

UNIVERSITY OF OKLAHOMA  
GRADUATE COLLEGE

EVALUATING THE ROLE OF QUANTUM ALGORITHMS IN  
SUPERVISED MACHINE LEARNING

A THESIS

SUBMITTED TO THE GRADUATE FACULTY  
in partial fulfillment of the requirements for the  
Degree of  
MASTER OF SCIENCE



By

PABLO GUTIÉRREZ FÉLIX  
Norman, Oklahoma

2024

EVALUATING THE ROLE OF QUANTUM ALGORITHMS IN  
SUPERVISED MACHINE LEARNING

A THESIS APPROVED FOR THE  
MICHAEL F. PRICE COLLEGE OF BUSINESS

BY THE COMMITTEE CONSISTING OF

Dr. Naveen Kumar, Chair

Dr. Radhika Santhanam

Dr. Rajendra Singh

© Copyright by PABLO GUTÉRREZ FÉLIX 2024

All Rights Reserved.

# Table of Contents

## Chapter 1: Introduction

1.1 Background.....	1
1.2 Objectives and Research Question.....	4
1.3 Key Contributions.....	5

## Chapter 2: Literature Review

2.1 Quantum Enhancement in Supervised Learning.....	7
2.2 Quantum Enhancement in Unsupervised Learning .....	10
2.3 Summary.....	11

## Chapter 3: Building Blocks of Quantum Machine Learning

3.1 Linear Algebra .....	13
3.1.1 Vectors .....	13
3.1.1.1 Vector Addition .....	15
3.1.1.2 Vector Multiplication .....	15
3.1.1.3 Vectors in Quantum Computing.....	17
3.1.2 Linear Independence and Vector Basis.....	18
3.1.2.1 Vector Basis in Quantum Computing.....	20
3.1.3 Linear Transformations.....	21
3.1.4 Matrixes.....	24
3.1.4.1 Matrixes in Quantum Computing.....	26
3.1.5 Dot Product .....	26
3.1.5.1 Dot Product in Quantum Computing.....	28

3.1.6 Eigenvectors and Eigenvalues.....	29
3.1.6.1 Eigenvectors and Eigenvalues in Quantum Computing.....	32
3.1.7 Summary .....	32
3.2 Quantum Physics .....	33
3.2.1 The Language of Quantum Mechanics .....	33
3.2.2 Bra-ket Notation.....	36
3.2.3 Entanglement.....	38
3.2.4 Observables .....	40
3.2.5 The Uncertainty Principle .....	41
3.2.6 Probability and the Bloch Sphere.....	43
3.2.7 Summary .....	46
3.3 Quantum Programming with Qiskit.....	46
3.3.1 Qiskit Introduction .....	47
3.3.2 Qiskit Basics.....	47
3.3.3 Quantum Gates.....	51
3.3.3.1 X Gate.....	52
3.3.3.2 Z Gate .....	54
3.3.3.3 H Gate.....	55
3.3.4 Qubit Entanglement.....	58
3.3.4.1 CNOT Gate.....	58
3.3.4.2 CZ Gate.....	60
3.3.4.3 Entangling Qubits .....	61
<b>Chapter 4: Experiments and Results</b>	
4.1 QKSVM Theory.....	64

4.1.1 SVM .....	64
4.1.2 Kernels .....	65
4.1.3 Quantum Kernels.....	68
4.2 Experiments .....	69
4.2.1 Methodology .....	70
4.2.1.1 Discrete Logarithm Problem Dataset .....	71
4.2.1.2 Experimental Design .....	72
4.3 Results.....	78

## **Chapter 5: Discussion and Conclusion**

5.1 Discussion.....	83
5.2 Conclusion .....	85
5.2.1 Challenges and Limitations.....	87
5.2.2 Future Work and Recommendations.....	87
5.2.3 Key Learnings .....	89
Acknowledgments.....	90
References.....	91
Appendix.....	95
A.1 Determinants .....	95
A.2 Determinants and eigenvectors .....	96
A.3 Eigenbasis.....	97
A.4 Imaginary and complex numbers .....	98
A.5 Complex Conjugate.....	99

## Abstract

Quantum computing (QC) has emerged as a disruptive technology, promising exponential speedups for certain computational problems. At the same time, machine learning (ML) continues its transformative journey across scientific domains with its ability to locate patterns within data. The intersection of both disciplines, Quantum Machine Learning (QML), offers efficiency and optimization speedups in certain learning tasks, captivating the interest of both researchers and business leaders. The power of QML lies in its ability to harness quantum properties, such as superposition and entanglement, to solve complex business problems more efficiently than classical algorithms. These quantum properties can be applied to ML in numerous ways, depending on the complexity of the problem.

This thesis delves into Quantum Kernel Support Vector Machines (QKSVMs), a specific QML technique that leverages quantum computing for supervised learning, particularly support vector machines (SVMs). The reason behind focusing on this technique specifically lies in the fact that quantum algorithms can map data points into a higher dimensional feature space. Through this process, quantum kernels aim to locate atypical patterns in data for classification tasks. The discrete logarithm problem (DLP) is a common mathematical problem widely used in cryptography due to its one-way-function nature and resistance to classical algorithms. It has been argued that the case of DLP is a scenario where quantum-inspired algorithms can enhance their classical counterparts in terms of accuracy and other machine learning performance metrics. Utilizing this mathematical problem as the distribution within a dataset could potentially prove quantum advantage, suggesting that using quantum algorithms in certain situations may be beneficial.

This thesis presents empirical evidence of quantum algorithms' performance enhancements within the DLP framework. This work concludes that in specific scenarios like DLP-distributed datasets, even near-term quantum algorithms operating with fewer qubits can have an advantage over existing algorithms. Furthermore, the findings from this work offer valuable insights to researchers and business leaders interested in investing in the design and implementation of QML models in scenarios beyond DLP, where quantum algorithms have an advantage over classical algorithms.

**Keywords:** quantum computing, kernels, support vector machines, discrete logarithm problem.

# Chapter 1: Introduction

Data science and information systems are constantly evolving and developing. With the emergence of Quantum Computing (QC) in recent years, it is imperative to understand its key aspects and evaluate its benefits and limitations. This thesis delves into the foundational concepts of QC and the benefits it can provide to Machine Learning, a subset of data science that has sparked interest in recent years (Cerezo et al., 2022). Understanding the intricacies of the convergence between both fields can have tremendous advantages for industry decision-makers and future researchers as quantum computers become more accessible to companies and research institutes worldwide in the coming years (Troyer, 2023).

## 1.1 Background

Quantum Computing is the field of computation that utilizes quantum mechanics principles to perform computational operations on data. Quantum computers, the devices that allow scientists to run quantum computations, are thought to have an advantage over classical computers for certain tasks (Troyer, 2023). On the other hand, ML focuses on finding patterns of data, which can be used for predictions, decision-making, or gaining insights about the data distribution in question (Baheti, 2022). Some business applications of ML are used to tackle classification tasks, like fraud detection or sentiment analysis (Mercha & Benbrahim, 2023) (Shehnepoor et al., 2023).

In recent years, the convergence of QC and ML has been of great interest to researchers, giving rise to a new realm: Quantum Machine Learning (QML). This field looks to devise learning algorithms more efficiently than their classical counterparts in certain tasks (Pushpak & Jain, 2021). Classical machine learning is about finding patterns in data to predict future patterns, and QML has the potential to assist in locating those specific patterns that classical machine learning cannot grasp, like discrete logarithmic problems (DLP) (Xiao et al., 2023). The technology that powers QML is of quantum nature, meaning that it takes advantage of quantum mechanics to apply quantum properties to its computations.

QC is different from classical computing in its mere essence since it uses qubits (quantum bits) instead of classical bits (Pushpak & Jain, 2021). While bits can only have two values (0 or 1),



qubits can have a superposition of both states at the same time. Moreover, qubits exhibit the property of entanglement, where the states of two qubits depend on each other, enabling parallel processing of such qubits. These properties are thought to provide exponential computational gains when comparing quantum to classical computing, as will be exposed in the literature review (Daley et al., 2022).

QML is being cataloged as a revolutionary technology in almost every existing industry (Troyer, 2023). For finance, it is argued that quantum computing algorithms can provide a 21% increase in returns when used in finance portfolio optimization (Shipilov & Furr, 2022). It is also suggested that this technology can assist supply chain activities, enabling up to 60% cost reduction in vehicle routing optimization (Shipilov & Furr, 2022). Also, QML has played an important role in fraud detection tasks, particularly through the use of a hybrid quantum-classical neural network that provided a 1.66% increase in accuracy (Shipilov & Furr, 2022).

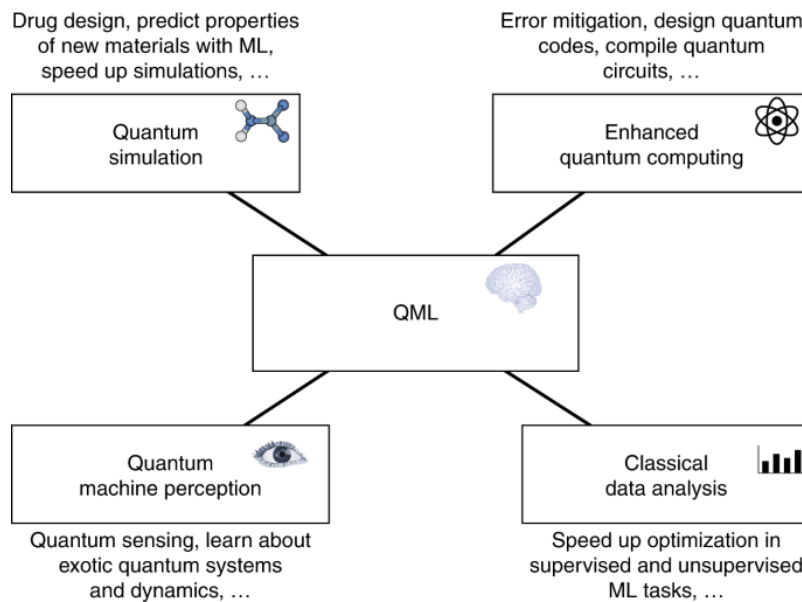
Quantum computing seems to be able to enhance business intelligence (BI), a field in charge of utilizing the necessary methods to transform raw data into meaningful insights (Heiner, 2022). This is due to the increased amount of data that quantum algorithms allow to manipulate simultaneously when paired with a classical algorithm. Therefore, quantum computers would allow businesses to calculate complex functions that use several variables, while classical algorithms are left with the simpler ones. Besides, BI processes will benefit from quantum technologies because the tools will, therefore, be dynamic and constantly updated (Heiner, 2022). Bova et al. (2023) suggest that it may be beneficial for companies to adopt quantum computing technology over classical, even if quantum advantage is not achieved. However, this quantum advantage has yet to be empirically proven for specific business cases. In search of that quantum advantage, Zhang and Ni (2020) differentiate between basic quantum machine learning algorithms and application algorithms for quantum machine learning. The first group is formed by Grover's swap test, phase estimation, and Harrow-Hassidim-Lloyd (HHL) algorithms, which operate with qubits to obtain an exponential or quadratic computational advantage over classical algorithms. These basic algorithms serve as the building blocks of the so-called "application algorithms."

ML is comprised of several different sections: supervised, unsupervised, and reinforcement learning. The applications of quantum computing to each of them open the door to several opportunities for QML enhancement (Pushpak & Jain, 2021). Figure 1.1 below shows some of

these opportunities, like simulation of quantum systems, enhancement in quantum computing techniques, quantum machine perception, and improvements in classical data analysis through quantum computing speedups (Cerezo et al., 2022).

This thesis focuses on supervised learning that utilizes kernels for classification. In ML, a classical kernel is a method of using a linear classifier to solve a non-linear problem, like separating the data of a dataset into parts depending on specific data features (Krunic et al., 2022). These kernels are utilized ML for classification purposes, a technique known as Kernel Support Vector Machines (KSVM). This thesis will examine the convergence of QML and KSVM. More specifically, the focus is on finding specific scenarios in which a quantum-KSVM (Q-KSVM) has an edge over classical KSVM.

**Figure 1.1: Practical implications of QML (Cerezo et al., 2022)**



This study allows practitioners and data science researchers to understand the benefits and limitations of this quantum classification task. The reason behind focusing on this specific algorithm lies in the fact that the intersection of QC and ML opens the door for many quantum implementations of existing algorithms, each with a unique structure and functioning. The scope of this thesis is restricted to KQSVM, although the conclusions drawn may prove to be helpful in the quantum “translation” of other classical algorithms. Furthermore, this thesis aims to contribute

to the discourse surrounding QML, unraveling the theoretical foundations and practical applications of QML to the realm of data science and information systems.

## 1.2 Objectives and Research Question

Classification tasks are a widely used technique in data science and information systems, but they can perform poorly when dealing with linear models (Ganguly, 2021, p. 271). Therefore, it is sometimes required to use functions that tackle non-linearly separable datasets, like kernel functions. The ability of quantum circuits to map data into a higher dimensional space, like the Hilbert space of the Bloch sphere, allows one to classify data in detail. Nonetheless, it is still to be determined what type of characteristics in real-world datasets will require such a classification technique.

As will be explored in Chapter 2, Q-KSVM has been proposed for solving classification tasks in different disciplines, with quantum success in only a handful of them (Gentinetta et al., 2024). It is fundamental to understand that current quantum computers are “noisy” due to their early stage of development. In this context, “noise” speaks about the physical disturbances that cause quantum information to degrade during a quantum computational process (Munro, 2018), affecting experimental outcomes. To address this limitation, computationally expensive quantum simulators allow researchers to understand the behavior of quantum systems with few qubits (Johnson et al., 2014).

The objectives of this thesis can be divided into two main objectives, given the theoretical and experimental approach that is being conducted throughout the exploration.

The first objective is to conduct experiments, share key results, and provide insights concerning the performance of Q-KSVM compared to KQSVM. These empirical explorations are performed on several types of datasets with different quantum circuit combinations and by adjusting these quantum circuits’ characteristics. Through these experiments, valuable insights will be gained about those scenarios in which ML can benefit from the use of QC, as well as those scenarios where QC is not necessary. The insights derived from said results are especially relevant to the field of cybersecurity and cryptography. As discussed in Chapter 4, the experiments will operate with datasets related to the DLP. Besides, although there is a theoretical approach to KQSVM applied to DLP-distributed datasets (Liu et al., 2021), a notable gap exists in the literature regarding

the practical application of KQSVM in scenarios where a substantial quantum advantage in performance is demonstrated.

The second objective is to investigate and explain the intricacies of quantum mechanics and quantum computing to business leaders. More specifically, this objective includes the identification of key concepts within quantum computing and machine learning that serve as building blocks for further exploration in QML. Furthermore, this thesis summarizes the key steps required to use *qiskit*, a Python-based library developed by IBM Quantum to access and interact with quantum devices.

This thesis will focus on the following research question: *In the context of supervised learning, and more specifically SVMs, under what scenarios do Quantum Kernel Support Vector Machines perform better than classical Kernel Support Vector Machines?*

### **1.3 Key Contributions**

The outcome of this research is of great support to the data science field and any scientific field where QML can be applied.

This research provides valuable insights into supervised classification methods that utilize kernels. The empirical results obtained through this objective hold the potential to support future research in QML, particularly through the discussion of those patterns or functions that are challenging for classical computing devices to detect. These insights contribute to the advancement of QML research and offer guidance for the development of quantum-based machine learning algorithms.

In addition, this thesis contributes to expanding quantum computing literature in data science and information systems literature. Because QML is a domain that has yet to be fully explored in the IS domain, the role of QC in Information Systems is still unknown. Therefore, the beginning chapters of this thesis aim to provide the foundational theoretical framework required to understand the role of QC in Data Science and Information Systems. Moreover, a practical demonstration illustrating the implementation of these theoretical concepts will be provided in detail, facilitating access for anyone interested in conducting quantum experiments.

Besides scientific purposes, the thesis's outcome intends to impact the academic and business community, which can benefit from the analysis and discussion of this exploration. Researchers in

academia and industry can use the experimental findings of this thesis to advance the development of quantum machine learning algorithms and methodologies. This thesis aims to provide thought-provoking experiments on how to implement Q-KSVM in different scenarios, which equips stakeholders with the necessary tools to deploy their data and frameworks onto quantum computing devices or simulators, facilitating experiments that could potentially benefit their organizations.

In addition, those sectors of industry that focus on ML solutions or that benefit from classification tasks in their operations will significantly benefit from the conclusions drawn from this thesis' experiments. While the thesis primarily focuses on data science, it delves into identifying problems where quantum computing can enhance existing solutions, offering a new perspective that can be extrapolated across numerous industry applications. Through this discussion, industries can strategically utilize these insights to optimize their operations and drive innovation in their respective domains.

## Chapter 2: Literature Review

Although QML is a relatively disruptive field, existing research has made considerable progress in its development so far. This chapter summarizes the literature and business applications concerning QML. Moreover, it analyzes how quantum computing has revolutionized two principal areas of ML, supervised and unsupervised learning, while discussing the impact they had on the scientific domains that have benefited from the adoption of this novel technology.

### 2.1 Quantum Enhancement in Supervised Learning

Zhang and Ni (2020) provide theoretical insights regarding the power of supervised quantum machine learning algorithms. They particularly deepen into Quantum Support Vector Machine for classification problems and the mathematical explanation behind the quadratic and exponential speedup awarded by both versions of the QSVM algorithm, which uses Grover's basic algorithm as a subroutine. In another paper, Zhang and Ni (2021) dive into the design of quantum neural networks with a particular approach to classification tasks. Their experiment shows that their quantum neural network design could predict new types of data based on the data points provided. Besides, by using this quantum neuron design, they were able to perform a classification task on real data. The model could classify the new inputs into two groups, based on twelve input features, by applying the logistic regression method. The results portray that the model needed twenty iterations for the predictions to be accurate to the actual results. Jiang et al. (2021) provide a deep understanding of this matter in a case study, exhibiting a full workflow on how to implement neural networks into quantum circuits.

The applications of supervised QML can also support several scientific fields other than Machine Learning and Data Science. For example, it takes ten hours to model one single cell of a bacteria called "mycoplasma genitalium," the simplest form of organism, genetically speaking (Byrum, 2022). It could take more than a human lifespan to model more complex living organisms like humans through classical supercomputers, making a compelling case in favor of quantum technology's computational power (Byrum, 2022). Similarly, Baiardi et al. (2023) make a theoretical approach weighing the possible problems that quantum-supervised machine learning

can tackle in the field of molecular biology by speeding up the training process and predicting more complex relationships between the variables in question. This can be applied to genome sequencing, a process that is truly challenging when approached classically but can be enhanced using quantum technology. Nonetheless, quantum advantage can be witnessed when manipulating large datasets because the previous processing required to utilize a quantum algorithm would nullify the advantage provided by such.

Besides biology, pharmacology can also benefit from quantum machine-learning techniques. Ganesh (2021) utilized a modified Quantum Support Vector Machine algorithm to identify compounds that could inhibit a promising target associated with Alzheimer's disease, MSUT-2. This supervised model based on quantum technology was trained with dozens of chemical features using qiskit, which allowed the authors to predict five possible inhibitors for MSUT-2. To validate these results, this methodology was used for other known chemical compounds and pharmaceuticals, providing the same positive result.

In the field of quantum chemistry, Anatole von Lilienfeld (2018) discusses these applications within chemical compound spaces. They conclude that QML will allow the scientific community to “rectify some of the issues which plague many of the common approximations made within conventional compute campaigns, such as the choice of density functionals,” placing special emphasis on supervised algorithms. They argue that supervised models, when applied to quantum systems, can allow quantum chemists to predict the quantum properties of chemical compounds.

Bernal et al. (2022) also provide a glimpse into the current perspective of quantum computing in chemical engineering. This article suggests that quantum computing will be able to assist in the process of measuring kinetic and thermodynamic properties, particularly those that require high accuracy for small molecules, like analyses of thermochemistry for gas-phase processes such as combustion, pyrolysis, and atmospheric chemistry. The authors emphasize the importance of linear regression methods within chemical engineering to describe the relationships between response and explanatory variables. They also suggest that quantum algorithms like HHL can assist in this process. In another paper, Rupp (2015) utilizes a ridge kernel regression model to predict the atomization energies of organic particles. Although this study does not focus explicitly on QML, it uses supervised learning methods to predict the atomization energies of small organic molecules. By using Gaussian and Laplacian kernel algorithms, the author provides insights into the

conjunction between computational quantum chemistry and its benefits for this specific prediction task, which could potentially be enhanced by quantum computing technology.

Supervised quantum machine learning has been widely used in finance research. Orus et al. (2019) approached this subfield by reviewing the potential applications of quantum computing to financial problems. Optimization tasks are extremely challenging for classical computers, which is fundamental for portfolio optimization. The author argues in favor of implementing quantum technology to optimize arbitrage opportunities and credit scoring. Besides, they make a case for the utilization of supervised QML techniques for specific financial tasks like data classification in credit scoring, regression algorithms for supply chains, pricing of financial derivatives, and risk analysis. Hybrid classification algorithms have been reviewed as well, particularly for finance datasets (Hellstern, 2021). This experiment confronted a hybrid quantum neural network, where the final layer of an all-classical neural network is substituted with a quantum network against an all-classical neural network. The outcome of this study suggests a slightly better performance of hybrid algorithms when compared to all-classical algorithms, given that the number of neurons in the last layer was tuned so that both networks had a similar number of parameters.

Sanz-Fernandez et al. (2021) created a quantum-enhanced Monte Carlo algorithm for price value forecasting. By executing this algorithm together with a Gordon-Shapiro model for asset value estimation, the authors were able to create a framework that forecasts the value of assets while considering the uncertainty of specific financial variables. The findings suggest that, for the same computational cost, the quantum model produces a similar result to that obtained through classical Monte Carlo while diminishing the statistical error by a quadratic factor.

Continuing within the finance domain, fraud detection in bank payments can benefit from quantum classification techniques. Innan (2023) compares four different types of quantum classification algorithms (Quantum Support Vector Classifier, Variational Quantum Classifier, Quantum Neural Network (QNN), and Sampler QNN) in the task of categorizing bank payments as fraudulent or not. The dataset contained ten features, with one of them being a binary for “fraud” or “not fraud.” The QSVC model scored a 0.98 precision score for both fraud and non-fraud cases, being the best of the four quantum algorithms. The findings suggest that this algorithm, within supervised models, promises to be the best alternative for classification tasks.



Quantum classification methods have a history in the manufacturing fields as well, with Guijo's study being the first one to show quantum algorithms outperform classical ones in terms of computer vision for product defect detection (Guijo et al., n.d.). The objective of this experiment was to train several models (quantum and classical) with labeled images as "with defect" and "without defect" so that the model can apply that same pattern to new images. The authors applied a Quantum Support Vector Machine and the QBoost algorithm and compared them to their classical counterparts: A Support Vector Machine and an AdaBoost algorithm. The experiment concluded that even after performing dimensionality reduction in all cases, the quantum algorithms provided more accurate classification results than their classical counterparts. This was also performed for the QBoost algorithm in the same amount of training time.

## **2.2 Quantum Enhancement in Unsupervised Learning**

Because supervised solutions are easier to implement, several publications suggest that the ongoing research should focus on the applications of quantum unsupervised models (Perdomo-Ortiz et al., 2018). Besides, the study suggests focusing on hybrid models: those that conjugate classical and quantum algorithms. These seem to be the research areas where scientists are struggling, and Perdomo-Ortiz claims that quantum technology may be particularly advantageous for those cases. Zhang and Ni (2020) expand on the theory that supports the quantum version of the k-means clustering algorithm, typically used for tasks that concern grouping sets of unlabeled data. Quantum k-means utilizes the swap-test basic algorithm to calculate the minimum distance between data points and their respective clusters. In this paper, two more application algorithms are explored, which centered around dimensionality reduction, a subfield within unsupervised learning that aims to reduce the number of variables within the data while conserving the fundamental variable relationships.

In their perspective article, Bernal et al. (2022) argue that chemical engineering can benefit from quantum unsupervised learning, mainly from dimensionality reduction techniques. This would allow scientists to speed up processes of "monitoring, chemometrics, and optimal control for energy management." Prior literature reveals that QML can also support the domain of oncology and cancer research. Jain et al. (2020) utilize a combination of classical and quantum algorithms to classify two subtypes of non-small cell lung cancer, known as Adenocarcinoma and Squamous Cell Carcinoma. The dataset was optimized by using a classical machine learning technique known

as “feature selection” so that the variables could fit into a D-Wave Quantum Processing Unit. The results of this experiment showed a 95.24% average score of prediction accuracy, matching the scores of an all-classical configuration. This suggests that classical-quantum configurations can be as beneficial as all-classical configurations.

Quantum computing Boltzmann Machines can also be utilized with other types of datasets, for example, in presidential election modeling (Henderson et al., 2019). This research utilizes D-Wave’s adiabatic quantum devices to train a set of Boltzmann machines to find correlated systems within data from the previous 11 US presidential elections. The results describe that this quantum machine learning model forecasted accurate results comparable to the current best election forecasting methods.

Quantum-enhanced unsupervised models are promising in finance applications as well. Palmer’s study depicts how a hybrid quantum annealing algorithm could be used to optimize financial portfolios (Palmer et al., 2021). The optimization of these large portfolios (with real data) took place in minutes per job. Therefore, portfolio optimization through quantum computing is a viable option. The authors suggest that using unsupervised methods together with this optimization technique can be beneficial when simulating similar financial indexes. Shifting to a different domain within finance, quantum unsupervised methods have exhibited enhanced efficiency when locating fraud detection in credit card datasets (Kyriienko & Magnusson, 2022). This study compares classical methods (SVM for supervised and OC-SVM for unsupervised) to their quantum version (QSVM). The results show that in supervised classification, as more features (characteristics or attributes of the data) are considered, the quantum algorithm is more accurate, while the classical algorithm performs worse due to overfitting. On the other hand, the unsupervised classification experiment concludes that quantum kernels also improve as features increase, reaching 15% more accuracy than classical kernels.

## **2.3 Summary**

Prior literature has shown the initial steps of domains where quantum algorithms may provide an edge over their classical counterparts, particularly for supervised applications. Although quantum-enhanced unsupervised models have not been as widely researched as quantum-enhanced supervised ones, previous publications suggest positive clues of beneficial applications regarding

quantum technology in this specific subtype of Machine Learning. Regardless of the field and domain of the publications reviewed in this section, all authors concluded that as quantum hardware and access continue to evolve, scientists will have a better chance to prove the potential of quantum algorithms compared to current classical algorithms.

Circling back to supervised learning, particularly kernelized SVMs, it has been argued that quantum kernels can exploit quantum feature spaces to map data into a quantum Hilbert space (Havlíček et al., 2019; Schuld & Killoran, 2019). Within this realm, this thesis will particularly examine the idea portrayed by Liu, Arunchalam, and Temme (2021) in their paper “A rigorous and robust quantum speed-up in supervised machine learning.” Although it is not a practical approach, they establish a concept class where quantum kernels have a better performance than classical kernels for datasets with a distribution following the DLP.

## Chapter 3: Building Blocks of Quantum Machine Learning

To understand QML, it is imperative to rely upon quantum mechanics, which is why this chapter focuses on covering the prior knowledge required to understand this domain. Overall, the process of acquiring this background can be separated into two steps: Linear Algebra and Quantum Physics. Section 3.1 covers the basic concepts of linear algebra, which will aid in comprehending the quantum mechanics concepts covered in Section 3.2 that apply to quantum computing. Finally, Section 3.3 will cover the fundamentals of coding in the realm of quantum computing.

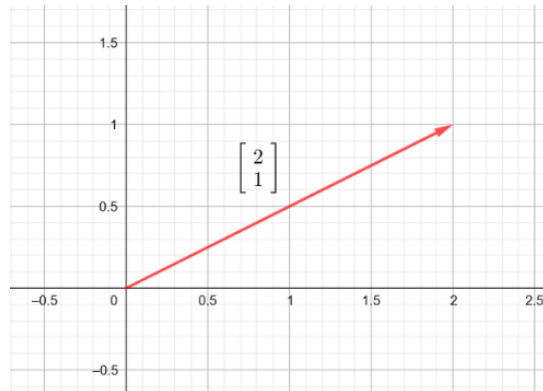
### 3.1 Linear Algebra

In quantum computing, key concepts are described and understood through linear algebra, which is considered the language of quantum computing. Under the broad umbrella of linear algebra, several underlying topics need to be comprehended sequentially. Therefore, this section covers the basic building blocks of linear algebra: vectors, vector bases, linear transformations, inner products, eigenvectors, and eigenvalues.

#### 3.1.1 Vectors

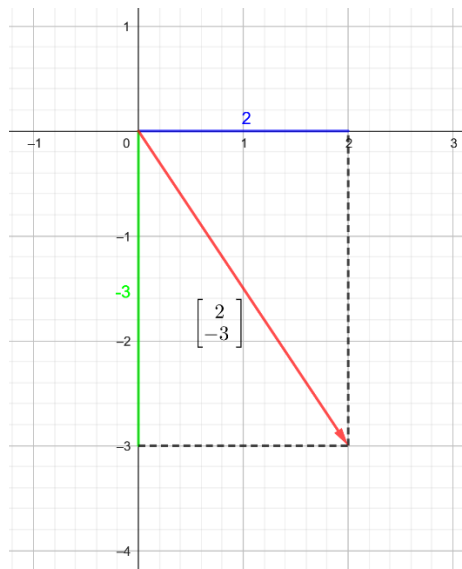
The fundamental unit of information within a quantum computer is known as a qubit (a quantum bit), and a qubit is represented through a vector. Vectors are the building blocks of linear algebra and are mathematical concepts used to describe objects with magnitude and direction. For example, a car travels at 50 km/h towards the east. This real-life example can be represented through a vector because it gives two pieces of information: the magnitude (50 km/h) and the direction in which the car is traveling (east). Although vectors can be used to describe physical concepts, like the velocity of a car, they can be understood as purely mathematical concepts. Figure 3.1 below shows an arrow pointing toward a specific direction within a coordinate system, which is a graphical representation of a vector.

**Figure 3.1: Vector Within a Coordinate System.**



To represent it in a 2-dimensional space, the information it conveys can be broken down into its X-axis and Y-axis components. As Figure 3.2 portrays, the original vector (in red) can be broken down into its X coordinate (in blue) and Y coordinate (green). This permits the tracking of the “path” needed to travel to get to the point to which the vector is pointing. Although there are numerous ways to notate vectors, the most common is to represent them with an arrow on top of the variable to which they are assigned ( $\vec{a}$ ).

**Figure 3.2: X and Y Components of a Vector**



Vectors are expressed through squared brackets, where the number on top represents the X-coordinate, and the number below represents the Y-coordinate. For the vector represented above, one of the possible representations is as depicted in Equation 3.1:

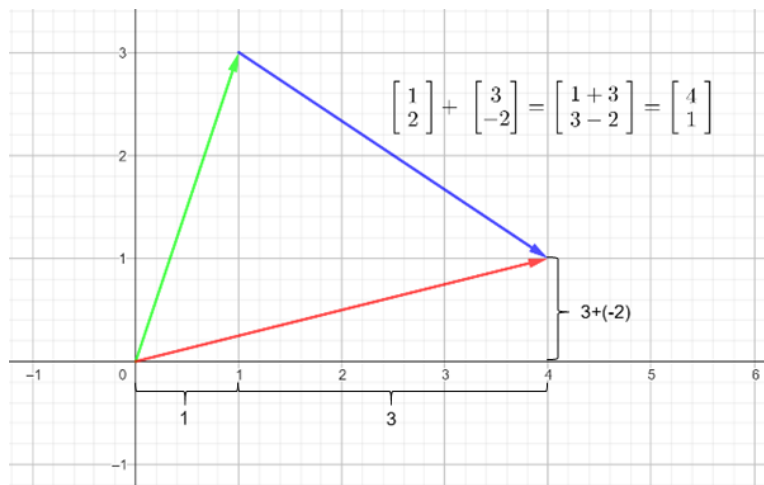
$$\vec{a} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.1}$$

### 3.1.1.1 Vector Addition

Next, this subsection reviews vector addition and multiplication since the other operations (like subtraction and division) are derived from the first two. When adding vectors, the second vector's arrowhead is positioned at the endpoint of the first vector's arrow, and then a new arrow (the resultant vector) is drawn from the origin of the plane (the point where the X and Y axes intersect) to the endpoint of the second vector, as illustrated in Figure 3.3. Note that Figure 3.3 is a consequence of the theoretical description in Equation 3.2:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix} \tag{3.2}$$

**Figure 3.3: Vector Addition**



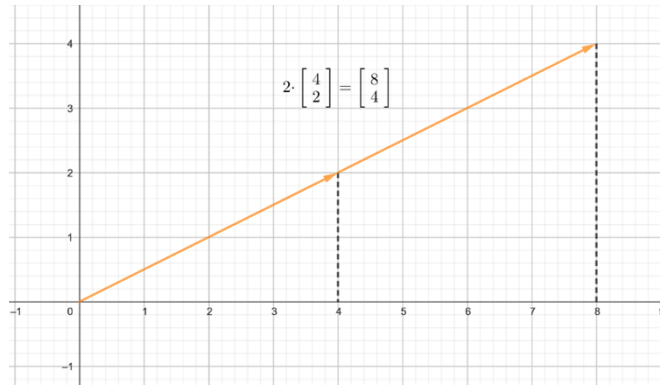
### 3.1.1.2 Vector Multiplication

Vector multiplication either increases or decreases the magnitude of the given vector by a specified scalar factor. Vector multiplication is often referred to as "scaling" because a scalar is a magnitude without direction, which multiplies the original vector. To find the resultant vector from the multiplication, trace the arrowhead starting at the origin, following the formula (see Equation 3.3 and Figure 3.4):

$$2 \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

(3.3)

**Figure 3.4: Vector Multiplication**

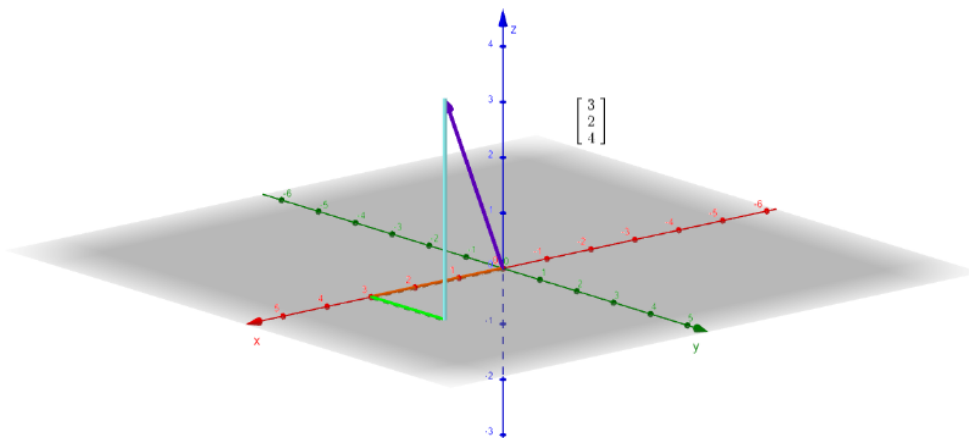


Vectors can also be represented in a 3-dimensional space. To represent this, a new coordinate (z) is introduced to the previously mentioned notation, which is placed perpendicular to the “y” coordinate. Figure 3.5 below shows the three coordinates (x, y, and z) components of a 3-dimensional vector derived from Equation 3.4:

$$\vec{b} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

(3.4)

**Figure 3.5: 3-Dimensional Vector**



### 3.1.1.3 Vectors in Quantum Computing

All this theoretical knowledge concerning vectors applies to quantum computing because this discipline can bring classical bits (which can only be either 0 or 1) into the quantum world, where elements can have several values at the same time. The result is the qubit, which can be in a superposition of 0 and 1. When a qubit is measured, the results can only be either 0 or 1, but the probability of getting a certain result is given by the state of the qubit, known as the “quantum state.” For a single qubit, the X component of the vector will hold the coefficient of the qubit’s probability of collapsing into 0, and the Y component will hold the coefficient for the probability of collapsing into 1.

Quantum physics uses bra-ket notation ( $|\psi\rangle$ ) to portray the values of the vector. This notation is formed by three different symbols: “|,”  $\psi$ ,” and “)” (Schwabl, 2007). First, the pipe “|” is used to indicate one of the edges, in this case, the start of a quantum state representation. The Greek letter “ $\psi$ ” (“psi”) is used as a label to portray the state of the quantum system itself. The symbol “)” closes the quantum state vector notation by matching the “|” (find a more detailed explanation of bra-ket notation in Section 3.2.2). The state of the qubit ( $\psi$ ), which has a probability  $\alpha$  of measuring 0 and a probability  $\beta$  of measuring 1, can be represented by the vector portrayed in Equation 3.5:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{3.5}$$

It is important to understand that when talking about the *state* of a qubit, the context refers to its vector expression. The probability of a qubit being 0 or 1 when it is measured is derived from this state, but the probability is not the state itself. Therefore, two different qubit states can have the same probability of collapsing into 0 or 1 when measured. When performing operations with a qubit, the qubit’s state changes (the coefficient holding the probabilities of the qubit collapsing into 0 or 1). Before collapsing, the qubit is both 0 and 1 at the same time, providing exponential computational advantage compared to classical computers. This means that, before collapsing, one qubit has the computing power of two bits, two qubits have the power of four bits, three qubits have the power of eight bits, and so on. The result of this superposition is a machine that is not only way faster than conventional computers but also can make calculations that our computers simply cannot (Ladd et al., 2010).



### 3.1.2 Linear Independence and Vector Basis

The next concept is “linear combination,” which is paramount to understanding the nature of linear algebra. As its name indicates, a linear combination is a function that operates with vectors and scalars to form new vectors. These resultant vectors are linear combinations of the “natural” vectors and scalars.

Colors can be a helpful metaphor for visualizing this (Beneschan 2020). When designing a painting, an artist typically includes five primary colors in their palette: red, yellow, blue, white, and black. By combining these fundamental colors, artists can create any other color in the spectrum. The spectrum in linear algebra is called *span*, which accounts for all the possible linear combinations within a vector set. What is crucial to understand here is that these five primary colors are completely *independent* of each other. Therefore, for example, no amount of red will ever be able to produce black paint. This same logic extends to the combination of all five primary colors. Applying this reasoning to linear algebra, two vectors ( $\vec{a}$  and  $\vec{b}$ ) are linearly independent when it is impossible to scale  $\vec{a}$  to obtain  $\vec{b}$ . Examine Equation 3.6, where  $c_n$  is a set of scalars, and  $\vec{v}_n$  is a set of vectors:

$$c_1 \cdot \vec{v}_1 + c_2 \cdot \vec{v}_2 + c_n \cdot \vec{v}_n = 0 \tag{3.6}$$

For a vector set to be linearly independent, the only way to satisfy this equation will be for all the constants ( $c_1, c_2, \dots, c_n$ ) to be 0. This is because each vector ( $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ ) is the only one in the vector set that contains information about a specific coordinate of the vector space. For example, vectors  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ . This vector set is linearly independent because  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$  has value in the Y-coordinate, but no value in the X-coordinate, and vice versa with vector  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ . This case, therefore, satisfies the previous equation, where the only solution to obtain 0 as a result is by setting the constants to 0, as seen in Equation 3.7:

$$c_1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + c_2 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

$$0 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

(3.7)

Now that the terms linear combination and linear independence are clear, it is possible to dive into the painting metaphor again. Following the logic explained before, green is a *combination* of blue and yellow, and pink is a *combination* of red and white. But how does one know whether a color is a primary color? Two rules need to be followed for this to take place:

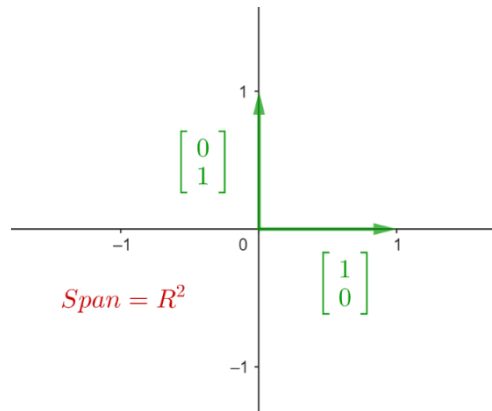
- 1- The given color must be independent from the other primary colors (i.e., no amount of red will ever produce black).
- 2- The given color, alongside the other primary colors, must be part of the color spectrum.

When transferring this logic to linear algebra, *linear combinations* can be obtained by adding and scaling “primary” vectors, which in linear algebra are called *basis*. A set of vectors  $n$  in a vectorial space  $S$  can be categorized as a basis if it follows these two requirements:

- 1- The vectors within the vector set are linearly independent.
- 2- The vectors within the vector set span within the vectorial space  $S$ .

Examine the example of vectors  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ . As shown previously, it has been proved that these vectors are a set of linearly independent vectors, but because they both exist in a 2-dimensional plane (vectorial space), they can also be recognized as a basis. This means that, by scaling and operating with the basis  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  as seen in Figure 3.6, any possible vector in the 2-D plane can be created.

**Figure 3.6: Span of Vector Basis**



### 3.1.2.1 Vector Basis in Quantum Computing

As mentioned previously, any given state for a qubit can be described by only using a linear combination of the states  $|0\rangle$  and  $|1\rangle$ . Therefore,  $|0\rangle$  and  $|1\rangle$  are the vectors that form a basis for single-qubit states. This is the main basis used for qubit representation, but there can be infinitely more bases. For example, imagine a qubit in a state that has the same probability of measuring 0 and measuring 1. This state is represented by the vector of Equation 3.8 (the coefficient  $\frac{1}{\sqrt{2}}$  will be talked about in a later chapter, but as it can be inferred, the coefficient is the same for 0 and 1, which therefore provides the same likeliness of measuring 0 or 1):

$$|+\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle \tag{3.8}$$

Assume the quantum state of Equation 3.8 as the state  $|+\rangle$ . Moreover, when the sign of the second coefficient is flipped, it is named  $|-\rangle$ , as depicted in Equation 3.9:

$$|-\rangle = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle \tag{3.9}$$

Although both states are different, the probability of obtaining 0 and 1 when the qubit is measured is the same (50% each). As surprising as it may be,  $|+\rangle$  and  $|-\rangle$  also form a basis for single-qubit states because any qubit state can be described using a linear combination of these two vectors.

For example, if adding  $|+\rangle$  and  $|-\rangle$ , the second coefficient will cancel out, leaving only  $\sqrt{2} |0\rangle$  as the resultant quantum state. Likewise, when subtracting them, the quantum state would be reduced to  $\sqrt{2} |1\rangle$  only.

Take the following state  $|z\rangle$  as an example, and substitute for the  $|+\rangle$  and  $|-\rangle$  basis, as portrayed in Equation 3.10:

$$\begin{aligned}
 |z\rangle &= \frac{\sqrt{3}}{2} |0\rangle + \frac{1}{2} |1\rangle = \frac{\sqrt{3}}{2} (\sqrt{2} |+\rangle + \sqrt{2} |-\rangle) + \frac{1}{2} (\sqrt{2} |+\rangle - \sqrt{2} |-\rangle); \\
 |z\rangle &= \left(\frac{\sqrt{3}+1}{\sqrt{2}}\right) |+\rangle + \left(\frac{\sqrt{3}-1}{\sqrt{2}}\right) |-\rangle = a |+\rangle + b |-\rangle
 \end{aligned}
 \tag{3.10}$$

The state  $|z\rangle$ , which was expressed through the basis of  $|0\rangle$  and  $|1\rangle$  can also be expressed through the basis of  $|+\rangle$  and  $|-\rangle$ . The state continues to be the same, but the coefficients have changed. Vectorial bases can be incredibly useful in quantum computing. Think about this analogy that compares vectorial bases to spoken languages: English and Spanish use different vocabulary and grammatical rules, yet both languages can portray the same ideas and feelings through speech and written text. Vectorial bases work similarly since different vectorial bases can represent the state of the same qubit under their own “rules” (Nielsen & Chuang, 2010, p. 14).

### 3.1.3 Linear Transformations

A linear transformation is an operation performed on a vector, which outputs a new “transformed” vector. This operation can be thought of as a function that operates on a vector, where the function specifies what needs to be done to the coordinates of the original vector to obtain the transformed vector. For example, see Equation 3.11:

$$T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ 3x_1 \end{bmatrix}
 \tag{3.11}$$

This vector transformation adds the X and Y components and sets the result as the X component of the transformed vector. Besides, it triplicates the X component of the original vector to produce

the Y component of the transformed vector. Vector transformations can also be notated through the following expressions in Equation 3.12:

$$T(x_1, x_2) = (x_1 + x_2, 3x_2)$$

$$T: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + x_2 \\ 3x_2 \end{bmatrix}$$

(3.12)

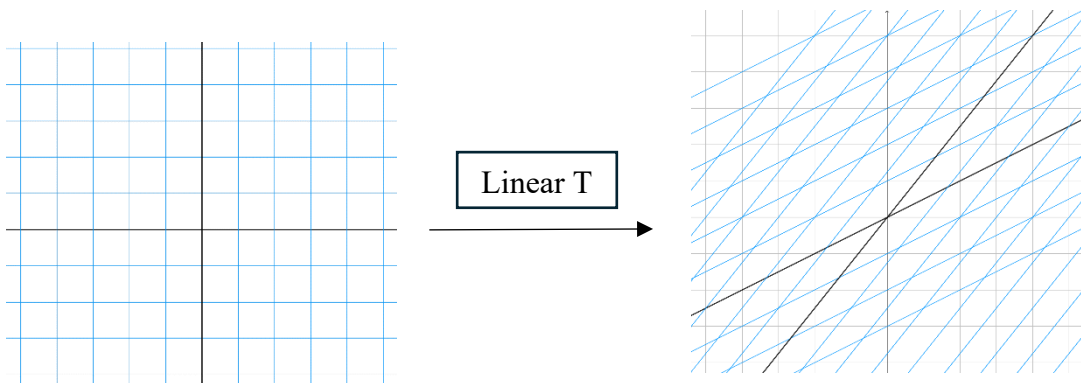
Now, the focus will be directed towards a type of vector transformation called *linear transformation* or  $T$ , which is used in several real-life applications, like in video game design when the game shows the same scenery from different perspectives depending on the character. For a vector transformation to be a linear transformation, it must hold these two specific linearity properties true:

- 1-  $T(\vec{a} + \vec{b}) = T(\vec{a}) + T(\vec{b})$
- 2-  $T(c \cdot \vec{a}) = c \cdot T(\vec{a})$ , where  $c$  is constant.

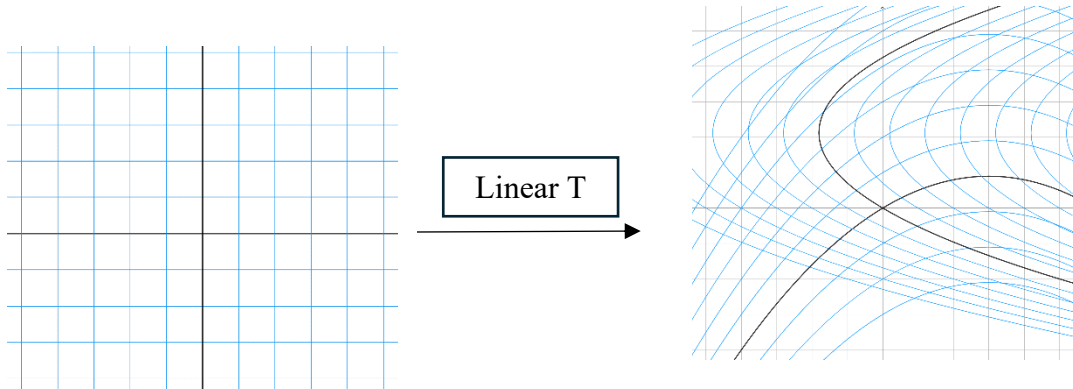
To observe these properties graphically, a specific vector transformation can be applied to all the vectors of a plane. Therefore, the resultant grid of the plane will be different from the original plane. Nonetheless, the properties mentioned above state that it will be a linear transformation only if, after the plane transformation, all lines within the plane remain lines (not curves) and if the origin remains fixed. Figure 3.7 shows graphical examples for both types of situations (linear and non-linear):

**Figure 3.7: Graphical Examples of Linear Transformations**

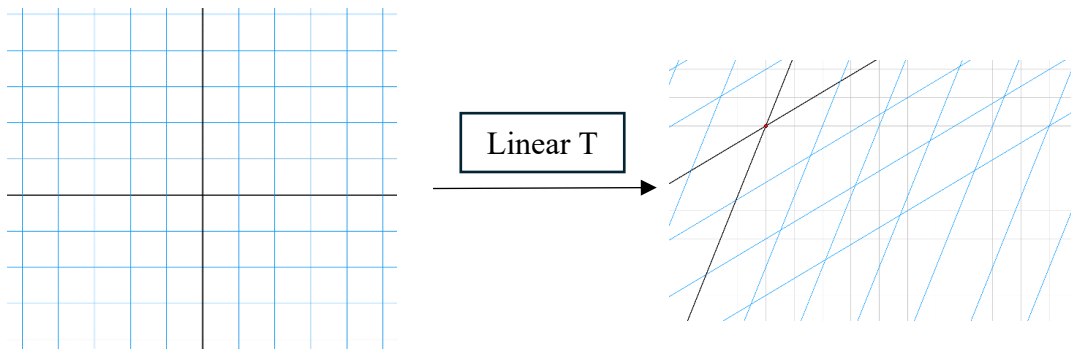
- This is a linear transformation:



- This is not a linear transformation (curves instead of lines within the plane):



- This is not a linear transformation (origin is not fixed):



But how does one find the transformation function? In other words, how does one know the formula that, when given the input vector, outputs the transformed vector? It is possible to do so through the basis vectors  $\hat{i} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$  and  $\hat{j} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ , by tracking where these vectors land once the transformation has taken place. For example, imagine a vector  $\vec{a} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ , which can be rearranged as  $\vec{a} = -1\hat{i} + 2\hat{j}$ . After the linear transformation, it is inferred that  $\hat{i}$  - transformed lands at  $\begin{bmatrix} 1 \\ -2 \end{bmatrix}$ , and that  $\hat{j}$  - transformed lands at  $\begin{bmatrix} 3 \\ 0 \end{bmatrix}$ . By applying this to the previous equation, the landing spot for  $T(\vec{a})$  can be deduced through the logic in Equation 3.13:

$$T(\vec{a}) = -1 \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 2 \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$T(\vec{a}) = \begin{bmatrix} -1(1) + 2(3) \\ -1(-2) + 2(0) \end{bmatrix}$$

$$T(\vec{a}) = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

(3.13)

The reason this process is so powerful lies behind the fact that any transformed vector can be obtained by only knowing three pieces of information: The original vector,  $\hat{i}$  - transformed, and  $\hat{j}$  - transformed.

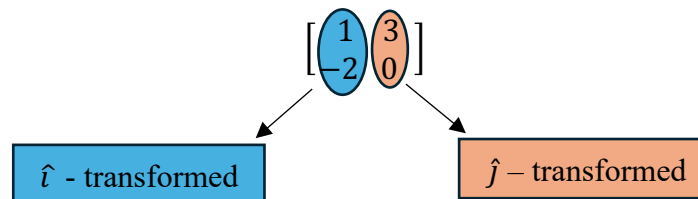
### 3.1.4 Matrixes

The term “matrix” refers to a combination of vectors which can be arranged in columns and rows. They are referenced to represent linear combinations, as mentioned in the previous section, and can have as many rows and columns as needed. Some examples of matrixes are depicted in Equation 3.14:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \begin{bmatrix} 5 & 1 & 4 \\ 2 & 3 & 6 \\ 2 & 6 & 4 \\ 3 & 9 & 5 \end{bmatrix}$$

(3.14)

Matrixes are especially useful for linear transformations since they are used to represent the final coordinates of the transformed basis vectors together. This permits the calculation of any resultant vector derived from the linear transformation with a simple two-by-two matrix. To set up this transformation matrix, the first column represents where  $\hat{i}$  - transformed lands, and the second column where  $\hat{j}$  - transformed lands. Using the example from the section above, Equation 3.15 shows the transformation matrix:



(3.15)

By multiplying the original vector ( $\vec{a} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ ) with this matrix, the resultant transformed vector is obtained. Hence, this two-by-two matrix is key for this specific linear transformation, which will unlock the resultant transformed vector within this 2-D space. Matrix multiplication follows the procedure portrayed in Equation 3.16:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} \quad (3.16)$$

Linear transformations can be applied one after another as well. This means there will be two two-by-two transformation matrixes ( $M1$  and  $M2$ ), where each one holds the values of where  $\hat{i}$  and  $\hat{j}$  would land in each case. Therefore, it is imperative to find the composition of both transformation matrixes that will unlock the destination of  $\hat{i}$  and  $\hat{j}$  after both linear transformations have taken place. This can be obtained by multiplying both transformation matrixes. It is important to note that the order in which the linear transformations are applied directly influences the outcome of the composite matrix. This is why  $M1$  will be located on the right and  $M2$  on the left (following the logic of composite functions), as shown in Equation 3.17:

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_M \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_M = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \quad (3.17)$$

To compute the composition matrix,  $M1$  is separated into two separate vectors. After that, calculate the position at which  $M1$ 's  $\hat{i}$  (in blue) and  $\hat{j}$  (in red) will land after  $M2$ 's (in yellow) transformation, as shown in Equation 3.18:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \hat{i} & \hat{j} \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \quad (3.18)$$

When translating this diagram into a formula with variables, obtaining Equation 3.19:



$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} \quad (3.19)$$

### 3.1.4.1 Matrixes in Quantum Computing

Linear transformations (represented through matrixes) are used to interact with qubits and get information from them. In classical computers, bits go through logical gates, and computations are performed on their value (0 or 1). The same can be said for quantum computers. Qubits (represented by vectors) go through quantum gates (which are linear transformations in their matrix form) that transform the qubit into its different possible states. Therefore, matrixes are fundamental to unleashing the power of quantum computers to their full potential. Because qubits can hold any value between 0 and 1, linear transformation matrixes will allow the manipulation of all the possible combinations between 0 and 1. For example, the quantum NOT gate (X), shown in Equation 3.20, switches the coefficients of  $|0\rangle$  and  $|1\rangle$  as follows:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \quad (3.20)$$

Another fundamental quantum gate is the *Hadamard* gate (see Equation 3.21 below), which turns  $|0\rangle$  into  $|+\rangle$  and  $|1\rangle$  into  $|-\rangle$  (Nielsen & Chuang, 2010, p. 18).

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (3.21)$$

### 3.1.5 Dot Product

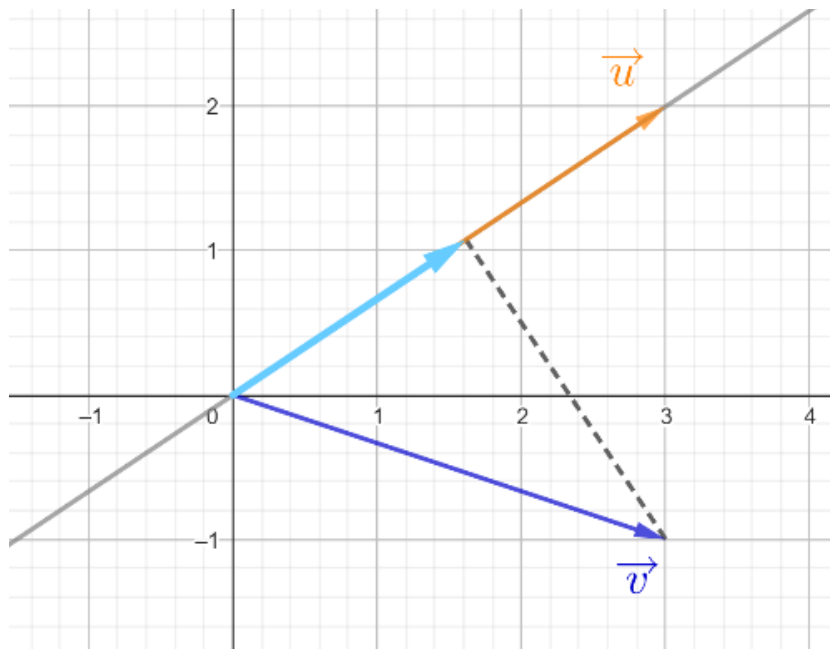
This subsection introduces another linear algebra concept that will be useful for the understanding of quantum computing in the future, known as the dot product. This mathematical operation takes two vectors within the same dimension as input and outputs a scalar, providing information about their relationship. Equation 3.22 portrays how the dot product is calculated:

$$\begin{bmatrix} a \\ c \end{bmatrix} \cdot \begin{bmatrix} b \\ d \end{bmatrix} = (a \cdot b) + (c \cdot d)$$

(3.22)

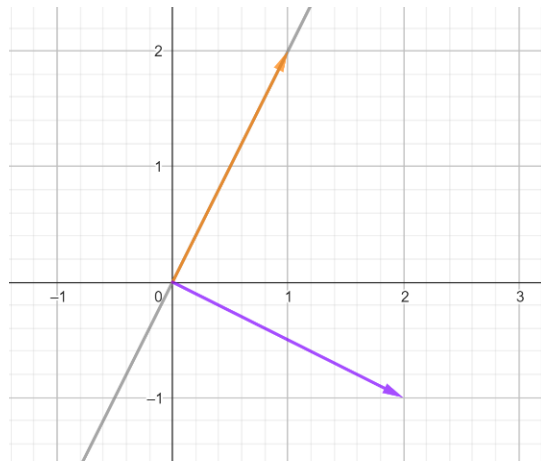
The dot product is related to the projection of a vector onto another vector. This idea can be approached as casting the shadow of a vector on top of another vector. Figure 3.8 below represents the projection of  $\vec{w}$  (pink) on  $\vec{v}$  (yellow). The length of the resulting shadow is directly related to the dot product's result.

**Figure 3.8: Projection of a Vector onto Another Vector**



When computing the dot product of a vector times itself (for example  $\vec{v}$  and  $\vec{v}$ ), the resultant “shadow” would be the length of  $\vec{v}$ . However, when calculating the dot product of perpendicular vectors, the result will be 0, as no “shadow” can be projected at all. Therefore, when the dot product of two vectors is 0, they are known as *orthogonal* and cannot be expressed as a linear combination of the other. As can be inferred from Figure 3.9, no shadow can be cast in this situation since both vectors are perpendicular. Figure 3.9 is an example of a pair of orthogonal vectors whose dot product equals 0:

**Figure 3.9: Orthogonal Vectors**



The dot product is particularly useful for calculating another property of vectors: the modulus. This concept stands for the magnitude (or length) of a vector, and its notation and formula are shown in Equation 3.23:

$$|\vec{v}| = \sqrt{\vec{v} \cdot \vec{v}} \tag{3.23}$$

Observe that the operation taking place within the square root is the vector's dot product with itself. Note that the length of the vector is related to the dot product.

### 3.1.5.1 Dot Product in Quantum Computing

There are multiple applications of the dot product in quantum computing, such as vector normalization. Take the following qubit state in Equation 3.24 as an example:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{3.24}$$

The probability of measuring 0 is given by  $\alpha \cdot \alpha^*$ , and the probability of measuring 1 is  $\beta \cdot \beta^*$ . Since the only possible outcomes once the qubit is measured are either of these two values (0 and 1), the probability of measuring 0 plus the probability of measuring 1 must always add up to 1, following the total probability theorem. Therefore, as it can be inferred, neither  $\alpha$  nor  $\beta$  is greater than 1. Because the modulus of an arbitrary vector can take any value, our scope is to scale it (multiply the vector by a factor) so that its modulus is equal to 1, and it will follow the probabilistic

logic desired. This means that the modulus of our vector must follow this condition, known as normalization, which all quantum states must fulfill. See Equation 3.25:

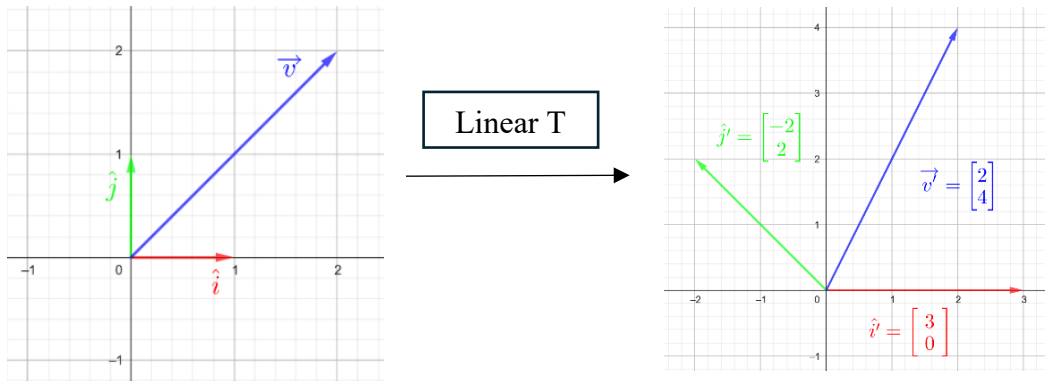
$$||\psi\rangle| = \sqrt{|\psi\rangle \cdot |\psi\rangle^*} = 1 \rightarrow |\psi\rangle \cdot |\psi\rangle^* = 1 \quad (3.25)$$

Note that the dot product is referred to as the “inner product” in the realm of quantum mechanics (Ganguly, 2021, p. 8). The main difference between them lies in the type of vectors they operate on. The inner product operates with real and complex vectors. The dot product is a type of inner product that operates only with real vectors (Nielsen & Chuang, 2010, p. 66). Because quantum mechanics operates in a complex vectorial space, the term “inner product” is more accurate.

### 3.1.6 Eigenvectors and Eigenvalues

This section will focus on the concepts of eigenvectors and eigenvalues, which are crucial for a deep understanding of linear algebra and, therefore, quantum computing. An eigenvector refers to a vector that remains unchanged after a linear transformation. This means that the vector continues to be within the same span as it was before the linear transformation, although it may have been scaled. An eigenvalue refers to the factor by which the eigenvector is scaled after the linear transformation has taken place. For example, refer to Figure 3.10 below:

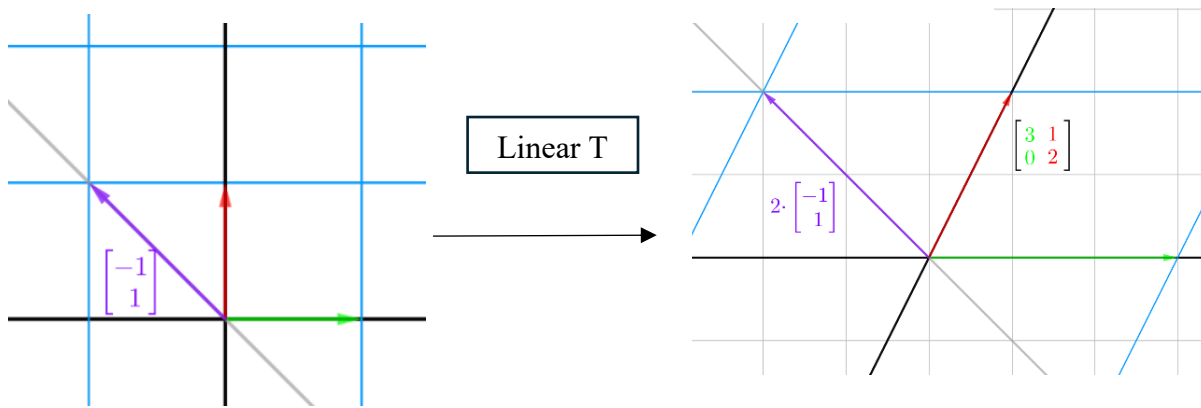
**Figure 3.10: Eigenvectors After a Linear Transformation (1)**



This example shows how, after the linear transformation,  $\hat{i}$  is an eigenvector because it continues pointing in the same direction as before the transformation. In this case, it can be quickly understood that the eigenvalue is 3, since  $\hat{i}$  was scaled by a factor of 3. Before the linear

transformation, its coordinates were  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ , and after the transformation they are  $\begin{bmatrix} 3 \\ 0 \end{bmatrix}$ . It is important to understand that there may be multiple eigenvectors (and therefore eigenvalues) in a linear transformation. Figure 3.11 is another example that has an eigenvector in the diagonal with an eigenvalue of 2 represented through a purple arrow:

**Figure 3.11: Eigenvectors After Linear Transformation (2)**



An equation that includes both variables must be established to algebraically determine both the eigenvector and the eigenvalue. On one side of the equation, there is the transformed eigenvector, and on the other side, the eigenvector scaled (multiplied by the eigenvalue), as seen in Equation 3.26:

$$A\vec{v} = \lambda\vec{v} \tag{3.26}$$

In this equation, A stands for the transformation matrix,  $\vec{v}$  is the eigenvector,  $\lambda$  is the eigenvalue, and the notation “|” will compute the determinant<sup>1</sup>. Therefore, both sides are representing the transformed eigenvector. After rearranging Equation 3.26<sup>2</sup>, Equation 3.27 below is obtained:

<sup>1</sup> See Appendix – Determinants.

<sup>2</sup> See Appendix - Eigenvectors and Eigenvalues formula.

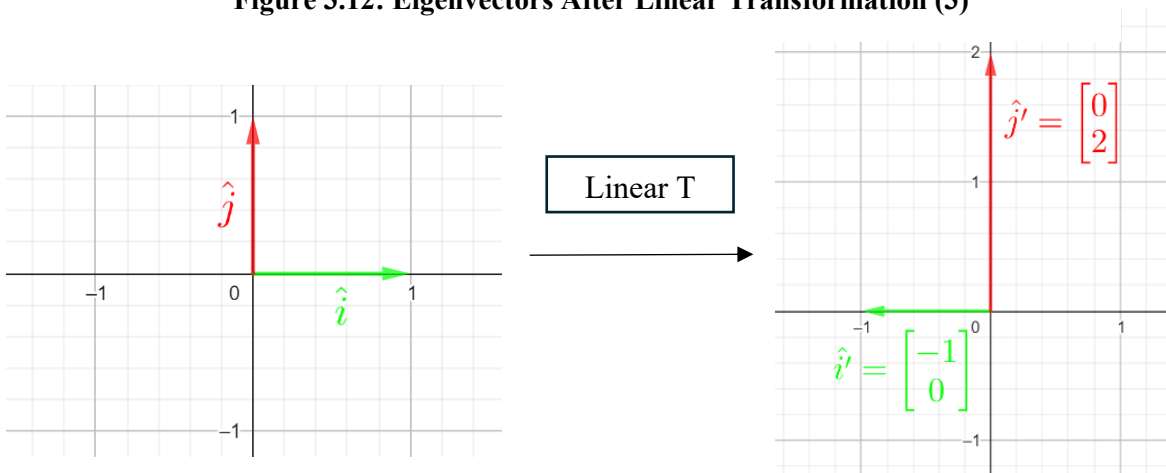
$$|A - \lambda I| = 0 \tag{3.27}$$

Once the value of  $\lambda$  is known, the eigenvalue is substituted in the equation and solved for  $\vec{v}$ , which provides the coordinates of the eigenvector.

Now, what if both  $\hat{i}$  and  $\hat{j}$  are eigenvectors? For example, given the linear transformation in Figure 3.12,  $\hat{i}$  (green) is scaled by -1 and  $\hat{j}$  (red) is scaled by 2. This means that the transformation matrix will have the form of a *diagonal matrix*, a matrix whose values can only be found in its diagonal. From the example above, the transformation matrix would be the following diagonal matrix portrayed in Equation 3.28:

$$\begin{bmatrix} -1 & 0 \\ 0 & 2 \end{bmatrix} \tag{3.28}$$

**Figure 3.12: Eigenvectors After Linear Transformation (3)**



These diagonal transformation matrixes are formed by eigenvectors ( $\hat{i}$  and  $\hat{j}$ ), where each of the values within that diagonal matrix accounts for each of their eigenvalues. Diagonal matrixes are of great interest in linear algebra since they allow for much faster matrix computations. For example, when multiplying a diagonal matrix multiplied by itself, it is only required to square the diagonal values. Therefore, escalating this reasoning to higher powers, the marginal benefit is exponential. To take advantage of transformation diagonal matrixes, it is imperative to find

eigenvalues of a linear transformation and use them as vector basis, or the “new”  $\hat{i}$  and  $\hat{j}$ <sup>3</sup>. These new vector bases, coming from the eigenvectors of a linear transformation, are known as *eigenbasis*.

### 3.1.6.1 Eigenvectors and Eigenvalues in Quantum Computing

Eigenvectors and eigenvalues are crucial to understand quantum physics. As will be shown in Section 3.2, one of the properties of quantum systems is that their magnitudes cannot take any value. Take, for example, the process of measuring the energy of a given quantum system. To find the possible energy values, one of the most important equations in quantum physics must be considered: the time-independent Schrödinger Equation, as described in Equation 3.29:

$$\hat{H} |\psi\rangle = E |\psi\rangle \tag{3.29}$$

As can be inferred, Equation 3.29 looks exactly like an eigenvalue equation. The Hamiltonian ( $\hat{H}$ ) is a linear transformation on a vector  $|\psi\rangle$ , which results in the same vector scaled by a factor of E (eigenvalue). E stands for the energy of the system, which means that the system’s energy can only take the values given by this equation.

### 3.1.7 Summary

As described at the beginning of this chapter, linear algebra is the language of quantum mechanics, and these sub-sections focused on providing a brief introduction to the concepts that serve as the foundation for that language. When applied to quantum computing, this theoretical background forms the basis for representing and manipulating quantum states, enabling the development of quantum algorithms and computational models.

The next chapter will focus on the concepts of quantum physics that are required to understand quantum computing clearly. The mathematical tools portrayed in this section will be relevant throughout the discussion in Section 3.2.

---

<sup>3</sup> See Appendix - Eigenbasis.

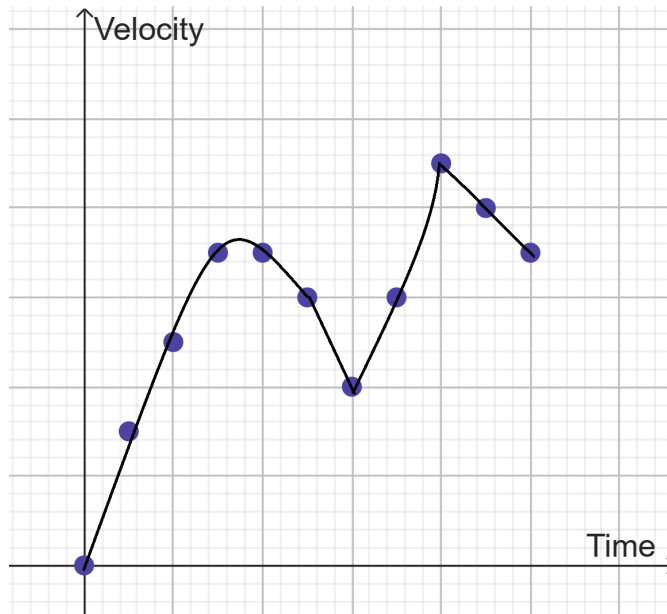
## 3.2 Quantum Physics

After reviewing the fundamental concepts outlined in Section 3.1, this section aims to provide background information concerning quantum physics. Continuing the analogy referenced earlier, where linear algebra serves as the language for manipulating “quantum” concepts in quantum computing, this section deepens into the main concepts of quantum physics (bracket notation, entanglement, observables, the uncertainty principle, and probabilities), which are to be explored through this same language.

### 3.2.1 The Language of Quantum Mechanics

Before the birth of modern physics, everything could be described using classical models. In classical physics, physical quantities are always single-valued and continuous. For example, it is impossible to conceive of a car that travels at two different velocities simultaneously (therefore, it is single-valued) or one that goes from 0 to 100 km/h instantaneously (therefore, it must be continuous). This is why physical quantities in classical mechanics are described using continuous and single-valued functions, which could represent the velocity of a car over time, as depicted in Figure 3.13:

**Figure 3.13: Example of the Velocity of a Car Over Time**

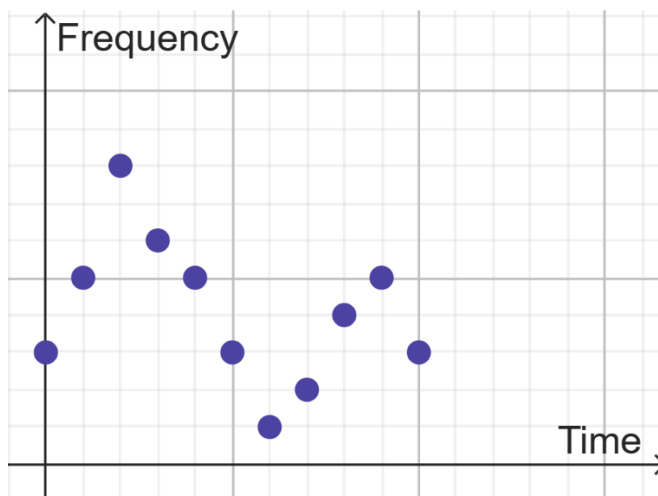




However, everything changes when venturing into the world of small physical systems. In the early 20<sup>th</sup> century, the model used to describe an atom was the Rutherford atomic model, which established that atoms consisted of a cloud of electrons orbiting a nucleus. When analyzing this system using classical mechanics, the electron will release electromagnetic radiation (light) while orbiting the nucleus. Since the electron loses energy while doing this, it will rapidly spiral inwards, releasing light with higher frequency until it collapses into the nucleus. Therefore, the graph that predicts the classical outcome of the system should be single-valued and continuous, as previously mentioned.

This, however, was not the result of the experiment. Instead of continuous light output with increasing frequency, only discrete (non-continuous) sets of light were obtained. When the experiment was repeated, light was discrete again, but surprisingly, the values were different from those of other runs of the same experiment (non-single valued). Therefore, the results seemed to suggest that outcomes were not single-valued nor continuous but probabilistic and discrete. Figure 3.14 portrays a hypothetical example of what the scientists retrieved from their experiments<sup>4</sup>:

**Figure 3.14: Hypothetical Example of a Quantum System Experiment**



This was a total revolution in the way physical quantities were conceived. If the light measured was discrete, then light must travel in bundles of energy. Each bundle is known as a photon or a

---

<sup>4</sup> Note that Figure 3.14 exemplifies the discrete nature of quantum mechanics only. In this case, the experiment was performed one time only. If the experiment was to be repeated, the values obtained would be different, showing a completely different distribution of the points depicted.

quantum of the electromagnetic field (the minimum amount of any physical entity). Scientists needed a new way to portray these physical phenomena. Thus, quantum mechanics was born.

Different than classical systems, quantum systems are probabilistic. When measuring a physical quantity in an experiment many times, the outcome will be single values that may be different each time they are measured (as mentioned before, some outcomes are more likely to occur, so they are random but probabilistic). This means that before making a measurement, the particle holds the information of all possible states, but when measuring, this superposition of states collapses into one single outcome. This is the key that holds the power behind quantum computing.

Since physical quantities are no longer single-valued and continuous, they cannot be represented using functions like in classical mechanics. To mathematically represent the quantum state, all possible states and their probabilities must be considered, and the best way to do this is by using vectors. For example, a quantum state which is in a superposition of energy states  $|E_1\rangle, |E_2\rangle\dots$  can be represented as depicted in Equation 3.30:

$$|\psi\rangle = c_1 |E_1\rangle + c_2 |E_2\rangle + \dots = \sum_i c_i |E_i\rangle \quad (3.30)$$

Where the coefficients  $c_1, c_2\dots$  provide information about the probability of obtaining each outcome. If instead of energy measurements, angular momentum was being measured, the *same* state  $|\psi\rangle$  can be described using a linear combination of possible angular momentum states  $|L_1\rangle, |L_2\rangle\dots$ etc. See Equation 3.31 below:

$$|\psi\rangle = c_1' |L_1\rangle + c_2' |L_2\rangle + \dots = \sum_i c_i' |L_i\rangle \quad (3.31)$$

Although it might seem counterintuitive, the quantum state expressed in both equations is the same, but it is written as a function of different possible measurement parameters (in this case, energy, and angular momentum).

To fix the problems that surged because of the Rutherford atom model, Bohr proposed a new atomic model in 1913 that included quantized electron orbits: they orbit the nucleus, but only at

certain distances. The possible orbits are given by the energy of the atom, which, as explained earlier, is discrete. Hence, the orbits must also be discrete.

However, is everything in quantum physics discrete? The answer is no. The electron cannot be in between orbits, as it automatically jumps from one to another when there is a change in energy. But nothing is stopping it from taking any position within the same orbit. Some physical properties can be discrete or continuous, depending on the quantum system at hand. Next, an example of continuous states to visualize how they function is presented. When representing a state given by a superposition of position, a discrete sum ( $\sum$ ) cannot be used because there are no discrete values of position (represented with the letter  $x$ ). The correct mathematical tool for this is the integral, which is the sum of a continuous set. Therefore, the integral represents the superposition of all individual position states. In this case, each continuous value also has a probability value associated, so the final expression for the quantum state is expressed through Equation 3.32:

$$|\psi\rangle = \int dx \psi(x) |x\rangle \tag{3.32}$$

In this scenario,  $|x\rangle$  are all the possible position states, and  $\psi(x)$  is related to the probability of each position state. As can be seen, the expressions for discrete and continuous variables are remarkably similar. It is possible to sum (in the case of discrete variables through conventional summation and in the case of continuous variables through an integral) the possible states multiplied by a coefficient related to the probability of measuring each state. In fact,  $\Psi(x)$  is called the position wavefunction, which provides the probability of measuring any position value.

### 3.2.2 Bra-ket Notation

As you may have noticed, the way vectors are notated in quantum mechanics is different than in mathematics. Why are these brackets used instead of the arrow on top for vectors? If it is a vector, how does the arrow look in this case? Why are functions sometimes used instead of real numbers?

Vectors are indeed used in quantum mechanics, but they are not within the same vectorial space that has been reviewed in Section 3.1.1 (arrows in three dimensions). “Quantum vectors” lie in an abstract mathematical space called a Hilbert space (Nielsen & Chuang, 2010, p. 66). A ket (known as  $|\psi\rangle$ ) is a mathematical object that represents a quantum state, and it can be a vector or a function.

A particle's spin state can be represented through  $|\varphi\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ , which is a vector. But, to represent a wavefunction, it can be done through  $|\psi\rangle = f(\vec{r}, t)$ . Both ways of portraying the quantum state are valid.

It is fundamental to understand that a ket's coefficient can be imaginary numbers<sup>5</sup>. An imaginary number contains the number  $i = \sqrt{-1}$ , and they can be expressed in the form  $z = a + i \cdot b$ . As explained in the appendix, "a" is the real part of  $z$  ( $a = \text{Re}\{z\}$ ), and "b" is its imaginary part ( $b = \text{Im}\{z\}$ ). Finally, the conjugate of an imaginary number consists of changing the sign of the imaginary part. This is used so that the outcome probability is not negative since multiplying the imaginary part of  $z$  times itself would result in  $-b^2$  (which is a negative probability). By multiplying with the complex conjugate, a positive output is ensured. Thus, the conjugate of  $z$  would be  $z^* = a - i \cdot b$ .

Given a ket " $|\Phi\rangle = \begin{bmatrix} 1 \\ -i \\ 0 \end{bmatrix}$ ," it can be transformed into its counterpart, the bra " $\langle\Phi|$ ." To do this,

the column vector must be converted into a row vector, taking the complex conjugates of every coefficient. Thus, the previous ket " $|\Phi\rangle$ " is turned into the bra " $\langle\Phi| = [1 \ i \ 0]$ ." Merging both together outputs a "bracket," which represents the inner product<sup>6</sup> of the two vectors. Given  $|\psi\rangle =$

$\begin{bmatrix} a \\ b \\ c \end{bmatrix}$  and  $|\Phi\rangle = \begin{bmatrix} d \\ e \\ f \end{bmatrix}$ , Equation 3.33 shows this process:

$$\langle\Phi|\Psi\rangle = a \cdot d^* + b \cdot e^* + c \cdot f^* \tag{3.33}$$

Remember that the kets represent the superposition of possible states, so in this case, although  $|\psi\rangle$  is a column vector, it is a linear combination of the possible measured states represented as  $|\psi\rangle = a|X\rangle + b|Y\rangle + c|Z\rangle$ . Therefore, the probability of measuring  $|X\rangle$  is given by  $|\langle X|\psi\rangle|^2 = a \cdot a^*$ .

---

<sup>5</sup> See Appendix – Imaginary and Complex Numbers.

<sup>6</sup> Recall the dot product from Section 3.1.5.

### 3.2.3 Entanglement

Entanglement is a phenomenon in quantum mechanics where two or more particles become correlated in such a way that the state of one particle provides information about the state of the other one. Although this may sound counterintuitive for classical physics, there is no independence of distant objects in quantum mechanics.

Picture having a pair of shoes and placing each of them in separate boxes. Then, you send each box to opposite ends of the universe, at an enormous distance from each other. If you open one of the boxes and find the right shoe, you will automatically know that the other box, despite how far it is, contains the left shoe. Quantum entanglement operates on a similar principle to particles. Once a particle's properties have been measured, there is an instantaneous transfer of information regarding the other entangled particles without having to measure them, regardless of how distant these particles are from each other. Because this example depicts a classical system, the item contained in the boxes is already "collapsed" into a state (left or right shoe) once it is placed in the box. However, it is fundamental to understand that in a quantum system, the boxes would contain a superposition of the left and right shoe until the box is opened.

Until now, quantum states of single qubits have been expressed through matrixes with two rows (representing the possible states: 0 and 1) and 1 column. Dealing with a higher number of qubits requires the representation of all possible combinations of states, known as the "basis states." Basis states refer to the possible outcomes that the qubit system might collapse into upon measurement (Nielsen & Chuang, 2010, p. 16). For example, for one single qubit, the basis states are  $|0\rangle$  and  $|1\rangle$ . However, for two qubits, the basis states are  $|00\rangle$ ,  $|01\rangle$ ,  $|10\rangle$ , and  $|11\rangle$ . Because both qubits can be either 1 or 0 after measurement, in a 2-qubit system, a combination of all possibility outcomes is referred to as the "basis states." Equation 3.34 depicts the matrix representation of the basis states of a 2-qubit system:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(3.34)

To represent the combination of two states (that is not a state as described in Equation 3.34), their coefficients must be multiplied. This is known as the tensor product, which allows for operations with different qubits that exist in different Hilbert spaces. For example, the combined state<sup>7</sup> of quantum states  $|x\rangle$  and  $|y\rangle$  is given by Equation 3.35:

$$|yx\rangle = \begin{bmatrix} y_0x_0 \\ y_1x_0 \\ y_0x_1 \\ y_1x_1 \end{bmatrix} \quad (3.35)$$

The following theoretical approach will provide an intuition of the meaning of entanglement in the quantum computing domain. When working with multiple qubits, their structure speaks about their behavior. Based on said structure, 2-qubit systems can be classified into two types:

- **Product states:** These multi-qubit states can be expressed through a sequence of single-qubit states. Imagine that the previous states of  $|x\rangle$  and  $|y\rangle$  are to be measured, aiming to calculate the probability of the qubit on the left (y) collapsing into 0. The only two possible results in the scenario would be if the outcome after measurement is  $|00\rangle$  or  $|01\rangle$ . Equation 3.36 portrays this when translated into an algebraic form:

$$\begin{aligned} p_{left-qubit}(|0\rangle) &= p_{|yx\rangle}(|00\rangle) + p_{|yx\rangle}(|01\rangle) \\ &= (y_0x_0)^2 + (y_0x_1)^2 \\ &= y_0^2x_0^2 + y_0^2x_1^2 \\ &= y_0^2(x_0^2 + x_1^2) \\ &= y_0^2 \end{aligned} \quad (3.36)$$

In the process of operating from the third step to the final result of Equation 3.36, it can be inferred that  $(x_0^2 + x_1^2)$  is, in fact, equal to 1, since it is certain that the right qubit will either collapse into 0 or 1. Therefore, the probability of the left qubit collapsing into a particular state is independent of the other qubit's coefficients. It can be inferred that product states are agglomerations of single

---

<sup>7</sup> Note the change in order, where X represents the qubit on the left (the first) and Y the one on the right (the second).

qubit states and can be represented as such. For example, the state  $| - + \rangle = \frac{1}{\sqrt{2}}(| 00 \rangle + | 01 \rangle - | 10 \rangle - | 11 \rangle)$  refers to a 2-qubit system whose first qubit is in state  $| + \rangle$ , and the second qubit is in state  $| - \rangle$ . It is possible to arrive at this conclusion by applying the reasoning explained in Equation 3.36.

- **Entangled states:** These states cannot be represented as a sequence of single qubits because each qubit's quantum state is dependent on the other, known as entanglement. For example, consider the state  $| \Phi^+ \rangle = \frac{1}{\sqrt{2}}(| 00 \rangle + | 11 \rangle)$ . To calculate the probability of this state collapsing into 0, there is only one possible outcome: both qubits collapsing into  $| 00 \rangle$ . Hence, if after measuring the left qubit, the output is 0 (collapsing  $| \Phi^+ \rangle$  into the state  $| 00 \rangle$ ), it can be concluded with a 100% probability that once the right qubit is measured, the output will be 0. Therefore, when working with entangled states, measuring a qubit will directly affect the outcome of the other qubit. Unlike product states, where the qubit's probability of collapsing into a particular state is completely independent of the other qubit's coefficients, entangled states are dependent on their entangled pair.

### 3.2.4 Observables

As previously mentioned, a quantum state can have several distinctive characteristics (energy, angular momentum, position, etc.). However, they are all represented through the wavefunction  $| \psi \rangle$ . Observables are the tools that enable the retrieval of one of these characteristics from the wavefunction. They work as filters that will only output information about said characteristics. For example, when using the “energy filter” on  $| \psi \rangle$ , it will only retrieve the information about  $| \psi \rangle$  concerning energy. But how is it possible to mathematically describe them? In previous points, how a quantum state can be in a superposition of many possible values of a physical quantity was discussed. Consider a state that can take four possible energy values:  $E_1$  with an associated vector state  $| E_1 \rangle$ ,  $E_2$  with a vector  $| E_2 \rangle$ , etc. These are called definite states because if the quantum state was, for example,  $| E_4 \rangle$ , it is known that the energy of the state is  $E_4$  with a 100% probability. Therefore, a list of kets and their associated values can be created. With this information, it is possible to construct a mathematical object that represents the desired physical quantity.

As reviewed in Section 3.1.3, a linear transformation (also known as an “operator” in quantum mechanics) can have eigenvectors and their corresponding eigenvalues associated with it. Therefore, physical values can be described as operators, known as observables. For example, the energy can be represented as the energy operator  $\hat{E}$ , which may have definite eigenvectors and eigenvalues, meaning discrete values of energy.

The possible measurable quantum states are given by the eigenstates of the observable, so all the possible superposition states must be given by a linear combination of those states. Since each eigenvector has a different eigenvalue (the measured quantity), each eigenvector must be independent from all other eigenvectors. This means that since you can represent any possible quantum state with the eigenvectors of the observable and since those eigenvectors are independent (therefore orthonormal, so their dot product will be 0), they form an eigenbasis. Going back to the example state with four possible energy states, as shown in Equation 3.37:

$$|\psi\rangle = a |E1\rangle + b |E2\rangle + c |E3\rangle + d |E4\rangle \tag{3.37}$$

Imagine the energy of the state is measured, and the value  $E3$  is the output. What would the quantum state be after the measurement? Since there is one single definite eigenvalue, the state must be the eigenvector associated with that eigenvalue and cannot be in a superposition of states. This means that after the measurement, the state will be  $|\psi\rangle = |E3\rangle$ , which is equivalent to projecting the quantum state onto the definite measured state. This is called quantum state collapse or collapse of the wavefunction.

### 3.2.5 The Uncertainty Principle

In quantum mechanics, the uncertainty principle describes how accurately two observables can be measured at the same time. To understand this matter, a new concept must be introduced: the commutator.

Two observables, for example,  $\hat{A}$  and  $\hat{B}$ <sup>8</sup>, commute if  $\hat{A}\hat{B} = \hat{B}\hat{A}$ . This means that the outcomes of measuring  $\hat{A}$  first and then  $\hat{B}$  will be the same as changing the order of measurements (this might

---

<sup>8</sup> This “triangle hat” denotes that the object refers to a linear transformation.



seem trivial, but it is not always the case). Therefore, the commutator can be introduced as described in Equation 3.38:

$$[\hat{A}, \hat{B}] = \hat{A}\hat{B} - \hat{B}\hat{A} \tag{3.38}$$

If the commutator is 0, then the observables commute, and if it is different than 0, then changing the order of measurements means changing the outcome of those measurements. It can be proven that if two observables commute, then they share the same eigenbasis. Take the example of a system in which energy and angular momentum commute:  $[\hat{E}, \hat{L}] = 0$ . Since they commute, they share an eigenbasis formed by the vectors  $|1\rangle, |2\rangle, |3\rangle, \dots$ . Remember that each of these vectors represents a measured state: for example, the state  $|1\rangle$  is the state corresponding to measuring  $E_1$  energy (its eigenvalue for energy), but it also corresponds to measuring  $L_1$  for angular momentum. Equation 3.39 shows the possible quantum state:

$$|\psi\rangle = c_1 |1\rangle + c_2 |2\rangle + c_3 |3\rangle + \dots \tag{3.39}$$

So, picture measuring energy for this system and obtaining the value  $E_3$ , which means that the value for angular momentum will be  $L_3$ . But what if  $L$  and  $E$  do not commute? In that case, they will not be sharing an eigenbasis, meaning that for each value of energy measured, there can be more than one possible angular momentum state. Imagine that the value  $E_2$  is measured. After measuring the state, it will collapse into  $|E_2\rangle$ , which is a definite state of energy but not of angular momentum (although it will still be in a momentum superposition). So,  $|E_2\rangle$  can be described as a combination of possible momentum states (see Equation 3.40 below):

$$|E_2\rangle = c_1 |L_1\rangle + c_2 |L_2\rangle + \dots \tag{3.40}$$

The momentum of the collapsed system can indeed be measured, but it would mean collapsing it again into a momentum state, which means having a superposition of energy states. As it can be inferred, if two observables do not commute, it means that it is impossible to know the value

measured by both observables with absolute certainty. The most famous example of this is Heisenberg's uncertainty principle, depicted in Equation 3.41:

$$\Delta x \cdot \Delta p \geq \frac{\hbar}{2} \tag{3.41}$$

This equation states that the position and linear momentum of a quantum particle cannot be measured at the same time, but this comes from the expression of the commutator of these two observables, shown in Equation 3.42, Where  $\hbar$  is a quantum constant, and  $\Delta$  is the statistical dispersion:

$$[\hat{x}, \hat{p}] = i\hbar \neq 0 \tag{3.42}$$

The dispersion is the size of the range of measurements of position and momentum, respectively. A lower position dispersion (meaning lower range, thus more accurate measurement) results in a higher momentum dispersion (less accurate momentum measurement).

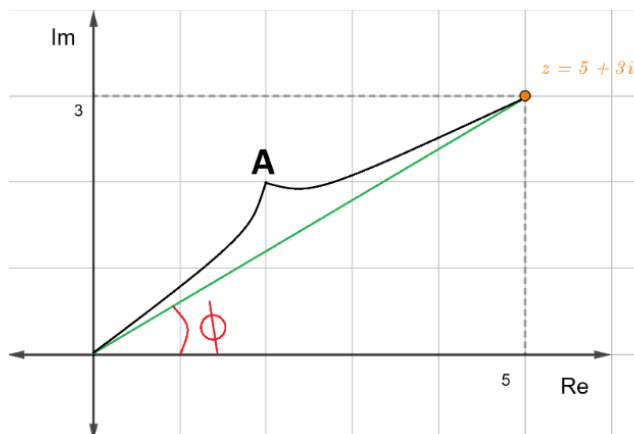
### 3.2.6 Probability and the Bloch Sphere

Probability enables the assessment of event likelihoods, such as coin flips or card games, allowing for the quantification of their occurrence. But does probability work the same way for classical and quantum systems? Picture two coins: a normal coin and a “quantum” coin. Would their probability be so different? The answer is quite “quantum:” Yes, but also no. Although their final probability is the same (50-50%), the roadmap used to arrive at this probability is different. But why are different roadmaps being used? The answer lies behind imaginary numbers. Probabilities in quantum systems are related to the coefficients of their possible states. For example, in state  $|A\rangle = \alpha |0\rangle + \beta |1\rangle$ ,  $\alpha$  and  $\beta$  are the coefficients for states 0 and 1. The key is that these coefficients are often expressed with imaginary numbers, unlike in classical physics (Zwiebach, 2022). A complex number<sup>9</sup> can be described in the complex plane. For example, Figure 3.15 shows the complex number  $z = 5 + 3i$ :

---

<sup>9</sup> See Appendix – Imaginary and Complex Numbers.

**Figure 3.15: Complex Number in the Complex Plane**



As can be inferred from Figure 3.15, any complex number has a length  $A$  and phase  $\phi$  in the complex plane. Phase is related to the direction in the complex plane. Therefore, if it is possible to use imaginary and complex numbers to describe probabilities in quantum systems, this means that  $\alpha$  and  $\beta$  must also have an amplitude and phase. Although it may sound counter-intuitive, coefficients in quantum mechanics have directions, and they play a huge role in quantum computing, as will be discussed in the following chapters. This is because, when adding or subtracting these coefficients, their phase can cancel out (if they point toward different directions). This behavior is known as interference in quantum mechanics. In quantum computing, it is accepted to use the term “amplitude” for the complex numbers  $\alpha$  and  $\beta$ , and the term “magnitude” for their absolute value, which, once they are squared, provides the probability of that state.

However, it is fundamental to understand that the amplitude of the quantum system’s coefficient (how large it is), although related to the probability, does not describe the probability itself. But what is their relation? This is what Max Born studied, creating the famous Born Rule, which states that the square of the coefficients (if they are real numbers<sup>10</sup>) is, in fact, the probability (Schlatter, 2017).

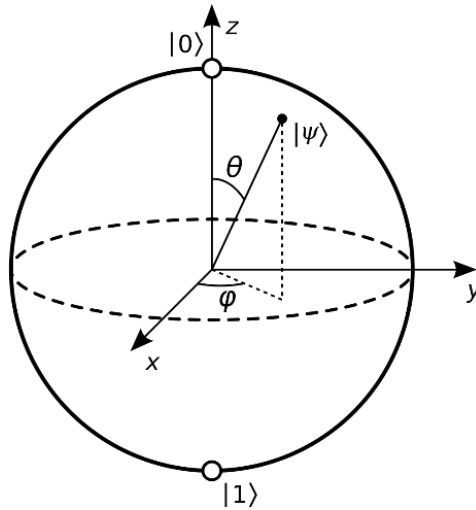
To visualize what was explained above, Felix Bloch came up with a great idea in 1946, which was later graphically represented by Feynman, giving birth to what is now known as the ***Bloch Sphere***

---

<sup>10</sup> If they are imaginary or complex numbers, the coefficient will be multiplied by its complex conjugate.

(Bloch, 1946; Feynman et al., 1957). This visual tool, represented in Figure 3.16 below, allows for the geometrical description of a qubit's quantum state:

**Figure 3.16: Bloch Sphere**

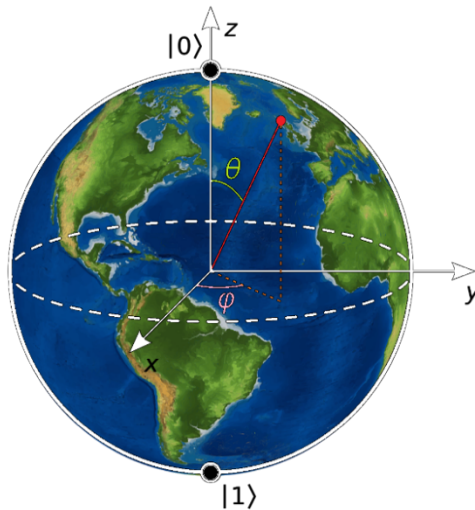


The state denoted by this sphere is described by a vector in a 3-dimensional space, given by  $|\psi\rangle$  in Figure 3.16. This vector will be a superposition of states  $|0\rangle$  and  $|1\rangle$ , but the total probability of the system must be 1. Therefore, the length of the vector representing the state of the qubit (the radius of the sphere) will always be 1. The angle  $\theta$  represents the probability of our state being 0 or 1. The closer the vector is pointing towards 0, the more likely it is to collapse into 0 when the qubit is measured, and vice versa. Besides, the angle  $\phi$  represents the difference in phase between both coefficients of our state.

Because these concepts can be rather challenging, the sphere of planet Earth can be used as a metaphor for the Bloch Sphere to understand each variable in play. In Figure 3.17 below, the quantum state is denoted by  $|\text{Scotland}\rangle$  because our quantum state vector points towards this region. This country has two properties that provide details about its position on Earth: Latitude and longitude. As in any world map, the Bloch sphere describes two fundamental properties of our quantum state as well. The latitude represents the probability of measuring 0 and 1 ( $\theta$ ), while the longitude is the difference in phase between both coefficients of our state ( $\phi$ ). In simple words, if our quantum state's "country" is in a colder region, it will be more likely to collapse into 0 or 1. In contrast, if our country is closer to the equator, it will be a total superposition (50-50%) between

0 and 1. On a different note, the difference in phase of our quantum state is given by the time zone of our country.

**Figure 3.17: Bloch Sphere Analogy**



In the example of quantum state  $| \textit{Scotland} \rangle$ , it is more likely to collapse into 0 because its latitude is closer to the north pole. Besides, if the prime meridian is time zone 1, and it is assigned to the start of our sphere, this example has phase 0. Hence, both coefficients of  $| \textit{Scotland} \rangle$  have the same phase because their subtraction is equal to 0.

### 3.2.7 Summary

This section has briefly introduced the main topics of quantum physics required for the understanding of quantum computing. From bracket notation to entanglement, observables, the uncertainty principle, and probabilities, all these concepts will be of crucial importance throughout the discussions offered in this thesis.

The following chapter focuses on the implementation of basic quantum computing techniques through *qiskit*, an environment offered by IBM to perform quantum computational tasks on actual quantum hardware on the cloud.

## 3.3 Quantum Programming with Qiskit

This section will deepen the programming knowledge required to engage with IBM's quantum computing devices. This will be done through *qiskit*, an open-source quantum computing software

development framework created by IBM. Qiskit was designed by researchers to allow users to interact with IBM’s quantum computers through a Python-based framework (*Qiskit*, n.d.). All the knowledge concerning quantum programming techniques with qiskit will have its foundations based on an educational course developed by IBM, which subdivided the required topics within quantum programming into different parts (*Quantum Explorers - IBM Quantum Experience*, n.d.).

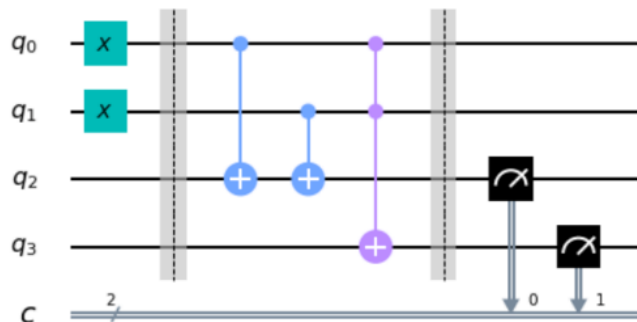
### 3.3.1 Qiskit Introduction

This section covers the fundamentals of qiskit, which will later be applied to the Quantum Machine Learning processes performed in the experiments. Through practice, the quantum concepts of superposition, entanglement, and interference are reviewed. This section also covers the logic behind quantum gates and how to manipulate them using qiskit and the IBM Quantum Platform. Finally, through qiskit, this section describes how to combine these quantum gates in order to run simple quantum circuits.

### 3.3.2 Qiskit Basics

In classical computing, circuits operate with bits through computing gates to obtain results. This same procedure is followed in quantum computing. Qiskit allows for the creation and usage of quantum circuits, which are models that represent operations being performed on qubits. These quantum circuits have the same three jobs that classical circuits perform: Encode the input, operate, and extract the output. The only difference is that these inputs, instead of bits, are qubits. Similar to bits, qubits are the most basic unit of information in quantum computers, and they have quantum properties. Quantum circuit diagrams are used to represent quantum circuits. Figure 3.18 below shows an example:

**Figure 3.18: Example of a Quantum Circuit Diagram**



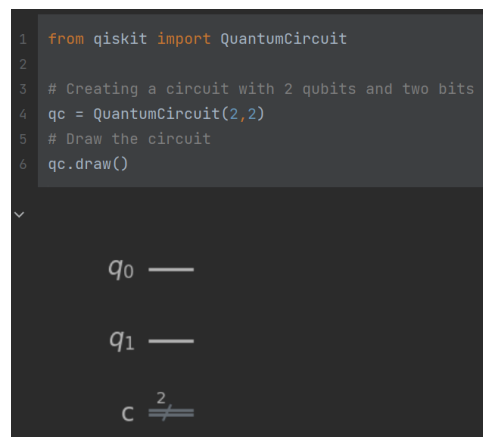
From this diagram, it can be inferred that there are three distinct stages of the circuit, separated by the dotted lines. Inputs are on the left, operations are performed in the middle, and output extraction occurs on the right. Each single line preceded by a “q” account for a qubit, starting at index 0. This circuit, for example, has four qubits. The double line at the bottom preceded by a “c” specifies the number of classical bits of the quantum circuit. Note that the number next to the “c” determines the number of bits, not the number of “double lines.” But why are classical bits required in a quantum circuit? They determine the value of a qubit once it is measured. In quantum circuits, bits are also indexed starting at 0.

To start developing quantum circuits with qiskit, you can find several tools in the IBM Quantum Platform (*IBM Quantum*, n.d.). In the “learning” section, click on the “IBM Quantum Lab” option. This is directed to a Jupyter notebook where the qiskit code can be run. Several important practices and techniques are required to understand how to interact with IBM’s quantum computers through qiskit.

To create a quantum circuit, the number of qubits and bits needed must be specified first. To do so, import the *QuantumCircuit* item from the *qiskit* library, which will allow for the creation of a *QuantumCircuit* class with, for example, two qubits and two bits. By using the *draw* method, create a diagram of the circuit, now encapsulated in the variable *qc*. See Figure 3.19:

**Figure 3.19: Creating a Quantum Circuit**

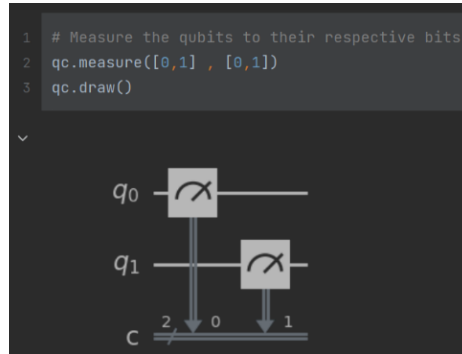
```
1 from qiskit import QuantumCircuit
2
3 # Creating a circuit with 2 qubits and two bits
4 qc = QuantumCircuit(2,2)
5 # Draw the circuit
6 qc.draw()
```



The *measure* method is used to measure the circuit’s qubits, recording their results into the circuit’s bits. Therefore, each qubit needs to be assigned to a bit of the quantum circuit. The first element

of the measure function should be an array of the qubits to be measured, and the second element an array of the bits they will be assigned to, as shown in Figure 3.20:

**Figure 3.20: Measuring Qubits**



For smaller circuits of less than 30 qubits, simulating a quantum computer can be beneficial in terms of computational power. That is why the *AerSimulator* is used to run certain circuits. For larger circuits, or if the user desires to run the quantum circuit on an actual quantum device, it is possible to use an IBM Quantum account. Figure 3.21 below imports the IBMQ module, which allows to connect to the IBM Quantum Experience. The *save\_account* method authenticates the access to IBM’s devices by passing the token assigned to the user account. The *load\_account* function is used to permit access to the quantum devices and simulators.

**Figure 3.21: Access to IBM Quantum Experience**

```
1 from qiskit import IBMQ
2 # Save account assigned to the token specified
3 IBMQ.save_account('enter_token_here')
4 # Load the account previously saved
5 IBMQ.load_account()
```

The code shown in Figure 3.22 creates QiskitRuntimeService instance that allows the user to connect to the hardware available through the token assigned to your IBM Quantum account. Once that is done, the backend is selected, with the parameter “simulator” set to false<sup>11</sup>. In this case, the quantum hardware set as the backend is “ibm\_brisbane.” As previously mentioned, quantum

---

<sup>11</sup> This version of the code will output the “least busy” hardware available.



computing (and, therefore, quantum circuits) works with probabilities. Hence, these circuits must be run several times to record the most probable outcomes.

**Figure 3.22: Backend Connection**

```
1 from qiskit_ibm_runtime import QiskitRuntimeService
2 # Select the 'ibm-quantum' channel to set up a service
3 service = QiskitRuntimeService(channel='ibm_quantum')
4
5 # Assign the first available quantum device as the backend
6 backend = service.backends(simulator = False)[0]
7 print(backend)

qiskit_runtime_service.__init__:INFO:2024-03-27 03:39:27,120:
<IBMBackend('ibm_brisbane')>
```

Figure 3.23 below shows the creation of an *AerSimulator* object stored in the variable *sim* after importing the *AerSimulator* class. Using this variable, the *run* method simulates the circuit in a quantum computer simulation. Next, the results are stored using the *result* method. Finally, the *get\_counts* function provides a visualization of the results, which will output the results in a dictionary. When designing circuits, all qubits start having a value of 0. In this case, the circuit did not contain any input encoding or gates, and it was run 1024 times. Therefore, all 1024 times measured 0 for both qubits.

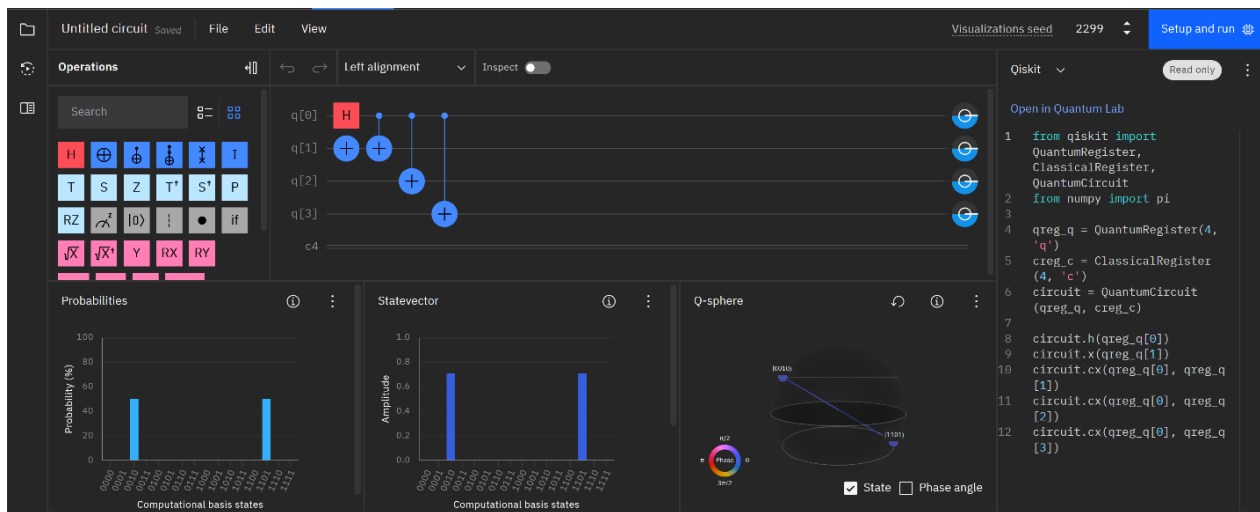
**Figure 3.23: Aer Simulator Example**

```
1 from qiskit.providers.aer import AerSimulator
2 # Create an instance of simulator
3 sim = AerSimulator()
4
5 # Run existing circuit on the simulator
6 job = sim.run(qc)
7 # Retrieve the job result
8 result = job.result()
9 # Count the results
10 result.get_counts()

{'00': 1024}
```

There is also an interactive way to manipulate quantum circuits besides raw programming with qiskit. In the “learning” section of the IBM Quantum Platform, find the Quantum Composer, which allows you to graphically build circuits. Figure 3.24 is an example of how a circuit is built by using the IBM Quantum Composer:

**Figure 3.24: IBM Quantum Composer**



In the top left corner, the application shows the different operations that can be performed on the circuit’s qubits. The top middle section is the composer, where the qubit operations can be dragged and dropped. You can add or delete qubits by clicking on the “q’s.” Follow the same procedure with the “c” to add bits. The left menu will translate whatever is built on the composer to code. The visuals displayed at the bottom provide insights into the possible outcomes of the circuit. As circuits get more complex, the “Quantum Composer” is truly a valuable tool since it makes the quantum circuit design process much quicker and intuitive.

### 3.3.3 Quantum Gates

However, as you may infer, it is sometimes the case that the input is different than 0. This can be done by using quantum gates. The behavior of quantum gates can be portrayed through an intuitive metaphor previously used<sup>12</sup>, with Earth as an analogy. When traveling between two different countries, a route traces the path connecting both points. This route covers all the necessary turns,

<sup>12</sup> See Section 3.2.6 – Probability and the Bloch Sphere.

distances, and travel directions required to arrive at the destination. That is exactly the job of a quantum gate. They alter the qubit’s state so that the Bloch sphere points towards a different direction. Nonetheless, instead of using a GPS, quantum gates are expressed through transformation matrixes. By multiplying the input state by the quantum gate’s transformation matrix, the resultant state of the qubit<sup>13</sup>, which is the encoded input, is obtained.

### 3.3.3.1 X Gate

For example, the X gate flips the value of the qubit in question from 0 to 1 or vice versa by using the following matrix as a function (see Equation 3.43):

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.43}$$

Imagine the qubit is in state  $|0\rangle$ . Therefore, the vector state is  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ . Picture passing the qubit through an X gate; the resultant state of the qubit will be given by the procedure shown in Equation 3.44:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} (0 \cdot 1) + (1 \cdot 0) \\ (1 \cdot 1) + (0 \cdot 0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.44}$$

Therefore, the transformation matrix of the X gate serves as a roadmap that states where the quantum state will land after “traveling” through the X quantum gate. These “trips” are often referred to as “rotations of the sphere.” In this case, a rotation of 180° is performed around the X-axis<sup>14</sup>. Figure 3.25 shows the “trip” performed by the quantum state, which, in this case, covers the distance from the north pole to the south pole (0 to 1).

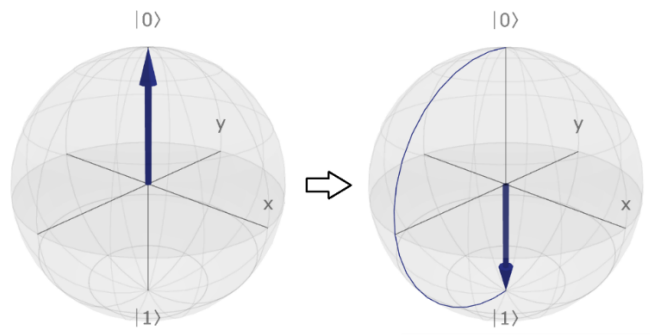
Now, it is possible to interact with this quantum gate through qiskit through the following quantum circuit. The `x` method performs the X gate on the qubits comprehended within the parameter, which must be represented in an array. Figure 3.26 portrays an example.

---

<sup>13</sup> See Section 3.1.4 – Matrixes.

<sup>14</sup> By convention, these rotations are always counterclockwise.

**Figure 3.25: X Gate on  $|0\rangle$**



**Figure 3.26: X Gate**

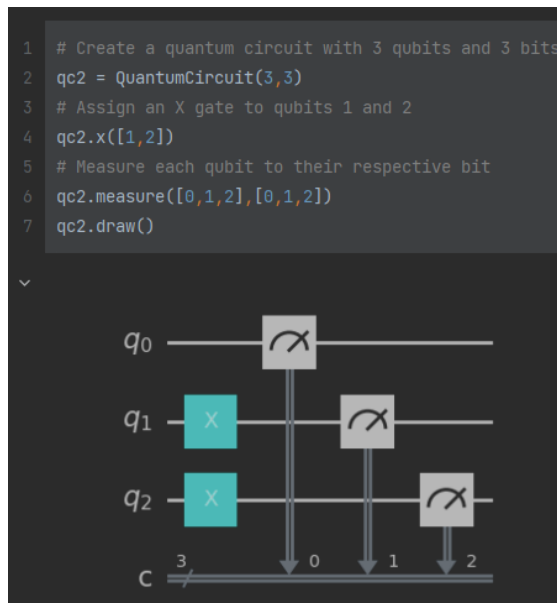


Figure 3.27 shows the simulator results after encoding two qubits as 1. Note how the result is “110” every time the circuit is run. This is because after measuring a circuit, results are portrayed from right to left. Therefore,  $q_0$  is stated in the furthest right of the results,  $q_1$  in the middle, and  $q_2$  in the furthest left:

**Figure 3.27: X Gate Simulation.**

```
1 # Retrieve results
2 job = sim.run(qc2)
3 result = job.result()
4 result.get_counts()

{'110': 1024}
```

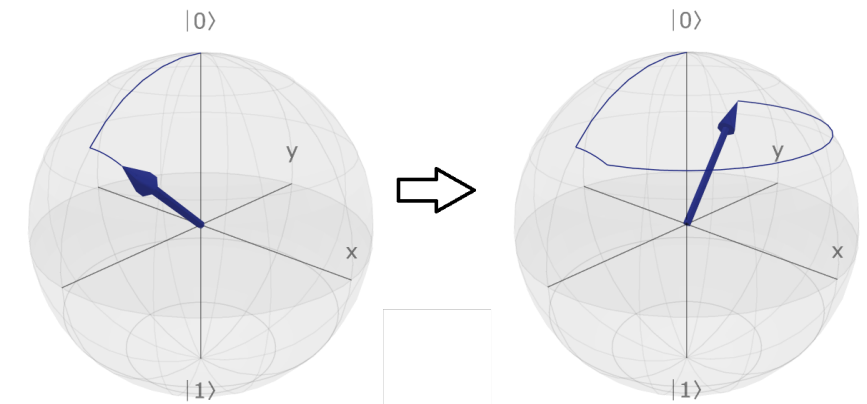
### 3.3.3.2 Z Gate

Similarly to the X gate, the Z gate is also used on a single qubit. This gate focuses on changing the phase of the qubit input through a rotation of  $180^\circ$  around the Z axis. Therefore, in this analogy, the itinerary set by the Z gate describes a route perpendicular to the equator without any movement in the north-south component. The Z gate is described by the following transformation matrix of Equation 3.45:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.45}$$

Note that if the input qubit is in states  $|0\rangle$  or  $|1\rangle$  the Z gate will not have any effect on the resultant state of the qubit. When performing the same mathematical procedure as before, the outcome is that the quantum state does not change. This process can be understood by considering that the vector states of  $|0\rangle$  and  $|1\rangle$  lie within the Z axis. This means that they do not have any component in the X and Y axes. Consequently, applying the Z gate brings no change in the resultant state vector. Figure 3.28 below describes the effect of the Z gate on a quantum state  $|W\rangle$ .

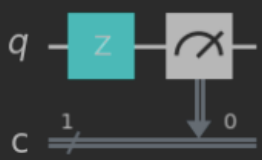
**Figure 3.28: Z Gate on  $|W\rangle$**



In qiskit, the `z` method implements a Z gate on a qubit, as shown in Figure 3.29. However, the nature of this gate will not affect the probability of the qubit collapsing into 0 or 1. Instead, it influences the phase of the qubit, as previously stated. This feature can be highly valuable when manipulating several qubits at the same time, as will be seen in future sections.

**Figure 3.29: Z Gate**

```
1 # Create a quantum circuit with 1 qubit and 1 bit
2 qc3 = QuantumCircuit(1,1)
3 # Apply the Z gate
4 qc3.z([0])
5 # Measure qubit 1 in bit 1
6 qc3.measure([0],[0])
7 qc3.draw()
```



The diagram shows a quantum circuit with one qubit line labeled 'q' and one classical bit line labeled 'c'. The qubit line passes through a green square gate labeled 'Z'. After the Z gate, the qubit line enters a measurement gate, represented by a square with a meter symbol. An arrow points from the measurement gate to the classical bit line 'c', which is labeled with the value '0'.

### 3.3.3.3 H Gate

All the operations performed on the bits so far are deterministic. This means, there is no randomness associated with the operation itself. However, randomness and probability are fundamental characteristics of quantum mechanics. Therefore, qubits take advantage of these properties to work with probabilities.

Nonetheless, an important question is how to interact with a qubit's probability. Examine the behavior of the most famous probabilistic gate in quantum computing: The H gate. This transformation matrix accounts for a rotation of the Y axis of 90° followed by a rotation of the X axis of 180°, described in Equation 3.46:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.46}$$

When applying the Hadamard gate to a qubit in state  $|0\rangle$ , it will transform the Bloch sphere as follows<sup>15</sup> (see Figure 3.30):

---

<sup>15</sup> After the Y axis rotation, the X axis rotation does not affect the state, because the vector state is part of the rotating axis. Therefore, there is no change after the X rotation is applied.

**Figure 3.30: H Gate on  $|0\rangle$**

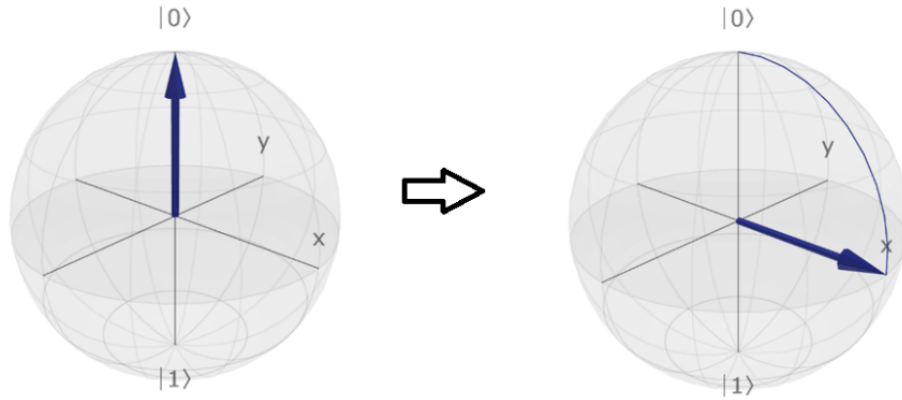
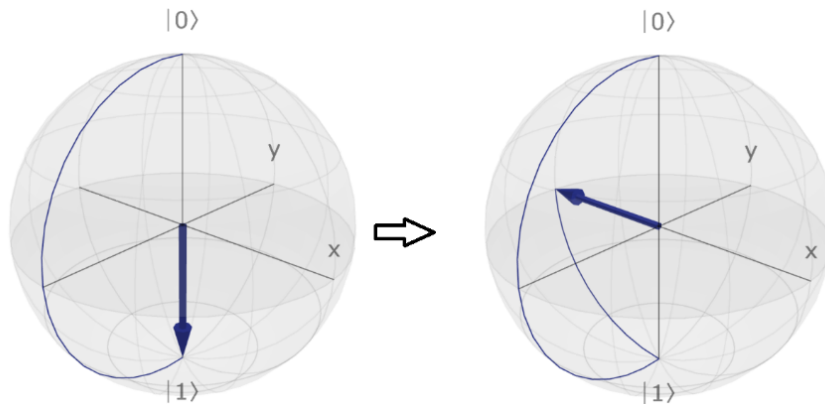


Figure 3.30 suggests that the H gate splits the probability of measuring 0 and 1 into 50-50%<sup>16</sup>. This can lead to believing that the H gate just randomizes the output of the qubit, but this is an incorrect assumption. As it can be inferred from Figure 3.31 below, when the input is in state  $|1\rangle$ , although the probability is split evenly as well, the phase is different from the result from Figure 3.31. This is because after applying the same “plane itinerary” but departing from various locations, the final destination is different:

**Figure 3.31: H Gate on  $|1\rangle$**



---

<sup>16</sup> Note that what is stated in Figure 3.30 are the quantum state’s coefficients. To retrieve their probability, square such coefficient.

Figure 3.32 shows how the H gate includes a phase change, which can be determined when applying multiple gates in a row. In fact, after applying two H gates consequently, there is no change in the output because the rotations will cancel each other out:

**Figure 3.32: Consecutive H Gates on  $|0\rangle$**

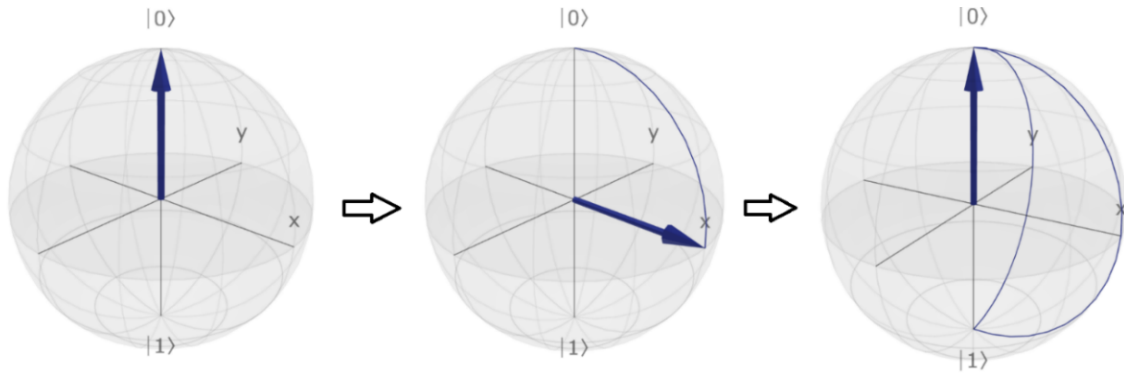
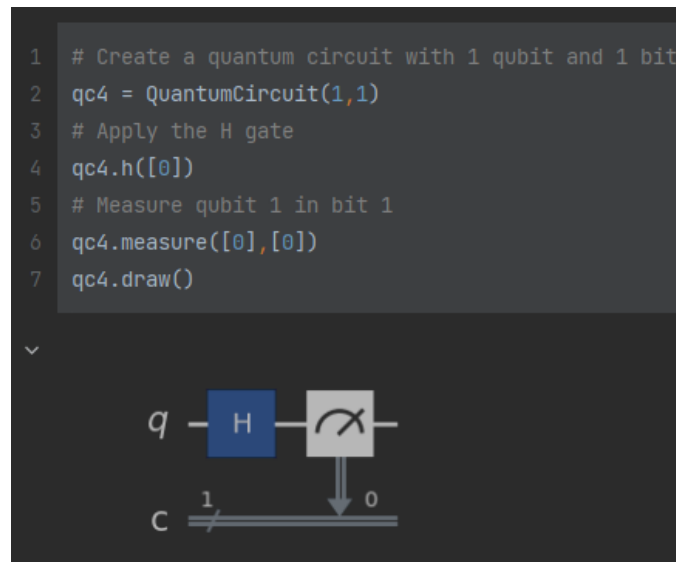


Figure 3.33 portrays the measurement and consequent results of applying the Hadamard gate on a single qubit, utilizing the same methods previously used with the X gate combined with the method *h*:

**Figure 3.33: H Gate**



The outcome shows an even distribution of 0 and 1, suggesting that every time the circuit was run (1024 times), the qubit had a 50-50% probability of collapsing into both possible values (See Figure 3.34):



**Figure 3.34: H Gate Simulation**

```
1 # Create a quantum circuit with 1 qubit and 1 bit
2 qc4 = QuantumCircuit(1,1)
3 # Apply the H gate
4 qc4.h([0])
5 # Measure qubit 1 in bit 1
6 qc4.measure([0],[0])
7 # Retrieve and count results through Aer simulator
8 sim = AerSimulator()
9 job = sim.run(qc4)
10 result = job.result()
11 result.get_counts()

{'1': 519, '0': 505}
```

### 3.3.4 Qubit Entanglement

Working with two qubits unleashes the power of entanglement, as reviewed in Section

3.2.3. Through entanglement, it is possible to manipulate qubits whose value has not yet collapsed but whose output is certain. Because entanglement is a multi-qubit property, multi-qubit gates must be applied to entangle several qubits. The CNOT and CX gates are foundational examples of multi-qubit gates.

#### 3.3.4.1 CNOT Gate

This gate works on two qubits, the “control” and “target.” If the control qubit is in state  $|0\rangle$ , the CNOT gate will not perform any operation on the target qubit. Whereas if the control qubit is in state  $|1\rangle$ , it will perform an X gate on the target qubit. Equation 3.47 portrays the CNOT gate’s transformation matrix:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.47}$$

Given state  $|10\rangle$ , once the CNOT gate is applied, follow the mathematical reasoning to arrive at the resultant quantum state (see Equation 3.48 below):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1(0) + 0(0) + 0(1) + 0(0) \\ 0(0) + 1(0) + 0(1) + 0(0) \\ 0(0) + 0(0) + 0(1) + 1(0) \\ 0(0) + 0(0) + 1(1) + 0(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(3.48)

As it can be inferred, because the target qubit (on the left) was in state  $|1\rangle$ , the target qubit was flipped from  $|0\rangle$  to  $|1\rangle$ . Similarly, the CNOT gate will convert the state  $|11\rangle$  into  $|10\rangle$ . However, it will not have any effect with the states  $|00\rangle$  and  $|01\rangle$  because the control qubits are in state  $|0\rangle$ . In qiskit, as shown in Figure 3.35, utilize the CNOT gate by using the method `cx`.

**Figure 3.35: CNOT Gate**

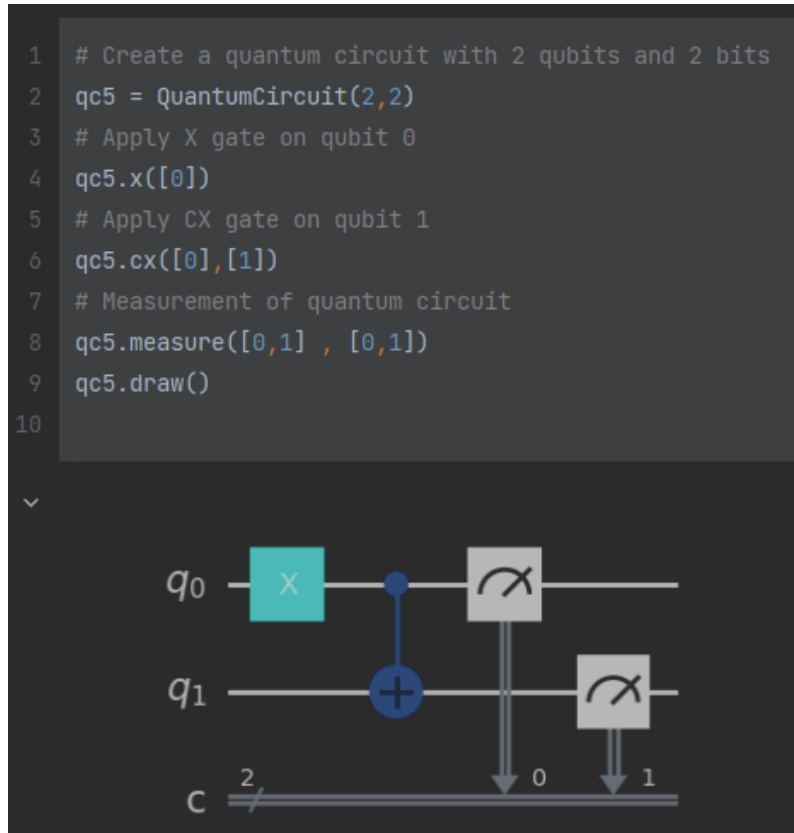


Figure 3.36 describes the process of simulating the results of circuit *qc5*. A quantum state encoded as  $|10\rangle$ , after going through a CNOT gate, will always output the state  $|11\rangle$  every single time the circuit is run.

**Figure 3.36: CNOT Gate Simulation**

```
1 # Create a quantum circuit with 2 qubits and 2 bits
2 qc5 = QuantumCircuit(2,2)
3 # Apply X gate on qubit 0
4 qc5.x([0])
5 # Apply CX gate with qubit 0 as control and qubit 1 as target
6 qc5.cx([0],[1])
7 # Measurement of the quantum circuit
8 qc5.measure([0,1] , [0,1])
9
10 # Retrieve and count results through Aer simulator
11 sim = AerSimulator()
12 job = sim.run(qc5)
13 result = job.result()
14 result.get_counts()

{'11': 1024}
```

### 3.3.4.2 CZ Gate

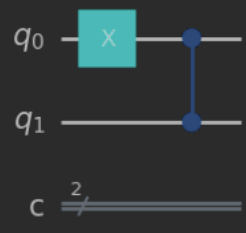
As with the X and Z gates, the CNOT and CZ gates operate similarly. The CZ gate takes effect by applying a Z gate on the control qubit only if the target qubit is in state  $|1\rangle$ . Otherwise, it will leave it unchanged. Therefore, the transformation matrix of the CZ gate can be formulated as in Equation 3.49:

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{3.49}$$

Use the method `cz` to call the CZ gate in qiskit, as portrayed in Figure 3.37:

**Figure 3.37: CZ Gate**

```
1 # Create a quantum circuit with 2 qubits and 2 bits
2 qc6 = QuantumCircuit(2,2)
3 # Apply X gate on qubit 0
4 qc6.x([0])
5 # Apply CZ gate with qubit 0 as control and qubit 1 as target
6 qc6.cz([0],[1])
7 qc6.draw()
8
```



### 3.3.4.3 Entangling Qubits

With the use of the CNOT gate and the CX gate, it is possible to create entangled states. For example, Figure 3.38 describes a circuit of two qubits, and an H gate is applied on the first qubit (specified in the far right of the bra-ket), leaving the second qubit unchanged in state  $|0\rangle$ . Note the use of the *Statevector* item from the *qiskit.quantum\_info* library, which quickly calculates and represents the state vector of the quantum state:

**Figure 3.38: Entangling Qubits (1)**

```
1 from qiskit import QuantumCircuit
2 from qiskit.quantum_info import Statevector
3
4 # Create a quantum circuit with 2 qubits
5 qc7 = QuantumCircuit(2)
6 # Apply H gate on qubit 0
7 qc7.h(0)
8 # Output the state vector representation
9 ket = Statevector(qc7)
10 ket.draw('latex')
11
```

$$\frac{\sqrt{2}}{2} |00\rangle + \frac{\sqrt{2}}{2} |01\rangle$$

Therefore, the only two possible resultant states are  $|00\rangle$  and  $|01\rangle$  since the rightmost qubit (the first qubit) is the only one that has been altered by the H gate. It is important to remember that this representation shows the quantum states' coefficients, not their probability. As explained in Section 3.2.6, it is required to square these coefficients to find the probability of the qubit collapsing into said quantum state. Now, the CNOT gate can be applied with the control on the first qubit and the target on the second qubit. See Figure 3.39 below:

**Figure 3.39: Entangling Qubits (2)**

```

1 from qiskit import QuantumCircuit
2 from qiskit.quantum_info import Statevector
3
4 # Create a quantum circuit with 2 qubits
5 qc7 = QuantumCircuit(2)
6 # Apply H gate on qubit 0
7 qc7.h(0)
8 # Apply CX gate with qubit 0 as control and qubit 1 as target
9 qc7.cx(0,1)
10 # Output the state vector representation
11 ket = Statevector(qc7)
12 ket.draw('latex')
13


```

$$\frac{\sqrt{2}}{2}|00\rangle + \frac{\sqrt{2}}{2}|11\rangle$$

It can be inferred that every time the first qubit is measured and outputs, for example,  $|0\rangle$ , the CNOT gate will operate on the second qubit, converting it into  $|0\rangle$  as well. However, if the first qubit is uncovered as  $|1\rangle$ , the CNOT gate will not perform any operation on the second qubit, leaving it in state  $|1\rangle$  too. As portrayed by the vector state in Figure 3.40, the only two possible states of this circuit after being measured are either  $|00\rangle$  or  $|11\rangle$ . Hence, once the value of one of the qubits is known, the value of the other one is intuitively known, too. This is the logic followed to successfully entangle two qubits by just using two gates through the following quantum circuit:

**Figure 3.40: Entangling Qubits (3)**

```
1 from qiskit import QuantumCircuit
2 # Create a quantum circuit with 2 qubits
3 qc7 = QuantumCircuit(2)
4 # Apply H gate on qubit 0
5 qc7.h(0)
6 # Apply CX gate with qubit 0 as control and qubit 1 as target
7 qc7.cx(0,1)
8 qc7.draw()
9
```



The diagram shows two horizontal lines representing qubits  $q_0$  and  $q_1$ . On the  $q_0$  line, there is a blue square labeled 'H'. A vertical line connects a blue dot on the  $q_0$  line to a blue circle with a plus sign on the  $q_1$  line, representing a CNOT gate.

This example highlighted how to entangle two qubits, but it is possible to entangle more as far as the correct circuit is designed. It has been shown that circuits with more than two qubits can be entangled through different circuit configurations. The more qubits in the circuit, the more possible configurations to achieve entanglement (Dür et al., 2000; Verstraete et al., 2002)

## Chapter 4: Experiments and Results

With a clear understanding of qiskit basics and how to utilize quantum gates, it is now possible to apply these concepts for QML purposes. This section is divided into three different subsections. First, it will discuss the theoretical principles of KQSVM required to understand the experiments, which will be discussed later. Following that, the methodology will present the process through which the experiments were conducted. Ultimately, the results obtained from such experiments will be reviewed.

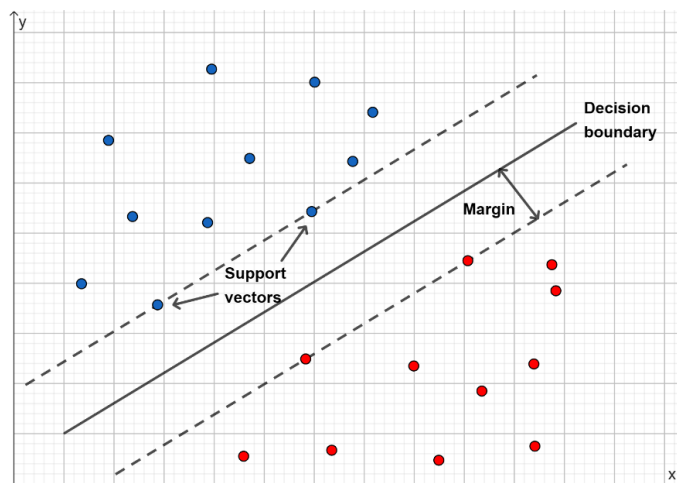
### 4.1 QKSVM Theory

As mentioned in Chapter 1, the experiments will specifically focus on QKSVM. This subsection provides an in-depth understanding of this technique's background.

#### 4.1.1 SVM

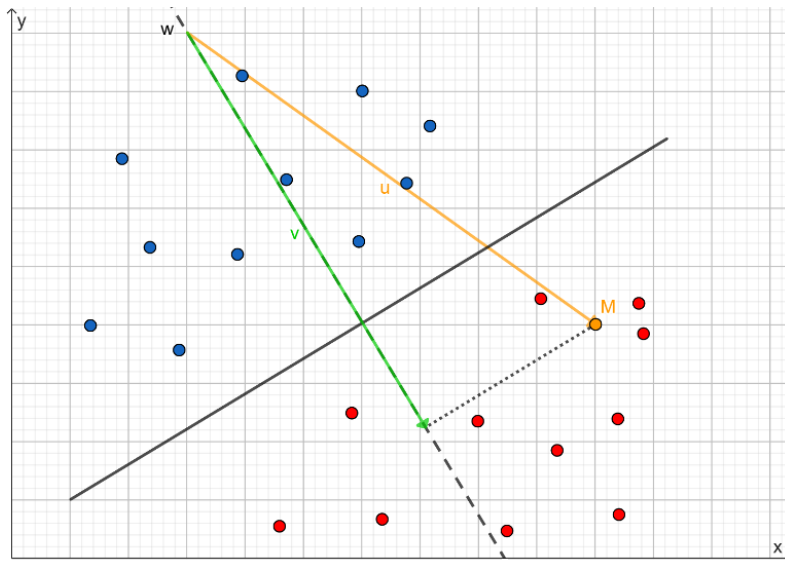
SVM is a supervised learning algorithm that focuses on binary classification (Rebentrost et al., 2013). This means that the algorithm's goal is to separate the dataset into two classes by comparing the similarity between the data points of the dataset. Through an optimization process, SVMs output the "decision boundary," the line that permits the successful categorization of data points (Protopapas & Rader, 2018). The vectors lying closer to the hyperplane are called "support vectors," and the distance between support vectors and the hyperplane is denominated "margin" (Figure 4.1).

**Figure 4.1: Example of SVM**



To create this decision boundary, the algorithm follows an optimization process that maximizes the distance between both classes of the dataset while maintaining the least possible number of misclassifications (Protopapas & Rader, 2018). This is known as the “widest street” approach (Winston, 2010). The power of SVM lies in the fact that they can also be rephrased into linear algebra terms, with the inner product as the main concept when calculating the decision boundary (Brownlee, 2020). Essentially, once the decision boundary is set, the algorithm defines the “weight vector” (“ $w$ ” in Figure 4.2), which is perpendicular to the decision boundary. It then creates a vector from a point within the “ $w$ ” to the new data point, “ $M$ .” This new vector is referred to as “ $u$ ” in Figure 4.2. After calculating the projection of “ $u$ ” onto “ $w$ ” (through the dot product), it is possible to determine on which side of the decision boundary “ $M$ ” lands. This projection is described by “ $v$ ” in Figure 4.2. In this example, because “ $v$ ” falls under the “red” class,  $M$  will be successfully labeled as “red.” The dot product, therefore, functions as a sort of “similarity score” between the data points and the weight vector, a concept that will be fundamental for the understanding of quantum kernels in Section 4.1.3 (Schuld, 2019a).

**Figure 4.2: SVM Classification of a New Data**

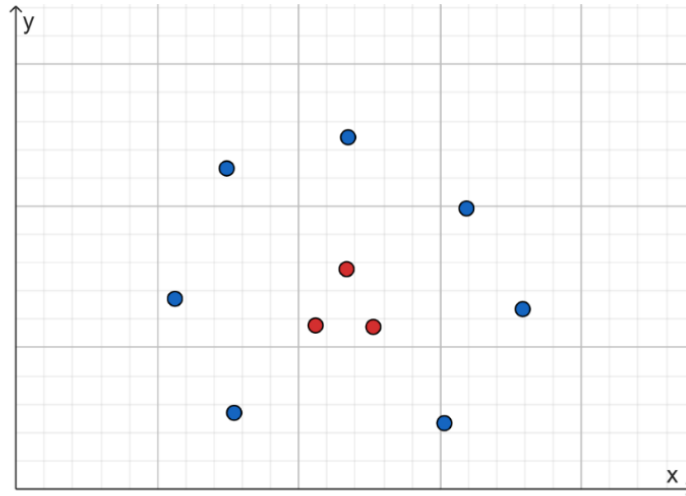


### 4.1.2 Kernels

However, the approach followed by SVMs can be truly challenging when trying to classify a non-linearly separable dataset. As can be seen, Figure 4.1 is easy to separate through a line, but certain datasets cannot be classified linearly. For example, Figure 4.3 shows an example where no line will be able to differentiate both classes (red and blue):

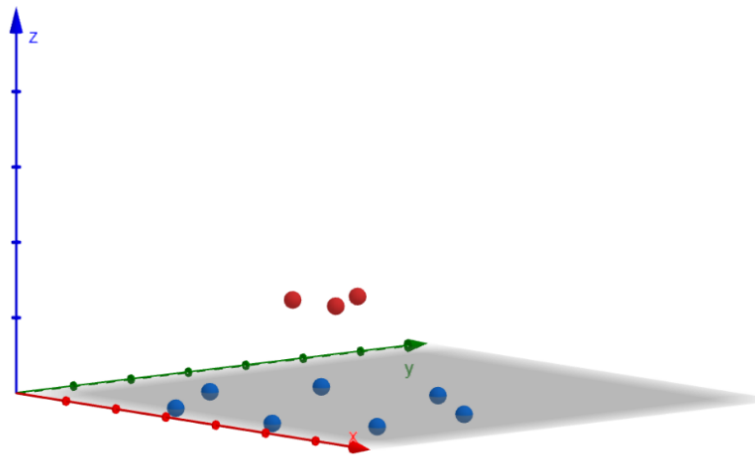


**Figure 4.3: Non-Linearly Separable Dataset**



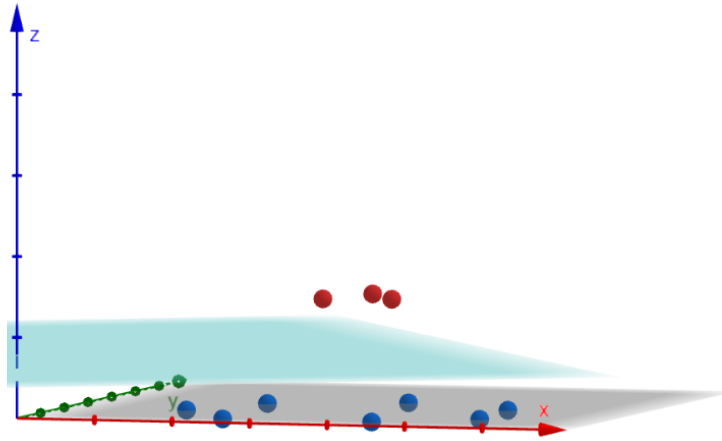
This sort of case can be dealt with by applying a linear transformation to the points to find whether, in a higher dimensional space, these classes can be accurately separated. For example, when transforming Figure 4.3’s data points into a three-dimensional space, the data point distribution found in Figure 4.4 can be obtained:

**Figure 4.4: Dataset Transformation into 3 Dimensions**



As can be inferred, this dataset is easily separable by a 3-dimensional plane, portrayed in Figure 4.5 below. This decision boundary is also known as a “hyperplane,” and it is the output of the SVM after the transformation has taken place (Protopapas & Rader, 2018).

**Figure 4.5: Hyperplane Separating Both Classes**



Nonetheless, there are two main problems with this approach:

- Finding the characteristics of the linear transformation that allows for hyperplane class separation.
- When dealing with datasets of high complexity, a considerable number of dimensions may be required.

Both challenges can be solved by using the kernel trick (El Khadir, 2022). This concept establishes that the algorithm behind SVM does not need to know the position where each data point is mapped after the linear transformation, which is a challenging task due to its computationally expensive nature (Weinberger, 2018). It is sufficient to know the similarity score provided by the inner product (represented through “<>” in Equation 4.1) of both data points in the new space provided by the kernel function (Winston, 2010). A kernel function “k” follows the logic portrayed in Equation 4.1, given a linear transformation “ $f(x)$ ” and data points  $x$  and  $x'$ :

$$k(x, x') = \langle f(x), f(x') \rangle \tag{4.1}$$

This equation states that it is possible to know the similarity score (inner product) of two vectors in a higher dimensional space with only three variables: Both data points and the kernel function.

Different types of kernels can be used to map data points to new feature spaces. The linear kernel operates in the original feature space. Referring to Figure 4.1, the decision boundary portrayed in

this graphic example would be what a linear kernel would output. A polynomial kernel of degree “d,” nonetheless, maps the data to a higher dimension, depending on the degree “d”<sup>17</sup> (Kutzkov, 2022). The sigmoid kernel operates through the sigmoid function, offering a mapping to a higher dimensional space, too (Sitorus, 2020).

### 4.1.3 Quantum Kernels

The kernel trick can serve as the bridge between SVM and quantum computing. To approach this idea, it is necessary to understand the concept of “overlap.” This term is referred to as the measurement of similarity between two quantum states. For example, picture two quantum states, as portrayed in Equation 4.1:

$$\begin{aligned} |\psi\rangle &= a |0\rangle + b |1\rangle \\ |\psi'\rangle &= c |0\rangle + d |1\rangle \end{aligned} \tag{4.2}$$

This is calculated through the inner product of both quantum states, known as “fidelity.” Recall the concept of the dot product in linear algebra, which projects a vector onto another. In this case, fidelity looks at how similar two quantum states are by projecting one onto the other. See Equation 4.3 below for the calculation of the inner product:

$$\langle \psi | \psi' \rangle = (a^* \cdot c) + (b^* \cdot d) \tag{4.3}$$

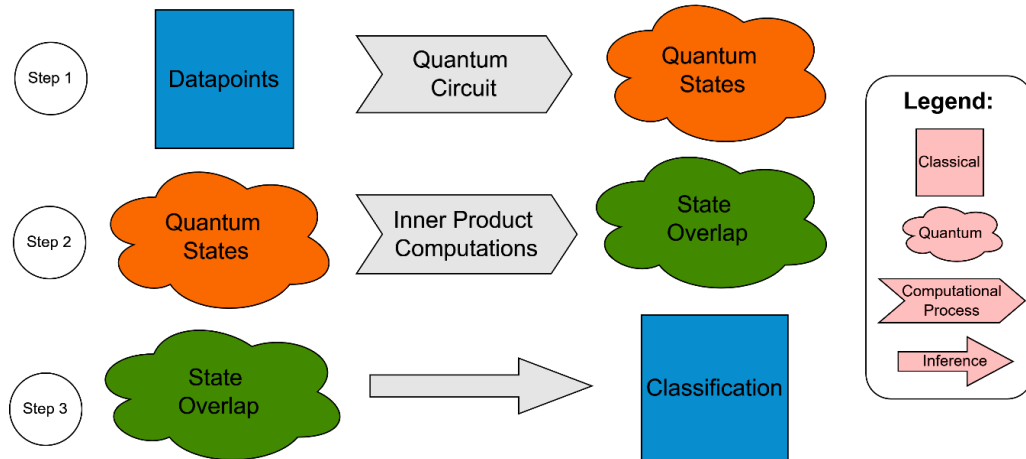
Essentially, the overlap measures how similar two quantum states are. And as you may recall, that is exactly what a kernel performs on data points. To translate data points into quantum states, quantum computers use quantum circuits. Using the same quantum circuit, data points are converted into quantum states. These quantum states are feature vectors of the new feature space (the Hilbert space). Quantum computers can then calculate the quantum states’ inner products (overlaps) to efficiently compute the similarity between data points, acting as a “quantum kernel” (Schuld, 2019b). Hence, while the classical kernels require a kernel function to compute the inner

---

<sup>17</sup> As it will be shown in the chapter 4, python sets it by default to d = 3 when it is not specified.

products, quantum kernels directly compute the inner products in the new feature space. Figure 4.6 describes the process outlined:

**Figure 4.6: Quantum Kernel Process**



## 4.2 Experiments

This section of the thesis covers all the information about the process of developing the experiments and how they were performed. It deepens into the details of how to load classical data into a quantum computer, setting examples of the actual experiments. Furthermore, it explains the necessary steps to create a quantum kernel and test it on a quantum simulator. Finally, the results of the experiments will be discussed, and the comparison between quantum and classical kernels will be analyzed.

Before starting the discussion concerning the framework of the experiments performed, it is fundamental to examine the first experiments conducted throughout the experimental process. Before developing this thesis, the aim was to find scenarios where QML can enhance current ML processes. That is why the first experiments performed operated with uncomplicated datasets, where a classical SVM was utilized before showing robust results in terms of accuracy and other performance metrics. Some of the datasets utilized in this first experimentation stage were the “iris,” “palmer penguins,” and “breast cancer” datasets. Nonetheless, the results when comparing classical and quantum algorithms in these scenarios did not provide any insights concerning quantum advantage. The results showed that QSVMs performed equally (if not worse in certain

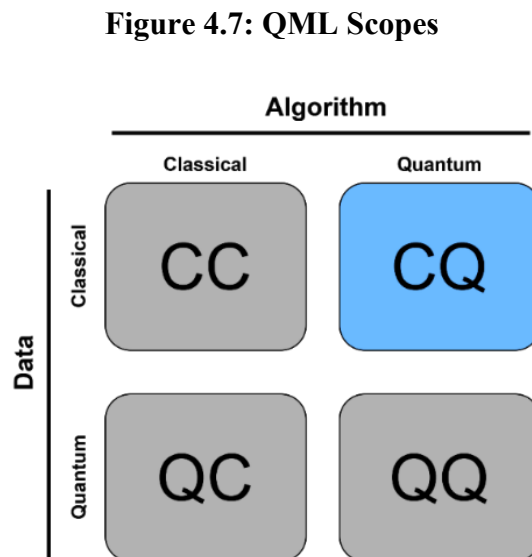
cases) than classical SVMs. In those situations, a QSVM does not seem to enhance the performance of a classical SVM.

In light of these findings, a new different type of dataset had to be found. As previously mentioned, this research aimed to show a quantum advantage for certain specific situations, and the answer did not seem to hide behind extensively used datasets. As will be explained later, Liu et al. (2021) mentioned a theoretical edge of QSVM over classical SVMs for datasets following a DLP distribution. Therefore, the experiment design structure pivoted to this novel approach.

Note that the code used to develop these experiments received support and intuitive guidelines from a large language model like ChatGPT, enabling the practical implementation of these theoretical concepts (OpenAI, 2023).

#### 4.2.1 Methodology

Depending on the classical nature (quantum or classical) of the resources (data and algorithm) of the machine learning process, there are four types of techniques that can be applied in QML (Dunjko et al., 2016). Figure 4.7 below shows the different options:



CC (classical-classical) concerns classical data processed by a classical unit, which is the general goal of ML. QC (quantum-classical) references quantum data processed by classical algorithms. This is denominated “quantum applied ML,” an area still in development. QQ (quantum-quantum) refers to quantum data being processed by a quantum algorithm or “Quantum generalized ML,” a field currently in ongoing research as well. Finally, CQ (classical-quantum) is a technique that

utilizes quantum algorithms to deal with classical data: “Quantum enhanced ML” (Pira & Ferrie, 2023). CQ is indeed the field that this thesis focuses on, analyzing whether quantum kernels can enhance SVM techniques in classical datasets.

The CQ technique requires two different elements, nonetheless: A classical dataset and a quantum algorithm. The next two subsections will dive into both components.

#### 4.2.1.1 Discrete Logarithm Problem Dataset

As Liu, Arunachalam, and Temme (2021) indicated, the DLP supposes a struggle for classical algorithms. This mathematical problem is used in cryptography, where messages are encrypted using the DLP, for example, in the ElGamal encryption scheme (Menezes et al., 2020, p. 297). Essentially, the DLP involves finding the exponent “ $x$ ” to which a given generator “ $g$ ” must be raised to be congruent with a specific element within a finite cyclic group “ $s$ ” modulo “ $p$ ,” where “ $p$ ” is a prime number (Galaviz, 2021; Trevisan, 2012). This relationship is described in Equation 4.4:

*Given  $g, p,$  and  $s,$  find  $x$ :*

$$g^x \equiv s \pmod{p} \tag{4.4}$$

In the paper “A rigorous and robust quantum speed-up in supervised machine learning,” a concept class is formulated so that a KQSVM can aim at classifying a DLP dataset (Liu et al., 2021). To perform the experiments, this thesis followed the same concept class to generate a DLP dataset through the following steps:

1. Selecting a prime number  $p$  and a generator (primitive root)  $g$ .
2. Randomly selecting an integer  $s$  for the set  $\{0, 1, \dots, p - 2\}$ .
3. For  $M$  data samples:
  - Randomly select an integer  $z$  from set  $\{0, 1, \dots, p - 2\}$ .
  - Store data points  $(z, 1)$  or  $(z, -1)$  depending on:
    - If  $z$  is in the range  $[s, s + \frac{p-3}{2}]$ , then label 1.

- If not, then label -1.
- Calculate  $x = g^z \bmod p$ .
- Associate  $x$  with the label obtained.

Note that for this thesis' experimental analysis  $M = 300$ . This is due to the limitations imposed by IBM's servers, which did not permit handling computations involving datasets with a higher number of data points.

#### 4.2.1.2 Experimental Design

Several variables and procedures were considered to perform a successful empirical analysis of how a quantum kernel can classify a DLP dataset compared to classical kernels.

First, three different DLP datasets were generated following the process outlined in Section 4.2.1.1. These datasets were derived from different prime numbers: small (64 bits), medium (512 bits), and large (1024 bits). Despite the attempt to increase the prime number's magnitude beyond 1024 bits, IBM's servers now allowed the manipulation of a dataset of such dimensions.

Then, for each of the three datasets, 25 different runs were performed. For each run, a different train-test seed was chosen. This allows for a change in the configuration of the records selected to be part of the train and test groups, which helps in understanding whether the models' performances are consistent and stable based on the data split.

Finally, the experiments involved the utilization of three classical kernels (linear, polynomial, and sigmoid) alongside a quantum kernel. This allows for a comparison between the conventional classical options and the alternative offered by quantum computing in terms of kernelized SVMs.

##### 4.2.1.2.1 Data Encoding

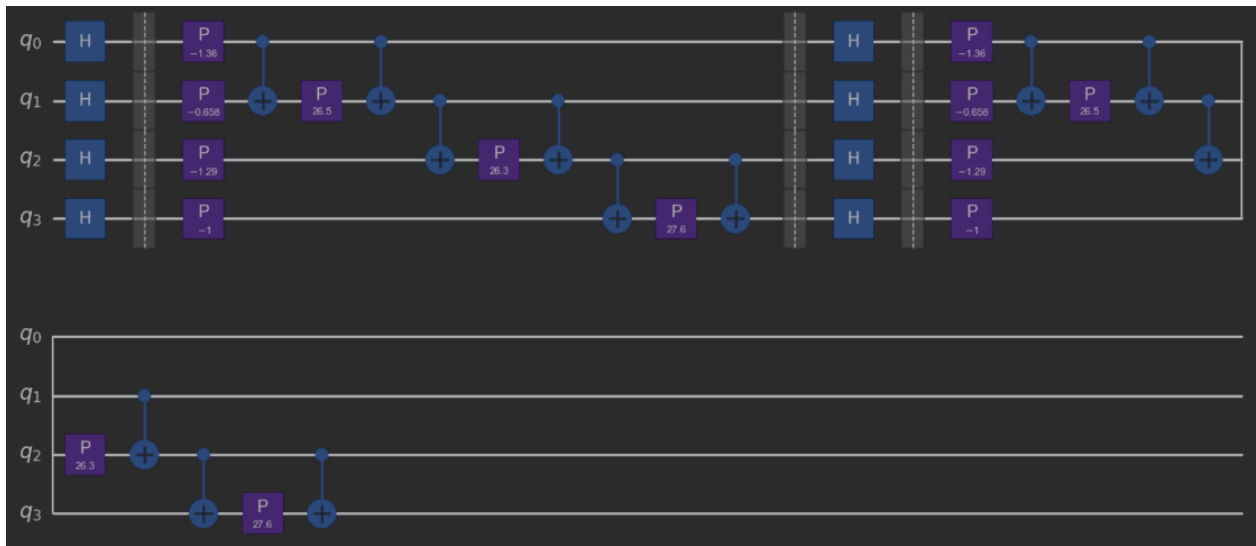
Before proceeding with the creation of a quantum kernel, there is one fundamental step to perform: Data encoding. This process is about translating classical data into a quantum computer. There are several suggested methods for data encoding, and each is used for diverse tasks (Ganguly, 2021, p. 216). This thesis will focus on "Basis Encoding" through the *ZZFeatureMap* quantum circuit found in qiskit.

Basis encoding focuses on assigning classical input (data points) into basis states. Recall the discussion about basis states from Section 3.2.3. For a 2-qubit system, the basis states are  $|00\rangle$ ,

$|01\rangle$ ,  $|10\rangle$ , and  $|11\rangle$ ). Basis encoding allows for the translation of four data points into quantum states with this system, one for each basis state. Given data points  $x_1, x_2, x_3$ , and  $x_4$ , they must be first transformed into a binary representation such that  $x_1 = 00$ ,  $x_2 = 01$ ,  $x_3 = 10$ , and  $x_4 = 11$ . Then, basis encoding creates a circuit to associate each binary form with its respective basis state. In this case, after a successful data encoding process,  $x_1 = |00\rangle$ ,  $x_2 = |01\rangle$ ,  $x_3 = |10\rangle$ , and  $x_4 = |11\rangle$ .

The ZZFeatureMap quantum circuit is designed to transform data into quantum states that a quantum computer can interpret. It utilizes a second-order Pauli-Z evolution circuit and several parameters that allow the user to personalize the circuit's features (IBM Quantum Documentation, n.d.). Figure 4.8 shows an example of a ZZFeatureMap circuit composed of four qubits, linear entanglement, and one repetition. As can be seen, the ZZ FeatureMap utilizes several gates reviewed in Section 3.3, like the H-gate and the CNOT-gate. Moreover, the purple gates are operations performed on the unique features of the dataset, which, together with the rest of the gates, will transform them into quantum states.

**Figure 4.8: ZZFeatureMap**



#### 4.2.1.2.2 Quantum Kernel Implementation

This subsection revises the necessary process to implement a KQSVM on qiskit for the dataset specified before. As shown in Figure 4.9, variables are separated into features ( $x, z, s$ ) and labels (label). The dataset is then split through sklearn's *train\_test\_split* method, 80% test and 20% train.



**Figure 4.9: Train-Test Split**

```
# Split data into features (X) and labels (y).
X = [(x, z, s) for x, z, s, label in generated_data]
y = [label for x, z, s, label in generated_data]

# Perform train-test split with 80% training and 20% testing.
sample_train, sample_test, label_train, label_test = \
    train_test_split(X, y, test_size=0.2, random_state=random_seed)
```

The dataset is then scaled using the *MinMaxScaler* method from sklearn, which allows for easier handling of the data points during the training process. In this case, the data points were scaled to numbers between -1 and 1, as shown in Figure 4.10.

**Figure 4.10: Dataset Scale**

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

samples = np.append(sample_train, sample_test, axis=0)
# Fit values between -1 and 1.
minmax_scale = MinMaxScaler((-1,1)).fit(samples)
# Transform samples.
sample_train = minmax_scale.transform(sample_train)
sample_test = minmax_scale.transform(sample_test)
```

Then, the process shown in Section 3.3.2 is followed to connect to an available backend that will perform the computational task. In this case, through a *QiskitRuntimeService* instance, the ‘*ibm\_qasm\_simulator*’ quantum computer simulator was chosen, as shown in Figure 4.11:

**Figure 4.11: Backend Connection**

```
from qiskit_ibm_runtime import QiskitRuntimeService
# Connect to QiskitRuntimeService through the channel ibm_quantum.
service = QiskitRuntimeService(channel='ibm_quantum')
# Assigning the desired simulator as our backend.
backend = service.backends('ibmq_qasm_simulator')
print(backend)
```

The next step consists of importing all the necessary packages required in the process of building a KQSVM, as described in Figure 4.12. They will be used for the following purposes:

- *Session* and *Sampler* classes from *qiskit\_ibm\_runtime*: Provide a connection to the IBM Runtime service (sampler), and sample measurements from quantum circuits (sampler).
- *ComputeUncompute* class from *qiskit\_algorithms.state\_fidelities*: Used to measure the state fidelity (inner product) between two quantum states.
- *ZZFeatureMap* class from *qiskit.circuit.library*: Quantum circuit used for feature map encoding. As portrayed in Section 4.2.1.2.1, it operates by encoding data points into quantum states.
- *FidelityQuantumKernel* class from *qiskit\_machine\_learning.kernels*: Kernel class that leverages the fidelity algorithm.
- *generate\_preset\_pass\_manager* class from *qiskit.transpiler.preset\_passmanagers*: Used for circuit transpilation. This is used to optimize the quantum circuit before it executes, matching the topology of the quantum device.
- *QSVC* class from *qiskit\_machine\_learning.algorithms*: Quantum version of the classical SVM algorithm.
- *classification\_report* and *confusion\_matrix* classes from *sklearn.metrics*: To measure the performance of the different algorithms.
- *matplotlib.pyplot* and *seaborn*: To perform visualizations out of the performance measurements.
- *time*: To compute training time.

**Figure 4.12: Necessary Packages for KQSVM**

```
from qiskit_ibm_runtime import Session, Sampler
from qiskit_algorithms.state_fidelities import ComputeUncompute
from qiskit.circuit.library import ZZFeatureMap
from qiskit_machine_learning.kernels import FidelityQuantumKernel
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_machine_learning.algorithms import QSVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import time
```

Then, with a *Session* class open, where the service and backend previously assigned are provided, the process of creating a quantum kernel begins. First, create a new quantum kernel with the *ComputeUncompute* class and the *Sampler* class as a parameter. This will allow the quantum kernel to compute the inner products required to measure the similarity between data points. Then, the *ZZFeatureMap* class allows for the creation of a quantum circuit (in this case, the *ZZFeatureMap* quantum circuit) that will be used as data encoding for the DLP dataset data points. The parameters of the *ZZFeatureMap* include “*feature\_dimensions*” (the number of features in the dataset, which will be the number of qubits required in the quantum circuit), “*reps*” (the number of repetitions of the circuit), “*entanglement*” (the type of entanglement desired in the circuit), and “*insert\_barriers*” (whether if the user desires to separate between layers). The *generate\_preset\_pass\_manager* class sets a transpiler for the quantum circuit after setting the “*optimization\_level*” required and the “*target*” backend as parameters. Once the quantum circuit is transpiled, the quantum kernel is finally defined through the *FidelityQuantumKernel* class. The parameters required for this class are “*feature\_map*” (the transpiled quantum circuit variable) and the “*fidelity*” variable. Figure 4.13 below shows the process outlined:

**Figure 4.13: Quantum Kernel**

```
with Session(service = service, backend = backend):

    # Creating a fidelity instance. To create a fidelity instance, pass a sampler.
    # The sampler in this example is from QiskitRuntimeService to leverage Qiskit runtime services.
    fidelity = ComputeUncompute(sampler=Sampler())

    # Create quantum circuit through ZZFeatureMap for data encoding.
    zz_map = ZZFeatureMap(feature_dimension=3, reps=2, entanglement='linear', insert_barriers=True)

    # Transpile quantum circuit.
    pm = generate_preset_pass_manager(optimization_level=1, target=backend.target)
    isa_circuit = pm.run(zz_map)

    # Create a quantum kernel through the FidelityQuantumKernel class and the fidelity instance.
    zz_kernel = FidelityQuantumKernel(feature_map=isa_circuit, fidelity=fidelity)
```

As portrayed in Figure 4.14, to create the QSVM algorithm, pass the *QSVC* class with the quantum kernel defined before as the “*quantum\_kernel*” parameter. Then, the method *fit* will start training the KQSVM algorithm with the training data points of the DLP dataset. In this case, the module *time* was used to record the algorithm’s training time.

**Figure 4.14: Quantum Support Vector Machine**

```
# Create the KQSVM model.
# Use the newly created quantum kernel as quantum_kernel.
qsvc = QSVC(quantum_kernel=zz_kernel)

start = time.time()
# Train the quantum model.
qsvc.fit(sample_train, label_train)
elapsed = time.time() - start
```

Figure 4.15 below shows the process of recording the results of the algorithm after it has been trained. By using the *predict*, *confusion\_matrix*, *classification\_report*, and visualization tools obtained from *pyplot* and *seaborn*, it is possible to successfully retrieve the performance measures of the KQSVM.

**Figure 4.15: Confusion Matrix**

```
# Predict on the test set.
predictions = qsvc.predict(sample_test)

# Calculate confusion matrix.
cm = confusion_matrix(label_test, predictions)

# Generate the classification report.
report = classification_report(label_test, predictions, output_dict=True)

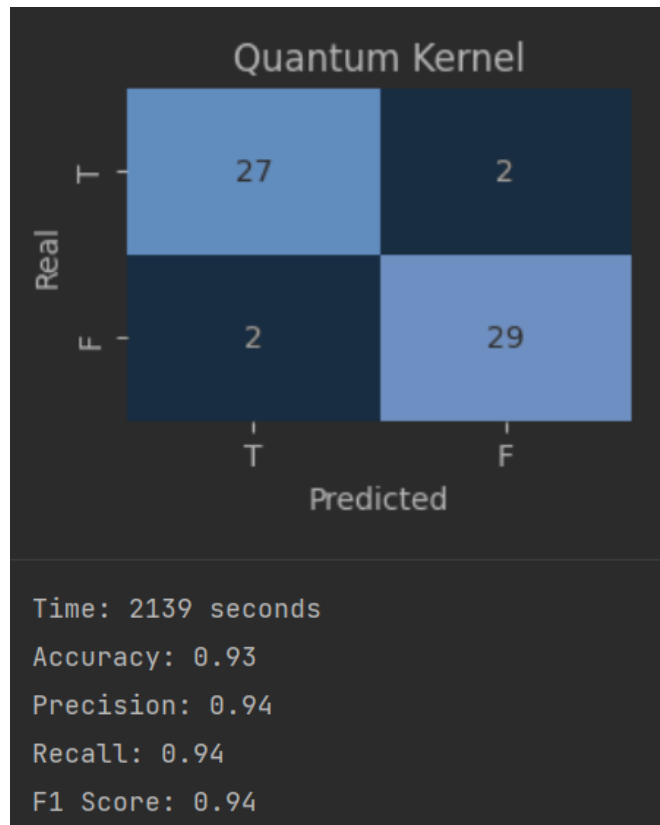
# Extracting metrics from the confusion matrix.
tn, fp, fn, tp = cm.ravel()
accuracy = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)

# Plot the confusion matrix.
plt.figure(figsize=(3,2))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False,
            xticklabels=['T', 'F'], yticklabels=['T', 'F'])
plt.xlabel('Predicted')
plt.ylabel('Real')
plt.title(f'Quantum Kernel')
plt.show()

# Print results.
print(f"Time: {round(elapsed)} seconds")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1_score:.2f}\n")
```

The outcome is shown in Figure 4.16. As can be inferred, the training process of the KQSVM took 2139 seconds. Moreover, the performance measurements were 93% accuracy, 94 % precision, 94 % recall, and 94 % F1 score. This is an example of a real experiment performed during the development of this thesis.

**Figure 4.16: Example of KQSVM Performance**

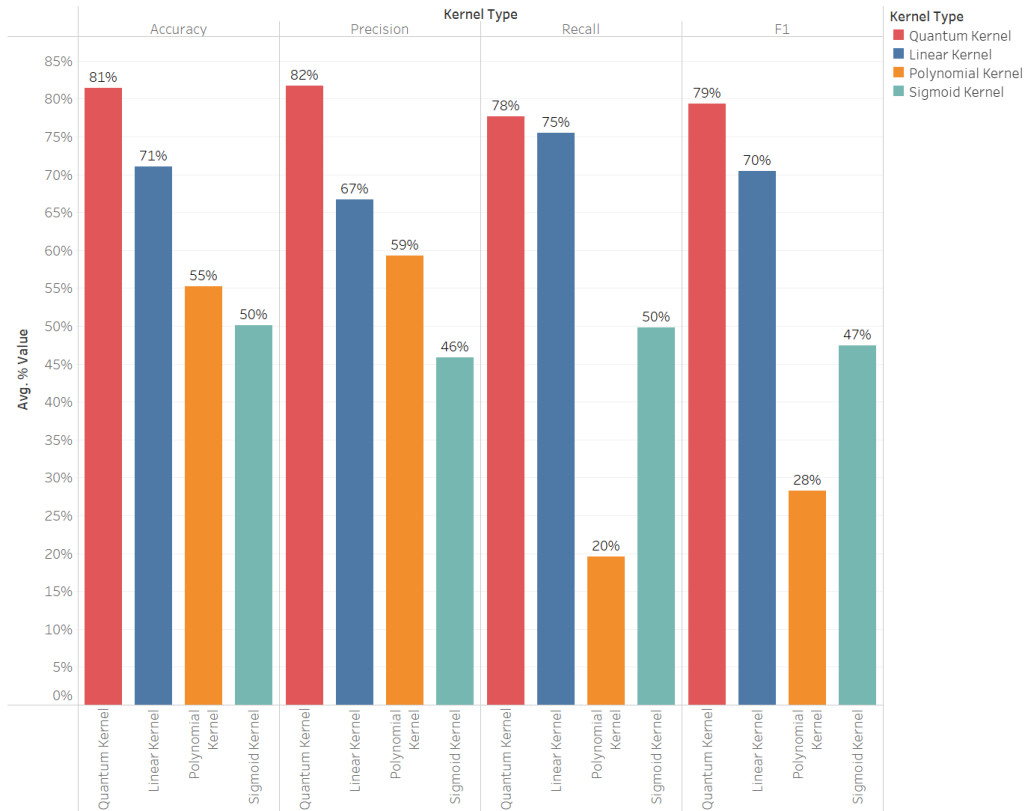


### 4.3 Results

This section describes the results obtained from the experiments performed, given the experimental design previously outlined. It will first examine the performance of the algorithms for each of the three datasets throughout all performance metrics (accuracy, precision, recall, and F1 score). Then, it will emphasize the accuracy score as the DLP dataset increases the prime number on which it is based. Finally, it will focus on the quantum kernel performance. The visualizations developed for this section are a result of calculating the average of the algorithms' performance for each dataset and performance metric. In total, 1200 data points were collected from the experiments (four algorithms, three datasets, 25 seeds, and four performance metrics).

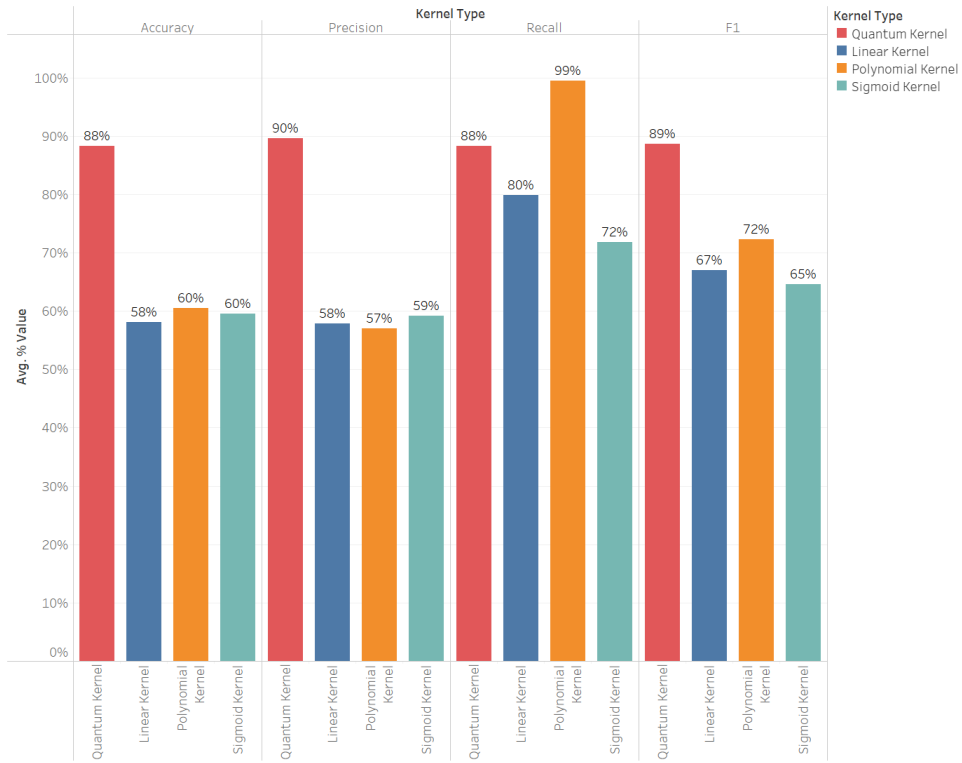
Figure 4.17 shows how the QKSVM outperformed its classical counterparts with a score of around 80% throughout all performance metrics for the DLP-64 dataset. The linear kernel was the highest scorer among the classical kernels.

**Figure 4.17: DLP-64 Dataset**

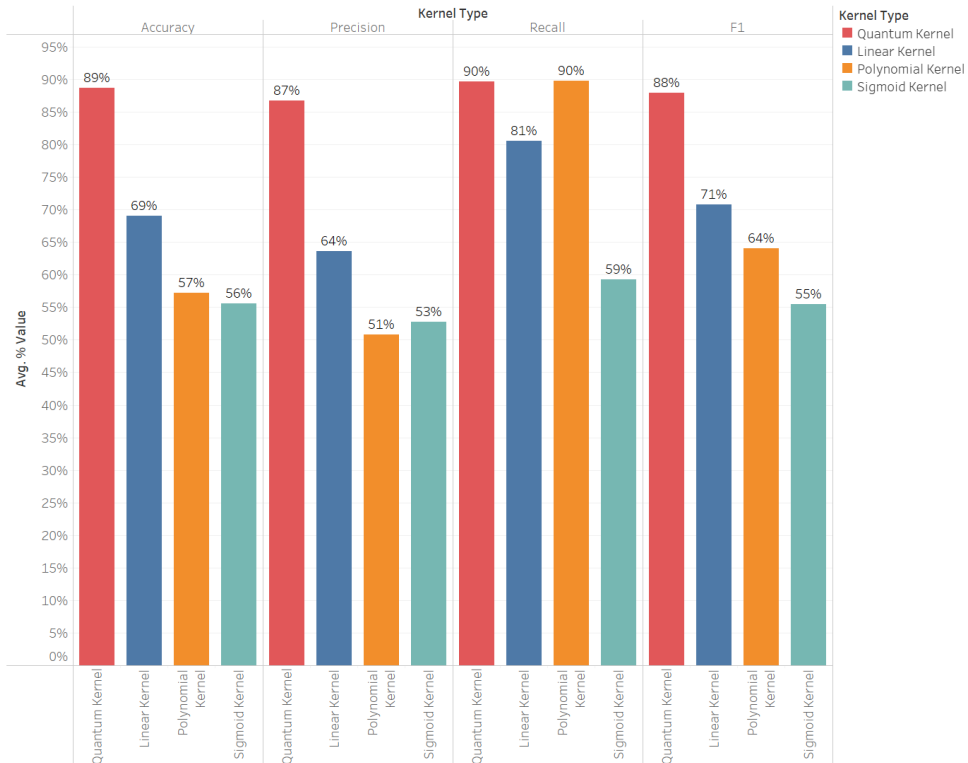


For the DLP-512 dataset, Figure 4.18 shows an improvement in the quantum kernel for all performance metrics, with all scores close to 90%. However, all classical algorithms continued with the same tendency of 60%-75% for all performance metrics. It is possible to notice a spike among all classical kernels for the recall measure, with the polynomial kernel scoring close to 99% on average. This suggests that classical kernels for this dataset seem to be better at identifying actual positive samples. However, although recall for classical kernels performs relatively high (in the case of polynomial even surpassing the quantum kernel, while the linear and sigmoid kernels are significantly poorer), the precision and accuracy scores are undoubtedly lower (50%-60%) for all classical kernels. Although the classical kernels correctly classify DLP-distribution data points, they return more irrelevant than relevant results. This suggests that classical kernels categorize DLP distribution to most data points, although not all follow that distribution.

**Figure 4.18: DLP-512 Dataset**



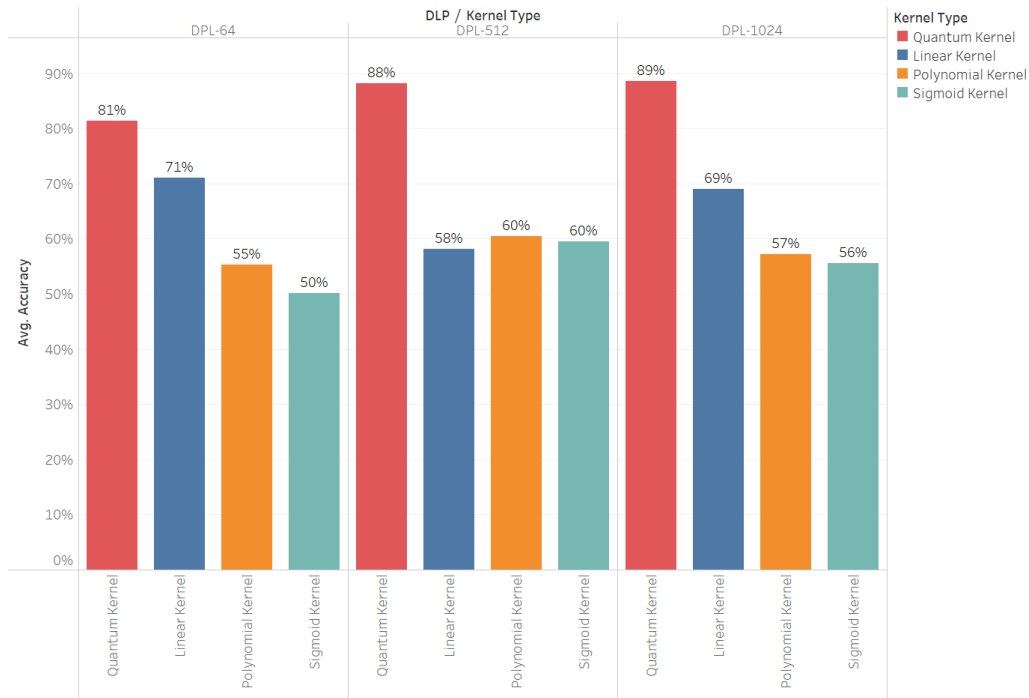
**Figure 4.19: DLP- 1024 Dataset**



The DLP-1024 dataset, as portrayed in Figure 4.19, has a similar performance as the DLP-512 dataset. The quantum kernel continues to outperform the classical kernels in all performance metrics, with a score close to 90%. Classical kernels have a higher score for recall again, with the polynomial kernel peaking at 90% (same as the quantum kernel). Nonetheless, as with the DLP-512 dataset, accuracy and precision are much lower than recall. As explained before, this behavior means that the model is classifying all data points as DLP-distributed data points, although they are all not.

When looking at the accuracy performance throughout all datasets in Figure 4.20, the quantum kernel exceeds the scores of the classical kernels. As the prime number grows, the quantum kernel seems to perform better, with an accuracy of almost 90%. Meanwhile, classical datasets always scored 71% in accuracy or less. The regular linear kernel scored around 70 % for the DLP-64 and DLP-1024 datasets, but besides that, the rest of the accuracy scores were between 50% and 60%.

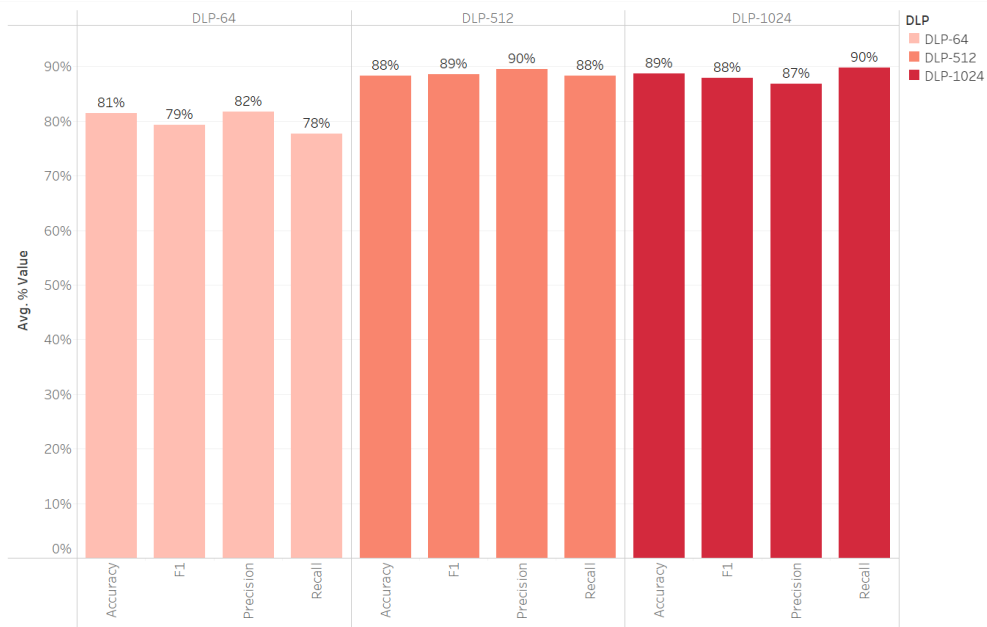
**Figure 4.20: Accuracy Across Datasets**



Taking a closer look at the performance of the quantum kernel, Figure 4.21 shows that the quantum kernel performed better for the DLP-512 and DLP-1024 datasets (around 90% for all metrics) compared to the DLP-64 (around 80%). In any case, all performance indicators have an average of 80% and 90% for the experiments performed. This behavior suggests a robust KQSVM.



**Figure 4.21: Quantum Kernel Performance**



In summary, the experiments performed were successful in determining that quantum kernels perform better than classical kernels in DLP-distributed datasets. The difference in accuracy between classical and quantum algorithms can range between 10-30%, with the quantum version outperforming the classical one. Besides, as the prime number of the DLP increases, quantum kernels continue to perform steadily.

## **Chapter 5: Discussion and Conclusion**

This chapter discusses the key findings and summarizes the conclusions drawn from the results of the experiments. Furthermore, it talks about probable future paths in the realm of KQSVM and QML in general. To conclude, this chapter delves into the key learning provided by the process of developing this thesis and reserves.

### **5.1 Discussion**

This thesis has reviewed several topics of significant importance to industry decision-makers and researchers in the field of QML. Throughout the chapters comprehended in this investigation, both theoretical and practical concepts are reviewed to offer a smooth yet robust introduction to the field of QML (and, more specifically, KQSVMs). Chapters 1 and 2 introduce QML and its role within IS while examining previous works in this realm. Chapter 3 equips the reader with the “building blocks” required to understand the technical topics discussed throughout the thesis. Sections 3.1 and 3.2 review the foundational knowledge of QML from a theoretical standpoint. With a particular focus on linear algebra and quantum mechanics, they aim to build a bridge that allows individuals to engage in the discussions that will follow later in the thesis. Furthermore, Section 3.3 provides a practical approach to the process of programming in a quantum environment. This section is targeted at both individuals and organizations interested in launching quantum initiatives. It covers the procedures for manipulating qubits, utilizing quantum gates, and establishing connections with quantum hardware. With this knowledge, now the reader has the necessary tools to dive into Chapter 4, which focuses on the experiments. Section 4.1 briefly introduces the logic of an SVM and how such an algorithm can be enhanced through quantum technology. Finally, Section 4.2 puts this idea into practice by delving into the procedure of programming a QSVM that utilizes a quantum kernel. This process culminates in the experimentation part of this thesis, resulting in the development of a quantum algorithm that companies can utilize for classification tasks.

The experiments outlined in Section 4.2 provided results that can be of important value to industry leaders, particularly those interested in classification tasks. These results can be summarized into two main takeaways, contingent upon the nature of the dataset that is being manipulated.

The first key takeaway is that utilizing QML techniques, such as QKSVM, does not seem to provide any benefits in certain cases. This conclusion extends to those datasets that are most common in daily classification tasks. In the pre-experimentation part of this project, several of these datasets were utilized, where quantum kernels had the same performance as classical kernels. This may be caused by the reduced setting in which the experiments were conducted, where the quantum circuits utilized a limited number of qubits, and where quantum advantage was not possible.

The second one is a consequence of the first one. The absence of quantum enhancement was fundamental in formulating an innovative approach to this thesis, which pivoted towards the use of a mathematically complex distribution dataset, like the DLP. This type of mathematical problem is known to be particularly complicated for classical algorithms due to its one-way function structure. As reviewed in Section 4.2.1.1, the DLP is particularly suited for scenarios where the user aims to transform a variable, yet the resulting output is difficult to trace back to the original variable. This difficulty arises because, while it is relatively simple to compute one side of the equation given the DLP congruence, determining the correct exponent to hold the DLP congruence true becomes a computationally intensive task involving trial and error (Menezes et al., 2020).

To conclude, as portrayed in Section 4.3, the quantum kernel had a better performance, particularly in accuracy, when compared to the three classical kernels selected. This behavior was extended throughout the different generated datasets and performance metrics. These results suggest that when it comes to classifying a DLP distributed dataset, a quantum kernel is much more accurate than any of the classical kernels utilized in these experiments. While quantum kernels scored 81-89% in accuracy throughout all datasets, classical kernels were more inconsistent with accuracy results that range between 50-71%. Besides, the spike in the recall metric for certain classical algorithms, paired with the still inferior performance in the F1 score, suggests that under certain scenarios, classical kernels tend to classify all the data points as “DLP-distributed,” incorrectly assigning labels to those data points that do not follow such distribution.

## 5.2 Conclusion

To summarize the main findings of this investigation, this section focuses on dissecting each of the results' components and its effect on stakeholders and current literature.

Returning to the original question posed in the introductory chapter of this thesis: *In the context of supervised learning, and more specifically SVMs, under what circumstances are Quantum Kernel Support Vector Machines better than classical Kernel Support Vector Machines?*

The satisfactory results provide direct insights into the research question: Datasets of a mathematically complex nature like the DLP are a scenario where quantum kernels perform better than classical kernels, even in near-term devices with a reduced number of qubits. With this, the empirical gaps that forged the motivation to draft this thesis outlined in Section 1.2 have been filled.

The experiments performed in this thesis showed to be successful in proving that KQSVMs can outperform classical kernelized SVMs for datasets whose distribution relies on the DLP, providing a practical example of the approach suggested by Liu, Arunachalam, and Temme (2021). This proves that it is possible to utilize a quantum feature space as a kernel, providing better results than classical kernels, as described in previous literature (Havlíček et al., 2019; Schuld & Killoran, 2019). Certain classical kernels, like the polynomial and sigmoid kernels, require parameters to function. These parameters are often arbitrarily assigned, sometimes providing poor results depending on the parameters chosen (Ganguly, 2021, p. 299). This process can be tedious and inefficient when classifying data points given a dataset distribution. A quantum kernel, however, can map the data into a quantum space without the need for any parameter in the process due to its quantum nature.

Given the constraints imposed by IBM cloud services in terms of the length of the datasets and the size of the prime number of the DLP (which Section 5.2.1 will discuss in detail), it has not been possible to determine whether quantum kernels are still a valid tool as the variables of the experiment grow. Nonetheless, the experiments showed that the quantum kernel obtained steady results as the prime number selected for the DLP increased in size. This evidence provides an intuition for potential further performances.

Direct applications from these conclusions can be applied to the domain of cryptography and IS, as previously mentioned in Section 1.2. The field of cryptography shares common ties with the realm of IS through cybersecurity and data privacy. While IS deals with storing and transmitting digital information, cybersecurity looks at protecting such information within those information systems. The core of this relationship is founded on the need to safeguard sensitive information from unauthorized access through cryptographic techniques, ensuring two of the three main cybersecurity pillars: integrity and confidentiality. Using different cryptographic tools makes it possible to encode and decode data, allowing for the transfer of sensitive information without the threat of its interception by an intermediary. This encryption process is fundamental in protecting data during storage, transmission, and processing. Essentially, cryptography serves as one of the building blocks of cybersecurity and data privacy, which are crucial within IS environments. The DLP datasets that were experimented with were inspired by cryptographic tools. This is because some algorithms base their security on the assumption that certain groups within the DLP have no efficient solution (Menezes et al., 2020). These experiments proved that quantum-inspired algorithms may have an edge over classical ones for these types of mathematical computation when extrapolated to the ML field. This has the potential to challenge security networks that heavily rely on DLP-based security tests, strengthening the argument of quantum computers being a threat to the cryptographic tools currently in use.

Furthermore, these findings can be of excellent value to other scientific fields that obtain value from classification tasks. Besides the existing literature on KQSVM outlined in Chapter 2, the insights drawn from this thesis can support fields like healthcare and life sciences, where a narrow quantum advantage has already been proved (Krunic et al., 2022), or in the financial sector, where KQSVM has been previously used for recession prediction (Rigetti Computing, 2023). Nonetheless, the results portrayed in this thesis suggest that quantum kernels seem to be an optimal solution not only for general datasets like these, where they have a slight or equal performance compared to classical algorithms, but for data that is generally complicated to classify. Therefore, the results of this thesis invite the utilization of KQSVM techniques for datasets within all scientific fields, where the data is apparently “randomly” distributed, yet quantum kernels can utilize their power to successfully classify new data points, as shown with the DLP dataset.

To conclude, it is crucial to understand that the essence of the experiments, results, and conclusions outlined in this thesis lies in addressing a challenge inherently difficult for classical computers like the DLP. In this investigation, the focus is not on refining or enhancing already-established ML algorithms with their quantum counterparts. Instead, it aims to illustrate the potential of quantum algorithms to tackle novel problems that classical algorithms struggle with. Therefore, these experiments highlight how quantum advantage may be obtained by using QML techniques for specific mathematical complex problems that scientists are not used to solving regularly because they were thought to be “unsolvable” by classical computers.

### **5.2.1 Challenges and Limitations**

Several constraints were imposed by IBM cloud services during the experimentation process of this investigation. The two main variables affected by these constraints are the length of the datasets and the size of the prime number utilized to generate the DLP dataset. Thus, it has not been possible to determine whether quantum kernels are still a valid tool as the variables of the experiment grow. Nonetheless, the experiments showed that the quantum kernel obtained steady results as the prime number selected for the DLP increased in size. This provides an intuition of possible further performances.

Another limitation of this thesis is the absence of a real-life dataset that follows a DLP distribution. Due to the complexity of this problem, at the time this thesis was published, it was impossible to determine a scenario where data was separated through this class distribution. Hence, the approach provided by Liu et al. (2021) was followed to generate an artificial dataset with these characteristics. This generated dataset can provide valuable insights regarding future explorations within the realm of QML after this thesis has shown quantum advantage for a dataset with said features.

### **5.2.2 Future Work and Recommendations**

Future works in the field may point to finding other mathematically complex tasks for classical algorithms where quantum computing can provide an advantage. Using quantum kernels to map data into a higher dimensional feature space may apply to other ML applications. In this case, quantum computers can easily compute the inner product of two quantum states, which essentially calculates the similarity between two data points. Finding other applications of this technique in near-term quantum computers would be of extreme interest for ongoing research and business

applications. Moreover, the experiments performed utilized a quantum simulator due to the small number of qubits required for the task. Finding a scenario in which more qubits are needed would allow more empirical experiments with actual quantum devices, which would be of great interest as quantum hardware becomes widely available and more powerful with time. Finally, there are other classical kernels besides those selected to be examined against the quantum kernel (linear, polynomial, and sigmoid). Continuing this line of research involves expanding its frontiers by incorporating classical kernels of other nature like the RBF, Laplacian, or exponential kernels.

Besides utilizing other classical kernels, several areas within the investigation portrayed in this thesis could be enhanced to obtain further results. These improvements will serve as recommendations for future projects in the field. Section 5.2.1 outlines the limitation of generating a dataset from a prime number greater than 1024 bits, a constraint imposed by the IBM server. Increasing the size of the prime number would allow for the exploration of quantum and classical algorithms as the DLP grows in complexity. This would provide further insights into the behavior of such algorithms in scenarios similar to those encountered in practical cryptographic applications. In terms of potential improvements to the algorithm, increasing the dataset size (as also mentioned in Section 5.2.1) would be of immense value for a more detailed and robust model. As the number of samples used to build the model increases, the accuracy and other performance metrics will likely increase too.

For those looking to replicate the experiments performed in this thesis, it is fundamental to note that qiskit 0.46 was the version utilized in the process of developing the quantum-oriented programming part contained in this exploration. Nonetheless, the qiskit community recently announced the release of qiskit 1.0. This updated version of qiskit is a foundational change aiming to wrap qiskit into the simple packaging structure used in most Python packages. This major update in qiskit might lead to certain sections of the code cited throughout the thesis becoming inoperable. Nonetheless, most of the quantum programming concepts outlined in this exploration are still valid and accurate, although they may be accessed through a different packaging structure in qiskit 1.0. Finally, as referenced in Section 2.2, there have been several projects discussing the role of QML in unsupervised learning. Although this thesis particularly focuses on supervised learning, it has provided significant insights into the theoretical scenarios where quantum technology can support

an algorithm's functioning logic. Therefore, finding similar situations where quantum algorithms can support unsupervised learning models can be the scope of future explorations in the field.

### **5.2.3 Key Learnings**

Developing this thesis started with a complete absence of knowledge concerning QML. The learning curve required for creating this work was immense. Before any substantial progress could be made, an extensive technical introduction to QC and QML paired with an initiation into conducting research in a scientific field was necessary. This subsection explores the key learnings and important processes that allowed this thesis to reach its final form.

The first stages of the project focused on understanding the foundations of linear algebra and quantum mechanics. During this time, Sections 3.1 and 3.2 were developed. These served as the pillars that sustain the following sections, where the focus shifts to quantum computing and KQSVMs. Resources from IBM Quantum enabled access to learning materials concerning QC, initially through a theoretical approach that would later be applied in QML projects. It was at this stage that the intricacies of quantum circuits, gates, and entanglement circuits were introduced, which were essential for understanding concepts later applied in the development of KQSVMs. Once this knowledge was solidified, concepts and techniques were applied to a specific niche of QML: quantum kernels. Only after delving into the details of QML it is possible to grasp the rationale behind why quantum kernels can provide an advantage over classical kernels. Furthermore, understanding the nature of the DLP was fundamental for comprehending the possible scenarios where quantum kernels can support existing techniques.

Parallely, a systematic research approach was crucial for integrating the theoretical knowledge mentioned above with existing literature and potential business applications. This approach commenced with a thorough literature review, encompassing scientific papers and studies on QML. Analyzing these works proved instrumental in identifying knowledge gaps and the main challenges faced by the QML community, which would later facilitate experiment development. Additionally, intensive research was conducted to find practical applications of KQSVMs in various scientific fields, establishing a clear goal for the relevance of the ongoing research in question within the current environment. The comprehensive approach outlined served as the project's guiding principle, resulting in the development of this thesis.



## Acknowledgments

I would like to express my sincere gratitude to the individuals whose contribution was key to the development of this thesis.

Dr. Naveen Kumar, the advisor of this thesis, played a fundamental role as my guiding force. His support and encouragement propelled me forward into the fascinating realm of QML for the first time. Besides, the research approach and guidance provided by Dr. Kumar served as the pathway toward the successful completion of this work.

This thesis partly owes its existence to the invaluable contributions of my brother, Samuel Gutiérrez. His insights, suggestions, and engaging discussions were paramount in shaping the foundations of this investigation. Samuel's profound knowledge of linear algebra and quantum mechanics provided the base upon which this work stands.

Additionally, I extend my gratitude to Kristan Temme for his crucial input in devising the programming framework necessary for creating the three DLP datasets. Without his exchange of ideas and suggestions, the experiments conducted would not have been feasible. It is worth noting that this work was inspired by their paper (Liu et al., 2021), which significantly influenced the experiments section of this thesis.

Finally, I wish to express my most sincere appreciation to Estela, my sister Eva, my mother María del Mar, and my father Rafael for supporting and encouraging me during this challenging yet fascinating journey.

## References

- Avramescu, C. (2023). Demystifying Complex Numbers: A Practical Guide. *Medium*. <https://medium.com/technological-singularity/demystifying-complex-numbers-a-practical-guide-453801e5dde4>.
- Baheti, P. (2022). Pattern Recognition in Machine Learning. *V7Labs*.
- Baiardi, A., Christandl, M., & Reiher, M. (2023). Quantum Computing for Molecular Biology\*\*. *ChemBioChem*, 24(13), e202300120.
- Bernal, D. E., Ajagekar, A., Harwood, S. M., Stober, S. T., Trenev, D., & You, F. (2022). Perspectives of quantum computing for chemical engineering. *AIChE Journal*, 68(6), e17651.
- Bloch, F. (1946). Nuclear Induction. *Physical Review*, 70(7–8), 460–474.
- Bova, F., Goldfarb, A., & Melko, R. G. (2023). Quantum Economic Advantage. *Management Science*, 69(2), 1116–1126.
- Brownlee, J. (2020). Support Vector Machines for Machine Learning.
- Byrum, J. (2022). A quantum future for medical breakthroughs. *Analytics*. <https://pubsonline.informs.org/doi/10.1287/LYTX.2022.01.02/full/>.
- Cerezo, M., Verdon, G., Huang, H.-Y., Cincio, L., & Coles, P. J. (2022). Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9), 567–576.
- Computing, R. (2023). Recession Prediction via Signature Kernels Enhanced with Quantum Features. *Rigetti*. <https://medium.com/rigetti/recession-prediction-via-signature-kernels-enhanced-with-quantum-features-a608995d48f7>.
- Daley, A. J., Bloch, I., Kokail, C., Flannigan, S., Pearson, N., Troyer, M., & Zoller, P. (2022). Practical quantum advantage in quantum simulation. *Nature*, 607(7920), 667–676.
- Dür, W., Vidal, G., & Cirac, J. I. (2000). Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6), 062314.
- El Khadir, B. (Director). (2022). The Kernel Trick in Support Vector Machine (SVM). <https://www.youtube.com/watch?v=Q7vT0--5VII>.
- Feynman, R. P., Vernon, F. L., & Hellwarth, R. W. (1957). Geometrical Representation of the Schrödinger Equation for Solving Maser Problems. *Journal of Applied Physics*, 28(1), 49–52.
- Galaviz, J. (Director). (2021). El problema del logaritmo discreto. <https://www.youtube.com/watch?v=rkA1PnvNmbU>.
- Ganesh, R. (2021). Computational identification of inhibitors of MSUT-2 using quantum machine learning and molecular docking for the treatment of Alzheimer's disease. *Alzheimer's & Dementia*, 17(S9), e049671.
- Ganguly, S. (2021). Quantum machine learning: An applied approach : the theory and application of quantum machine learning in science and industry. Apress.
- Gentinetta, G., Thomsen, A., Sutter, D., & Woerner, S. (2024). The complexity of quantum support vector machines. *Quantum*, 8, 1225.
- Guijo, D., Onofre, V., Bimbo, G. D., Mugel, S., Estepa, D., Carlos, X. D., Adell, A., Lojo, A., Bilbao, J., & Orus, R. (n.d.). Quantum artificial vision for defect detection in manufacturing.
- Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., & Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747), 209–212.

- Heiner, M. (2022). Quantum Business Intelligence and Quantum Augmented Digital Twins. *ORMS*. <https://pubsonline.informs.org/doi/10.1287/orms.2023.03.07/full/>.
- Hellstern, G. (2021). Analysis of a hybrid quantum network for classification tasks. *IET Quantum Communication*, 2(4), 153–159.
- Henderson, M., Novak, J., & Cook, T. (2019). Leveraging Adiabatic Quantum Computation for Election Forecasting. *Journal of the Physical Society of Japan*, 88(6), 061009.
- IBM Quantum. (n.d.). IBM Quantum. Retrieved December 12, 2023, from <https://quantum.ibm.com/>.
- IBM Quantum Documentation. (n.d.). ZZFeatureMap. IBM Quantum Documentation. Retrieved April 1, 2024, from <https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.ZZFeatureMap>.
- Jain, S., Ziauddin, J., Leonchuk, P., Yenkanchi, S., & Geraci, J. (2020). Quantum and classical machine learning for the classification of non-small-cell lung cancer patients. *SN Applied Sciences*, 2(6), 1088.
- Jiang, W., Xiong, J., & Shi, Y. (2021). When Machine Learning Meets Quantum Computers: A Case Study. Proceedings of the 26th Asia and South Pacific Design Automation Conference, 593–598.
- Johnson, T. H., Clark, S. R., & Jaksch, D. (2014). What is a quantum simulator? *EPJ Quantum Technology*, 1(1), 10.
- Krunić, Z., Flöther, F. F., Seegan, G., Earnest-Noble, N., & Shehab, O. (2022). Quantum kernels for real-world predictions based on electronic health records. *IEEE Transactions on Quantum Engineering*, 3, 1–11.
- Kutzkov, K. (2022). Explicit feature maps for non-linear kernel functions. Towards Data Science.
- Kyriienko, O., & Magnusson, E. B. (2022). Unsupervised quantum machine learning for fraud detection (arXiv:2208.01203). arXiv.
- Liu, Y., Arunachalam, S., & Temme, K. (2021). A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9), 1013–1017.
- Menezes, A. J., Oorschot, P. C. van, & Vanstone, S. A. (2020). Handbook of Applied Cryptography. CRC Press.
- Mercha, E. M., & Benbrahim, H. (2023). Machine learning and deep learning for sentiment analysis across languages: A survey. *Neurocomputing*, 531, 195–216.
- Munro, E. (2018). Quantum computing: Near- and far-term opportunities. *Medium*. [https://medium.com/@quantum\\_wa/quantum-computing-near-and-far-term-opportunities-f8ffa83cc0c9](https://medium.com/@quantum_wa/quantum-computing-near-and-far-term-opportunities-f8ffa83cc0c9).
- Nielsen, M. A., & Chuang, I. L. (2010). Quantum computation and quantum information (10th anniversary ed). Cambridge University Press.
- OpenAI. (2023). *ChatGPT (GPT-3.5)* [Large language model]. <https://chat.openai.com/chat>
- Orus, R., Mugel, S., & Lizaso, E. (2019). Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4, 100028.
- Palmer, S., Sahin, S., Hernandez, R., Mugel, S., & Orus, R. (2021). Quantum Portfolio Optimization with Investment Bands and Target Volatility (arXiv:2106.06735). arXiv.
- Perdomo-Ortiz, A., Benedetti, M., Realpe-Gómez, J., & Biswas, R. (2018). Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers. *Quantum Science and Technology*, 3(3), 030502.
- Pira, L., & Ferrie, C. (2023). Explicability and Inexplicability in the Interpretation of Quantum Neural Networks (arXiv:2308.11098). arXiv.

- Protopapas, P., & Rader, K. (2018). Support Vector Machines (SVMs).
- Pushpak, S. N., & Jain, S. (2021). An Introduction To Quantum Machine Learning Techniques. 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 1–6.
- Qiskit. (n.d.). Retrieved December 12, 2023, from <https://qiskit.org>.
- Quantum Explorers—IBM Quantum Experience. (n.d.). Retrieved December 12, 2023, from <https://challenges.quantum.ibm.com/quantum-explorers-23#captain>.
- Rebentrost, P., Mohseni, M., & Lloyd, S. (2013). Quantum support vector machine for big data classification.
- Rupp, M. (2015). Machine learning for quantum mechanics in a nutshell. *International Journal of Quantum Chemistry*, 115(16), 1058–1073.
- Sanz-Fernandez, C., Hernandez, R., Marciniak, C. D., Pogorelov, I., Monz, T., Benfenati, F., Mugel, S., & Orus, R. (2021). Quantum portfolio value forecasting (arXiv:2111.14970). arXiv.
- Schlatter, A. (2017). On the Nature of the Born Probabilities. *Journal of Modern Physics*, 8(5), Article 5.
- Schuld, M. (Director). (2019a). Quantum Machine Learning—32—Quantum-Enhanced Kernel Methods. <https://www.youtube.com/watch?v=uDAAi5aQbMU>.
- Schuld, M. (Director). (2019b). Quantum Machine Learning—33—Quantum-Enhanced Kernel Methods 2. <https://www.youtube.com/watch?v=pfGHJivyzHA&t>.
- Schuld, M., & Killoran, N. (2019). Quantum Machine Learning in Feature Hilbert Spaces. *Physical Review Letters*, 122(4), 040504.
- Schwabl, F. (2007). Quantum Mechanics. Springer Science & Business Media.
- Shehnepoor, S., Togneri, R., Liu, W., & Bennamoun, M. (2023). Social Fraud Detection Review: Methods, Challenges and Analysis (arXiv:2111.05645). arXiv.
- Shipilov, A., & Furr, N. (2022). Making Quantum Computing a Reality. *Harvard Business Review*. <https://hbr.org/2022/04/making-quantum-computing-a-reality>.
- Sitorus, I. (2020). Support Vector Machine (SVM) and Kernels Trick. *Medium*.
- Trevisan, L. (2012). Lecture 10: The Discrete Log Problem. <https://theory.stanford.edu/~trevisan/quantum/lecture10.pdf>
- Troyer, M. (2023). Quantum Advantage: Hope and Hype. *Microsoft Azure Quantum Blogs*. <https://cloudblogs.microsoft.com/quantum/2023/05/01/quantum-advantage-hope-and-hype/>.
- Verstraete, F., Dehaene, J., De Moor, B., & Verschelde, H. (2002). Four qubits can be entangled in nine different ways. *Physical Review A*, 65(5), 052112.
- von Lilienfeld, O. A. (2018). Quantum Machine Learning in Chemical Compound Space. *Angewandte Chemie International Edition*, 57(16), 4164–4169.
- Weinberger, K. (2018). *Lecture 13: Kernels*. <https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote13.html>.
- Winston, P. (Director). (2010). MIT OpenCourseWare: Support Vector Machines. [https://www.youtube.com/watch?v=\\_PwhiWxHK8o&t](https://www.youtube.com/watch?v=_PwhiWxHK8o&t).
- Xiao, T., Zhai, X., Wu, X., Fan, J., & Zeng, G. (2023). Practical advantage of quantum machine learning in ghost imaging. *Communications Physics*, 6(1), 171.
- Zhang, Y., & Ni, Q. (2020). Recent advances in quantum machine learning. *Quantum Engineering*, 2(1). <https://doi.org/10.1002/que2.34>.

- Zhang, Y., & Ni, Q. (2021). Design of quantum neuron model for quantum neural networks. *Quantum Engineering*, 3(3).
- Zwiebach, B. (2022). Mastering quantum mechanics: Essentials, theory, and applications. The *MIT Press*.

# Appendix

## A.1 Determinants

A determinant is a property of squared matrixes (two-by-two, three-by-three, ...), which has the form of a scalar. This scalar refers to the factor by which the plane is scaled after a linear transformation. Determinants are of major help when studying linear transformations and other linear algebra domains, like systems of linear equations. The determinant of the matrix  $A$  can be notated as  $\det(A)$ , or  $|A|$ . Calculating determinants can be rather complicated as the matrix grows larger, but this process is simplified by assessing two-by-two and three-by-three matrixes, which are the ones that will be managed more often in the context of this thesis. For squared matrix  $A$  of two-by-two, the determinant is calculated as shown in Equation A.1.1:

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb \tag{A.1.1}$$

Similarly, for a squared matrix  $B$  of three-by-three, the determinant is calculated through the following formula in Equation A.1.2:

$$|B| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + dhc + bfg - gec - hfa - dbi \tag{A.1.2}$$

The relevance behind calculating determinants of a vector set (i.e., transformation matrix, or  $\hat{l}$  and  $\hat{j}$  transformed) is that they directly provide the information on whether that vector set is linearly dependent or not. Therefore, the main takeaways are:

- If the determinant is equal to 0, the vectors within the vector set are linearly dependent on each other and cannot be used as a base.
- If the determinant is not 0, the vectors within the vector set are linearly independent and can be used as a base.

Going back to the previous definition, these statements logically support the definition of determinants. If the determinant is 0, this means that after the linear transformation, the plane is

scaled by 0, reducing the 2D or 3D plane to a single line or a single point. If the determinant is not 0, the plane is being scaled by a number, which means that it continues to be a 2D or 3D plane.

## A.2 Determinants and eigenvectors

Because  $\lambda$  is a scalar, it is required to find a way to represent it as a matrix, which can be done by multiplying it by the identity matrix (a matrix that has the basis vectors as its components), known as “I” in Equation A.2.1:

$$sI = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ therefore:}$$

$$\lambda \text{ as a matrix} = \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}$$

(A.2.1)

Therefore, all the necessary variables are present in order to rewrite the previous equation into Equation A.2.2:

$$A\vec{v} = (\lambda I)\vec{v}, \text{ therefore:}$$

$$(A - \lambda I)\vec{v} = 0$$

(A.2.2)

To solve this equation, either  $\vec{v}$  has to equal zero, or  $A - \lambda I$  has to equal zero. The first option is not viable because that would mean the vector would not exist. Therefore, the value of  $A - \lambda I$  through the second option. Nonetheless, this expression is still in a matrix form, which is not how eigenvalues are represented. Hence, the determinant is used to find the scalar representation of this matrix. After following these steps, it is concluded that to find the eigenvalue, Equation A.2.3 is used for solving  $\lambda$  :

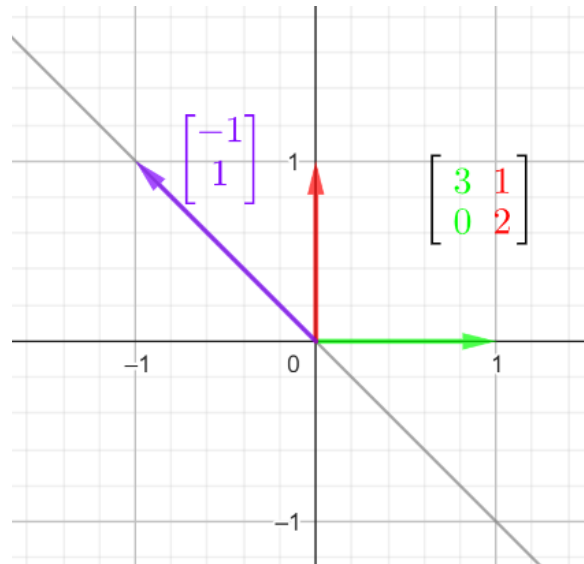
$$\det(A - \lambda I) = 0$$

(A.2.3)

### A.3 Eigenbasis

A change of basis is computed by multiplying three matrixes: the change of basis matrix (with the coordinates of your desired vector basis), its inverse, and the original transformation matrix. Refer to the example from Section 3.1.6 (depicted in Figure A.3.1), where the eigenvectors of a linear transformation are  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  and  $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ :

**Figure A.3.1: Eigenvector After Linear Transformation (1)**



In this example, it is desired for the vectors  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  and  $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$  to become a vector basis. This can be done through the change of matrix basis  $\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ . To compute the change of basis, Equation A.3.1 below shows the process of multiplying the inverse of the change of matrix basis times the original transformation matrix times the change of matrix basis:

$$\underbrace{\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}^{-1}}_{\text{Change of matrix basis inverse}} \underbrace{\begin{bmatrix} 3 & 1 \\ 0 & 1 \end{bmatrix}}_{\text{Original transformation matrix}} \underbrace{\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}}_{\text{Change of matrix basis}} = \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}}_{\text{Eigenbasis}} \quad (\text{A.3.1})$$

This computation will always output a diagonal matrix, which can be used as the “new” basis.



## A.4 Imaginary and complex numbers

The last mathematical domain covered concerns imaginary and complex numbers. The intuition behind these concepts comes from the idea of calculating the square root of a negative number. Although it may seem abstract at first, when applied to science, imaginary numbers do follow the laws of nature, for example, in alternate current circuits or mathematical algorithm development. Therefore, even though they cannot be logically explained, they are fundamental in solving real-life problems (Avramescu, 2023). The basic expression of an imaginary number starts with the determination of “i,” as seen in Equation A.4.1:

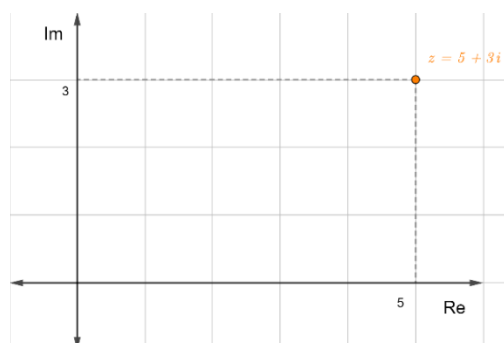
$$i = \sqrt{-1} \tag{A.4.1}$$

The different expressions in which “i” can be represented ( $-i, \pi i, ei \dots$ ) are known as imaginary numbers. Creating expressions that blend imaginary numbers with real numbers is known as complex numbers; for example, see Equation A.4.2:

$$\begin{array}{ccc} \text{Complex number} & \{ z = 5 + 3i \} & \text{Imaginary number} \\ & \underbrace{\hspace{1.5cm}}_{\text{Real number}} & \end{array} \tag{A.4.2}$$

Two main functions provide information about a complex number. The first one is  $Re()$ , which outputs the real part of a complex number from the example in  $z = 5 + 3i$ ,  $Re(z) = 5$ . The second function is  $Im()$ , which refers to the multiple by which the imaginary part is scaled. In the previous example,  $Im(z) = 3$ . These functions can be depicted in a complex-number graph, as can be seen in Figure A.4.1, where  $Re()$  is the X-axis, and  $Im()$  is plotted on the Y-axis. The equation  $z = 5 + 3i$  would be graphically represented as depicted in Figure A.4.1:

**Figure A.4.1: Graphical Representation of Complex Number**



## A.5 Complex Conjugate

The *complex conjugate* stands for the inverse of an imaginary number, where the real part remains unchanged, and the sign of the imaginary part is inverted. The notation used to represent the complex conjugate is “ $z^*$ ” and its example applied to the complex number referred to in Equation A.4.2 is represented in Equation A.5.1:

$$z^* = 5 - 3i \tag{A.5.1}$$

The utility of the complex conjugate lies behind the fact that it provides the resources to deal with complex functions and complex vectorial spaces. These tools are fundamental for quantum mechanics and quantum computing because both of these fields heavily rely on quantum wave functions.