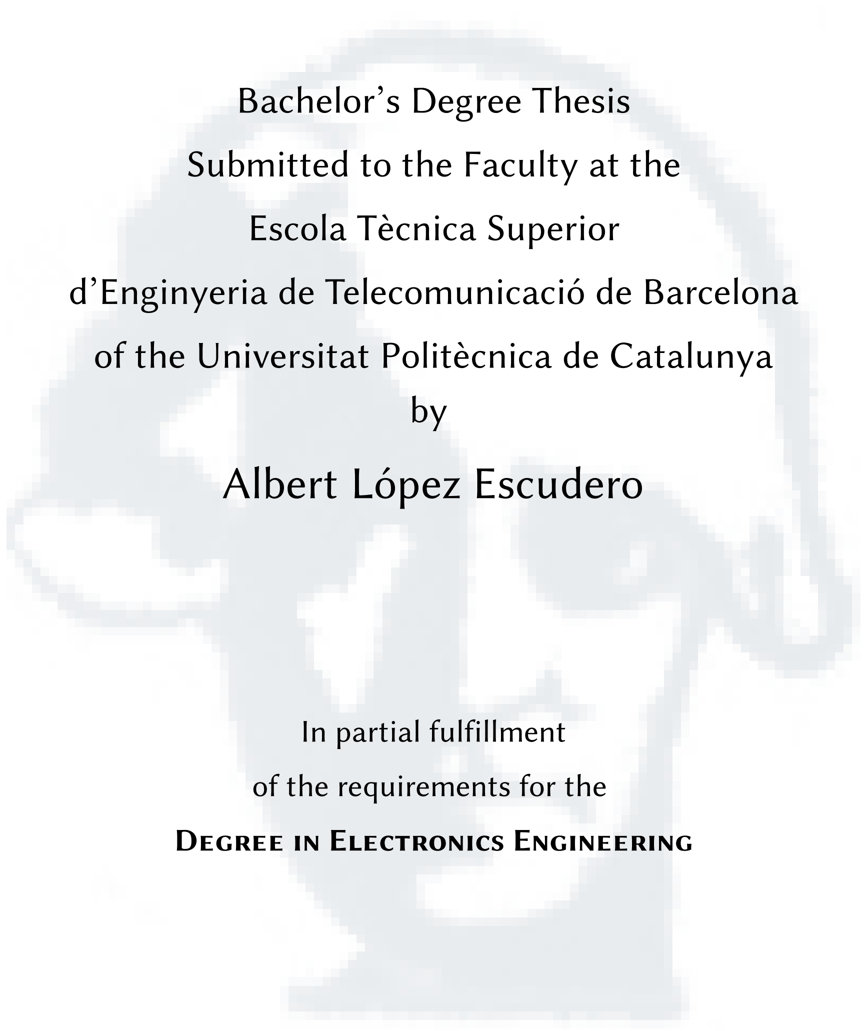
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

telecos
BCN

Quantum Simulation



Bachelor's Degree Thesis
Submitted to the Faculty at the
Escola Tècnica Superior
d'Enginyeria de Telecomunicació de Barcelona
of the Universitat Politècnica de Catalunya
by

Albert López Escudero

In partial fulfillment
of the requirements for the
DEGREE IN ELECTRONICS ENGINEERING

Advisor: Vincenzo de Maio & Ivona Bandric
Reviewer: Javier Rodrigez Fonollosa

Barcelona, December 2024

ETSETB – Degree in Electronics Engineering

Resum

Cada exemplar del Treball de Fi de Grau (TFG) ha de contenir un Resum, que és un breu extracte del TFG. En termes d'estil, el Resum hauria de ser una versió reduïda del projecte: una introducció concisa, un compendi dels resultats i les principals conclusions o arguments presentats en el projecte. El Resum no ha de superar les 150 paraules i cal que estigui traduït al català, castellà i anglès.

Resumen

Cada ejemplar del Trabajo de Fin de Grado (TFG) debe incluir un Resumenque es un breve extracto del TFG. En cuanto al estilo, el Resumen debería ser una versión reducida del proyecto: una introducción breve, un resumen de los resultados principales y las conclusiones o argumentos principales presentados en el proyecto. El Resumen no debe exceder las 150 palabras y debe estar traducido al catalán, castellano e inglés.

Summary

Each copy of the Bachelor's Thesis (TFG) must include a Summary, which is a concise abstract of the TFG. In terms of style, the Summary should be a condensed version of the project: a brief introduction, a summary of the main results, and the conclusions or key arguments presented in the project. The Summary should not exceed 150 words and must be translated to catalan, spanish and english.

A Dedication page may be included in your thesis just before the Acknowledgments page, but it is not a requirement.

Acknowledgements

It is appropriate, but not mandatory, to declare the extent to which assistance has been given by members of the staff, fellow students, technicians or others in the collection of materials and data, the design and construction of apparatus, the performance of experiments, the analysis of data, and the preparation of the thesis (including editorial help). In addition, it is appropriate to recognize the supervision and advice given by your advisor.

Revision history and approval record

Revision	Date	Author(s)	Description
1.0	dd/mm/yyyy	AME	Document creation
1.1	dd/mm/yyyy	AME, JPV	Error correction
2.0	dd/mm/yyyy	AME, MLO	Revised after review
4.0	dd/mm/yyyy	AME	Final version

DOCUMENT DISTRIBUTION LIST

Role	Surname(s) and Name
[Student]	
[Project Supervisor 1]	
[Project Supervisor 2 (if applicable)]	

Written by:		**Reviewed and approved by:**	
Date	dd/mm/yyyy	Date	dd/mm/yyyy
Name	Xxxxxxx Yyyyyyy	Name	Xxxxxxx Yyyyyyy
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

Abbreviations

ETSETB Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

EU European Union

GEF Grau en Enginyeria Física

GREELEC Grau en Enginyeria Electrònica de Telecomunicació

GRETST Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació

CHAPTER 1

Introduction

An Introduction that clearly states the rationale of the thesis that includes:

1. Statement of purpose (objectives).
2. Requirements and specifications.
3. Methods and procedures, citing if this work is a continuation of another project or it uses applications, algorithms, software or hardware previously developed by other authors.
4. Work plan with tasks, milestones and a Gantt diagram.
5. Description of the deviations from the initial plan and incidences that may have occurred.

The minimum chapters that this thesis document should have are described below, nevertheless they can have different names and more chapters can be added.

1.1 Work goals

The primary objective of this project is to contribute a new methodology to the realm of molecular simulations by:

- Developing a quantum simulation program capable of efficiently modeling molecules using quantum algorithms.
- Introducing innovative code that implements adaptive circuit construction and advanced optimization strategies.
- Demonstrating the effectiveness of our approach by simulating selected molecular systems and comparing the results with classical computational methods.
- Enhancing the integration between quantum and classical computations to optimize both electronic and nuclear degrees of freedom.

1.2 Requirements and specifications

To achieve these objectives, the following requirements and specifications have been established:

- **Quantum Computing Frameworks**: Utilize quantum computing libraries such as PennyLane and JAX for quantum circuit simulation and automatic differentiation.
- **Algorithm Implementation**: Implement the VQE algorithm with adaptive circuit construction, allowing the selection of the most significant excitations based on energy gradients.
- **Optimization Techniques**: Employ efficient optimizers, including gradient-based methods like gradient descent and advanced techniques like the Quantum Natural Gradient optimizer.
- **Hamiltonian Construction**: Accurately build molecular Hamiltonians for various molecular geometries and ensure compatibility with standard quantum chemistry basis sets.
- **Visualization Tools**: Develop modules for visualizing energy convergence, parameter evolution, and molecular geometries throughout the optimization process.
- **Computational Resources**: Ensure the code is optimized for performance, making effective use of computational resources and supporting parallel execution where possible.
- **Extensibility**: Design the codebase to be modular and extensible, allowing future enhancements and adaptation to other molecular systems or quantum algorithms.

1.3 Methods and procedures

This project introduces innovative methods and procedures to enhance molecular simulations using quantum computing. The key innovations in our code are centered around adaptive circuit construction, advanced optimization strategies, and the seamless integration of quantum and classical computations.

Adaptive Circuit Construction

Traditional VQE implementations often rely on fixed ansätze, which may not efficiently capture the complexities of all molecular systems. Our code implements an adaptive approach where the quantum circuit is dynamically constructed by selecting excitations (operators) from a predefined pool based on their contribution to lowering the system's energy. This selection is guided by computing the energy gradients with respect to each operator, ensuring that only the most impactful excitations are included in the circuit. This method enhances computational efficiency and can lead to faster convergence to the ground state energy.

Advanced Optimization Techniques

Optimizing the parameters of the quantum circuit is crucial for the success of the VQE algorithm. Our implementation leverages both gradient-based and gradient-free optimization methods. We utilize optimizers like the *Gradient Descent Optimizer* and explore advanced techniques such as the *Quantum Natural Gradient* optimizer, which accounts for the geometry of the parameter space and can provide faster convergence. Additionally, our code extends the optimization process to include the nuclear coordinates, allowing simultaneous optimization of the electronic structure and molecular geometry.

Integration of Quantum and Classical Computation

Our code exemplifies a robust integration of quantum and classical computational techniques. By employing automatic differentiation tools provided by frameworks like JAX and PennyLane, we can efficiently compute gradients of the energy with respect to both circuit parameters and nuclear coordinates. This integration facilitates the optimization process and enables the handling of complex systems that are challenging for classical methods alone.

Innovations in Code Implementation

Key innovations in our code include:

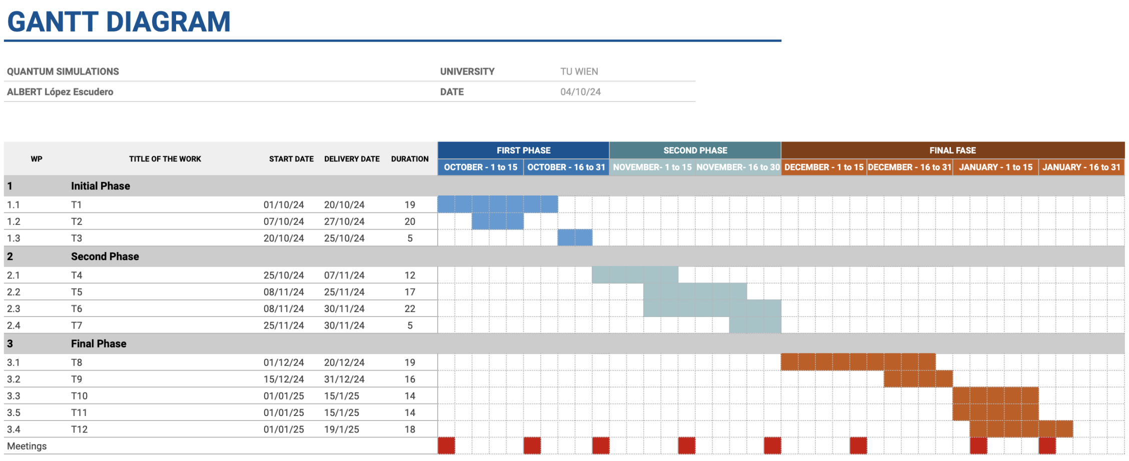
- **Dynamic Operator Selection**: A procedure to compute energy gradients for each operator in the pool and select the one with the highest impact, thus adaptively constructing the quantum circuit.
- **Hybrid Optimization Loop**: An iterative loop that updates both quantum circuit parameters and nuclear positions, enhancing the ability to find the global minimum energy configuration.
- **Efficient Gradient Computation**: Implementation of numerical methods to compute gradients with respect to nuclear coordinates, enabling geometry optimization within the VQE framework.
- **Visualization and Analysis Tools**: Development of comprehensive visualization functions to analyze the optimization process, including energy evolution plots and 3D representations of molecular geometries.
- **Modularity and Extensibility**: A modular code structure that allows for easy adaptation to different molecules, basis sets, and quantum devices, facilitating future research and development.

Utilization of Existing Frameworks and Contribution to the Field

While our work builds upon established quantum computing frameworks such as PennyLane and utilizes existing algorithms like the VQE, the innovations introduced in our code represent significant advancements in the field of molecular simulations. By enhancing the efficiency of quantum simulations and providing new methods for adaptive circuit construction and parameter optimization, this project contributes valuable tools and methodologies to researchers and practitioners in quantum chemistry and quantum computing.

1.4 Work plan

Normally the figures and tables are put in `\figure` and `\table` environments, that can float freely in the document. You can identify each float with a `\label`


GANTT DIAGRAM

QUANTUM SIMULATIONS		UNIVERSITY	TU WIEN
ALBERT López Escudero		DATE	04/10/24

WP	TITLE OF THE WORK	START DATE	DELIVERY DATE	DURATION
1	**Initial Phase**			
1.1	T1	01/10/24	20/10/24	19
1.2	T2	07/10/24	27/10/24	20
1.3	T3	20/10/24	25/10/24	5
2	**Second Phase**			
2.1	T4	25/10/24	07/11/24	12
2.2	T5	08/11/24	25/11/24	17
2.3	T6	08/11/24	30/11/24	22
2.4	T7	25/11/24	30/11/24	5
3	**Final Phase**			
3.1	T8	01/12/24	20/12/24	19
3.2	T9	15/12/24	31/12/24	16
3.3	T10	01/01/25	15/1/25	14
3.5	T11	01/01/25	15/1/25	14
3.4	T12	01/01/25	19/1/25	18
Meetings				

Figure 1.1: *Project's Gantt diagramGantt diagram of the project. For more information read the manual [**skalagantt**] of Skala..*

CHAPTER 2

State of the Art of the Technology Used or Applied in this Thesis

In recent years, computing has undergone significant evolution, reaching a point where improving the hardware of traditional devices presents considerable challenges. This has driven research in quantum computing, a technology that promises to revolutionize the field by enabling much more efficient calculations and superior processing capabilities. In applications such as molecular simulation, quantum computing has proven to be remarkably more efficient than classical computing, justifying investment in its development.

However, one of the main obstacles of quantum computing is the high cost associated with its devices. To run programs on a quantum computer, it must operate at extremely low temperatures, close to 0.1 Kelvin, which significantly increases the cost and technical complexity. Therefore, methods are being investigated to improve and optimize these devices, making them more accessible and viable for broader use.

Currently, one of the most practical ways to harness quantum computing's potential is through **quantum simulation**. Quantum simulation allows us to study complex quantum systems using either quantum simulators or classical computers. By emulating the behavior of quantum systems, we can explore quantum algorithms and applications effectively without necessarily requiring a fully functional quantum computer.

In this project, we focus on the operation of quantum simulators applied to molecular simulation, an area where quantum computing offers significant advantages. The main objective is to develop a program capable of simulating different molecules in a simple and efficient manner.

To this end, we will review the basic concepts of quantum mechanics that are essential to understand the fundamentals and potential of quantum computing, and explore the techniques of quantum simulation that allow us to harness quantum computing capabilities even with current technological limitations.

2.1 Quantum Simulation

Quantum simulation has emerged as an advanced and essential technique for studying complex quantum systems, especially those that are inaccessible or present great challenges for direct analysis using classical methods. Based on the proposal of Richard Feynman, who postulated that a computer built from quantum elements could overcome the limitations of classical computers in simulating quantum phenomena, quantum simulation has progressed significantly. It encompasses both digital and analog simulations and has expanded its applicability in various scientific areas.

There are mainly two approaches in quantum simulation: **Digital Quantum Simulation (DQS)** and **Analog Quantum Simulation (AQS)**. DQS employs the quantum circuit model, where systems are represented by qubits that evolve through quantum gates to reproduce the dynamics of the target system. This approach is universal, as it can, in principle, simulate any quantum system, although not always efficiently. On the other hand, AQS involves creating a quantum system that directly emulates the Hamiltonian of the system under study, allowing certain properties of the simulated system, such as time evolution, to be reproduced approximately. This method is particularly useful when a qualitative representation is required rather than high precision.

In addition to these approaches, there are algorithms inspired by quantum information theory that facilitate the classical simulation of quantum systems. Techniques such as **Matrix Product States (MPS)** and **Projected Entangled Pair States (PEPS)** allow representing particle systems on classical computers more efficiently than standard classical methods, optimizing the calculation of properties of complex quantum systems.

The applications of quantum simulation are broad and encompass multiple scientific fields. In condensed matter physics, it allows the study of models such as the Hubbard model and quantum phase transitions, fundamental for understanding phenomena like superconductivity. In quantum chemistry, it facilitates the calculation of molecular energies and complex chemical reactions. In high-energy physics and cosmology, it emulates particles in high-energy fields and cosmological phenomena. Furthermore, quantum simulation is instrumental in the analysis of open quantum systems and in the investigation of quantum chaos, allowing exploration of interactions with the environment and chaotic dynamics in the quantum realm.

However, quantum simulation faces significant challenges related to the precise control of the quantum simulator systems and the management of decoherence and errors, which can affect the accuracy of the results. The amount of required resources, such as the number of qubits and quantum gates, also depends on the size and complexity of the system to be simulated. It is estimated that quantum simulators require between 40 and 100 qubits to surpass the computational power of classical computers in specific problems. Despite these challenges, technological advances continue to improve the viability and efficiency of quantum simulation, promising to transform research in natural sciences and expand our understanding of quantum phenomena.

2.2 Key Concepts in Quantum Mechanics

It is essential to understand the difference between bits in classical computing and qubits in quantum computing to delve into this new technological paradigm.

In classical computing, the basic unit of information is the **bit**, which can take the value of 0 or 1. These bits are the foundation upon which conventional computers operate, processing information through combinations of these binary states.

In contrast, quantum computing uses the **qubit** or quantum bit as its basic unit. Unlike the classical bit, a qubit can exist in a superposition of states, meaning it can simultaneously represent the values 0 and 1 thanks to the principle of superposition in quantum mechanics. This property, along with phenomena such as quantum entanglement and interference, allows quantum computers to process information exponentially more efficiently for certain problems.

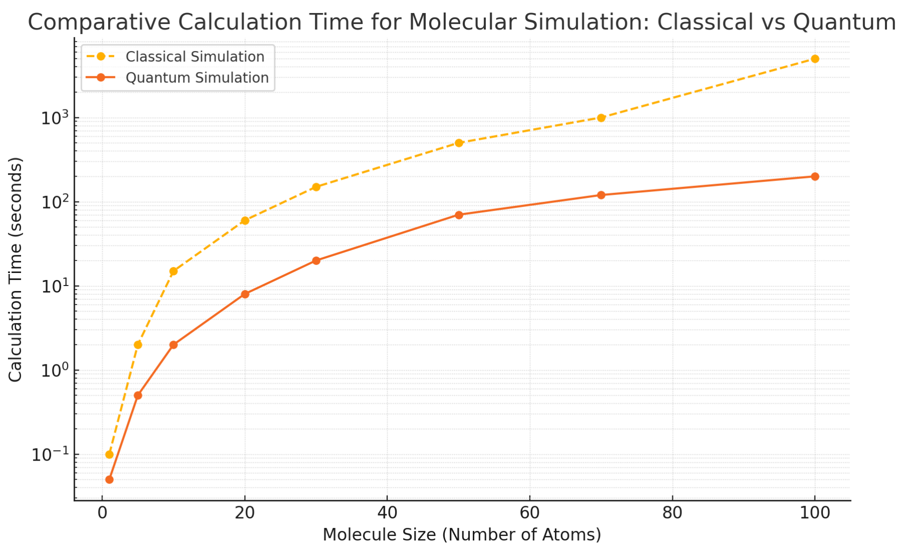


Figure 2.1: *Comparison of computation time for molecular simulations: classical vs quantum.*

Understanding how qubits operate and their differences from classical bits is essential to appreciate the revolutionary potential of quantum computing.

2.2.1 Qubit

The **qubit** is the basic unit of information in quantum computing. While the classical bit can only be in one of two states (0 or 1), a qubit can be in a superposition of both states simultaneously. This is due to the principle of quantum superposition, one of the fundamental characteristics of quantum mechanics.

Mathematically, a qubit is represented as a linear combination of the basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers that satisfy the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. These coefficients indicate the probability amplitudes of finding the qubit in the states $|0\rangle$ or $|1\rangle$ upon measurement.

In addition to superposition, qubits can exhibit **quantum entanglement**, a property that allows creating strong correlations between qubits that cannot be explained by classical physics. Entanglement is essential for the computational power of quantum computers, as it enables processing and storing an exponentially larger amount of information than classical systems.

For example, while a classical system of n bits can represent one of 2^n possible state combinations, a quantum system of n qubits can represent a superposition of all those combinations simultaneously. This capability is what allows quantum computers to tackle complex problems more efficiently.

However, manipulating and maintaining qubits is a significant technical challenge. Qubits are extremely sensitive and can be affected by interactions with the environment, leading to **quantum decoherence**. To minimize this effect and preserve quantum properties, it is necessary to keep systems in controlled conditions, such as very low temperatures, close to absolute zero.

2.2.2 Quantum Superposition

Quantum superposition allows a quantum system to exist in multiple states simultaneously until a measurement is performed. This characteristic is key to the functioning of quantum computers, as it enables processing a large amount of information in parallel.

In quantum systems, superposition is combined with **quantum interference**, where the probability amplitudes of states can reinforce or cancel each other out. This phenomenon is exploited in quantum algorithms to increase the probability of obtaining the correct result. For example, in Grover's algorithm, constructive interference amplifies the probability of the desired state, significantly improving the efficiency of searching for elements in an unsorted database.

Superposition is especially useful in simulating complex molecular systems. Quantum computers can naturally model the superpositions of electronic states in molecules, which is crucial for studying chemical reactions and molecular properties that are difficult to address with classical methods due to the exponential growth of computational resources required.

2.2.3 Quantum Decoherence

Quantum decoherence is one of the main challenges in quantum computing. It refers to the loss of a system's quantum properties, such as superposition and entanglement, due to unwanted interactions with the environment. This loss causes the quantum system to transition toward classical behavior, affecting the accuracy and reliability of quantum calculations.

Qubits are extremely sensitive to external disturbances, such as electromagnetic fluctuations, vibrations, and temperature changes. These interactions can cause quantum states to mix with those of the environment, leading to a loss of coherence that is irreversible and degrades the stored quantum information.

To mitigate the effects of decoherence, various strategies are implemented:

- **System Isolation**: Designing physical systems that minimize unwanted interactions with the environment, using materials and techniques that protect qubits from external disturbances.
- **Quantum Error Correction**: Implementing error correction codes that allow detecting and correcting errors without directly measuring the qubit's state, thereby preserving quantum information.
- **Dynamic Control**: Applying techniques such as pulse refocusing and dynamic pulse sequences that actively compensate for disturbances and extend the coherence time of qubits.

Controlling and mitigating decoherence are essential for the advancement of quantum computing and its application in areas like molecular simulation, where the precision of calculations is fundamental.

2.3 The Hamiltonian in Quantum Mechanics

The Hamiltonian is a fundamental concept originating from classical mechanics, introduced by William Rowan Hamilton in 1833. Hamiltonian mechanics is a reformulation of classical mechanics that provides powerful tools for studying the dynamics of systems. The Hamiltonian function represents the total energy of the system, expressed in terms of generalized coordinates and momenta, and is given by the sum of the kinetic and potential energies.

In quantum mechanics, the **Hamiltonian operator** plays a central role in describing the energy and time evolution of quantum systems. It represents the total energy of the system, including both kinetic and potential energies, and is essential for formulating the Schrödinger equation.

2.3.1 Mathematical Definition

The Hamiltonian operator, commonly denoted as $\hat{H}$, is a self-adjoint operator acting on the Hilbert space associated with the quantum system. For a single particle in one dimension, the Hamiltonian is expressed as:

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

where:

- $\hat{T}$ is the kinetic energy operator.
- $\hat{V}$ is the potential energy operator.

In terms of the position $\hat{x}$ and momentum $\hat{p}$ operators, these are defined as:

$$\hat{T} = \frac{\hat{p}^2}{2m} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2}$$

$$\hat{V} = V(\hat{x})$$

Here, m is the mass of the particle, $\hbar$ is the reduced Planck constant, and $V(\hat{x})$ is the potential energy function depending on position.

2.3.2 Role in the Schrödinger Equation

The Hamiltonian is central to the Schrödinger equation, which describes how the quantum state of a system evolves over time. The time-dependent Schrödinger equation is expressed as:

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle$$

where $|\psi(t)\rangle$ is the state vector of the system at time t. For time-independent systems, the Schrödinger equation reduces to the eigenvalue equation:

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

Here, E represents the eigenvalues of the Hamiltonian, corresponding to the allowed energy levels of the system, and $|\psi\rangle$ are the associated eigenstates.

2.3.3 Hamiltonian in Multi-Particle Systems

For systems with multiple particles, the Hamiltonian includes additional terms representing interactions between particles. For example, for a system of two particles, the Hamiltonian is expressed as:

$$\hat{H} = \hat{T}_1 + \hat{T}_2 + \hat{V}_1 + \hat{V}_2 + \hat{V}_{12}$$

where:

- $\hat{T}_1$ and $\hat{T}_2$ are the kinetic energy operators of particles 1 and 2, respectively.
- $\hat{V}_1$ and $\hat{V}_2$ are the individual potential energy operators.
- $\hat{V}_{12}$ represents the potential interaction between the two particles.

2.3.4 Importance in Quantum Simulations

In quantum simulations, especially in algorithms like the Variational Quantum Eigensolver (VQE), the Hamiltonian is decomposed into a sum of simpler terms, often expressed in terms of Pauli operators. This decomposition facilitates implementation on quantum circuits and allows estimating the system's energy through measurements on qubits.

Understanding the structure and properties of the Hamiltonian is essential for modeling and simulating quantum systems, as it determines the possible energies and dynamics of the system under study.

2.4 Algorithms

Once we have understood the basic concepts, we focus on the quantum algorithms used in particle simulation. These algorithms make use of quantum logic gates, which are detailed in Appendix A.

2.4.1 VQE: Variational Quantum Eigensolver

The VQE is a hybrid quantum-classical algorithm designed to find the minimum energy of a quantum system, such as a molecule. This algorithm combines quantum state preparation with classical optimization. It leverages the quantum properties of qubits to find the ground state of the molecule more rapidly, followed by classical optimization to refine the solution.

Stages of the Algorithm

El VQE is based on the variational principle, which states that the expected energy of any approximate state $|\psi(\theta)\rangle$ is always greater than or equal to the energy of the true ground state E_0:

$$E(\theta) = \langle \psi(\theta)|H|\psi(\theta)\rangle \geq E_0$$

1. Quantum State Preparation A parameterized quantum circuit known as an *ansatz* is used, defined by a set of adjustable parameters $\vec{\theta}$. This circuit applies a sequence of quantum gates, such as rotations and entangling gates, to generate a quantum state:

$$|\psi(\vec{\theta})\rangle$$

2. Measurement of the Expected Energy For a given quantum state $|\psi(\vec{\theta})\rangle$, the expected energy of the system under a Hamiltonian H is measured:

$$E(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle$$

The Hamiltonian H is decomposed into terms of Pauli operators representing the electronic and nuclear interactions in the system.

3. Classical Optimization The parameters $\vec{\theta}$ are iteratively adjusted using a classical optimizer to minimize $E(\vec{\theta})$.

4. Iteration of the Process The steps of state preparation, measurement, and optimization are repeated until $E(\vec{\theta})$ converges to a minimum value. This value corresponds to the ground state energy of the system.

2.4.2 Adaptive Circuits

Adaptive circuits allow optimizing the design of quantum circuits for specific problems. They dynamically adjust their structure based on feedback and optimization criteria during the execution process.

An example is found in *Variational Quantum Algorithms* (VQA), which use classical optimization techniques to adjust the parameters of a quantum circuit and minimize a cost function. These algorithms are especially useful in current quantum devices, which are noisy and of limited size (NISQ).

For more information and practical examples on adaptive circuits:

https://pennylane.ai/qml/demos/tutorial_adaptive_circuits/

2.5 Optimizers

In the realm of quantum simulation algorithms like the **Variational Quantum Eigensolver** (VQE), optimizers are essential components that facilitate the minimization of the expected energy of a quantum system. Following the preparation of quantum states and the measurement processes described earlier, optimizers are employed in the classical computation stage to adjust the parameters of the quantum circuit, known as the *ansatz*.

The theoretical purpose of optimizers in this project is to solve a continuous optimization problem. They aim to find the optimal set of parameters $\vec{\theta}$ that minimize the cost function $E(\vec{\theta})$, which represents the expectation value of the Hamiltonian H with respect to the quantum state $|\psi(\vec{\theta})\rangle$:

$$E(\vec{\theta}) = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle$$

Optimizers utilize mathematical techniques to navigate the high-dimensional parameter space effectively. Depending on the specific characteristics of the problem, different optimization methods can be employed:

- **Gradient-based methods**: These methods compute the gradient of the cost function with respect to the parameters and use this information to guide the search towards the minimum energy. Examples include gradient descent and its variants.
- **Gradient-free methods**: In cases where calculating the gradient is impractical or the cost function is noisy, gradient-free methods like Nelder-Mead or COBYLA can be used.
- **Second-order methods**: These methods, such as the BFGS algorithm, approximate the second derivatives (Hessian) of the cost function to achieve faster convergence.

Within the iterative loop of the VQE algorithm, the optimizer updates the parameters $\vec{\theta}$ after each quantum measurement based on the chosen optimization strategy. This process continues until convergence is achieved, meaning the expected energy $E(\vec{\theta})$ reaches a minimum value that approximates the ground state energy of the system.

Integrating optimizers into the quantum-classical workflow is crucial because they connect quantum computations with classical numerical methods. They enable effective exploration of the parameter space, addressing challenges such as multiple local minima and flat regions in the energy landscape that are inherent in quantum systems.

By employing appropriate optimization techniques, this project aims to enhance the efficiency and accuracy of molecular simulations. Optimizers play a pivotal role in leveraging the capabilities of quantum computing to achieve results that are difficult to obtain with classical computational methods alone, thus contributing to the advancement of quantum simulation despite current technological limitations.

CHAPTER 3

Methodology / project development

In this chapter, the methodology used in the completion of the work will be detailed. Its aim is to offer a thorough account of the approaches and techniques used, ensuring replicability and academic rigor. It will not only cover the research methods and measurement techniques employed but will also delve into the specifics of software and hardware development. Whether the project involves qualitative analysis, quantitative measurements, computational modeling, or physical prototyping, this chapter should elucidate how each component contributes to the overall objectives.

In addition to describing the methods themselves, the chapter will also provide justifications for why specific methods were chosen over others. For example, it may explain the choice of a particular programming language, statistical test, or experimental setup. The chapter will also address the limitations of the methodology and how these have been mitigated or accounted for. Readers should come away with a clear understanding of how the project's development has been carried out, why certain choices were made, and how these methods serve to fulfill the initially established objectives.

3.1 Framework Selection

To make the decision on which framework to use, we compared the documentation of the two quantum simulation frameworks available in the market: PennyLane and Qiskit. These are the most comprehensive frameworks with similar features available at the time of creating this project. After reviewing the documentation, we ultimately chose to use PennyLane for two reasons.

The first reason was the amount of documentation related to quantum simulation. Once we started looking into how others were using these resources, we realized that in the field of molecular simulation, the existing documentation—both theoretical and especially practical—was substantially greater. This provided us with more examples to begin developing our project.

The second reason for our choice was the frequent major changes implemented by Qiskit.

We realized that while Qiskit is a tool that promises to be very good, it has historically undergone significant structural changes.

For these reasons, this project has been developed using the PennyLane framework. Below, we will observe how the project has been developed and explain the reasons behind the decisions made.

3.2 Project Structuring

3.2.1 Code Organization

```
                    quantum_simulation_project/
config/
     config_functions.py: Configuration functions for the project.
     molecules.json: Molecule data.
     __pycache__: Python cache files.
main.py: Main program file.
modules/
     ansatz_preparer.py: Quantum ansatz preparation.
     hamiltonian_builder.py: Molecular Hamiltonian construction.
     molecule_manager.py: Molecular data management.
     opt_mol.py: Molecular optimization.
     optimizer.py: Optimization algorithms.
     visualizer.py: Visualization tools.
     __pycache__: Python cache files.
temp_results_autograd/
     energy_evolution.png: Energy evolution graph.
     filtered_report_autograd.txt: Filtered results report.
     final_geometries_3D.png: Image of the final 3D geometries.
     nuclear_coordinates.png: Nuclear coordinates.
     output.txt: Program data output.
     profile_output_autograd.txt: Autograd profile output.
test/: Directory for tests.
```

Modular Structure

After deciding on the interface to use and implementing the first version of the code, we decided to reorganize the project to make it more precise and modular. This structure offers the possibility to easily add more lines of code and functionalities.

Description of Files and Directories

- **Main File (`main.py`)**: This file serves as the entry point of the program. It initializes the initial conditions, sets up the molecule's configuration, defines optimization options, and orchestrates the execution of the quantum simulation and geometry optimization process. It is the first file to execute when running the application.

- **Auxiliary Modules (`modules/`)**: This directory contains various modules that encapsulate the core logic and operations of the project:
  - `ansatz_preparer.py`: Includes functions for constructing the quantum circuit from the variational ansatz, preparing states, and applying excitations.
  - `hamiltonian_builder.py`: Responsible for generating the molecular Hamiltonian based on nuclear coordinates and the selected electronic basis.
  - `molecule_manager.py`: Manages information related to the molecule, such as its charge, multiplicity, number of electrons, and required orbitals.
  - `opt_mol.py`: Functions that orchestrate the molecular optimization process, invoking the optimizer and recording the simulation's progress.
  - `optimizer.py`: Implements the optimization logic, combining routines for evaluating the cost (energy) with the selected optimization algorithms, updating parameters, and geometries.
  - `visualizer.py`: Generates graphical outputs and reports that display the evolution of energy, nuclear coordinates, and other relevant data during optimization.
- **Temporary Results Directories (`temp_results_autograd`)**: Several directories with names starting with `temp_results_autograd` store output data, time logs, and visualizations generated for different simulations and configurations. For documentation purposes, these directories are treated as a single repository for temporary results.
- **Dependencies (`requirements.txt`)**: This file lists the required libraries and their versions to reproduce the project's execution environment. Keeping it updated ensures reproducibility of the simulation across different systems, facilitating the installation of necessary dependencies.

3.2.2 Version Control

Version control was managed using Git, allowing detailed tracking of changes and facilitating continuous collaboration with the supervisors on the project. Primarily, at the start of the project, a single branch was used to develop the project and explore the framework's possibilities. Once a stable version was achieved, branches were created to conduct tests and develop new functionalities. The first branches created were for the different interface versions. In each branch, the code was refined so that the same code would run across the various interfaces. Finally, only the interface changes that proved most suitable for the project were merged back into the main branch.

3.3 Development and Implementation

In our simulator, the method we use to find the energy of the ground state of a quantum system is the VQE. We have already explained the concept of VQE in the state of the art chapter; now we will explain how we have implemented it in our project and how we have integrated it.

Principle of VQE: The VQE is based on the variational principle, which states that the expected energy of any approximate state $|\psi(\theta)\rangle$ is always greater than or equal to the real ground state energy E_0:

$$E(\theta) = \langle\psi(\theta)|H|\psi(\theta)\rangle \geq E_0$$

We have already discussed this concept, but it is necessary to emphasize it as it is the foundation of the entire algorithm. The idea is to find the parameters θ that minimize the expected energy, thereby approaching the real value of the ground state energy.

Next, we will detail how the VQE is implemented in our project, explaining how each component has been developed.

3.3.1 Implementation of VQE in Our Simulator

The VQE algorithm in our simulator consists of several key steps, which we describe below:

1. **Preparation of the Quantum Ansatz**

   The first step to implement the VQE algorithm (Variational Quantum Eigensolver) is to prepare the quantum state of the system. This is achieved using an **ansatz**, which is a parameterized quantum circuit designed to approximate the system's ground state. In our project, we have implemented an adaptive ansatz that selects the most relevant excitations, thus allowing a more efficient and accurate representation of the ground state.

Preparation of the Quantum Ansatz

```
def prepare_ansatz(params, hf_state, selected_excitations,
spin_orbitals):
    qml.BasisState(hf_state, wires=range(spin_orbitals))
    for i, exc in enumerate(selected_excitations):
        if len(exc) == 2:
            qml.SingleExcitation(params[i], wires=exc)
        elif len(exc) == 4:
            qml.DoubleExcitation(params[i], wires=exc)
```

This ansatz starts from the Hartree-Fock state, represented by hf_state, and applies a series of single and double excitation operations, each parameterized by an angle θ. The selected excitations are applied using PennyLane's SingleExcitation and DoubleExcitation gates, allowing the construction of a quantum state that captures the system's electronic correlations.

2. **Definition of the Cost Function**

   With the ansatz defined, the next step is to establish a cost function that evaluates the expected energy of the system given a set of parameters θ. In our implementation, this cost function is defined within update_parameters_and_coordinates and calculates the expected value of the molecular Hamiltonian:

   Definition of the Cost Function

```python
@qml.qnode(dev, interface=interface)
def cost_fn(params):
    prepare_ansatz(params, hf_state, selected_excitations, spin_orbitals)
    return qml.expval(hamiltonian)
```

   This function is essential for evaluating $E(\theta)$. By calculating the expected value of the Hamiltonian, we can quantify how close our approximate state is to the true ground state.

3. **Optimization of the Parameters**

   Finally, we use classical optimizers, such as GradientDescentOptimizer, to adjust the parameters θ and minimize the expected energy $E(\theta)$. This process is critical for finding the quantum state that best approximates the system's ground state.

   Optimization of the Parameters

```python
params, energy = opt.step_and_cost(cost_fn, params)
```

   In this step, the parameters θ are iteratively updated to reduce the value of the cost function. The step_and_cost method performs a parameter update and returns the associated energy, facilitating the tracking of the optimization progress.

Optimization Cycle

In our simulation, we have utilized these basic concepts; however, since the objective of our simulation is to optimize the molecular geometry, we have had to modify the algorithm somewhat. In our case, we have implemented an optimization loop where, in each iteration, the ansatz parameters and the nuclear coordinates are optimized. Below, we will explain the steps of the loop and delve deeper into the methodology used and the steps of each part of the loop.

- **Construction of the Molecular Hamiltonian:** The Hamiltonian of the system is generated for the current molecular geometry, incorporating any changes in the nuclear coordinates.
- **Calculation of the Operator Gradients:** The energy gradients with respect to each operator in the excitation pool are calculated, identifying which excitations contribute most to the energy reduction.
- **Selection of the Most Significant Operator Based on Gradients:** The operator with the largest gradient (in absolute value) is selected to be included in the ansatz, ensuring that the updates are the most effective.
- **Updating the Ansatz Parameters and Nuclear Coordinates:** The ansatz parameters θ and, if necessary, the positions of the nuclei are adjusted using optimization techniques, aiming to minimize the system's total energy.

The cycle continues until the predefined convergence criteria are met or the established maximum number of iterations is completed. This iterative approach ensures that the ansatz is progressively refined, incorporating the most relevant excitations and adjusting the parameters to accurately approximate the system's ground state.

3.3.2 Hamiltonian Construction Process

1. **Definition of Molecular Geometry:**

   The geometry is specified by the atomic symbols and the Cartesian coordinates of each atom in the molecule:

   Geometry Definition

   ```
   symbols = [’H’, ’H’]
   x_init = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.74])
   ```

2. **Hamiltonian Construction:**

   The `build_hamiltonian` function generates the molecular Hamiltonian using PennyLane's functions:

   Hamiltonian Build

   ```
   def build_hamiltonian(x, symbols, charge=0, mult=1, basis_name=’sto-3g’):
       x = np.array(x)
       coordinates = x.reshape(-1, 3)
       hamiltonian, qubits = qml.qchem.molecular_hamiltonian(
           symbols, coordinates, charge=charge, mult=mult, basis=basis_name
       )
       h_coeffs, h_ops = hamiltonian.terms()
   ```

```python
    h_coeffs = np.array(h_coeffs)
    hamiltonian = qml.Hamiltonian(h_coeffs, h_ops)
    return hamiltonian
```

Note: A basis set, such as 'sto-3g', is selected, which is a predefined set of basis functions to represent atomic orbitals in a simplified manner. This makes the simulation more efficient.

Function Description:

This function generates the qubit Hamiltonian of a molecule by transforming the electronic Hamiltonian in second quantization into the Pauli matrix framework. Additionally, it allows for the incorporation of net charge effects, spin multiplicity, and an active space defined by a specific number of electrons and orbitals, optimizing the quantum simulation of molecular systems.

3.3.3 Parameter and Geometry Optimization

In this phase of the simulation, the algorithm refines both the electronic and nuclear degrees of freedom within a unified optimization loop. As previously discussed in the optimization cycle, each iteration not only adjusts the parameters θ of the variational ansatz but also updates the nuclear coordinates $\mathbf{X}$. By doing so, the approach brings the system closer to its true ground state energy and equilibrium molecular geometry simultaneously.

Integration into the Optimization Cycle: The optimization cycle described above involves four key steps: constructing the molecular Hamiltonian, computing operator gradients, selecting the most relevant operator, and finally updating both the ansatz parameters and the nuclear coordinates. While the initial steps focus on identifying and incorporating crucial excitations into the ansatz, this last step ensures that the molecular structure itself becomes more stable. By optimizing θ and $\mathbf{X}$ together, the algorithm effectively searches the coupled electronic-nuclear energy landscape, guiding the molecule toward a configuration that yields a lower overall energy.

Parameter Update Mechanism: After selecting the most significant operator, we append a new parameter corresponding to that operator to the ansatz. This step, as seen in the code snippet below, ensures that at each iteration the ansatz gains complexity in the most relevant direction:

Parameter Extension

```python
# Inside the main optimization loop:
selected_excitations.append(selected_gate)

```

```
# Append a new parameter for the chosen operator
params = np.append(params, 0.0)
params = np.array(params, requires_grad=True)
```

Here we re-initialize the optimizer after extending the parameter vector:

Optimizer Re-initialization

```
opt = type(opt)(stepsize=STEP_SIZE)
```

Justification: Re-initializing the optimizer ensures that changes in parameter dimensionality do not disrupt the internal state of algorithms like Adam or RMSProp, which store momentum terms. Although this approach discards historical gradients, it guarantees that each iteration starts consistently, preventing errors that could arise from dimension mismatches. While this may slow convergence slightly, it prioritizes correctness and stability of the optimization process.

Nuclear Geometry Updates: In parallel with parameter updates, the nuclear geometry is also refined. The code below illustrates how the nuclear gradients are computed and applied:

Updating Parameters and Coordinates

```
params, x, energy_history, x_history, opt_state = update_parameters_and_coordinates(
    opt, opt_state, cost_fn, params, x, symbols, selected_excitations, dev, hf_state, spin_o
    learning_rate_x, convergence, interface, charge, mult, basis_name
)
```

Within `update_parameters_and_coordinates`, both θ and $\mathbf{X}$ are optimized:

Parameter and Geometry Steps

```
for opt_step in range(10):
    params, energy = opt.step_and_cost(cost_fn, params)
    grad_x = compute_nuclear_gradients(params, x, symbols, selected_excitations,
                                       dev, hf_state, spin_orbitals, interface, charge, mult
    x = x - learning_rate_x * grad_x
```

Justification: Performing multiple optimization sub-steps (`opt_step`) in each main iteration smooths out fluctuations and ensures a more stable descent in the nuclear coordinates. The choice of `learning_rate_x` is crucial: a too-large value may drive the molecule into unrealistic configurations, while a too-small one may slow down convergence. After experimenting with various rates, we chose a moderate value (e.g., `0.01`) to balance speed and stability.

Coupling Electronic and Nuclear Optimizations: By updating the electronic parameters and the molecular geometry within the same loop, the algorithm follows a path that reflects physical reality—electrons and nuclei influence each other. This integrated approach ensures that the final state is not only electronically optimized but also structurally sound, representing a stable molecular configuration close to the equilibrium structure. As the number of iterations progresses, both parameters and coordinates converge, indicated by diminishing energy gradients and more stable final geometries.

Convergence and Termination: As the optimization proceeds, criteria such as the maximum gradient value falling below a predefined threshold (`CONV`) or reaching the maximum number of iterations (`MAX_ITER`) trigger termination. This ensures that the algorithm stops when it either finds a stable solution or when further progress becomes negligible.

Justification: Defining explicit convergence criteria prevents infinite loops and ensures that the simulation delivers a result within a reasonable time. For this project, we have chosen a stringent `CONV` value (e.g., 10^{-8}) to ensure the final geometry and energy parameters are well-converged. This high precision may increase computational cost but results in more reliable and reproducible outcomes.

Selection of the Most Significant Operator Based on Gradients

Identifying which operator to add to the ansatz at each iteration is a crucial step in refining the variational representation of the molecular ground state. In our approach, we compute the gradient of the expected energy with respect to each candidate operator in the excitation pool. This procedure provides a quantitative measure of how much impact a given operator has on lowering the system's energy if included in the ansatz.

Gradient-Based Operator Ranking: By evaluating the energy gradients, the algorithm can distinguish which excitations are most promising. The operator associated with the largest absolute gradient value is the one that, if turned on (i.e., given a nonzero parameter), offers the steepest descent in energy. This ensures that each added operator contributes effectively to reducing the total energy, rather than randomly guessing or adding operators that have minimal effect.

Operator Selection Code Snippet

```
selected_gate, max_grad_value = select_operator(gradients, operator_pool_copy, convergence)
if selected_gate is None:
    # No more significant operators to add
    break
```

```
selected_excitations.append(selected_gate)
```

Justification: This gradient-driven choice is not arbitrary; it ensures that resources (in terms of computational effort and circuit complexity) are invested in the most impactful directions. By systematically adding operators that yield the greatest immediate reduction in energy, the ansatz grows in complexity in a controlled and meaningful manner. This strategy strikes a balance between accuracy and computational efficiency, as it avoids introducing unnecessary parameters that would provide negligible energy improvements.

Updating the Ansatz Parameters and Nuclear Coordinates

Once the most relevant operator has been selected and incorporated into the ansatz, the next step is to refine both the electronic and nuclear configurations. This refinement process involves adjusting the variational parameters θ associated with the chosen operators and, if needed, the nuclear coordinates $\mathbf{X}$ of the molecule.

Joint Electronic-Nuclear Optimization: In many simulations, molecular structure and electronic configurations are interdependent. The electronic state depends on the positions of the nuclei, and the equilibrium nuclear geometry is influenced by the electronic distribution. By updating θ and $\mathbf{X}$ together within the same optimization loop, the algorithm can move the system toward a self-consistent solution that minimizes the total energy. This integrated approach ensures that improving the variational state is not done in isolation but rather aligned with finding the best molecular geometry.

Parameter and Geometry Update Code Snippet

```
params = np.append(params, 0.0)
params = np.array(params, requires_grad=True)

# Re-initialize optimizer after changing parameter dimension
opt = type(opt)(stepsize=STEP_SIZE)

params, x, energy_history, x_history, opt_state = update_parameters_and_coordinates(
    opt, opt_state, cost_fn, params, x, symbols, selected_excitations, dev, hf_state, spin_o
    learning_rate_x, convergence, interface, charge, mult, basis_name
)
```

Justification: This process is guided by gradient-based optimizers that adjust θ to decrease the energy. Simultaneously, nuclear gradients inform how to shift the atoms' positions, nudging the molecule toward a more stable, lower-energy structure. The chosen learning rates for parameters and nuclear coordinates have been tuned to ensure smooth convergence. If the learning rates are too high, the system may exhibit oscillatory or erratic behavior; if too low, convergence may be excessively slow. By carefully selecting these optimization parameters, we achieve a stable and efficient path toward a well-converged solution.

Outcome: The combined approach of adding impactful operators at each iteration and updating both electronic and nuclear degrees of freedom results in a methodical journey toward a more accurate molecular ground state. With each iteration, the ansatz becomes richer, and the geometry refines, culminating in a final state that closely approximates the molecule's true equilibrium configuration and ground-state energy.

Has to be finished

To be continued...

3.4 Elección de Optimizador

Para poder conseguir el objetivo de desarrollar un simulador cuántico de la manera más optimizada, el primer paso fue generar una comparación entre distintas simulaciones utilizando los mismos parámetros y condiciones, pero variando el optimizador. De esta forma, podríamos determinar cuál de ellos convergía de manera más estable y rápida, sin incrementar excesivamente el tiempo de cómputo. Además, esta comparación nos permitiría contar con una base sólida para futuras decisiones, como la elección del ansatz más adecuado o la optimización de la geometría molecular.

Al inicio, el código no contemplaba la ejecución con múltiples optimizadores a la vez. Estaba enfocado en un único optimizador, dificultando el análisis comparativo. Para solventar esto, se implementó una modificación en el script principal que:

1. Define un diccionario con diversos optimizadores disponibles.
2. Itera sobre cada uno de ellos para ejecutar la misma simulación.
3. Registra y compara los resultados (energía final, número de iteraciones, tiempo total, etc.).

A continuación se muestran las líneas de código clave que se han añadido o modificado para realizar dicha comparación. Cabe destacar que el siguiente fragmento se integra en la función principal de optimización (por ejemplo, `optimize_molecule`), en el punto donde antes se usaba un único optimizador.

Como se puede observar, el cambio principal radica en el uso de un bucle `for optimizer_name, opt in optimizers.items():` que nos permite ejecutar el mismo procedimiento de optimización molecular con cada optimizador de la lista. Esto no sólo agiliza la comparación, sino que también evita duplicar código.

Al generar la simulación con cada uno de los optimizadores, se puede observar cómo estos se comportan a medida que la ejecución avanza. Cada optimizador presenta ligeras diferencias en:

- La velocidad a la que disminuye la energía.
- El número de iteraciones requeridas para alcanzar la convergencia.
- La estabilidad ante fluctuaciones en los parámetros.

Posteriormente, comentaremos los resultados de las distintas simulaciones que efectuamos. Pero podemos adelantar que, en las pruebas realizadas, el optimizador que mejor se comporta ha resultado ser **RMSProp**. El hecho de que RMSProp funcione de manera particularmente efectiva en este contexto podría atribuirse a su capacidad de adaptar la tasa de aprendizaje por parámetro, estabilizando así el descenso hacia el mínimo energético.

Un dato a tener en cuenta es que, como indicaremos en el apartado de resultados y discusión, el tiempo total de ejecución entre los distintos optimizadores apenas varió. Este punto es relevante ya que, si todos tardan un tiempo similar, el factor clave pasa a ser *la calidad y la velocidad de la convergencia*. En otras palabras, lo que buscamos es el optimizador que alcance el valor mínimo de energía con menos iteraciones y mayor estabilidad, y en nuestras corridas experimentales, RMSProp destaca en este sentido.

Si bien se podría considerar también la influencia del ansatz, la cantidad de parámetros y otros factores, esta primera comparación se centró exclusivamente en el impacto de distintos algoritmos de optimización, manteniendo invariables otros aspectos. Así, el usuario dispone ahora de una referencia clara sobre qué optimizador ofrecería un mejor punto de partida para simulaciones más complejas.

3.5 Foundations of VQE and Justification for Its Selection

The *Variational Quantum Eigensolver* (VQE) was chosen as the primary method to estimate the ground state energy of the studied quantum system. VQE combines limited quantum processing (measurements and applicability in moderately deep circuits) with classical optimization techniques. Its selection is justified by:

- **Suitability for NISQ devices:** VQE is particularly well-suited for noisy intermediate-scale quantum (NISQ) devices, as it requires circuits of relatively low depth.
- **Flexible Ansatz:** It allows the use of various adaptive variational ansätze that capture essential electronic correlations.
- **Direct coupling to classical optimizers:** The VQE cost function (the expected energy) can be minimized with a wide range of classical methods, making it easy to experiment with different optimizers.

The core principle of VQE is the variational theorem, which guarantees that the expected energy of the ansatz is always an upper bound to the true ground state energy. By optimizing the ansatz parameters, the algorithm progressively approaches the actual energy minimum.

3.6 Implementing VQE in the Simulator

The integration of VQE in the simulator follows a step-by-step scheme detailed below:

3.6.1 Preparing the Quantum Ansatz

The first step is to define the ansatz, i.e., the parameterized quantum circuit that aims to approximate the system's ground state. Our approach starts from the Hartree-Fock state (`hf_state`) and applies a series of electronic excitations (single and double excitation operators) whose amplitudes are controlled by parameters θ:

Quantum Ansatz Preparation

```python
def prepare_ansatz(params, hf_state, selected_excitations, spin_orbitals):
    qml.BasisState(hf_state, wires=range(spin_orbitals))
    for i, exc in enumerate(selected_excitations):
        if len(exc) == 2:
            qml.SingleExcitation(params[i], wires=exc)
        elif len(exc) == 4:
            qml.DoubleExcitation(params[i], wires=exc)
```

This adaptive ansatz grows in complexity as new excitations, selected based on their energy gradients, are added. Thus, not only predefined excitations are used, but the algorithm itself determines which are most relevant for lowering the energy.

3.6.2 Defining the Cost Function

The cost function in VQE is the expected energy of the molecular Hamiltonian for a given set of parameters θ. Through this function, we evaluate the quantum state prepared by the ansatz:

Cost Function Definition

```python
@qml.qnode(dev, interface=interface)
def cost_fn(params):
    prepare_ansatz(params, hf_state, selected_excitations, spin_orbitals)
    return qml.expval(hamiltonian)
```

Based on this function, the classical optimizer adjusts θ to minimize the energy. In this manner, VQE forms a bridge between quantum mechanics (state preparation and Hamiltonian measurement) and classical optimization (searching for the energy minimum).

3.6.3 Parameter Optimization of the Ansatz

Parameter optimization (θ) is performed with classical optimizers, for example:

Parameter Optimization

```
params, energy = opt.step_and_cost(cost_fn, params)
```

In each iteration, the optimizer updates θ to reduce the cost function. This process repeats until a convergence criterion is met (e.g., an energy variation below a given threshold).

3.7 The Optimization Cycle: Parameters and Nuclear Geometry

Our approach is novel in that we not only optimize the electronic state but also the molecular geometry. The optimization cycle integrates several steps:

1. **Molecular Hamiltonian Construction:** The qubit Hamiltonian is computed for the current geometry.
2. **Operator Gradient Calculation:** Energy derivatives with respect to each candidate operator in the excitation pool are determined.
3. **Selection of the Most Relevant Operator:** The operator with the highest gradient is chosen for inclusion in the ansatz, increasing its variational expressiveness.
4. **Updating Parameters and Nuclear Coordinates:** Both θ and the nuclear positions X are optimized simultaneously. This brings the system closer not only to the electronic minimum but also to a more stable molecular structure.

This cycle repeats until the convergence criteria are met, whether energetic, geometric, or related to the maximum number of iterations.

3.8 Molecular Hamiltonian Construction

The molecular Hamiltonian is generated from the atomic geometry definition. For example:

Geometry Definition

```
symbols = ['H', 'H']
x_init = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.74])
```

The Hamiltonian construction function:

Hamiltonian Construction

```python
def build_hamiltonian(x, symbols, charge=0, mult=1, basis_name='sto-3g'):
    x = np.array(x)
    coordinates = x.reshape(-1, 3)
    hamiltonian, qubits = qml.qchem.molecular_hamiltonian(
        symbols, coordinates, charge=charge, mult=mult, basis=basis_name
    )
    h_coeffs, h_ops = hamiltonian.terms()
    h_coeffs = np.array(h_coeffs)
    hamiltonian = qml.Hamiltonian(h_coeffs, h_ops)
    return hamiltonian
```

A chosen basis set (e.g., 'sto-3g') represents atomic orbitals efficiently, balancing precision and complexity to avoid a combinatorial explosion in the number of qubits.

3.9 Simultaneous Parameter and Geometry Updates

During each main optimization iteration, multiple sub-steps are performed on θ and the nuclear gradients are recalculated. This allows updating atomic coordinates to minimize the total energy (both electronic and geometric) of the molecule:

Joint Parameter and Coordinate Update

```python
params, x, energy_history, x_history, opt_state = update_parameters_and_coordinates(
    opt, opt_state, cost_fn, params, x, symbols, selected_excitations, dev, hf_state, spin_o
    learning_rate_x, convergence, interface, charge, mult, basis_name
)
```

The choice of the learning rate for nuclear coordinates (`learning_rate_x`) and the number of sub-steps ensures a smooth and stable descent, avoiding unrealistic physical configurations or numerical divergences.

3.10 Selecting the Most Significant Operator from Gradients

To determine which operator to add to the ansatz at each iteration, we compute the energy gradient with respect to each operator in the excitation pool. The operator with the largest absolute gradient offers the greatest potential for reducing the energy if assigned a nonzero parameter:

Operator Selection

```python
selected_gate, max_grad_value = select_operator(gradients, operator_pool_copy, convergence)
if selected_gate is None:
    # No more relevant operators
    break
selected_excitations.append(selected_gate)
```

This approach ensures that the ansatz grows in the most effective direction, optimizing computational resources and improving convergence.

3.11 Optimizer Comparison and Selection

To refine the methodology, we evaluated different classical optimizers (e.g., `GradientDescentOptimizer`, `RMSProp`, `Adam`) under the same initial conditions. This comparative analysis identified the most robust optimizer, one that converges quickly and stably toward the energy minimum.

Initially, the code was oriented toward a single optimizer, making comparison difficult. We introduced a dictionary of optimizers and a loop to iterate over them, running the same simulation:

1. Define a dictionary with various optimizers.
2. Iterate over each and run the simulation.
3. Compare results (final energy, iterations, total time).

The results showed that **RMSProp** performed excellently, combining speed and stability in convergence. This can be attributed to its ability to adapt the learning rate per parameter, preventing oscillations and improving the descent toward the energy minimum. Additionally, the total execution time did not significantly differ among optimizers, so the decisive factor was the quality of convergence.

3.12 Limitations and Mitigation Measures

Among the limitations of this approach are:

- **Scalability:** As the system grows in the number of electrons and orbitals, the complexity of Hamiltonian construction and the excitation space increases exponentially.
- **Quantum Noise and Errors:** On real devices, noise affects measurement fidelity. Our work, primarily simulation-oriented, plans to integrate mitigation techniques in future studies.

- **Ansatz Choice:** Although the adaptive ansatz helps, there is no guarantee that the excitation selection is optimal. Future work might explore more complex heuristics.

To mitigate these issues, we opted for reduced basis sets, strategies such as re-initializing the optimizer when increasing the parameter space, and verifying convergence through multiple criteria (energetic and geometric).

CHAPTER 4

Results

This chapter should encompass your data analysis and findings. Additionally, include relevant tables, figures, and citations to support your results and interpretations. Here is a suggested list of topics to discuss:

4.1 Experiments and Tests

Describe the experiments conducted to assess the performance of your project. Explain how you collected and processed the data.

4.2 Data Visualization

Create visual representations of the results (e.g., scatter plots, bar charts). Interpret the visualizations and relate them to the research questions.

4.3 Limitations

Acknowledge any limitations in the data or analysis. Explain how these limitations may have influenced the results.

CHAPTER 5

Sustainability Analysis and Ethical Implications

Starting from the academic year 2023-24, the TFG regulations of ETSETB require the inclusion of a sustainability report in the project's documentation. This analysis involves an assessment of environmental, social, and economic impacts, as well as potential ethical implications resulting from the completion of the TFG. In the case where the TFG involves a product/service/system/building, etc., that could be implemented, the analysis should also address the impacts that the proposal would have during the various stages of its lifecycle.

Detailed instructions on what the sustainability report should contain and how to prepare it can be found on the ATENEA platform.

IMPORTANT: Please note that the previous chapter on "Project Budget" is now integrated into the sustainability analysis, specifically in the cells "Economic Cell/Development of BT" and "Economic Cell/Project Execution".

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

- Summarize the main results of your work.
- Discuss the degree of achievement in relation to the objectives set at the beginning of the work.
- Highlight the contributions of your work to the field of study.

6.2 Future Directions

- Identify areas for future research or development based on your work.
- Discuss possible ways to expand or improve the project.
- Consider questions that remained unanswered and opportunities for future exploration.

APPENDIX A

Logic Gates

A.1 Simple Logic Gates

Below are detailed the simple logic gates essential for constructing more complex quantum algorithms:

X Gate (Pauli-X) The **Pauli-X** gate is the quantum analog of the classical NOT gate. It performs a bit flip on the qubit, transforming the state $|x\rangle$ into $|\neg x\rangle$.

Representative Matrix

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

Effect on Basis States

- $X|0\rangle = |1\rangle$
- $X|1\rangle = |0\rangle$

Y Gate (Pauli-Y) The **Pauli-Y** gate performs a rotation of π around the y-axis. It transforms the state $|x\rangle$ into $i(-1)^x|\neg x\rangle$.

Representative Matrix

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Effect on Basis States

- $Y|0\rangle = i|1\rangle$
- $Y|1\rangle = -i|0\rangle$

Z Gate (Pauli-Z) The **Pauli-Z** gate is known as the phase inversion gate. It transforms the state $|x\rangle$ into $(-1)^x|x\rangle$.

Representative Matrix

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Effect on Basis States

- $Z|0\rangle = |0\rangle$
- $Z|1\rangle = -|1\rangle$

Hadamard Gate (H) The **Hadamard** gate creates an equal superposition of the computational basis states. It transforms the state $|x\rangle$ into $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^x |1\rangle)$.

Representative Matrix

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Effect on Basis States

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

A.2 Multi-Qubit Logic Gates

Controlled-NOT Gate (CNOT) The **CNOT** or **Controlled-X** gate is a two-qubit gate that flips the second qubit (target) if and only if the first qubit (control) is in the state $|1\rangle$. It transforms the state $|x, y\rangle$ into $|x, x \oplus y\rangle$, where $\oplus$ denotes the XOR operation.

Representative Matrix

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

Effect on Basis States

- $\text{CNOT}|00\rangle = |00\rangle$
- $\text{CNOT}|01\rangle = |01\rangle$
- $\text{CNOT}|10\rangle = |11\rangle$
- $\text{CNOT}|11\rangle = |10\rangle$

Single Excitation Gate (*SingleExcitation*) This gate performs a rotation in the two-dimensional subspace $\{|01\rangle, |10\rangle\}$. It transforms the state $|10\rangle$ into $\cos\left(\frac{\phi}{2}\right)|10\rangle - \sin\left(\frac{\phi}{2}\right)|01\rangle$.

Representative Matrix

$$U(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\frac{\phi}{2}\right) & -\sin\left(\frac{\phi}{2}\right) & 0 \\ 0 & \sin\left(\frac{\phi}{2}\right) & \cos\left(\frac{\phi}{2}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Effect on Basis States

It affects the subspace $\{|01\rangle, |10\rangle\}$, performing a rotation parameterized by ϕ.

Double Excitation Gate (*DoubleExcitation*) This gate performs a rotation in the subspace of states $\{|0011\rangle, |1100\rangle\}$. It specifically affects these states, leaving the others unchanged.

Representative Matrix

$$U(\phi) = \begin{pmatrix} I_{12} & 0 & 0 \\ 0 & \begin{pmatrix} \cos\left(\frac{\phi}{2}\right) & -\sin\left(\frac{\phi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right) & \cos\left(\frac{\phi}{2}\right) \end{pmatrix} & 0 \\ 0 & 0 & I_2 \end{pmatrix}$$

Effect on Basis States

It performs a rotation parameterized by ϕ in the subspace $\{|0011\rangle, |1100\rangle\}$.

These gates are implemented in PennyLane as `qml.SingleExcitation` and `qml.DoubleExcitation`, and are essential in quantum chemistry algorithms such as the *Unitary Coupled-Cluster Singles and Doubles* (UCCSD).