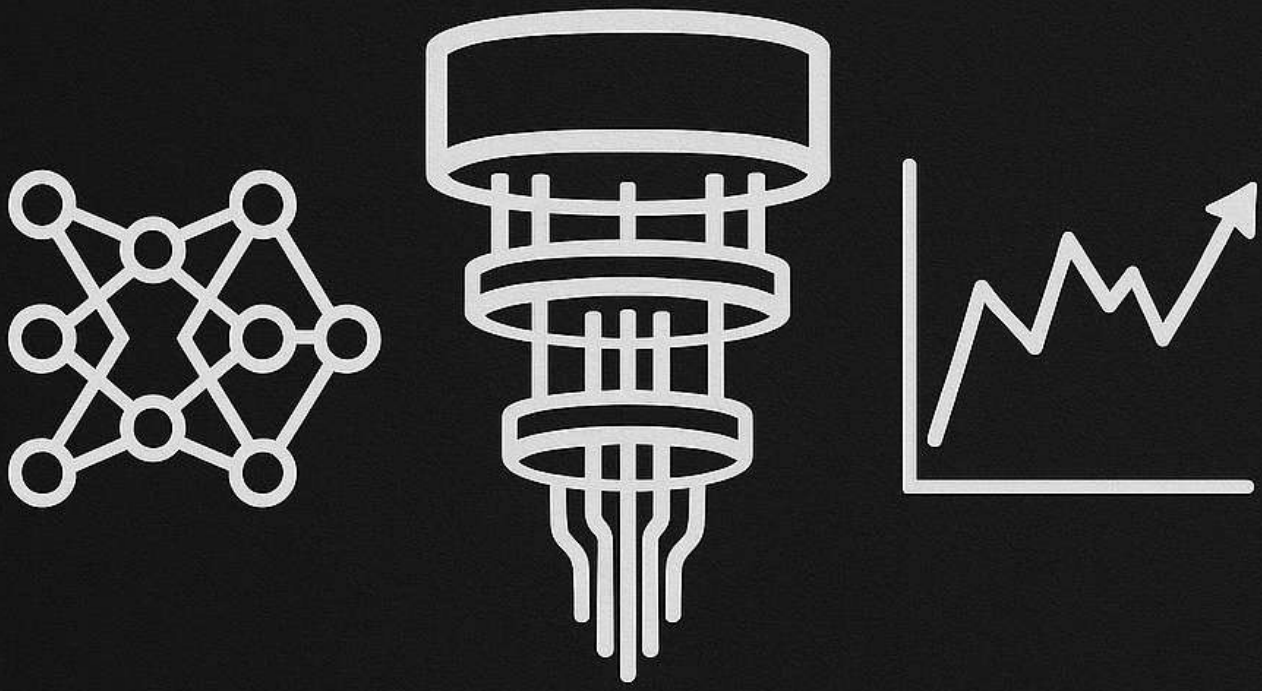
# QUANTUM COMPUTING IN FINANCE

## An Introductory Guide



OSWALDO ZAPATA, PhD

# Quantum Computing for Finance

## An Introductory Guide

Oswaldo Zapata, PhD

The purpose of this short book is to illustrate how quantum computing is transforming the future of finance and to outline the actions that professionals and financial institutions can—and must—take to prepare for the upcoming revolution. It is specifically intended for finance students and professionals who are interested in learning how quantum computing is transforming their industry. Quantum computing experts with an interest in finance may also find it valuable.

# Chapter 1

# Introduction

The purpose of this short book is to illustrate how quantum computing is transforming the future of finance and to outline the actions that professionals and financial institutions can—and must—take to prepare for the upcoming revolution.

It is specifically intended for finance students and professionals who are interested in learning how quantum computing is transforming their industry. Quantum computing experts with an interest in finance may also find it valuable.

Chapter 2 provides a brief overview of quantum computing. Since I have already explained the basics of quantum computing in detail in previous notes,[1] this chapter focuses only on the key concepts most relevant to the upcoming chapters.

In Chapter 3, I discuss some of the most pressing challenges currently faced by the financial industry and how experts are addressing them. Specifically, I highlight areas where quantum computing shows promise in offering significant advantages. Key topics include portfolio optimization, Monte Carlo simulations, and various problems involving artificial intelligence models, particularly machine learning. Without delving too deeply into technical details, I also touch on the programming skills necessary to tackle these problems.

Chapter 4 discusses how quantum computing can help address the challenges mentioned in the previous chapter, particularly in portfolio optimization and the integration of quantum computing with machine learning. It concludes with a survey of the most popular frameworks used to program several quantum hardware available today in the market.

Chapter 5 provides an overview of the quantum computing landscape specifically for finance. I discuss the maturity of quantum hardware, concrete examples of major firms and startups driving the adoption of quantum technology, as well as practical advice on what professionals and financial institutions can begin doing now to avoid falling behind and gain a competitive advantage.

A final note on how to approach this guide: I suggest focusing on the chapters and sections that align with your interests and background. For instance, if you are a chief innovation officer at a bank and find the chapter on quantum computing too challenging, feel free to skip ahead to the parts most relevant to you. Conversely, if you are a quantum computing physicist and find this chapter too basic, move on to the material that captures your interest. That said, for a comprehensive understanding of the subject, I recommend reading and digesting the entire book.

[1] See "An Introduction to Quantum Computing for Physicists" and "A Second Course on Quantum Computing for Physicists." You can find them on my LinkedIn. In the following, I will refer to them as QC1 and QC2.

Feel free to connect with me on LinkedIn and let me know if you have any feedback: https://www.linkedin.com/in/oswaldo-zapata-phd-quantum-finance.

# Chapter 2

# Review of Quantum Computing

Quantum computers are expected to solve some problems significantly faster and more accurately than classical devices. This chapter provides a brief summary of their key features.

*Quantum computation* is a quantum mechanical approach to solving computational problems. It uses the principles and mathematical formalism of quantum mechanics—such as state vectors, unitary operators, and measurements—to arrive at logical solutions to given computational problems. A *quantum computer*, on the other hand, is the physical device that implements the quantum computational processes of interest. For decades, it was believed that quantum computations could only be performed on quantum computers built entirely from quantum components. In this chapter, we will see that this is no longer the case. Today, experts are convinced that the first machines to surpass classical supercomputers will be hybrid devices, composed of both quantum and classical components working together in tandem.

This chapter is organized as follows. Sections 2.1 and 2.2 review fundamental concepts of quantum computing, with a focus on the circuit model of quantum computation and error correction. Section 2.3 offers a preliminary introduction to the hybrid quantum-classical computational models just mentioned.

## 2.1 Quantum Circuits

Let us begin with the most fundamental concepts. A *model of computation*, broadly speaking, is a logical framework that defines how to proceed given a set of basic elements and instructions. More precisely, a model of computation is characterized by a set of abstract objects and a collection of elementary operations on these objects. An *algorithm* is a sequence of precise instructions defined within a model of computation, designed specifically to solve a computational problem.

For example, the *Boolean* or *binary model of computation* is based on the so-called *Boolean algebra*. In Boolean algebra, there are only two elements, conventionally denoted by 0 and 1, and three elementary operations, referred to as NOT, AND, and OR. If we denote any two arbitrary elements as $i, j = 0, 1$, and use the standard

notation of the arithmetic system, the elementary operations are defined as follows:

$$\mathrm{NOT}\,(i) = \mathrm{NOT}\,i = \bar{i} = 1 - i\,, \tag{2.1.1}$$

$$\mathrm{AND}\,(i,j) = i\,\mathrm{AND}\,j = ij\,, \tag{2.1.2}$$

$$\mathrm{OR}\,(i,j) = i\,\mathrm{OR}\,j = i + j - ij\,. \tag{2.1.3}$$

In the context of computer science, an element $i$ of Boolean algebra is known as a *binary digit*, or *bit* for short. The elementary operations are called *Boolean logical gates*.

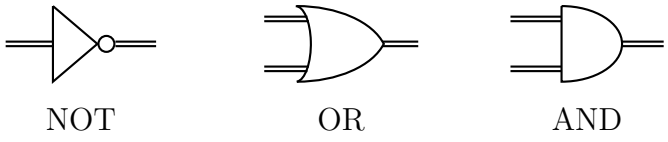



**Fig. 2.1.** The three fundamental Boolean gates.

A *Boolean circuit* is a sequence of Boolean gates. As is typically the case with electric circuits, Boolean circuits are often represented visually by *circuit diagrams*. Since every Boolean circuit is a deterministic process (because the action of each individual gate is deterministic), it follows that for every string of bits entering a Boolean circuit (the *input*), there is a unique string of bits that exits it (the *output*). Thus, any Boolean circuit defines a unique *Boolean function*. The reverse problem is also of interest: given a Boolean function, what is the Boolean circuit that realizes it? In this case, however, it can be proven that the circuit is not unique. For example, some circuits may contain more logical gates than others. A circuit that solves a problem using fewer logical gates is said to be more *efficient*.

The analysis of the depth (size) and complexity of a circuit, that is, the number of gates employed to perform a computation, is called *circuit complexity*. For example, it determines whether a problem can or cannot be solved by a certain model of computation. In addition to studying theoretical problems, circuit complexity is also of major practical importance. Suppose, for instance, that the NOT gate we have built sometimes gives the wrong result. That is, when the bit $i$ enters our NOT gate, on the other side, we sometimes read the same bit $i$ instead of $\bar{i}$. This is known as a *computational error*. If we use many of these faulty NOT gates in a circuit, the risk is that the error propagates through the circuit, potentially leading to an incorrect computational result. Obviously, the deeper the circuit, the greater the probability that the final computational result will be incorrect. It is, therefore, crucial to understand the level of error an algorithm can tolerate. Since errors are practically unavoidable, it is clear that this posed a serious concern in the earlier stages of our modern digital era.

Fortunately, by the mid-20th century, physicists had already discovered the electronic components needed to build reliable machines based on the binary circuit model of computation. These are what we call today *digital computers* or, simply, *computers*. The solution to any computational problem was thus reduced to the following steps: 1. translate the computational problem into an equivalent Boolean function, 2. find the instructions, that is, the algorithm, that solves it, and 3. wait

for the machine to do its job. It was believed that, thanks to digital computers, the solution to any solvable problem was merely a matter of time. The goal of computer science was then to discover more efficient algorithms and build more powerful computers.

The *quantum model of computation* is a completely different logical system. In contrast to the binary model of computation, the fundamental ideas of this model are based, as its name suggests, on the principles of quantum mechanics. Essentially, in quantum computation, we do not work with two distinct sets of objects, such as ones and zeros, but with their linear superposition. Instead of bits, we now use *qubits* (quantum binary digits). Moreover, the *quantum logical gates* are unitary transformations on qubits. A *quantum circuit* is a sequence of quantum gates connected by quantum channels, through which the qubits are transferred. As in any quantum experiment, a measurement is performed at the end of the quantum circuit. According to the postulates of quantum mechanics, when a qubit passes through a quantum gate or circuit, there is, in general, no certainty about the result of the measurement. In other words, the result is probabilistic, rather than deterministic, as in the binary (classical) case. A *quantum algorithm* is a specific set of instructions, including the initial qubit, the arrangement of gates, and the appropriate measurements at the end, intended to solve a computational problem. All these concepts form the foundation of a contemporary scientific paradigm known as the *quantum circuit model of computation*. The purpose of this model, particularly for scientists interested in the physical applications of the theory, is the development of machines that will implement it. These devices are the so-called *quantum computers*.

Let us examine in more detail some of the basic components of the quantum circuit model of computation.

A *single qubit* is the simplest element of this model. If we denote by $|0\rangle$ and $|1\rangle$ the two measurable states, any single qubit $|q\rangle$ will be a linear superposition of these states,

$$|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \sum_{i=0}^{1} \alpha_i |i\rangle \ , \tag{2.1.4}$$

where $\alpha_0$ and $\alpha_1$ are complex numbers. The states $|0\rangle$ and $|1\rangle$ are called the *computational basis states*. Quantum mechanics affirms that the probability of measuring the single qubit $|q\rangle$ in state $|i\rangle$ is $|\alpha_i|^2$. Instead of using complex numbers to specify the single qubit, two real numbers associated with angles in spherical coordinates can be used. The general expression for the single-qubit state vector in these new variables is:

$$|q(\vartheta, \phi)\rangle = \cos(\vartheta/2)|0\rangle + e^{i\phi} \sin(\vartheta/2)|1\rangle \, . \tag{2.1.5}$$

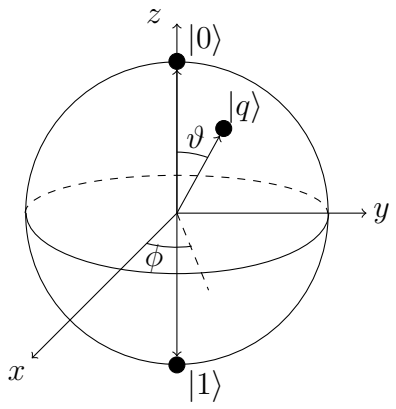



**Fig. 2.2.** Image representation of a single qubit in the Bloch sphere.

An *n-qubit* state is, in general, an entangled state of $n$ single qubits. It is expressed as follows,

$$|Q\rangle = \sum_{i_1,\ldots,i_n} \alpha_{i_1\ldots i_n} |i_1 \ldots i_n\rangle \ , \tag{2.1.6}$$

where each bit $i_1, \ldots, i_n$ in the string $i_1 \ldots i_n$ can take the values 0 and 1, and the coefficients $\alpha_{i_1\ldots i_n}$ are complex numbers. Note that some states of the $n$-qubit system are not entangled. For example, the following product states are non-entangled states:

$$|i \ldots i\rangle = |i\rangle^{\otimes n} \ . \tag{2.1.7}$$

A *quantum gate* is a unitary transformation applied to an input qubit. For instance, a *single-qubit gate* is a unitary transformation applied to a single qubit,

$$|q\rangle \longmapsto U\,|q\rangle = \sum_{i=0}^{1} \alpha_i\, U\,|i\rangle \ , \tag{2.1.8}$$

where, by the definition of unitary operator, $\langle q'|U^\dagger U|q\rangle = \langle q'|q\rangle$. The circuit element corresponding to the gate $U$ is shown below:

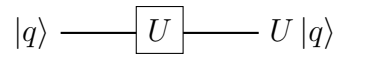



**Fig. 2.3.** Circuit diagram of a single-qubit gate.

Examples of such single-qubit gates are the *Pauli gates*,

$$X\,|0\rangle = |1\rangle \ , \qquad X\,|1\rangle = |0\rangle \ , \tag{2.1.9}$$

$$Y\,|0\rangle = i\,|1\rangle \ , \qquad Y\,|1\rangle = -i\,|0\rangle \ , \tag{2.1.10}$$

$$Z\,|0\rangle = |0\rangle \ , \qquad Z\,|1\rangle = -\,|1\rangle = e^{i\pi}\,|1\rangle \ . \tag{2.1.11}$$

The Pauli gate $Z$ is a special case, when $\phi = \pi$, of the *relative phase gate (phase shift gate)*,

$$P(\phi)\,|0\rangle = |0\rangle \ , \qquad P(\phi)\,|1\rangle = e^{i\phi}\,|1\rangle \ . \tag{2.1.12}$$

Each of the following operators, known as *rotation gates*, is a single-qubit rotation of $\theta_a$ radians about the $a$ axis,

$$R_a(\theta_a) = \cos(\theta_a/2)I - i\sin(\theta_a/2)\sigma_a\,, \tag{2.1.13}$$

where $I$ is the identity operator and $\sigma_a$, with $a = x, y, z$, is another notation for the Pauli gates ($\sigma_x = X$, $\sigma_y = Y$ and $\sigma_z = Z$).

In general, for an *n-qubit gate* acting on an $n$ qubit,

$$|Q\rangle \longmapsto U\,|Q\rangle = \sum_{i_1,\ldots,i_n} \alpha_{i_1\ldots i_n} U\,|i_1\ldots i_n\rangle\ . \tag{2.1.14}$$

For instance, the *CNOT gate* is an example of a two-qubit gate,

$$\mathrm{CNOT}\,|i\ j\rangle = |i\ j\oplus i\rangle\ , \tag{2.1.15}$$

where the symbol $\oplus$ denotes the binary sum, $j\oplus i = (j+i) \bmod 2$.

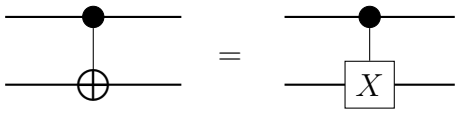



**Fig. 2.4.** Circuit representation of the CNOT gate.

Any unitary transformation, as can be shown, can be approximated by a finite quantum circuit (that is, a circuit made of a finite number of quantum gates),

$$|Q\rangle \longmapsto U\,|Q\rangle \approx U_N\ldots U_1\,|Q\rangle\ . \tag{2.1.16}$$

Furthermore, just as any classical circuit can be decomposed into a sequence of NOT, AND, and OR gates, any quantum circuit can be thought of as a composition of a finite number of elementary quantum gates. Any such finite set of gates is called a *set of universal quantum gates*. For example, the rotation gates, the phase shift gate, and the CNOT gate together form a universal set of quantum gates.

In quantum mechanics, any physical *measurement* is associated with a Hermitian operator called an *observable*, $M = M^\dagger$. Since the Pauli operators, as well as their tensor products, are Hermitian, we have

$$\sigma_a = \sigma_a^\dagger \tag{2.1.17}$$

$$\sigma_{a_1}\ldots\sigma_{a_N} = (\sigma_{a_1}\ldots\sigma_{a_N})^\dagger\,. \tag{2.1.18}$$

For this reason, they are often used to express any measurement performed at the end of a quantum circuit. Unlike classical systems, though, where the effects of a measurement are negligible, a measurement in quantum mechanics generally has an unpredictable effect on the physical system. This makes measurement an integral part of the quantum computational process. Therefore, we must always specify the measurements to be performed at the end of a quantum circuit! The outcomes are probabilistic rather than deterministic.

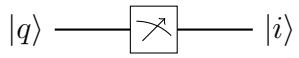



**Fig. 2.5.** A single-qubit measurement gate.

After this brief survey of both the binary and quantum models of computation, you may be asking yourself: why do we need quantum computers if we already have classical (digital) computers, which have proven to be highly reliable for solving most practical computational problems? There are two main reasons to believe that, in some cases, quantum computers will surpass classical computers. The first reason is that quantum computers may solve certain problems much faster than classical computers. This is what is meant when it is said that quantum computers will be more "efficient" or "powerful" than classical computers. In practical terms, this means that, in the future, quantum computers could be much smaller (and less complex) than classical computers designed for the same computational tasks. The second reason for the growing interest in quantum computing is that quantum computers will likely (though there is no formal proof yet) be able to solve computational problems that classical computers cannot solve. In other words, scientists expect them to tackle problems that even the most powerful digital computers imaginable may never be able to solve.

## 2.2 Computational Errors and Fault Tolerance

It is expected that future quantum computers, at least in the early stages of development, will not function exactly as described above. In other words, they will be prone to *computational errors*. Managing the propagation of these errors is a critical task in the progress toward a reliable quantum computer.

At the beginning of the modern digital era, digital components were also far from perfect, and there was a need for error correction—methods to detect and correct errors in the computational process. Nowadays, however, electronic components are so accurate that the probability of a computational error is negligible, and there is no longer a need for error correction.

One of the reasons quantum circuit components are so unreliable is that it is extremely difficult to isolate them from their environments. In fact, these interactions are so significant that they destroy the quantum mechanical behavior of the circuit elements. This phenomenon, well-known in quantum mechanics, is referred to as *decoherence*. The destructive influence of the environment has driven physicists and engineers to develop new techniques to mitigate the impact of external factors on circuit components. Meanwhile, theoretical physicists and computer scientists have devised procedures to detect and correct the various types of errors that may occur in these "noisy" quantum computers.

As an example of how experts cope with decoherence, consider the *bit-flip error* that may occur in the propagation of a qubit between two gates.

In the classical case, a bit flip is the only possible error, $i \to \bar{i}$. To protect it, we create multiple identical copies and send them. That is, instead of sending just $i$, we send $i\,i\,i$. If one of the bits is flipped—say, we receive $i\,\bar{i}\,i$ instead of $i\,i\,i$—we can measure the three bits, detect that the second bit has been flipped, and then correct it. The entire process can be schematically summarized as follows,

$$i \xrightarrow{encode} i\,i\,i \xrightarrow{send\ (error\ occurs)} i\,\bar{i}\,i \xrightarrow{detect} \xrightarrow{correct} i\,i\,i \xrightarrow{decode} i \tag{2.2.1}$$

Note that this method would not work if several errors could occur simultaneously. In fact, the *majority voting* strategy we just used will only correct a single wrong

bit. For example,

$$i \xrightarrow{encode} i\,i\,i \xrightarrow{send\ (errors\ occur)} i\,\bar{i}\,\bar{i} \xrightarrow{detect} \xrightarrow{correct} \bar{i}\,\bar{i}\,\bar{i} \xrightarrow{decode} \bar{i} \tag{2.2.2}$$

By repeating the initial bit many more times and assuming that the probability of an error occurring is very small, this correction procedure becomes completely reliable.

The *bit-flip quantum repetition code* is similar. However, there are some fundamental differences. Like in the classical case, instead of sending a single qubit $|q\rangle$, we send a three-qubit state $|q\rangle_L$,

$$|q\rangle = \sum_{i=1}^{2} \alpha_i |i\rangle \xrightarrow{encode} |q\rangle_L = \sum_{i=1}^{2} \alpha_i |i\,i\,i\rangle \xrightarrow{send} \tag{2.2.3}$$

To distinguish $|q\rangle_L$ from the initial qubit $|q\rangle$, the former is called the *logical qubit*, and each of the state vectors in $|i\rangle\,|i\rangle\,|i\rangle$ is referred to as the *physical qubit*. Now, during the process of transmitting the qubit, errors may occur. In our example, suppose a bit flip occurs. For the initial single qubit $|q\rangle$, this would have meant

$$|q\rangle = \sum_{i=1}^{2} \alpha_i |i\rangle \longrightarrow \sum_{i=1}^{2} \alpha_i |\bar{i}\rangle\ ; \tag{2.2.4}$$

however, since we are sending a logical three-qubit state, the error can now occur in any of the three physical qubits. Suppose that it is the second physical qubit that gets flipped,

$$|q\rangle = \sum_{i=1}^{2} \alpha_i |i\rangle \xrightarrow{encode} |q\rangle_L = \sum_{i=1}^{2} \alpha_i |i\,i\,i\rangle \xrightarrow{send\ (error\ occurs)} \sum_{i=1}^{2} \alpha_i |i\,\bar{i}\,i\rangle \tag{2.2.5}$$

The detection and correction of an error in a qubit is not as simple as in the classical case. In fact, we can measure a classical bit and leave it almost undisturbed. However, according to the principles of quantum mechanics, the measurement of a qubit will project it onto one of the observable states. To avoid this, we perform a *parity check* (see QC1, Subsection 5.2). After identifying the error, we correct it and recover the initial qubit. The entire process can be summed up as follows,

$$|q\rangle = \sum_{i=1}^{2} \alpha_i |i\rangle \xrightarrow{encode} |q\rangle_L = \sum_{i=1}^{2} \alpha_i |i\,i\,i\rangle \xrightarrow{send\ (error\ occurs)} \sum_{i=1}^{2} \alpha_i |i\,\bar{i}\,i\rangle$$

$$\xrightarrow{parity\ check} \xrightarrow{correct} |q\rangle_L = \sum_{i=1}^{2} \alpha_i |i\,i\,i\rangle \xrightarrow{decode} \sum_{i=1}^{2} \alpha_i |i\rangle = |q\rangle$$

Other types of errors can occur, and similar procedures have been developed to detect and correct them. In addition to errors related to the transmission of qubits, gates can also produce errors. For example, imagine you expect a qubit $|i\rangle$ to exit a gate, but instead, the qubit $|\bar{i}\rangle$ exits. Worse yet, a qubit transferred through a noisy channel might enter a faulty gate. If the two errors combine, the final qubit could become unrecognizable. Therefore, if we are not careful, errors can propagate throughout the circuit, making the final computation unreliable.

Finally, since the error correction process—namely, the encoding, detection, and correction of errors—is performed by quantum devices, these devices will inevitably introduce additional errors. As a result, error correction requires larger quantum computers, which increases the probability that the final computational result will be incorrect. The good news is that it has been proven that, under certain conditions, error correction codes can reduce computational errors to arbitrarily small levels. A *fault-tolerant quantum computer* is a quantum computer designed such that the errors occurring in the logical qubits at each stage of the process are corrected along the way, ensuring the final computational result is reliable. However, this technology remains far from being realized.

## 2.3 Hybrid Quantum-Classical Algorithms

For years, it was thought that quantum computers had to be built exclusively with quantum components. This seemed obvious: given that we wanted to demonstrate the superiority of quantum over classical computation, the quantum device had to be purely quantum. The main obstacles to this ideal goal were, and still are, the development of new hardware (less noisy) and the invention of appropriate quantum error-correcting codes. This future situation is known as the *fault-tolerant quantum era*. However, in recent years, scientists have come to accept that fault-tolerant quantum computers will not be available anytime soon. As a result, they began looking for more realistic algorithms that could be implemented on near-term quantum computers, which are characterized by a moderate amount of noise and a relatively small number of qubits and gates. This is the stage we are currently in, known as the *Noisy-Intermediate Scale Quantum* (*NISQ*) *era*. According to experts, we will likely remain in this phase for several years (even decades) before achieving fault tolerance.

The algorithms expected to be implemented in the near term are known as *hybrid quantum-classical algorithms*. These combine quantum and classical components: the quantum subroutines address problems that are hard for classical computers, while the classical computer handles tasks where its efficiency is well-established. For example, the variational quantum algorithms (VQAs) discussed below are NISQ algorithms designed to demonstrate *quantum advantage* (practical *quantum supremacy*) in the near future. Since many problems—not only in physics and chemistry but also in finance—share a common basic structure, the techniques used in VQAs can be applied to a wide variety of situations. It is common to hear that some variational quantum algorithms, particularly the quantum approximate optimization algorithm (QAOA) we will present below, are *heuristic*. This means that, while there is currently no rigorous proof they are more efficient than known classical algorithms, there are strong theoretical reasons to be optimistic. The promise is that future results may demonstrate their advantage.

# Chapter 3

# Non-Quantum Approaches

With the basic understanding of quantum computing gained in the previous chapter, we are now ready to explore some of the most complex computational problems in finance. In the following chapter, we will examine how quantum computers can enhance these methods. We begin with a brief overview of some elementary financial concepts.

In finance, a *portfolio* refers to a collection of assets that an investor owns, such as stocks, bonds, or real estate.[1] For simplicity, let us assume that these assets are held for a fixed period. At the end of this period, some assets will have increased in value, while others will have decreased. This means that the investor will profit from some assets and experience losses on others. The objective of a portfolio manager is to maximize overall returns while managing risk. Balancing assets to achieve the optimal tradeoff between risk and return is a mathematical challenge known as *portfolio optimization*. This is far from trivial, as it requires analyzing large volumes of data, including historical performance, correlations among assets, and risk factors. When investments span multiple successive periods, the problem becomes even more complex, as it requires predicting future market conditions and adjusting strategies accordingly.

*Derivatives* are financial instruments whose values are derived from the price of one or more underlying assets. These underlying assets can be stocks, bonds, commodities, currencies, or interest rates. For example, the value of a derivative tied to a car manufacturing company's stock could depend on factors such as the prices of raw materials, the cost of production components, market demand for vehicles, and broader economic, geopolitical, and environmental influences. The pricing of derivatives is particularly complex because it involves predicting the value of the underlying assets, which are influenced by multiple interconnected variables. This makes derivative pricing one of the most intricate problems in finance. Sophisticated mathematical models and powerful computational techniques, such as numerical methods and simulations, have been developed to estimate prices and manage the risks associated with derivatives.

An *option* is a special type of derivative. In simple terms, an option gives its owner the right, but not the obligation, to buy or sell an underlying asset at a predetermined price within a specified time frame. A *call option* gives the holder the right to buy the asset at a fixed price, known as the *strike price*, before the option's expiration date. If the market price of the asset rises above the strike price,

[1] For most of the financial concepts, I have relied on J. C. Hull's classic book, *Options, Futures, and Other Derivatives*.

the option holder can exercise the option, purchase the asset at the lower fixed price, and then sell it at the higher market price to make a profit. A *put option*, on the other hand, gives the holder the right to sell the asset at a fixed price before the expiration date. If the market price of the asset falls below the strike price, the option holder can sell the asset at the higher fixed price, thereby avoiding losses that would occur by selling it at the lower market price. While the basic concept of options is relatively straightforward, predicting their future prices is extremely challenging. The price of an option is influenced by several factors, such as its volatility, the time remaining until expiration, and prevailing interest rates. These interdependencies make option pricing a complex mathematical and computational problem that requires advanced modeling techniques.

*Financial services* refer to the various offerings provided by financial institutions to their clients. It is essential for these institutions to thoroughly assess the potential consequences and risks associated with delivering these services. For example, when individuals or businesses apply for loans from a bank, the bank must evaluate their creditworthiness and ability to repay. This process involves analyzing factors such as income, outstanding debts, repayment history, and credit behavior. The outcome of this evaluation is a numerical score, known as the *credit score*, which quantifies the risk that the borrower might default. This area of financial theory is referred to as *credit risk assessment*. Optimizing the loan evaluation process is critical for banks, as they need to minimize financial losses while providing fair and timely services to customers. The goal is to identify and reject high-risk loan applicants, while avoiding the denial of loans to eligible borrowers who genuinely need financial support.

Another area of concern for financial institutions is *credit card fraud*. Preventing fraudulent transactions is essential, but unnecessary interruptions in legitimate transactions can frustrate customers. For example, if someone takes a flight and uses their credit card in two geographically distant locations within a short period, the bank may flag the activity as suspicious. To reduce fraud while ensuring a smooth customer experience, banks rely on advanced techniques to identify suspicious behaviors and anomalies.

In addition to satisfying the ever-growing demands imposed by customers, banks must also comply with governments and regulatory bodies. One of the critical concerns in this context is *anti-money laundering*. Financial institutions are responsible for ensuring that the money they manage does not originate from illegal activities such as tax evasion, corruption, or trafficking. To meet this requirement, banks employ systems to monitor and flag suspicious transactions. Alongside anti-money laundering, *risk management* is another crucial responsibility for banks. Financial organizations are subject to regulations that limit the level of risk they can assume when investing customer funds. To ensure compliance, they use sophisticated models to evaluate and mitigate risks, ensuring that investments remain within acceptable thresholds.

To sum up, financial institutions can enhance both their internal operations and client services by adopting innovative technologies. As we will explore in the next chapter, quantum computers have the potential to significantly advance classical methods—an ambitious pursuit in today's increasingly complex and competitive environment.

## 3.1 Optimization Theory

Optimization theory is a vast mathematical field with numerous applications in pure science, engineering, industry, and finance. In this section, we focus solely on key concepts and techniques relevant to financial portfolio optimization.[2]

From a mathematical perspective, *optimization* is a straightforward concept. Given a real-valued function of a set of independent variables, the goal is to determine the values of these variables that minimize (or maximize) the function. In some cases, the independent variables are subject to constraints, which can make the problem extremely complex. The specific form of the function relative to the independent variables, as well as the constraints imposed on those variables, depends on the particular problem being considered. Due to the diversity and complexity of such problems, a variety of methods have been developed to address different types of optimization problems.

Suppose a real-valued function $f$ is defined on an $n$-dimensional space of binary variables; that is, they can only take the values 0 or 1. If $f$ is linear, we have that

$$f(b_1, b_2, \ldots, b_n) = c_1 b_1 + c_2 b_2 + \ldots + c_n b_n = \sum_{i=1}^{n} c_i b_i \,, \tag{3.1.1}$$

where the $c_i$'s are real constants and $b_i \in \{0, 1\}$, for $i = 1, 2, \ldots, n$. For a quadratic function $f$, quadratic terms must also be included:

$$\begin{aligned} &b_1 Q_{11} b_1 + b_1 Q_{12} b_2 + \ldots + b_1 Q_{1n} b_n + b_2 Q_{21} b_1 + b_2 Q_{22} b_2 + \ldots + b_n Q_{nn} b_n = \\ &\quad = b_1 Q_{11} + b_1 Q_{12} b_2 + \ldots + b_1 Q_{1n} b_n + b_2 Q_{21} b_1 + b_2 Q_{22} + \ldots + b_n Q_{nn} \,, \end{aligned} \tag{3.1.2}$$

where the $Q_{ij}$'s are constants. In the second line, we have used that $b_i^2 = b_i$. Using index notation, this sum can be written as

$$\sum_{i,j=1}^{n} b_i Q_{ij} b_j \,. \tag{3.1.3}$$

A quadratic function $f$ then takes the following form

$$f(b_1 \ldots b_n) = \alpha \sum_{i,j=1}^{n} b_i Q_{ij} b_j + \beta \sum_{i=1}^{n} c_i b_i \,, \tag{3.1.4}$$

where the constants $\alpha$ and $\beta$ have been included for greater generality. We will assume that, as is the case in most practical situations, $Q_{ij} = Q_{ji}$.

It is common practice to arrange the $n$ independent variables $b_1$, $b_2$, …, $b_n$ into a column vector $\mathbf{b} = [b_1 \; b_2 \; \ldots \; b_n]^T$. Similarly, we define the column vector $\mathbf{c} = [c_1 \; c_2 \; \ldots \; c_n]^T$ and the $n \times n$ symmetric matrix $Q = [Q_{ij}]$, allowing the quadratic function to be expressed as

$$f(\mathbf{b}) = \alpha \, \mathbf{b}^T Q \, \mathbf{b} + \beta \, \mathbf{c}^T \mathbf{b} \,. \tag{3.1.5}$$

[2]For more details, see my notes, "An Introduction to Portfolio Optimization with Quantum Computers," hereafter referred to as QC3. You can find it on my LinkedIn.

The purpose of a *quadratic unconstrained binary optimization problem* (*QUBO problem*) is to minimize (or maximize) a function of this form

$$\min_{\mathbf{b}} \alpha\, \mathbf{b}^T Q\, \mathbf{b} + \beta\, \mathbf{c}^T \mathbf{b} \qquad \Big(\text{or} \quad \max_{\mathbf{b}} \alpha\, \mathbf{b}^T Q\, \mathbf{b} + \beta\, \mathbf{c}^T \mathbf{b}\Big)\,. \tag{3.1.6}$$

Let us now briefly explore how this relates to the optimization of financial portfolios.

### 3.1.1 Portfolio Optimization

Imagine the following scenario: You have a certain amount of money—cash for that matter—and you wish to invest it in the stock market, with the expectation, of course, of making a profit over a given period of time. The problem you face is this: What is the best selection of stocks, and what is the optimal amount of money to invest in each, so that, at the end of the period, you maximize the return on your investment? In simple terms, this is the *portfolio optimization problem*.

If you begin with a portfolio of $S$ stocks, the initial value of the portfolio is given by

$$p_P(0) = \sum_{s=1}^{S} p_s(0)\,, \tag{3.1.7}$$

where $s = 1, 2, \ldots, S$, and $p_s(0)$ is the initial amount of money invested in the $s$th stock. After a time $T$, the value of the investment in the $s$th stock becomes $p_s(T)$, and the total value of your portfolio at that time is given by

$$p_P(T) = \sum_{s=1}^{S} p_s(T)\,. \tag{3.1.8}$$

The *portfolio return rate* is defined as follows:

$$\begin{aligned} R_P(T) &= \frac{p_P(T) - p_P(0)}{p_P(0)} = \frac{\sum_{s=1}^{S} \big(p_s(T) - p_s(0)\big)}{p_P(0)} \\ &= \sum_{s=1}^{S} \frac{p_s(0)}{p_P(0)} R_s(T) = \sum_{s=1}^{S} w_s(0) R_s(T)\,, \end{aligned} \tag{3.1.9}$$

where

$$R_s(T) = \frac{p_s(T) - p_s(0)}{p_s(0)}\,, \tag{3.1.10}$$

is the $s$th *stock return rate*, and

$$w_s(0) = \frac{p_s(0)}{p_P(0)}\,. \tag{3.1.11}$$

Note that $w_s(0)$ represents the proportion of the initial capital invested in the $s$th stock; it is known as the $s$th *stock weight*.

The portfolio return rate (3.1.9) can be written in vector notation as follows:

$$R_P(T) = \mathbf{w}^T(0)\mathbf{R}(T)\,. \tag{3.1.12}$$

Your challenge as an investor is that, due to unknown and unpredictable factors, the future values of the stocks in your portfolio are uncertain. They may increase as you expect, but they could also decrease, resulting in a loss. There are so many variables at play that even predicting the short-term future value of a single stock seems impossible.

The belief that future prices are unpredictable traces back to the pioneering observations made by Louis Bachelier at the beginning of the 20th century. More than a century of historical records has confirmed (though, it is fair to say, not conclusively) that stock price movements do not follow any recognizable pattern. In fact, stock price movements are often described by a random walk (the same phenomenon studied in many areas of physics). Because of this, we should focus on the *expected portfolio return rate*,

$$\mathbb{E}\big(R_P(T)\big) = \sum_{s=1}^{S} w_s(0)\, \mathbb{E}\big(R_s(T)\big)\,, \tag{3.1.13}$$

where $\mathbb{E}\big(R_s(T)\big)$ denotes the expected return rate of the $s$th stock. A more concise notation is

$$\mu_P(T) = \sum_{s=1}^{S} w_s(0)\, \mu_s(T)\,, \tag{3.1.14}$$

where the Greek letter $\mu$ stands for *mean* in probability theory. In vector notation,

$$\mu_P(T) = \mathbf{w}^T(0)\boldsymbol{\mu}(T) = \boldsymbol{\mu}^T(T)\mathbf{w}(0)\,. \tag{3.1.15}$$

Associated with the uncertainty of a portfolio's return rate is the concept of risk. *Risk*, simply put, is the possibility of losing money on an investment. In *modern portfolio theory*, also known as the *mean-variance model* (developed by Harry Markowitz in the 1950s), risk is assumed to be directly proportional to the uncertainty of the portfolio's return rate. In other words, the higher the uncertainty of the portfolio return rate, the greater the risk. Note that, according to this definition, risk also encompasses the possibility of earning more than expected. However, risk is generally understood with a negative connotation, often interpreted as the likelihood of earning less than expected or even incurring a loss.

Since the price movements of two or more stocks in a portfolio can be correlated, the risk of an investment portfolio must account for these correlations. The mathematical object that incorporates these correlations is the *covariance matrix* $\Sigma\big(\mathbf{R}(T)\big)$. Modern portfolio theory defines the *variance of the portfolio return rate* as

$$\sigma_P^2(T) = \mathbf{w}^T(0)\, \Sigma\big(\mathbf{R}(T)\big)\, \mathbf{w}(0)\,, \tag{3.1.16}$$

and the risk is directly proportional to it.

Given an expected portfolio return rate $\mathcal{M}$, the goal of a portfolio manager is to minimize the risk. That is,

$$\min_{\mathbf{w}(0)} \alpha\, \mathbf{w}^T(0)\, \Sigma\big(\mathbf{R}(T)\big)\, \mathbf{w}(0)\,, \tag{3.1.17}$$

given the expected return

$$\boldsymbol{\mu}^T(T)\mathbf{w}(0) = \mathcal{M}\,, \tag{3.1.18}$$

and, of course,

$$\mathbf{1}^T \mathbf{w}(0) = 1\,. \tag{3.1.19}$$

The positive number $\alpha$ is called the *risk aversion coefficient* and measures the investor's tolerance for risk.

Suppose that, for instance, we are interested in constructing a portfolio where each stock is either included or excluded. In this case, we are dealing with a *binary portfolio optimization problem*. The goal is to minimize the risk,

$$\min_{\mathbf{b}(0)} \alpha\, \mathbf{b}^T(0)\, \Sigma\big(\mathbf{R}(T)\big)\, \mathbf{b}(0)\,, \tag{3.1.20}$$

given the expected return

$$\boldsymbol{\mu}^T(T)\mathbf{b}(0) = \mathcal{M}\,. \tag{3.1.21}$$

Using the Lagrange multipliers method, we can restate the problem as

$$\min_{\mathbf{b}(0)} \alpha\, \mathbf{b}^T(0)\, \Sigma\big(\mathbf{R}(T)\big)\, \mathbf{b}(0) - \lambda\, (\boldsymbol{\mu}^T(T)\mathbf{b}(0) - \mathcal{M})\,, \tag{3.1.22}$$

or, equivalently,

$$\min_{\mathbf{b}(0)} \alpha\, \mathbf{b}^T(0)\, \Sigma\big(\mathbf{R}(T)\big)\, \mathbf{b}(0) - \lambda\, \boldsymbol{\mu}^T(T)\mathbf{b}(0)\,. \tag{3.1.23}$$

Recall that the cost function of a QUBO problem is given by (3.1.5),

$$f(\mathbf{b}) = \alpha\, \mathbf{b}^T Q\, \mathbf{b} + \beta\, \mathbf{c}^T \mathbf{b}\,. \tag{3.1.24}$$

Comparing these two functions, we note that

$$\alpha = \alpha\,, \quad \mathbf{b}(0) = \mathbf{b}\,, \quad \Sigma\big(\mathbf{R}(T)\big) = Q,\,, \quad -\lambda = \beta\,, \quad \boldsymbol{\mu}(T) = \mathbf{c}\,. \tag{3.1.25}$$

We conclude that the binary portfolio optimization problem is a QUBO problem.

QUBO problems are known to be extremely difficult to solve ("NP-hard"), and various computational techniques have been developed to approximate solutions. In the following chapter, we will explore how quantum computers, specifically hybrid classical-quantum systems, can assist in solving them.

## 3.2 Monte Carlo Simulation

Monte Carlo simulation (MCS) has a fascinating history that dates back to one of the most remarkable periods in the history of science and technology: the discovery of nuclear energy's potential and the invention of the first electronic computers. This era, marked by groundbreaking innovation during and after World War II, saw the emergence of computational techniques that would revolutionize not only science and technology but also society as a whole. Before delving into how finance connects to this, let us take a moment to explore the origins of Monte Carlo simulation.

### 3.2.1 Stochastic Processes

Simulating physical phenomena with the help of computers began with the advent of the first electronic programmable computers.[3] The essential idea is to reproduce the behavior of phenomena that, either due to their complexity (as with many processes in hydrodynamics and atomic physics) or because they cannot be reproduced in the laboratory (such as nuclear processes inside stars or the collision of two black holes), would otherwise be inaccessible. By simulating these phenomena, predictions can be made that theoretical or experimental physicists can later test or prove. Thanks to this method, computer scientists themselves can gain access to knowledge that would otherwise be impossible to obtain.

Suppose that a neutron is injected into a fissile material (for example, Uranium-235 or Plutonium-239). Due to the probabilistic nature of quantum phenomena, the neutron will interact with its environment in different ways, each event occurring with a certain probability. Being electrically neutral, the primary force at play is the strong nuclear force. The neutron can, for instance, be scattered or absorbed by a nucleus of the fissile material. Scattering can be of two types: elastic, where the neutron is deflected by the nucleus without transferring energy to it, or inelastic, where the neutron transfers some energy to the nucleus. Each of these scenarios has an associated probability. Similarly, the absorption process can occur in one of two ways: the neutron can be absorbed, producing an isotope of uranium, or it can induce fission. Finally, the neutron may also escape the fissile material and collide with the walls of the containing vessel. Each of these outcomes has its own probability. After analyzing these initial possibilities, we then proceed to examine the subsequent reactions. As you can see, as we account for more and more interactions, the process becomes increasingly complex to study. In principle, only by tracking each neutron through all possible timelines can we establish a statistical picture of the fate of the initial neutron.

To address the complexity of the task, nuclear physicists in the late 1930s and 1940s developed precise differential equations to describe the interaction of large numbers of neutrons within fission material, whether in nuclear reactors or weapons. These equations, considered by many to be the pinnacle of theoretical nuclear physics, are the *neutron diffusion equation* and the *transport equation*. They are solvable in simple cases, such as those involving basic geometries and materials. However, for more complex situations, analytical solutions are extremely difficult or impossible to find with current techniques. This is where Monte Carlo simulation comes to the rescue.

When applied to nuclear fission, *Monte Carlo simulation* does not rely on the behavior of a large number of neutrons. Instead, the method simulates the timelines of many individual neutrons, with each simulation randomly determining the outcome based on defined probabilities. By aggregating the results of these simulations, analysts can accurately estimate the likelihood of various outcomes, particularly the likelihood of a nuclear explosion.

[3] A brief history of the Monte Carlo method, written by one of its participants, can be found in N. Metropolis, "The Beginning of the Monte Carlo Method". The article by the historian and philosopher of physics P. Galison, "Computer Simulations and the Trading Zones", is an excellent account of the many factors that fostered the propagation of Monte Carlo simulation during the postwar period. For its many uses in contemporary research, see D. P. Kroese, "Why the Monte Carlo Method is So Important Today".

Conceptually, the application of MCS in finance stems from its origins in nuclear physics. Suppose you have a portfolio of options. For simplicity, assume that the options depend on a single stock.[4] Let us use the nuclear fission analogy to understand how MCS naturally applies to this case. The goal is to follow the history of the stock. Initially, the stock has several possibilities: it can either remain unchanged, go up in price, or go down, each with a certain probability. After the first outcome, for each of the previous possibilities, the stock can again remain unchanged, go up, or go down, with probabilities depending on the new market conditions. As time passes, the stock price has more and more possible histories. At every moment, the MCS estimates the probability associated with the different prices of the underlying stock. Note that this is similar to the diffusion of neutrons in fissile material. MCS does not attempt to solve a differential equation, such as the Black-Scholes equation (1973) for options, but instead provides a probabilistic description of the option's behavior, considering the multiple price paths of a statistical sample.

These are examples of Monte Carlo simulation applied to stochastic processes. The canonical example of a stochastic process is the *random walk*. Suppose you are standing on a straight line, which we associate with the axis $x$, and you can only step in the positive $(+)$ or negative $(-)$ direction with a fixed length $L$. For mathematical simplicity, let $L$ be the unit length, that is, $L = 1$. To decide whether you move to the right $(+)$ or to the left $(-)$, you toss a fair coin. If the result is heads $(H)$, you move to the right, and if it is tails $(T)$, you move to the left. Since the coin is fair, the probability of moving to the right or to the left is $p_H = 1/2$ and $p_T = 1/2$, respectively. The probability distribution is thus given by $\{H, p_H = 1/2; T, p_T = 1/2\}$. After tossing the coin $K$ times, your distance from the origin will be

$$\begin{aligned} D_K &= (\delta_{H1} - \delta_{T1}) + (\delta_{H2} - \delta_{T2}) + \ldots + (\delta_{HK} - \delta_{TK}) \\ &= \sum_{k=1}^{K} \delta_{Hk} - \sum_{k=1}^{K} \delta_{Tk} \,. \end{aligned} \tag{3.2.1}$$

In other words, it is the difference between the number of steps taken to the right and the number of steps taken to the left.

In probability language, if $\{H, T\}$ is the sample space, that is, the set of possible outcomes of a coin toss, then the step taken is a random variable,

$$X \colon \{H, T\} \to \{1, -1\} \,, \tag{3.2.2}$$

with $X(H) = 1$ and $X(T) = -1$. A random walk is defined as a sequence of random steps,

$$\{X_1, \ldots, X_k, \ldots, X_K\} \,, \tag{3.2.3}$$

where $X_k$ is the value of the $k$th step, taking the value 1 if the coin shows $H$ and $-1$ if it shows $T$. A set of random variables like this is called a *stochastic process*.

In general, the distance from the origin after $K$ steps is given by

$$D_K = \sum_{k=1}^{K} X_k = \sum_{k=1}^{K-1} X_k + X_K = D_{K-1} + X_K \,. \tag{3.2.4}$$

[4] Refer to the seminal paper by P. Boyle, "Options: A Monte Carlo Approach" (1977). The following textbooks provide a technical introduction to MCM in finance: P. Glasserman, *Monte Carlo Methods in Financial Engineering*, and P. Jäckel, *Monte Carlo Methods in Finance*.

This simple random walk can be generalized in several ways, such as by extending it to two or more dimensions.

*Brownian motion*, discovered by botanist Robert Brown in the late 19th century while observing pollen grains suspended in a liquid, is a two-dimensional example of a random walk. In this case, we need a tetrahedron to decide whether to move right, left, up, or down. The resulting position of the suspended particle evolves as a combination of these independent motions, creating a trajectory that reflects its random nature.

We can also generalize the simple random walk by rolling a die instead of tossing a coin. For instance, we could roll a fair die and define the following random variable:

$$X(1) = 1\,, \qquad X(2) = 2\,, \qquad X(3) = 3\,, \tag{3.2.5}$$

$$X(4) = -1\,, \qquad X(5) = -2\,, \qquad X(6) = -3\,. \tag{3.2.6}$$

Returning to finance, since the pioneering work of Bachelier, stock price movements have been regarded as stochastic processes. Suppose we want to analyze the movement of a stock. Let $p_s(0)$ denote the initial price. Since the price at time $T$ is a random variable, there are several possible prices at that time,

$$^1p_s(T), \ldots, ^i p_s(T), \ldots, ^N p_s(T). \tag{3.2.7}$$

The index $i$, ranging from 1 to $N$, represents the $i$th possible price. Each of the prices $^ip_s(T)$ occurs with some probability. More generally, at any time $kT$, where $k = 1, \ldots, K$, we have $^1p_s(kT), \ldots, ^i p_s(kT), \ldots, ^N p_s(kT)$, where $^ip_s(kT)$ represents the $i$th possible price of stock $s$ at time $kT$. The stochastic movement of the stock price will be described by a set of random variables representing the price at different time points,

$$\{^i p_s(kT)\}\,, \tag{3.2.8}$$

where $k = 1, \ldots, K$ and $i = 1, \ldots, N$. This set of potential prices can be conveniently represented by an $N \times K$ matrix,

$$\begin{bmatrix} ^1p_s(T) & ^1p_s(2T) & \cdots & ^1p_s(KT) \\ \vdots & \vdots & \ddots & \vdots \\ ^Np_s(T) & ^Np_s(2T) & \cdots & ^Np_s(KT) \end{bmatrix}. \tag{3.2.9}$$

Suppose that $^ip_s((k-1)T)$ represents the price of the stock at time step $(k-1)T$, and $^jp_s(kT)$ denotes the price at time $kT$, where $i, j = 1, \ldots, N$. With a slight modification to definition (3.1.10), the stock return rate during this time period is given by,

$$\begin{aligned} R_s^{ij}((k-1)T, kT) &= \frac{^jp_s(kT) - {}^ip_s((k-1)T)}{^ip_s((k-1)T)} \\ &= \frac{^jp_s(kT)}{^ip_s((k-1)T)} - 1\,. \end{aligned} \tag{3.2.10}$$

Simplifying the notation by introducing $R_s^{ij}(\Delta T_k) = R_s^{ij}((k-1)T, kT)$,

$$R_s^{ij}(\Delta T_k) = \frac{^jp_s(kT)}{^ip_s((k-1)T)} - 1\,. \tag{3.2.11}$$

More generally, given two moments in time, $mT$ and $nT$, where $m < n$, the stock return rate during this period is

$$\begin{aligned} R_s^{i_m i_n}(mT, nT) &= \frac{{}^{i_n}p_s(nT)}{{}^{i_m}p_s(mT)} - 1 \\ &= \frac{{}^{i_n}p_s(nT)}{{}^{i_{n-1}}p_s((n-1)T)} \frac{{}^{i_{n-1}}p_s((n-1)T)}{{}^{i_{n-2}}p_s((n-2)T)} \cdots \frac{{}^{i_{m+1}}p_s((m+1)T)}{{}^{i_m}p_s(mT)} - 1\,. \\ &= \prod_{k=1}^{n-m} R_s^{i_{n-k}i_{n-k+1}}(\Delta T_{n-k+1}) - 1\,. \end{aligned} \tag{3.2.12}$$

In order to make this equation more manageable, the *log stock return rate* is defined:

$$\log\left(R_s^{i_m i_n}(mT, nT) + 1\right) = \sum_{k=1}^{n-m} \log R_s^{i_{n-k}i_{n-k+1}}(\Delta T_{n-k+1})\,. \tag{3.2.13}$$

Let us pause the mathematical presentation here and shift focus to how Monte Carlo simulation extracts useful statistical information from the random walk. The application to stock prices is analogous.

Suppose you are playing the random walk game, but you are unaware that the coin is biased. After tossing the coin $K$ times, suppose you have followed a certain stochastic path,

$$\{D_1, \ldots, D_k, \ldots, D_K\}\,, \tag{3.2.14}$$

where $D_k = \sum \delta_{Hk} - \sum \delta_{Tk}$, for every $k = 1, \ldots, K$. The question is: by analyzing this stochastic path, how much can you learn about the probability distribution $\{H, p_H \neq 1/2; T, p_T = 1 - p_H \neq 1/2\}$ underlying it? Once you have discovered the probability distribution, you can make statistical predictions about the future behavior of the path.

MCS proceeds as follows: It selects a probability distribution, say $\{H, p_H; T, p_T\}_1$, and generates many possible stochastic paths. At each time step, the simulation computes the corresponding value of the random walk and averages these values across all generated paths to obtain an estimate of the expected behavior for the given probability distribution. This procedure is repeated for different probability distributions, $\{H, p_H; T, p_T\}_2$, ..., $\{H, p_H; T, p_T\}_G$. All these statistical models are compared with the observed behavior (as described in (3.2.14)). The one that most closely replicates the observed behavior is chosen as the underlying probability distribution that governs the random walk. Future predictions are then based on this selected probability distribution.

### 3.2.2 Monte Carlo and the Greeks

When a business or financial organization acquires or sells an option, it assumes the risk of potential financial loss. The so-called *Greeks* (or Greek letters) are various measures of the risks involved. Proper evaluation and management of the Greeks enable businesses to eliminate or reduce risk to a tolerable level. In most cases, the objective is not to generate profit but to minimize potential losses. This approach is referred to as *hedging*.

The most elementary of the Greeks is *delta* ($\Delta$). It measures how the price of an option, $C$, varies in response to changes in the price of the underlying asset, $s$ (in

previous sections denoted by $p_s$). In mathematical terms, delta is the derivative of the option price with respect to the price of the underlying asset:

$$\Delta_C = \frac{\partial C(s)}{\partial s}\,. \tag{3.2.15}$$

We use a partial derivative because the option price, in practice, depends on multiple parameters. The delta of a call option ranges from 0 to 1, reflecting the fact that when the price of the underlying asset is below the strike price, the probability of the option expiring "in-the-money" is close to zero. On the other hand, as the underlying asset's price rises above the strike price, the probability of expiring in-the-money increases, causing the delta to steadily approach 1. For a put option, the behavior is the opposite of that of a call option. Its delta ranges from $-1$ to 0. In fact, when the price of the underlying asset is below the strike price, the probability of the option expiring in-the-money is close to 1, resulting in a delta near $-1$. As the underlying asset's price approaches or exceeds the strike price, the probability of expiring in-the-money decreases, and the delta moves toward 0. The negative sign is a convention indicating the inverse relationship between the put option's price and the underlying asset's price. Let us illustrate this with a simple example of a portfolio of options.

Suppose you own a portfolio of $n$ options, all based on the same underlying asset, $s$. The price of your portfolio is expressed as:

$$p_P(s) = \sum_{i=1}^{n} w_i\, C_i(s)\,. \tag{3.2.16}$$

Using the definition, delta is then given by,

$$\Delta_P(s) = \frac{\partial p_P(s)}{\partial s} = \frac{\partial}{\partial s}\sum_{i=1}^{n} w_i\, C_i(s) = \sum_{i=1}^{n} w_i\, \frac{\partial C_i(s)}{\partial s} = \sum_{i=1}^{n} w_i\, \Delta_i(s)\,. \tag{3.2.17}$$

Here, $\Delta_i$ represents the delta of the $i$th option. If the goal of your portfolio is to hedge the investment by minimizing exposure to price movements of the underlying asset, you need an appropriate combination of put and call options, with some $\Delta_i$'s being positive and others negative. This creates a *delta-neutral portfolio*, where the total delta is close to zero.

The next Greek is *gamma* ($\Gamma$). Using a physical analogy, if $\Delta$ represents velocity, then, $\Gamma$ represents acceleration,

$$\Gamma_C(s) = \frac{\partial^2 C(s)}{\partial s^2} = \frac{\partial \Delta_C(s)}{\partial s}\,. \tag{3.2.18}$$

For a portfolio of options,

$$\Gamma_P(s) = \frac{\partial^2 p_P(t,s)}{\partial s^2} = \sum_{i=1}^{n} w_i\, \Gamma_i(s)\,, \tag{3.2.19}$$

where $\Gamma_i$ denotes the gamma of the $i$th option. Note that when the gammas of all the options in a portfolio are approximately zero, the gamma of the portfolio is also nearly zero. This implies that the portfolio's delta remains nearly constant. Consequently, the portfolio's price changes at a constant rate with respect to the

price of the underlying asset. In contrast, when the gammas of the individual options are significantly greater than zero, the portfolio's delta becomes highly sensitive to changes in the underlying asset's price. This increased sensitivity makes the portfolio's price less predictable, introducing greater risk when forecasting its value.

Suppose now that the value of the portfolio of options, $p_P$, depends not only on the price of the underlying asset, $s$, but also explicitly on time, $t$. We thus have that,

$$p_P(t,s) = \sum_{i=1}^{n} w_i \, C_i(t,s) \, . \tag{3.2.20}$$

The *Black-Scholes equation* states that the value of the portfolio satisfies the following partial differential equation:

$$\frac{\partial p_P(t,s)}{\partial t} + rs \, \frac{\partial p_P(t,s)}{\partial s} + \frac{1}{2} \, \sigma^2 s^2 \, \frac{\partial^2 p_P(t,s)}{\partial s^2} = r \, p_P(t,s) \, , \tag{3.2.21}$$

where $r$ is the (risk-free) *interest rate* and $\sigma$ is the *volatility* (in statistical jargon, the standard deviation over a certain period of time). Using that $\Delta_P = \partial p_P / \partial s$ and $\Gamma_P = \partial^2 p_P / \partial s^2$, we can rewrite the Black-Scholes equation as,

$$\frac{\partial p_P(t,s)}{\partial t} + rs \, \Delta_P(t,s) + \frac{1}{2} \, \sigma^2 s^2 \, \Gamma_P(t,s) = r \, p_P(t,s) \, . \tag{3.2.22}$$

The first term defines the Greek *theta* ($\Theta$),

$$\Theta(t,s) = \frac{\partial p_P(t,s)}{\partial t} \, . \tag{3.2.23}$$

Using this definition and simplifying the notation by omitting the dependence on time and the asset price, the Black-Scholes equation simplifies to:

$$\Theta + rs \, \Delta_P + \frac{1}{2} \, \sigma^2 s^2 \, \Gamma_P = r \, p_P \, . \tag{3.2.24}$$

For a delta-neutral portfolio ($\Delta_P = 0$), that is, a portfolio constructed to be insensitive to small changes in the price of the underlying stock, the Black-Scholes equation reduces to:

$$\Theta + \frac{1}{2} \, \sigma^2 s^2 \, \Gamma_P = r \, p_P \, . \tag{3.2.25}$$

There are other Greeks, and all of them are necessary in one way or another to appropriately manage a portfolio of options. The discussion above, though, is enough to highlight the complex mathematical nature of option pricing.

Let us finally see how Monte Carlo simulation can be applied to the evaluation of Greeks, focusing specifically on delta. Each MCS generates a possible trajectory for the asset price over time, and this process is repeated many times to form a statistical sample. At each point in time, from the purchase of the option to its expiration, the average asset price across all simulations is calculated. To do this, assumptions are typically made about market conditions, such as constant volatility, a fixed risk-free rate, and using models like geometric Brownian motion to describe price movements.

The resulting information can be visualized on a two-dimensional graph with the asset price on the $x$-axis and the option price on the $y$-axis. Since delta represents

the rate of change of the option price with respect to the underlying asset price, we can approximate it by evaluating the price change over small increments in the asset price. Assuming a linear relationship for small changes, we can express the option price as:

$$\widehat{C}(s_1) = \Delta_C(s_1 - s_0) + C(s_0)\,, \tag{3.2.26}$$

where $C(s_0)$ is the historical price of the option at the initial asset price $s_0$, $\Delta_C$ is the delta of the option, and $\widehat{C}(s_1)$ is the estimated option price at the new asset price $s_1$. For every asset price $s_i$, with $i = 1, \ldots, n$, the estimated value of the option is given by:

$$\widehat{C}(s_i) = \Delta_C(s_i - s_{i-1}) + C(s_{i-1})\,, \tag{3.2.27}$$

where $C(s_{i-1})$ is the option price at the previous asset price $s_{i-1}$. Using the *least squares error* (*LSE*) method, the best fit is determined by minimizing the error, $E$, between the observed option prices and the predicted values:

$$\min_{\Delta_C, C(s_0)} E = \min_{\Delta_C, C(s_0)} \sum_{i=1}^{n} E_i\,, \tag{3.2.28}$$

where

$$E_i = \left(\widehat{C}(s_i) - C(s_i)\right)^2. \tag{3.2.29}$$

The slope of the best fit line is the delta $\Delta_C$ we are seeking.

## 3.3 Machine Learning

As you may be aware, artificial intelligence (AI) has taken our technological societies by storm.[5] Countless industries have seen their productivity and services completely revolutionized by the new capabilities of AI models. Some of these sectors include the automotive industry, energy production, and, of course, finance. Many banks today use AI models to identify investment opportunities, detect fraud, spot money laundering activities, assess credit risk, and more. In this section, we will touch upon several of these aspects. To be more precise, we will focus on machine learning, which is a subfield of AI. While AI is a broad field encompassing various domains such as natural language processing, robotics, and computer vision, ML specifically deals with algorithms that enable computers to learn from data and improve over time without further programming. After reviewing the basic principles of ML, we will explore how some of these models are applied to financial problems.

As we just said, the primary goal of machine learning (ML) is to identify patterns within data in order to make predictions or decisions. The data is used to both discover underlying patterns and to test and refine these patterns through training and validation processes. In the financial industry, data—such as stock movements and customer information—are regularly collected, and standard ML algorithms are commonly used to analyze this data and make predictions. Note that in ML, there is an implicit assumption that the collected data follows certain underlying laws or functions, and that the algorithm's task is to uncover them. The parameters of the multivariable functions modeling these underlying patterns are adjusted by the

[5] A few accessible books on the subject include A. Agrawal et al., *Prediction Machines*, N. Bostrom, *Superintelligence*, and S. Gerrish, *How Smart Machines Think*.

algorithm until the best fit to the data is achieved. It is important to note that, however, the goal of ML is not to derive an exact function, but to extract practical and actionable insights from the data. In this section, we will review in some detail several ML algorithms currently used in finance that have been identified as potential candidates for enhancement by quantum computational methods.[6].

If, as stated above, ML aims at discovering patterns in data, it is crucial to begin clarifying what we mean by data. *Data* refers to information associated with physical objects or abstract concepts. The data collected comes in various forms: text, audio, video, images, and more. It is generally categorized into two main types: structured and unstructured.

*Structured data* is highly organized and typically stored in formats such as matrices for numerical data or tables for more general datasets. For instance, numerical data can be represented in matrix format, where rows correspond to *samples* (e.g., individual records), the first $I$ columns represent *features* (also referred to as inputs), and the last $O$ columns correspond to *labels* (outputs). For $N$ samples, the corresponding matrix is:

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_I^{(1)} & y_1^{(1)} & \cdots & y_O^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_I^{(N)} & y_1^{(N)} & \cdots & y_O^{(N)} \end{bmatrix}. \tag{3.3.1}$$

This format is essential for many ML tasks, especially in supervised learning, as we will discuss shortly. Often, we refer to the column vector

$$\mathbf{x}_i = \begin{bmatrix} x_i^{(1)} & \dots & x_i^{(N)} \end{bmatrix}^T, \tag{3.3.2}$$

where $i$ ranges from 1 to $I$, as the *ith feature vector*, and to

$$\mathbf{x}^{(n)} = \begin{bmatrix} x_1^{(n)} & \dots & x_I^{(n)} \end{bmatrix}^T, \tag{3.3.3}$$

as the *nth sample feature vector*. It is straightforward to verify that the submatrix of inputs can be written as:

$$\begin{bmatrix} (\mathbf{x}^1)^T \\ \vdots \\ (\mathbf{x}^N)^T \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_I \end{bmatrix}. \tag{3.3.4}$$

*Unstructured data*, on the other hand, refers to information that lacks a predefined structure, such as raw text, audio, or video files. This type of data is often the most common form of collected data. Before it can be used in ML algorithms, unstructured data must be cleaned, organized, and converted into a structured format. Data scientists, or more specifically, data engineers, are responsible for addressing issues such as duplicates, missing values, and inconsistent formatting. These errors must be corrected or removed to ensure the integrity of the dataset. Given the vast amounts of data involved—often millions or even billions of data points—this

[6] Andrew Ng's CS229 course on machine learning at Stanford is a classic. The videos and lecture notes are available online. A. Burkov's *The Hundred-Page Machine Learning Book* is an excellent complement to Ng's course. The well-known quant Paul Wilmott also offers a concise and instructive introduction to the subject in his book *Machine Learning*.

process requires powerful programming tools such as Python, along with specialized libraries like Pandas, NumPy, or TensorFlow (see below). Techniques like Natural Language Processing (NLP) for text or feature extraction for images play a crucial role in transforming unstructured data into a usable form.

Data science is a vast and varied field. For now, the key takeaway is that the data used in ML algorithms is not the raw, noisy, unprocessed data that is initially collected. Instead, it must be carefully preprocessed to ensure it is suitable for analysis. Poor data selection or inadequate preprocessing can lead to inaccurate descriptions of the data or faulty predictions and decisions based on it.

### 3.3.1 ML Algorithms

#### Supervised Learning

*Supervised Learning* (*SL*) is by far the most widely used ML method. It assumes that labeled data—comprising features and labels—has already been organized into a tabular format. For instance, when there is a single feature and a single label, the data can be visualized as points on a two-dimensional plane. For two samples, this data is represented in the matrix:

$$\begin{bmatrix} x_1^{(1)} & y_1^{(1)} \\ x_2^{(2)} & y_2^{(2)} \end{bmatrix}, \tag{3.3.5}$$

which simply represents the points $(x_1^{(1)}, y_1^{(1)})$ and $(x_2^{(2)}, y_2^{(2)})$ in a more manageable form. This structured approach enables supervised learning algorithms to effectively identify patterns and relationships between features and their corresponding labels.

Given a labeled dataset with feature vectors $\mathbf{x}_1$, ..., $\mathbf{x}_I$ and label vectors $\mathbf{y}_1$, ..., $\mathbf{y}_O$, here is how these algorithms work: the complete dataset is first divided into two groups; the first group, known as the *training data*, is used to train the model,

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_I^{(1)} & y_1^{(1)} & \cdots & y_O^{(1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & \cdots & x_I^{(n)} & y_1^{(n)} & \cdots & y_O^{(n)} \end{bmatrix}, \tag{3.3.6}$$

and the other, the *test data*, is used to evaluate the model's performance after training,

$$\begin{bmatrix} x_1^{(n+1)} & \cdots & x_I^{(n+1)} & y_1^{(n+1)} & \cdots & y_O^{(n+1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & \cdots & x_I^{(N)} & y_1^{(N)} & \cdots & y_O^{(N)} \end{bmatrix}. \tag{3.3.7}$$

During the training phase, the algorithm learns to predict the labels $y_j^{(k)}$ based on the inputs $x_i^{(k)}$ by minimizing a loss function that measures the difference between the predicted labels $(\hat{y}_1^{(k)}, \ldots, \hat{y}_O^{(k)})$ and the actual labels $(y_1^{(k)}, \ldots, y_O^{(k)})$ for every $k = 1, \ldots, n$. The model is iteratively refined by adjusting its parameters to minimize this error, thereby improving accuracy over time. After training, whenever a new set of features $(x_1^{(n+1)}, \ldots, x_I^{(n+1)})$ is introduced, the algorithm uses the learned model to predict the corresponding labels $(\hat{y}_1^{(n+1)}, \ldots, \hat{y}_O^{(n+1)})$. The test phase evaluates the model's ability to generalize to unseen data. This is done by comparing its

predictions to the true labels, without making further adjustments to the already trained model. It is important to note that the training and testing phases serve distinct purposes: the training phase focuses on learning and optimizing the model, while the testing phase assesses its generalization performance on new, unseen data.

What we have just seen is an example of *regression*. The simplest form of regression you may be familiar with is *linear regression*. While it shares similarities with the classical statistical approach (refer to equation (3.2.28)), there are key differences. In brief, the standard statistical approach is more theoretical and focused on understanding the relationship between variables, whereas linear regression in ML is more data-driven and aimed at minimizing prediction error. Furthermore, in ML, regularization techniques such as *Lasso* and *Ridge* regression are commonly applied to linear regression models to prevent overfitting and enhance their ability to generalize to unseen data.

Let us now consider the second major area of supervised learning: *classification*. In classification, unlike regression, the labels are discrete rather than continuous. The goal of a classification algorithm is to assign one of a finite number of possible labels, $(y_1^{(n)}, \dots, y_O^{(n)})$, to a given set of features, $(x_1^{(n)}, \dots, x_I^{(n)})$. For simplicity, let us assume that we are dealing with a single-label dataset. This means that each sample, characterized by its feature set, $(x_1^{(n)}, \dots, x_I^{(n)})$, is associated with exactly one label, $y^{(n)}$. In matrix form,

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_I^{(1)} & y^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(N)} & \cdots & x_I^{(N)} & y^{(N)} \end{bmatrix}, \tag{3.3.8}$$

where $y^{(1)}, \dots, y^{(N)}$ are elements of a discrete set. If there are more than two classes, we refer to this as *multiclass classification*. This means that each feature point $(x_1^{(n)}, \dots, x_I^{(n)})$, where $n = 1, \dots, N$, is associated with one of the $C$ categories, $y_1, y_2, \dots, y_C$, where $C > 2$. In other words, $y^{(1)}, \dots, y^{(N)} \in \{y_1, y_2, \dots, y_C\}$. If there are only two categories, $C = 2$, it is a *binary classification*. In this case, each point $(x_1^{(n)}, \dots, x_I^{(n)})$ is associated with one of the two labels, $y_1$ or $y_2$, which are often expressed as $-$ and $+$, or 0 and 1. In symbols, $y^{(1)}, \dots, y^{(N)} \in \{+, -\}$.

### *k-Nearest Neighbors*

The first classification algorithm we want to introduce is the *k-Nearest Neighbors* ($k$NN) *algorithm*. Due to its simplicity and effectiveness, it is one of the most widely used algorithms in classification tasks. The $k$NN algorithm groups feature data points based on their proximity and uses this grouping to make predictions for new data points. Intuitively, if a point in $(x_1^{(n)}, \dots, x_I^{(n)})$ corresponds to a certain class, such as $-$ or $+$, it is because it is surrounded by points in the same class. However, ambiguity arises when the point lies in a region where the surrounding points belong to multiple classes. The $k$NN algorithm resolves this ambiguity by assigning the label of a point based on the majority class among its $k$-nearest neighbors. Specifically, if there are $N_+$ neighboring points with the label $+$ and $N_-$ points with the label $-$, with $N_+ + N_- = k$, the classification is determined by the sign of $N_+ - N_-$. If $N_+ > N_-$, the new point is classified as $+$. If $N_- > N_+$, it is classified as $-$. This process captures the essence of the $k$NN algorithm. To train and test the $k$NN algorithm for data classification, we follow the same general steps outlined

above for supervised learning.

*Neural Networks*

Let us examine in detail *Neural Networks*, one of the most powerful methods in machine learning. While they can be applied to both supervised and unsupervised learning, we will focus on the supervised case here.

For over a century, biologists have understood that neurons in the human brain communicate with each other to process information. Inspired by this biological mechanism, computer scientists in the mid-20th century proposed the concept of artificial neural networks—now commonly referred to as Neural Networks. These computational models mimic certain aspects of how biological neurons function and are used not only to simulate brain activity, as was originally intended, but also for a wide range of applications in science and industry.

In these models, biological neurons are replaced by interconnected *nodes*, typically organized into *layers*. Each layer of nodes serves a specific purpose: some nodes receive input data (the *input layer*, analogous to sensory neurons), others process this data through intermediate layers (the *hidden layers*), and some nodes produce the final output or prediction (the *output layer*, akin to the response of a biological organism). The network's behavior is determined by a set of parameters, the number of which depends on the number of layers and the nodes within each layer. If the network's prediction does not match the actual outcome, the model learns by adjusting these parameters. This adjustment is guided by a *cost* or *loss function*, which quantifies the discrepancy between the predicted and actual outcomes. To minimize this loss during training, the network iteratively updates its parameters using optimization techniques such as backpropagation and gradient descent. These are the fundamental ideas behind neural networks.

Let us begin with the most elementary model. If there is a single input and a unique output, linear regression proposes a linear equation to fit the data: $\hat{y} = wx + b$, where the parameters $w$ and $b$ are adjusted through minimization of the loss function, ensuring the line is the best fit for the data in the two-dimensional $x$-$y$ plane. When there are multiple inputs, $x_1, \ldots, x_I$, the geometry of the model becomes a hyperplane: $\hat{y} = \mathbf{w}^T\mathbf{x} + b$, where the vector $\mathbf{x} = [x_1 \ \ldots \ x_I]^T$ collects the input features and $\mathbf{w} = [w_1 \ \ldots \ w_I]^T$ represents the vector of *weights*. The parameters $w_1, \ldots, w_I$, which determine the contribution of each input to the next node, and $b$, the *bias* term, are adjusted to minimize the loss function, ensuring the hyperplane best fits the data. If the relationship between the inputs and the output is non-linear (i.e., not a hyperplane), the model can be enhanced with the inclusion of an *activation function* $f$. This introduces non-linearity into the predictions, enabling the model to capture more complex patterns in the data. Thus, instead of

$$\hat{y} = \sum_{i=1}^{I} w_i x_i + b\,, \tag{3.3.9}$$

we use

$$\hat{y} = f\Big(\sum_{i=1}^{I} w_i x_i + b\Big)\,. \tag{3.3.10}$$

This simple model, known as the *perceptron*, consists of only two layers: the input layer and the output layer (with only one node). We now introduce a hidden layer

with $N_1$ nodes. The value associated with the $n_1$th node, where $n_1 = 1, \ldots, N_1$, is given by

$$h_{n_1} = f^1\Big(\sum_{i=1}^{I} w_{n_1,i}\, x_i + b_{n_1}\Big)\,. \tag{3.3.11}$$

Considering the contribution from all the nodes in the hidden layer,

$$\hat{y} = f^o\Big(\sum_{n_1=1}^{N_1} w_{o,n_1}\, h_{n_1} + b\Big)\,. \tag{3.3.12}$$

By combining these two results, we arrive at the prediction,

$$\hat{y} = f^o\Big(\sum_{n_1=1}^{N_1} w_{o,n_1}\, f^1\Big(\sum_{i=1}^{I} w_{n_1,i}\, x_i + b_{n_1}\Big) + b\Big)\,. \tag{3.3.13}$$

If there are two hidden layers, the first with $N_1$ nodes and the second with $N_2$ nodes, a straightforward extension of what we just did gives,

$$\hat{y} = f^o\Big(\sum_{n_2=1}^{N_2} w_{o,n_2}\, f^2\Big(\sum_{n_1=1}^{N_1} w_{n_2,n_1} f^1\Big(\sum_{i=1}^{I} w_{n_1,i}\, x_i + b_{n_1}\Big) + b_{n_2}\Big) + b\Big)\,. \tag{3.3.14}$$

In general, if there are $H$ hidden layers and $O$ possible outputs, denoted by $j = 1, \ldots, O$, the predicted value for each of them is

$$\hat{y}_j = f^o\Big(\sum_{n_H=1}^{N_H} w_{j,n_H}\, f^H\Big(\sum_{n_{H-1}=1}^{N_{H-1}} w_{n_H,n_{H-1}} f^{H-1}\Big(\ldots f^1\Big(\sum_{i=1}^{I} w_{n_1,i}\, x_i + b_{n_1}\Big)\ldots + b_{n_H}\Big) + b_j\Big)\,. \tag{3.3.15}$$

The crucial role of activation functions in neural networks dates back to the late 1980s. By that time, researchers began to understand that, under certain conditions, neural networks could approximate most practical functions. This property is known as the *Universal Approximation Theorem*. The theorem asserts that a neural network with a sufficient number of hidden nodes and appropriate activation functions can approximate any continuous function $y = f(x_1, \ldots, x_I)$ to an arbitrary degree of accuracy. This foundational result underscores the flexibility of neural networks in capturing complex, non-linear relationships between inputs and outputs. The choice of activation functions is critical. Among them, *sigmoid functions*, characterized by their S-shaped curve, are particularly effective. Another widely used activation function is the *Rectified Linear Unit* (*ReLU*), defined as

$$\mathrm{ReLU}(x) = \max(x, 0)\,. \tag{3.3.16}$$

ReLU is computationally efficient, simple to implement, and helps mitigate the vanishing gradient problem that often affects the training of deep neural networks.

To conclude, let us briefly discuss the optimization of the neural network's parameters. As described above, given the input values $x_1, \ldots, x_I$ and fixing the values for all the weights and biases, a general neural network predicts the output values $\hat{y}_j$, where $j = 1, \ldots, O$. In supervised learning, the true values, denoted as $y_j$, are known for the training dataset. This allows us to compute the error made by the

neural network for each data point, $\hat{y}_j - y_j$. The total error across the dataset can be quantified using a loss function, such as the *Mean Squared Error*(*MSE*) formula,

$$\text{MSE} = \frac{1}{O}\sum_{j=1}^{O}(\hat{y}_j - y_j)^2\,. \tag{3.3.17}$$

Minimizing this error is the goal during training, as it ensures the network's predictions $\hat{y}_j$ closely match the true values $y_j$. The key to minimizing this error—and thus improving the predictions of our neural network—lies in the dependence of each $\hat{y}_j$ on the weights and biases. If the neural network has $H$ hidden layers with $N_1, N_2, \ldots, N_H$ nodes in each respective layer, the total number of parameters (weights and biases) can be calculated as,

$$\begin{aligned}&(N_I N_1 + N_1) + (N_1 N_2 + N_2) + \ldots + (N_h N_{h+1} + N_{h+1}) + \ldots + (N_H N_o + N_o)\\ &\quad = N_1(N_I + 1) + N_2(N_1 + 1) + \ldots + N_{h+1}(N_h + 1) + \ldots + N_o(N_H + 1)\\ &\quad = \sum_{h=0}^{H} N_{h+1}(N_h + 1)\,,\end{aligned} \tag{3.3.18}$$

For notational convenience, we introduce $N_0 = N_I$ to represent the number of input features and $N_{H+1} = N_o$ to represent the number of output nodes. For example, if there are no hidden layers, the total number of parameters is,

$$\sum_{h=0}^{H=0} N_{h+1}(N_h + 1) = N_{0+1}(N_0 + 1) = N_o(N_I + 1)\,. \tag{3.3.19}$$

These include the weights connecting the input nodes to the output nodes, as well as the biases for the output nodes. Specifically, for the perceptron, where $N_o = 1$, there are $N_I + 1$ parameters. For a single input feature and a single output label, there are $1 + 1 = 2$ parameters to adjust: the weight $w$ and the bias $b$. We can use

$$\frac{d}{dw}\text{MSE} = \frac{1}{O}\frac{d}{dw}\sum_{j=1}^{O}(\hat{y}_j(w,b) - y_j)^2\,. \tag{3.3.20}$$

Given the initial values of the parameters, we optimize them using methods such as the gradient descent algorithm. The process is similar when there are more parameters. However, in this case, the change in the error function depends on all the parameters, and the partial derivatives with respect to each parameter capture the effect of small variations in each one. *Backpropagation* is the procedure that guides the algorithm in optimizing the parameters iteratively until the best fit is achieved.

*Recurrent Neural Network (RNN)*

A special type of neural network is the *Recurrent Neural Network (RNN)*. The key difference between standard (feedforward) neural networks, as presented above, and RNNs is that the latter incorporate contributions from previous time steps within a sequence of data. Suppose we construct a neural network at the initial time $t_0$ (i.e., there is no previous data available). At this point, the predicted output is

$$\hat{\mathbf{y}}_0 = f^0(W_0\,\mathbf{x}_0 + \mathbf{b}_0)\,, \tag{3.3.21}$$

where we have simplified the transformation of the input $\mathbf{x}_0$ into the output $\hat{\mathbf{y}}_0$ through the action of the weight matrix $W_0$. The vector $\mathbf{b}_0$ is a bias vector. At a later time $t_1$, the input vector is $\mathbf{x}_1$, with the corresponding weight matrix $W_1$ and bias vector $\mathbf{b}_1$. If we ignore the effect of what happened at $t_0$, the expected value at $t_1$ would be

$$\hat{\mathbf{y}}_1 = f^1(W_1\,\mathbf{x}_1 + \mathbf{b}_1)\,. \tag{3.3.22}$$

However, recurrent neural networks aim to include the contribution from previous time steps. In this context, the predicted value $\hat{\mathbf{y}}_0$ becomes the *hidden state* at $t_0$,

$$\mathbf{h}_0 = f^0(W_0\,\mathbf{x}_0 + \mathbf{b}_0)\,, \tag{3.3.23}$$

and the prediction at time $t_1$ is

$$\hat{\mathbf{y}}_1 = f^1\big(V_0\,\mathbf{h}_0 + (W_1\,\mathbf{x}_1 + \mathbf{b}_1)\big)\,, \tag{3.3.24}$$

where $V_0$ is the weight matrix for the contributions coming from time $t_0$. For time $t_2$, we proceed in the same way. We define the hidden state at $t_1$ as

$$\mathbf{h}_1 = f^1\big(V_0\,\mathbf{h}_0 + (W_1\,\mathbf{x}_1 + \mathbf{b}_1)\big)\,, \tag{3.3.25}$$

and

$$\hat{\mathbf{y}}_2 = f^2\big(V_1\,\mathbf{h}_1 + (W_2\,\mathbf{x}_2 + \mathbf{b}_2)\big)\,. \tag{3.3.26}$$

RNNs assume that there is nothing inherently unique about any specific time step. As a result, the weight matrices—both those associated with the input features and the hidden states—and the activation functions remain the same across all time steps. In other words, if there are $n$ time steps,

$$V_0 = V_1 = \ldots = V_{n-2} = V_{n-1}\,, \tag{3.3.27}$$

$$W_0 = W_1 = \ldots = W_{n-1} = W_n\,, \tag{3.3.28}$$

and

$$f^0 = f^1 = \ldots = f^{n-1} = f^n\,. \tag{3.3.29}$$

This, moreover, reduces the number of parameters and makes training computationally more efficient. In conclusion, the predicted output vector at $t_n$ is

$$\hat{\mathbf{y}}_n = f\big(V\,\mathbf{h}_{n-1} + (W\,\mathbf{x}_n + \mathbf{b}_n)\big)\,, \tag{3.3.30}$$

where the hidden state at $t_{n-1}$ is given by

$$\mathbf{h}_{n-1} = f\big(V\,\mathbf{h}_{n-2} + (W\,\mathbf{x}_{n-1} + \mathbf{b}_{n-1})\big)\,. \tag{3.3.31}$$

There is, however, a potential problem with *sharing weights* across all hidden states. If the elements of the weight matrix $V$ are much smaller than 1 at each step, during the backpropagation process, the gradients for parameter adjustment become progressively smaller. As we backpropagate through the sequence backward in time, the product of gradients across time steps diminishes exponentially, leading to the *vanishing gradient problem*. Consequently, the parameters are only slightly adjusted—or not adjusted at all—leading to slow learning and effectively neglecting the contributions from earlier time steps in the sequence of temporal data. In order to solve the vanishing gradient problem, we can introduce the *Long Short-Term*

*Memory* (*LSTM*) cell. Its purpose is to guide the network at every step on how much information from previous time steps should be retained. The LSTM cell prioritizes recent information while gradually forgetting or neglecting contributions from steps far back in time, ensuring that the model focuses on the most relevant context for its predictions.

### Unsupervised Learning

Unlike supervised learning, *Unsupervised Learning* (*UL*) algorithms are provided with datasets that do not include labeled data. Given a set of input data, their goal is to independently identify hidden patterns within the data.

In many situations, the input feature space, $\mathbb{R}^I$, can be reduced to a lower-dimensional space, $\mathbb{R}^{\hat{I}}$, where ideally $\hat{I} \ll I$, without losing significant information that could affect the algorithm's predictions. For example, consider a simple case where the data is three-dimensional, $(x^{(n)}, y^{(n)}, z^{(n)})$, where $n = 1, \ldots, N$ labels the sample. If it turns out that the $z$ feature exhibits minimal variation across all samples compared to the variations in the $x$ and $y$ features, we can safely ignore $z$. The data can then be simplified to $(x^{(n)}, y^{(n)})$. This reduction eliminates the need to process the superfluous $z$ feature, which neither contributes meaningful information nor affects the model's predictive quality. When dealing with high-dimensional data, where the input space is enormous, removing irrelevant or redundant features becomes a crucial preprocessing step. This dimensionality reduction not only accelerates the algorithm but also minimizes computational costs without sacrificing predictive performance.

#### *Principal Component Analysis*

One systematic method to perform such feature reduction is *Principal Component Analysis* (*PCA*), a key technique under the broader category of *Dimensionality Reduction*. PCA precisely quantifies the amount of information discarded during the reduction process. Instead of arbitrarily removing less significant components, PCA ensures that the remaining components retain as much information as possible from the original data. In other words, if we reduce the input space from $\mathbb{R}^I$ to $\mathbb{R}^{\hat{I}}$ using the PCA algorithm, it guarantees that the transformed dataset $(y_1^{(n)}, \ldots, y_{\hat{I}}^{(n)})$ retains most of the significant information from the original data. Here is how PCA works conceptually: Consider two features $i, j \in \{1, \ldots, I\}$. If the samples exhibit a strong linear relationship between $i$ and $j$, this indicates redundancy, as the information provided by one feature can largely be inferred from the other. PCA begins by shifting the data with respect to the mean of the $x_i$-$x_j$ plane and then rescales the axes (this is analogous to studying the physics of a massive system from the perspective of its center of mass), simplifying the analysis while preserving key relationships,

$$\bar{x}_i^{(n)} = \frac{x_i^{(n)} - \mu_i^{(n)}}{\sigma_i^{(n)}}\,, \tag{3.3.32}$$

where

$$\mu_i^{(n)} = \frac{1}{N}\sum_{n=1}^{N} x_i^{(n)}\,, \qquad \sigma_i^{(n)} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(x_i^{(n)} - \mu_i^{(n)}\right)}\,, \tag{3.3.33}$$

Similar for the $j$ feature. The $x_i$-$x_j$ plane can be effectively reduced to a single

dimension by identifying the line that best fits the data, a process guided by the MSE minimization method discussed earlier. This line corresponds to the direction along which the data exhibits the greatest variance. In other words, the line aligns with the principal eigenvector (the eigenvector with the largest eigenvalue), $\boldsymbol{e}_{ij}$, of the covariance matrix $\Sigma_{ij}$ of the data. This covariance matrix is computed after the data has been shifted to its mean and rescaled,

$$\Sigma_{ij} = \frac{1}{N}\sum_{n=1}^{N}\left(\bar{x}_i^{(n)}\bar{x}_i^{(n)} + \bar{x}_j^{(n)}\bar{x}_j^{(n)}\right). \tag{3.3.34}$$

If we define the two-dimensional feature vector $\bar{\mathbf{x}}^{(n)} = [\bar{x}_i^{(n)}\ \bar{x}_j^{(n)}]^T$, its component along the principal eigenvector $\boldsymbol{e}_{ij}$ is given by the dot product

$$y_{ij} = \bar{\mathbf{x}}^{(n)} \cdot \boldsymbol{e}_{ij}\,. \tag{3.3.35}$$

By comparing $y_{ij}$ with respect to $\bar{x}_{ij}$, we can quantify the amount of information lost when substituting $\bar{x}_{ij}$ with $y_{ij}$ and ignoring the transverse component. This comparison provides a measure of the variance retained in the reduced representation versus the variance discarded. This process can be systematically applied to as many pairs of features as possible, iteratively reducing the dataset's dimensionality. The reduction continues until the number of features is brought to a manageable size, balancing computational efficiency with minimal information loss. A common best practice is to reduce the original feature space to a subspace that captures 90-95% of the total variance, ensuring that the reduced dataset retains most of the critical information while discarding noise and less relevant variability.

*k-Means Clustering*

Given a dataset $\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(n)},\ldots,\mathbf{x}^{(N)}\}$, where $N$ is the number of samples and $\mathbf{x}^{(n)}$ is the feature vector of the $n$th record, i.e., $\mathbf{x}^{(n)} = [x_1^{(n)}\ \ldots\ x_i^{(n)}\ \ldots\ x_I^{(n)}]^T$, the *k-Means Clustering* algorithm aims to group the data into $K$ distinct clusters identified by the algorithm. The number of clusters, $K$, is a hyperparameter chosen by the data analyst, and the algorithm automatically groups the data points. The first step is to select $K$ random points in the feature space $\mathbb{R}^I$,

$$\boldsymbol{\mu}_k^0 = [\mu_{k,1}^0\ \ldots\ \mu_{k,i}^0\ \ldots\ \mu_{k,I}^0]^T\,, \tag{3.3.36}$$

where $k = 1,\ldots,K$. Here, $\mu_{k,i}^0$ is the $i$th component of the $k$th vector. These vectors are referred to as the *clustering centroids*, for reasons we will discuss shortly. For each feature vector $\mathbf{x}^{(n)}$ and each centroid $\boldsymbol{\mu}_k^0$, the algorithm computes the Euclidean distance between them,

$$d\big(\mathbf{x}^{(n)},\boldsymbol{\mu}_k^0\big) = \sqrt{\sum_{i=1}^{I}\left(x_i^{(n)} - \mu_{k,i}^0\right)^2}\,. \tag{3.3.37}$$

Every feature vector is then assigned to the cluster whose centroid is the closest, that is, the one for which $d\big(\mathbf{x}^{(n)},\boldsymbol{\mu}_k^0\big)$ is minimal. At this stage, the dataset is partitioned into $K$ clusters, denoted as $C_1^0,\ldots,C_K^0$, each containing $N_1^0,\ldots,N_K^0$

points, respectively. Next, the algorithm updates the centroids $\boldsymbol{\mu}_k^1$ of these clusters by computing

$$\mu_{k,i}^1 = \frac{1}{N_k^0} \sum_{\mathbf{x}^{(n)} \in C_k^0} x_i^{(n)}, \tag{3.3.38}$$

for each component $i = 1, \ldots, I$. This update ensures that each centroid represents the mean position of all points in its respective cluster. The distances $d\big(\mathbf{x}^{(n)}, \boldsymbol{\mu}_k^1\big)$ are then recomputed for every feature vector and the updated centroids. Each data point is reassigned to the cluster with the nearest centroid, forming new clusters $C_1^1, \ldots, C_K^1$. This process of recalculating centroids and reassigning data points is repeated iteratively. The algorithm terminates when subsequent iterations result in minimal or no change in the centroids, indicating convergence.

### 3.3.2 ML in Finance

The applications of ML to finance are many, and our intention here is not to present a comprehensive account of the subject.[7] Instead, let us focus on few applications of each of the algorithms presented above. For regression, we will consider credit scoring and risk assessment. For the classification algorithms, we will consider: credit risk assessment, transaction fraud detection, credit card fraud detection, money laundering detection, market behavior and sentiment analysis for the $k$NN algorithm; credit scoring and risk assessment for the SVM algorithm; fraud detection for NNs; and algorithmic trading and sentiment analysis for RNNs. For unsupervised learning algorithms, we will see the application of PCA in portfolio management, credit risk analysis, and algorithmic trading. Finally, we will examine how the $k$-means algorithm is used in fraud detection and anti-money laundering.

Regression models are used in *credit scoring* to predict the probability of a borrower repaying their loan (output close to 1) or defaulting (output close to 0). The raw dataset typically contains information about borrowers, including demographic details (such as age and gender) and financial data (such as loan amount, credit history, and repayment records). This data is often used to derive additional features, such as the length of credit history or the debt-to-income ratio, which can enhance the model's predictive power. Using this enriched dataset, the model is trained to discover patterns between the input features and the target variable (e.g., the probability of repayment or default). The model's performance is then evaluated on a test dataset to ensure it generalizes well. Once trained and tested, the regression model can predict the probability of repayment for new applicants. This helps financial institutions decide whether to approve a loan or determine credit limits based on the applicant's predicted creditworthiness. We have referred to probability as a measure of the borrower's creditworthiness. Banks, however, may instead use a different metric known as the *credit score*. Credit scores have a defined range with a minimum (indicating a higher risk of default) and a maximum (indicating a lower risk of default). While credit scores and probabilities are expressed differently, they are indirectly related, as both reflect the borrower's credit risk.

Risk assessment has a broader meaning than simply evaluating a customer's credit risk. In finance, *risk assessment* aims to predict both the likelihood and potential cost of adverse events that may impact an companies' financial stability. The specific

[7] See, for example, the book by M. F. Dixon et al., *Machine Learning in Finance*.

dataset and features used depend on the type of risk being assessed, which could include market risk, credit risk, operational risk, or country risk. For instance, when assessing risks to a multinational company, factors such as financial performance, political stability, social trends, and global economic conditions may be considered. Suppose we are interested in evaluating the risk an event poses to the company's market valuation. After collecting all relevant data, input features are defined, and the target variable (e.g., changes in the company's valuation or earnings volatility) is established. The regression model is trained on this data to discover patterns and then tested to evaluate its predictive accuracy. This insight enables financial institutions or companies to make informed decisions to mitigate potential risks. In the context of a large financial institution or a multinational corporation, country risk assessment can be seen as an extension of credit risk analysis, incorporating a broader set of variables, such as political instability, regulatory changes, and currency fluctuations.

That the $k$NN algorithm can be used for credit risk assessment is quite obvious. Suppose we are interested in the binary classification version, namely, predicting whether a potential borrower will repay the loan (1 if they will repay, and 0 if they will default). After collecting and cleaning the data, which includes personal and financial information from previous borrowers along with their repayment history, the algorithm is trained to recognize the customer's profiles associated with successful loan repayment. The data of a new applicant is then compared to the $k$ nearest neighbors in the training dataset and, using majority voting, the algorithm predicts whether the applicant will repay the loan or default. Based on this prediction, the bank can decide whether to approve or reject the loan application. It is easy to see how $k$NN can be used to evaluate the legitimacy of a transaction by comparing it to previous transactions, a process known as *transaction fraud detection*. In this case, the dataset consists of bank transactions, labeled as either valid or fraudulent. When a new transaction is classified as fraudulent, the bank can issue a warning to the account holder or block the transaction. *Credit card fraud detection* and *money laundering detection* using machine learning work in a similar manner, although money laundering detection typically involves more complex feature engineering and domain-specific knowledge.

Market movements are highly complex, and sophisticated techniques are often required to attempt predictions of their future behavior. However, it can provide a pedagogical example to illustrate the $k$NN algorithm. Suppose we wish to predict the movement of a stock—whether it will go up, down, or remain stable—at some future time $t_{n+1}$. To do this, we can collect historical financial data over a certain period. For instance, we might gather the trading volume and the stock's open, high, low, and close prices at times $t_0, t_1, \ldots, t_i, \ldots, t_n$. Additionally, incorporating macroeconomic indicators (such as interest rates, inflation data, or GDP growth) and other relevant market data can enhance the model's predictive capabilities. At every time step $t_i$, for $i = 1, \ldots, n$, we define the input features as the financial and economic data collected up to that point. The target variable can be specified based on the price difference between $t_i$ and $t_{i+1}$, categorizing it as "up," "down," or "stable." The $k$NN algorithm is then trained to learn the relation between the input features at each time $t_i$ and the corresponding movement label for $t_{i+1}$. When applied to new data, the algorithm identifies the $k$ nearest data points in the feature space and uses majority voting to predict the stock's movement at $t_{n+1}$. Another variant of the $k$NN algorithm can be used to forecast future asset prices based

on the historical price movements of similar stocks. However, as noted, this is a naive approach to market analysis. For instance, this method does not account for temporal dependencies or evolving trends over time, which are critical in financial time-series analysis. More advanced techniques, such as RNNs and LSTM networks, are better suited for capturing these temporal dynamics (see below).

PCA can be used for credit risk assessment. In fact, datasets used for credit risk analysis often contain features that are highly correlated, such as income, debt-to-income ratio, credit utilization, and payment history. PCA addresses this issue by reducing the original feature space to a much smaller subspace while retaining most of the variance. This helps risk analysts focus on the most significant factors driving credit risk. For example, reducing the dataset to three dimensions can facilitate the visualization of borrower clusters with similar risk profiles, as well as the identification of outliers that might represent unique risks or anomalies. It is not difficult to understand why PCA is also useful for risk management in general and fraud detection. For example, let us see the case of *portfolio management*. Many factors contribute to the construction of a portfolio, ranging from macroeconomic and social influences to political events and global conditions. Reducing the feature space can help portfolio managers identify the most important risk factors and understand how these factors translate into asset allocation decisions. By simplifying the covariance matrix, PCA reveals the correlation between pairs of assets, helping to identify and remove highly correlated assets, which in turn reduces the portfolio's size without sacrificing diversification. Although there may be some loss of information during dimensionality reduction, the efficiency of the calculations and the ability to focus on the most significant factors make the approximation introduced by PCA worthwhile. The use of PCA for market analysis is somewhat similar. A large amount of data, such as stock prices and economic factors, is reduced by performing PCA on historical data to project it onto a subspace of nearly uncorrelated principal components. By reducing the number of features, the market analyst or algorithmic trading system can better identify and focus on key market drivers, while ignoring irrelevant factors. This reduction helps improve efficiency and accuracy over time (crucial, for instance, for high-frequency trading strategies).

As we saw previously, a key distinction between $k$-means clustering and supervised learning algorithms is that $k$-means, being an unsupervised learning method, does not rely on labeled data. This enables it to discover unexpected patterns or previously unseen suspicious behaviors, making it especially valuable in evolving scenarios where fraudsters use increasingly sophisticated tactics. Applications include fraud detection and anti-money laundering, where grouping similar data points and highlighting outliers is crucial. While $k$-means is not explicitly designed as an anomaly detection algorithm, it can help uncover unusual behaviors by identifying data points that do not belong to any cluster or are far from cluster centroids.

## 3.4 Computational Finance

One natural question that arises at this point is: how do we implement these models? To answer this, let us first briefly review the evolution of quantitative finance over the past hundred years. We will then provide a concise introduction to the computational tools commonly used to solve the financial problems discussed earlier.

### 3.4.1 Historical Background

Despite Bachelier's seminal thesis dating back to 1900, quantitative finance emerged as a formal field of study around the mid-20th century. Prior to this, investors and financial institutions primarily relied on experience and practical judgment to guide their activities. By the 1950s, finance had become increasingly mathematical. Significant advancements were made, such as Markowitz's Mean-Variance Model (1952) for portfolio optimization and the Black-Scholes equation (1973) for option pricing. From the 1950s to the early 1980s, financial mathematicians focused on refining existing models and creating new ones. For example, Itô's calculus, developed in the 1940s, became a cornerstone of financial mathematics. The 1980s marked a turning point with the advent of powerful and accessible computers. Financial organizations shifted their focus from mathematical modeling to solving existing equations more efficiently. On the software side, programming languages like Fortran and C (introduced in 1957 and 1972 by IBM and Bell Labs, respectively) enabled financial professionals to perform complex computations on computers developed by companies such as IBM and HP. At the same time, the growing availability of financial data, facilitated by platforms like Bloomberg Terminal (launched in 1982), spurred a shift toward a more data-driven approach to finance.

On a personal note, I remember that in the 1990s, when I was studying physics at university, the field of Econophysics was gaining popularity. The idea behind this field was that physicists, with their mathematical and computational expertise in solving differential equations and understanding stochastic processes, could tackle some of the most challenging problems in finance.

The 21st century, largely driven by the internet, has been marked by the ever-growing availability of financial data, as well as increasingly powerful computers and advanced software. In recent years, we have witnessed the latest computational revolution in quantitative finance with the advent of artificial intelligence, particularly machine learning. In conclusion, over the past half-century, we have seen a shift from mathematical finance to computational finance—a transition from the development of theoretical models based on rigorous assumptions to a more pragmatic, data-driven approach, where the goal is to uncover patterns in the data.

### 3.4.2 Python's Dominance

In the early days of computational finance, during the 1980s and 1990s, C became the dominant programming language due to its fast execution, minimal resource usage, portability across different hardware and operating systems, and the control and adaptability it offers in software development. However, by the 2010s, Python emerged as the preferred programming language in finance. Unlike C, Python's syntax is more intuitive and concise, facilitating faster development and easier collaboration among teams. With libraries like Pandas, NumPy, and SciPy, Python has become the ideal tool for handling large datasets and performing complex calculations efficiently. Moreover, machine learning libraries such as scikit-learn, TensorFlow and PyTorch have significantly boosted Python's adoption within the finance community. As a result, Python's ability to handle large amounts of data, intuitive syntax, rich ecosystem, and strong community support has made it the dominant language in finance.

In the following pages, we will introduce some basic code examples from Python's

most common libraries, including machine learning libraries. (If you are already well-versed in Python, you may skip the rest of this chapter.)

### NumPy

*NumPy* (Numerical Python) is a library for numerical computing in Python. It supports not only one-dimensional arrays of data (a simple list of values, like a row of numbers) but also multi-dimensional arrays (for example, a two-dimensional array is a table or matrix with rows and columns). In NumPy, these arrays are called *ndarray*. A wide range of mathematical functions can be applied to them. We will explore some examples below. NumPy's advantages include improved computation speed, reduced memory usage, and seamless integration with other powerful libraries like Pandas and TensorFlow, which we will examine below. Today, NumPy is an essential library used by data scientists. In the financial industry, as well as in any sector that works with data, proficiency in Python implies expertise in NumPy.

To use NumPy, it must first be installed (if it has not already) using the following command:

```
pip install numpy
```

Then, it must be imported into the environment where Python code is written and executed (for example, a *Jupyter notebook*). The conventional way is:

```
import numpy as np
```

After importing NumPy, we can—for example—define a general 2D array (or matrix) as follows:

$$\texttt{X} = \texttt{np.array}([[\texttt{a}_{11}\ \texttt{a}_{12}\ \texttt{a}_{1\texttt{m}}], \ldots, [\texttt{a}_{\texttt{n}1}\ \texttt{a}_{\texttt{n}2}\ \texttt{a}_{\texttt{nm}}]])$$

To see the shape of the array we have defined, we use the following function:

```
X.shape
```

which, in our case gives `(n,m)`. Be aware that all the elements in an ndarray must be of the same type. This means that they must all be integers, floats, complex numbers, booleans, etc. To see the type of elements in the ndarray `X` defined above, use the following command:

```
X.type
```

Regarding the mathematical operations we can perform with ndarrays, addition and multiplication work as expected. For example, we can compute `cX+dX`, where `c` and `d` are two numbers, by typing the following line of code in the Jupyter notebook:

```
c*X+d*X
```

Given another ndarray `Y` with the same shape and type of elements as `X`, we can compute the sum `cX+dY` by writing:

```
c*X+d*Y
```

If the shape of the matrix `X` is `(n,p)` and that of the matrix `Y` is `(p,m)`, we can multiply the matrices by using:

```
X.dot(Y)
```

Alternatively, we could have used:

```
np.dot(X,Y)
```

Many more operations can be performed on NumPy's ndarrays, for example, matrix transposition and computing the inverses and determinants of matrices. One of the limitations of NumPy is that it cannot handle non-numerical data. This is certainly a limitation because much of the data collected does not come in numerical form. For example, dates and text data cannot be directly processed using NumPy arrays. In such cases, libraries like Pandas are often used, as they provide better support for handling mixed data types.

### Pandas

*Pandas* is a data manipulation and analysis library in Python. Like NumPy, it is widely used for handling structured data in tables, particularly large datasets. One of the advantages of Pandas is its seamless integration with other Python libraries, such as NumPy and machine learning libraries like Scikit-learn. It also enables data visualizations by integrating with Matplotlib. Pandas is more powerful than Numpy and has become an essential tool in financial data analysis, commonly expected to be known by anyone applying for a data analyst role. In finance, Pandas enables financial analysts to import historical stock market data, clean and preprocess it, compute moving averages, measure volatility, and analyze correlations between different assets.

### SciPy

While Pandas is primarily designed for data manipulation and analysis, *SciPy* is focused on scientific computing. It provides advanced mathematical functions for tasks like optimization, integration, and solving differential equations. While Pandas is great for preparing and analyzing data, SciPy extends these capabilities with specialized tools for more complex mathematical operations. The two libraries can be used together, with Pandas handling data manipulation and SciPy offering advanced computations.

### Matplotlib

The preferred tool for graphing in Python is the library *Matplotlib*. The advantages of Matplotlib include its flexibility in creating a wide range of static, animated, and interactive plots. It offers extensive customization of plot elements, such as labels, titles, and axes, enabling users to create highly specific visualizations. Additionally, it integrates seamlessly with other libraries such as NumPy and Pandas for data analysis. How Matplotlib compares to Excel is a natural question since Excel is known for its user-friendly interface and powerful graphical tools. The

main difference between the two is that Excel is more accessible for beginners but also more limited in terms of customization. Matplotlib, on the other hand, being a Python library, provides much greater flexibility and programmability, allowing developers to create more complex plots and handle large datasets with greater ease.

Before concluding this section, let us mention the most commonly used Python library in machine learning.[8]

### Scikit-learn

*Scikit-learn* is a widely used Python library for machine learning. It provides a complete and well-rounded set of tools for data preprocessing, model training, and evaluation. It supports both supervised and unsupervised learning algorithms. Since it is built on NumPy, SciPy, and Matplotlib, it ensures efficient performance and seamless integration with other data science libraries. Scikit-learn is optimized for speed and handles moderate-sized datasets efficiently. However, it lacks support for deep learning and may not be ideal for very large datasets compared to frameworks like TensorFlow or PyTorch. Despite these limitations, its ease of use, strong community support, and extensive documentation make it a go-to library for traditional machine learning tasks.

[8] An introductory ML book with a strong emphasis on programming with Python is A. C. Müller & S. Guido, *Introduction to Machine Learning with Python*. When it comes to finance, refer to Y. Hilpisch, *Python for Finance*, and A. Nag, *Stochastic Finance with Python*.

# Chapter 4

# Quantum-Enhanced Solutions

Like any other business, the goal of a financial institution is to provide the best possible service to its customers while maintaining profitability. In a competitive free-market environment, achieving this is a significant challenge, due to factors such as intense competition, evolving local and global regulations, and complex ethical considerations. Meeting customer expectations under these conditions is no easy task. This is where quantum computing could make a substantial difference in the future.

The rationale for adopting quantum computing is clear: modern industries rely heavily on digital technologies, and since quantum computers are expected to surpass classical systems in both efficiency and security, these industries—and modern society as a whole—will inevitably be impacted by this emerging technology. However, this reasoning both overestimates and oversimplifies the true potential of quantum technologies. While it is certain that quantum computers will impact some industries sooner than others, some sectors may experience little to no change at all. Moreover, the impact of quantum technologies is believed to extend beyond mere computational efficiency. Only future research will reveal the full extent of their influence.

What seems undeniable is that quantum computers will profoundly impact every industry that relies on machine learning (ML). This is largely because two fundamental mathematical pillars of ML—linear algebra and probability theory—are also central to quantum mechanics. As a result, industries that use ML in their production or operations will likely be disrupted by the current generation of NISQ (Noisy Intermediate-Scale Quantum) computers, as well as by more advanced versions on the horizon. Finance, which heavily depends on ML for portfolio optimization, market prediction, and other financial challenges, is no exception. Some experts predict that finance will be the first industry to be revolutionized by quantum computing.[1]

Before exploring how quantum computing can be applied in finance, it is important to recall that the current NISQ era is characterized by noisy quantum devices and a relatively small number of coherent qubits. In fact, there is broad consensus among experts that fully reliable, fault-tolerant quantum computers will only become available in the long term. Because of this, researchers have developed a new class of

[1]Some important papers on the subject are: A. Bouland et al., "Prospects and Challenges of Quantum Finance" (2020), D. J. Egger et al., "Quantum Computing for Finance" (2020), D. A. Herman et al., "A Survey of Quantum Computing for Finance" (2022), D. A. Herman et al., "Quantum Computing for Finance" (2023) and R. Orús et al., "Quantum Computing for Finance" (2018).

algorithms known as *hybrid quantum-classical algorithms*. The core idea behind these models is to delegate the most computationally complex aspects of a problem to a quantum computer while leveraging classical machines—whose efficiency is well established—for the remaining tasks. From a practical standpoint, the advantage of these hybrid approaches is that the quantum subroutine requires only a small number of coherent qubits and a shallow circuit, a technological threshold expected to be achievable in the near future.

## 4.1 Quantum Portfolio Optimization

The mathematical foundations of modern portfolio theory were presented in a previous section. Here, we aim to explain how two quantum algorithms, the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA), could improve the optimization process. In fact, classical algorithms often struggle with the scalability of large portfolios, while quantum algorithms may be able to process these massive datasets more efficiently, potentially unlocking new levels of optimization that are difficult for classical algorithms to achieve. As mentioned earlier, there is currently no formal proof that these quantum algorithms will provide a speedup compared to the faster classical algorithms available today. However, it is expected that future developments will shed light on how and in which areas these quantum algorithms may indeed be beneficial.

### 4.1.1 The VQE Algorithm

The *Variational Quantum Eigensolver* (*VQE*) leverages the variational principle of quantum mechanics to approximate solutions to the time-independent Schrödinger equation for complex systems. This equation can describe real quantum systems, such as complex molecules or the electronic configurations of new materials. Alternatively, as we will see below, it can represent the quantized version of a classical system. What makes these algorithms particularly interesting is that they are designed to run on hybrid classical-quantum computers. The quantum subroutine prepares quantum states and computes Hamiltonian expectation values, while a classical device performs the optimization process. In summary, this is how it operates.[2]

Let us say we want to find the minimum eigenvalue $E_0$ of a known Hamiltonian, $\hat{H}$. Assume that, based on traditional theoretical methods, we have determined that the minimum eigenstate is close to the state $|\widetilde{Q}\rangle$. In the present context, this state is known as the *ansatz state*. As single qubits can be parameterized by two angles in the Bloch sphere, we can assume that any quantum state can be expressed in terms of a set of parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)$. Thus, in theory, the ansatz state can be written as $|\widetilde{Q}\rangle = |Q(\widetilde{\boldsymbol{\theta}})\rangle$. We further assume that the ansatz state can be prepared by applying a *parameterized circuit* $U(\widetilde{\boldsymbol{\theta}})$ to an initial $n$-qubit state $|0\rangle^{\otimes n} = |\mathbf{0}\rangle$,

$$|\widetilde{Q}\rangle = |Q(\widetilde{\boldsymbol{\theta}})\rangle = U(\widetilde{\boldsymbol{\theta}})\,|\mathbf{0}\rangle\ . \tag{4.1.1}$$

$$|\mathbf{0}\rangle \mapsto U(\widetilde{\boldsymbol{\theta}})\,|\mathbf{0}\rangle = |Q(\widetilde{\boldsymbol{\theta}})\rangle\,. \tag{4.1.2}$$

[2]For additional explanations, refer to QC2, Subsection 4.2.

The parameterized circuit $U(\widetilde{\boldsymbol{\theta}})$ is constructed by incorporating the parameters into an appropriate combination of Pauli gates, single-qubit gate rotations, CNOT gates, and so on. The expectation value of the Hamiltonian in this state is computed as follows. First, recall that any Hamiltonian operator can be expressed as a linear combination of Pauli operators,[3]

$$\hat{H} = \sum_{A_1,\ldots,A_n} h_{A_1\ldots A_n}\, \sigma_{A_1} \otimes \ldots \otimes \sigma_{A_n}\,, \tag{4.1.3}$$

where the coefficients $h_{A_1\ldots A_n}$ are real numbers, and the $\sigma_A$'s are Pauli gates $X$, $Y$, $Z$ (for $A = X$, $Y$, $Z$) or the identity operator (for $A = I$). That is, we measure

$$E(\widetilde{\boldsymbol{\theta}}) = \sum_{A_1,\ldots,A_n} h_{A_1\ldots A_n} \langle \mathbf{0}|\, U^\dagger(\widetilde{\boldsymbol{\theta}})\, \sigma_{A_1} \otimes \ldots \otimes \sigma_{A_n} U(\widetilde{\boldsymbol{\theta}})\, |\mathbf{0}\rangle\,. \tag{4.1.4}$$

The VQE tells us that, from a computational point of view, it is more convenient to estimate each of the terms with a quantum device,

$$\langle \mathbf{0}|\, U^\dagger(\widetilde{\boldsymbol{\theta}})\, \sigma_{A_1} \otimes \ldots \otimes \sigma_{A_n} U(\widetilde{\boldsymbol{\theta}})\, |\mathbf{0}\rangle\,, \tag{4.1.5}$$

and let the sum be computed by a classical computer. This process is then repeated for other parameters in the neighborhood of $\widetilde{\boldsymbol{\theta}}$,

$$\langle \mathbf{0}|\, U^\dagger(\Delta\widetilde{\boldsymbol{\theta}})\, \sigma_{A_1} \otimes \ldots \otimes \sigma_{A_n} U(\Delta\widetilde{\boldsymbol{\theta}})\, |\mathbf{0}\rangle\,. \tag{4.1.6}$$

All these results (obtained from the quantum device) are then sent to a classical optimizer. In summary,

$$\min_{\boldsymbol{\theta}} \sum_{A_1,\ldots,A_n} h_{A_1\ldots A_n} \langle \mathbf{0}|, U^\dagger(\boldsymbol{\theta}), \sigma_{A_1} \otimes \ldots \otimes \sigma_{A_n} U(\boldsymbol{\theta}), |\mathbf{0}\rangle = E_{VQE} \gtrsim E_0\,. \tag{4.1.7}$$

Let us illustrate how the VQE could be applied to the optimization of portfolios. As we discussed above (see equation (3.1.23)), a binary portfolio optimization problem aims at

$$\min\, \alpha \sum_{s,s'=1}^{S} b_s \Sigma_{ss'} b_{s'} - \sum_{s=1}^{S} \mu_s b_s\,, \tag{4.1.8}$$

where $b_s$ is a binary variable, $b_s \in \{0, 1\}$, and $\Sigma_{ss'}$ is a symmetric matrix, $\Sigma_{ss'} = \Sigma_{s's}$. For completeness, let us write the objective function,

$$f(b_1, \ldots\, b_s\, \ldots, b_S) = \alpha \sum_{s,s'=1}^{S} b_s \Sigma_{ss'} b_{s'} - \sum_{s=1}^{S} \mu_s b_s\,. \tag{4.1.9}$$

The variational quantum method establishes that we can minimize this function by constructing a *cost Hamiltonian*, $\hat{H}_C$, and finding its minimum expectation value. Minimizing the objective function is equivalent to minimizing the expectation value of the cost Hamiltonian:

$$\min f(b_1, \ldots\, b_s\, \ldots, b_S) = \min \langle Q|\hat{H}_C|Q\rangle\,, \tag{4.1.10}$$

[3] See QC1, equation (4.71).

where $|Q\rangle \in \mathcal{H}^{2^S}$. If we choose the computational basis $\{|b_1 \cdots b_s \cdots b_S\rangle\}$ for $\mathcal{H}^{2^S}$, where the entries of each vector $|b_1 \cdots b_s \cdots b_S\rangle$ are associated to their corresponding stocks ($b_s = 1$ if the stock $s$ is in the portfolio and $b_s = 0$ if it is not), the vector $|Q\rangle$ can be expressed as a linear superposition of these basis vectors,

$$|Q\rangle = \sum_{s=1}^{S} \alpha_{\dots b_s \dots} \, |\cdots b_s \cdots\rangle \, . \tag{4.1.11}$$

The construction of the Hamiltonian operator in terms of Pauli gates is relatively simple. For this, recall that,

$$Z\,|0\rangle = |0\rangle = (1 - 2\cdot 0)\,|0\rangle \, , \tag{4.1.12}$$

$$Z\,|1\rangle = -\,|1\rangle = (1 - 2\cdot 1)\,|1\rangle \, . \tag{4.1.13}$$

In more compact notation,

$$Z\,|b\rangle = (1 - 2b)\,|b\rangle \, , \tag{4.1.14}$$

or,

$$b\,|b\rangle = \frac{1}{2}\,(I - Z)\,|b\rangle \, . \tag{4.1.15}$$

Hence, for every basis vector we have that,

$$b_s\,|\cdots b_s \cdots\rangle = \frac{1}{2}\,(I - Z_s)\,|\cdots b_s \cdots\rangle \, , \tag{4.1.16}$$

where $Z_s = I \otimes \ldots \otimes Z_s \otimes \ldots \otimes I$. The operator associated with the linear term in (4.1.9) is then:

$$-\sum_{s=1}^{S} \mu_s b_s \longmapsto -\sum_{s=1}^{S} \mu_s \, \frac{1}{2}(I - Z_s) \, . \tag{4.1.17}$$

The operator associated with the quadratic term in (4.1.9) is obtained in a similar fashion,

$$\begin{aligned} \alpha \sum_{s,s'=1}^{S} b_s \Sigma_{ss'} b_{s'} \mapsto \alpha \sum_{s,s'=1}^{S} \frac{1}{2}\,(I - Z_s)\,\Sigma_{ss'}\,\frac{1}{2}\,(I - Z_{s'}) \\ = \alpha \sum_{s,s'=1}^{S} \frac{\Sigma_{ss'}}{4}\,(I - Z_{s'} - Z_s + Z_s Z_{s'}) \, . \end{aligned} \tag{4.1.18}$$

The cost Hamiltonian is thus,

$$\hat{H}_C = \alpha \sum_{s,s'=1}^{S} \frac{\Sigma_{ss'}}{4}\,(I - Z_{s'} - Z_s + Z_s Z_{s'}) - \sum_{s=1}^{S} \mu_s \, \frac{1}{2}(I - Z_s) \, . \tag{4.1.19}$$

Suppose the case of only two stocks, $s$ and $s'$. The corresponding cost Hamiltonian is,

$$\begin{aligned} \hat{H}_{C_{ss'}} &= \alpha \frac{\Sigma_{ss'}}{2}\,(I - Z_{s'} - Z_s + Z_s Z_{s'}) - \mu_s \, \frac{1}{2}(I - Z_s) - \mu_{s'} \, \frac{1}{2}(I - Z_{s'}) \\ &= \alpha \frac{\Sigma_{ss'}}{2} Z_s Z_{s'} + \Big(\frac{\mu_s}{2} - \alpha \frac{\Sigma_{ss'}}{2}\Big) Z_s + \Big(\frac{\mu_{s'}}{2} - \alpha \frac{\Sigma_{ss'}}{2}\Big) Z_{s'} \\ &\quad + \Big(\alpha \frac{\Sigma_{ss'}}{2} - \frac{\mu_s}{2} - \frac{\mu_{s'}}{2}\Big) I \, . \end{aligned} \tag{4.1.20}$$

Since we can always shift the minimum of the Hamiltonian expectation value, we simply consider

$$\hat{H}_{C_{ss'}} = \alpha \frac{\Sigma_{ss'}}{2} Z_s Z_{s'} + \frac{1}{2}\Big(\mu_s - \alpha\Sigma_{ss'}\Big) Z_s + \frac{1}{2}\Big(\mu_{s'} - \alpha\Sigma_{ss'}\Big) Z_{s'} \,. \tag{4.1.21}$$

### 4.1.2 The QAO Algorithm

The *Quantum Approximate Optimization Algorithm* (*QAOA*) is a variational algorithm designed to solve classical combinatorial optimization problems, such as portfolio optimization.[4] We start by rewriting the cost Hamiltonian (4.1.19) as follows:

$$\hat{H}_C = \alpha \sum_{s,s'=1}^{S} \frac{\Sigma_{ss'}}{4} Z_s Z_{s'} + \sum_{s=1}^{S} \frac{1}{2}\Big(\mu_s - \alpha \sum_{s'=1}^{S} \Sigma_{ss'}\Big) Z_s \,, \tag{4.1.22}$$

where—for the same reason discussed above—we have omitted the term proportional to the identity operator. To make this expression more manageable, we define

$$\hat{H}_{ZZ} = \alpha \sum_{s,s'=1}^{S} \frac{\Sigma_{ss'}}{4} Z_s Z_{s'} \,, \qquad \hat{H}_Z = \sum_{s=1}^{S} \frac{1}{2}\Big(\mu_s - \alpha \sum_{s'=1}^{S} \Sigma_{ss'}\Big) Z_s \,. \tag{4.1.23}$$

We thus have that

$$\hat{H}_C = \hat{H}_{ZZ} + \hat{H}_Z \,. \tag{4.1.24}$$

The evolution operator corresponding to this Hamiltonian is given by

$$U_C(\gamma) = e^{-i\gamma \hat{H}_C} = e^{-i\gamma \hat{H}_{ZZ}} e^{-i\gamma \hat{H}_Z} \,, \tag{4.1.25}$$

where $\gamma$ is a positive parameter. Note that we have applied the Campbell-Hausdorff formula and used the fact that Pauli-$Z$ operators commute with each other. More explicitly,

$$U_C(\gamma) = \exp\Big[ - i\gamma\, \alpha \sum_{s,s'=1}^{S} \frac{\Sigma_{ss'}}{4} Z_s Z_{s'}\Big] \exp\Big[ i\gamma \sum_{s=1}^{S} \frac{1}{2}\Big(\alpha \sum_{s'=1}^{S} \Sigma_{ss'} - \mu_s\Big) Z_s\Big] \,. \tag{4.1.26}$$

Let us pause here to understand the quantum circuit corresponding to this unitary. First, recall that the rotation of a single qubit around the $a$-axis, where $a = x, y, z$, is given in equation (2.1.13). The circuit corresponding to the exponential containing $\hat{H}_Z$ is obtained in the following manner. We begin by rewriting the exponential as follows:

$$\begin{aligned} e^{-i\,\gamma \hat{H}_Z} &= \exp\Big[ \sum_{s=1}^{S} i\,\frac{\gamma}{2}\Big(\alpha \sum_{s'=1}^{S} \Sigma_{ss'} - \mu_s\Big) Z_s\Big] \\ &= \bigotimes_{s=1}^{S} \exp\Big[ i\,\frac{\gamma}{2}\Big(\alpha \sum_{s'=1}^{S} \Sigma_{ss'} - \mu_s\Big) Z_s\Big] \,. \end{aligned} \tag{4.1.27}$$

To simplify the notation, we define

$$a_s = -\frac{1}{2}\Big(\alpha \sum_{s'=1}^{S} \Sigma_{ss'} - \mu_s\Big) \,, \tag{4.1.28}$$

[4] For more details, refer to section 5 of QC3.

so that

$$e^{-i\gamma\hat{H}_Z} = \bigotimes_{s=1}^{S} e^{-i\gamma a_s Z_s} \,. \tag{4.1.29}$$

This unitary is equivalent to[5]

$$e^{-i\gamma\hat{H}_Z} = \bigotimes_{s=1}^{S} e^{-i\gamma a_s Z_s} = \bigotimes_{s=1}^{S} R_z(2\gamma a_s) \,. \tag{4.1.30}$$

This result tells us that the exponential involving $\hat{H}_Z$ is equivalent to rotating each qubit $s = 1, 2, \ldots, S$ by an angle of $2\gamma a_s$ about the $z$-axis. The circuit corresponding to the exponential containing $\hat{H}_{ZZ}$ is a bit more complicated to figure out. Since, as can easily be shown,

$$e^{-i\delta Z_s Z_{s'}} |b_s\rangle |b_{s'}\rangle = \text{CNOT}\,(I \otimes R_z(2\delta))\,\text{CNOT}\;|b_s\rangle |b_{s'}\rangle \;, \tag{4.1.31}$$

we conclude that

$$\begin{aligned} e^{-i\gamma\hat{H}_{ZZ}} &= \exp\Big[\sum_{s,s'=1}^{S} -i\gamma a_{ss'}\, Z_s Z_{s'}\Big] = \bigotimes_{s,s'=1}^{S} e^{-i\gamma a_{ss'}\, Z_s Z_{s'}} \\ &= \bigotimes_{\substack{s=1 \\ s'<s}}^{S} \text{CNOT}_{ss'}\,(I \otimes R_z(2\gamma a_{ss'}))\,\text{CNOT}_{ss'} \,, \end{aligned} \tag{4.1.32}$$

where, to simplify the notation, we have introduced $a_{ss'} = \alpha\Sigma_{ss'}/4$. The evolution operator in equation (4.1.25) is thus realized by the circuit in (4.1.32), followed by the circuit in (4.1.30).

Now that we know how to realize the evolution operator corresponding to the cost Hamiltonian, we are interested in the qubit that enters the circuit, how to prepare it, and the qubit that exits it. It is common to choose the input qubit as $|Q\rangle_{in} = |+\rangle^{\otimes S}$. This is done using Hadamard gates. Recall that a Hadamard gate acts on a computational basis vector as follows:

$$H\,|b\rangle = \frac{1}{\sqrt{2}} \sum_{b'} (-1)^{bb'} |b'\rangle \;. \tag{4.1.33}$$

More generally,

$$H^{\otimes S}\,|b_1 \ldots b_S\rangle = \frac{1}{\sqrt{2^S}} \sum_{b'_s} (-1)^{b_1 b'_1 + \ldots + b_S b'_S} |b'_1 \ldots b'_S\rangle \;. \tag{4.1.34}$$

The initial qubit $|Q\rangle_{in} = |+\rangle^{\otimes S}$ is prepared by applying a Hadamard gate to each individual qubit $|b_s\rangle = |0\rangle$:

$$|Q\rangle_{in} = |+\rangle^{\otimes S} = H^{\otimes S}\,|0 \ldots 0\rangle = \frac{1}{\sqrt{2^S}} \sum_{b'_s} |b'_1 \ldots b'_S\rangle \;. \tag{4.1.35}$$

[5] See equation (4.72) of QC1.

The output state is then given by

$$|0\dots0\rangle \xmapsto{H^{\otimes S}} |Q\rangle_{in} \xmapsto{U_C(\gamma)} U_C(\gamma)\,\frac{1}{\sqrt{2^S}}\sum_{b'_s}|b'_1\dots b'_S\rangle\ . \tag{4.1.36}$$

Regardless of the input qubit state, the QAOA stipulates that we follow the unitary $U_C(\gamma)$ with a second unitary, associated with the so-called *mixer Hamiltonian*,

$$\hat{H}_M = \sum_{s=1}^{S} X_s\,. \tag{4.1.37}$$

The corresponding parameterized unitary is given by

$$U_M(\beta) = e^{-i\beta\hat{H}_M} = e^{-i\beta\sum_{s=1}^{S}X_s} = \bigotimes_{s=1}^{S} e^{-i\beta X_s}\,. \tag{4.1.38}$$

Note that, since $U_M(\beta)$ does not commute with $U_C(\gamma)$, the strict order is as follows: first, $U_C(\gamma)$ acts on the input qubit, and then $U_M(\beta)$. It can easily be shown that

$$U_M(\beta) = \bigotimes_{s=1}^{S} R_x(2\beta)\,. \tag{4.1.39}$$

The qubit state that exits the quantum circuit, formed by the unitaries $U_C(\gamma)$ followed by $U_M(\beta)$, is now parameterized by the angles $\gamma$ and $\beta$,

$$|Q\rangle_{in} \xmapsto{U_M(\beta)U_C(\gamma)} |Q(\gamma,\beta)\rangle = U_M(\beta)\,U_C(\gamma)\,|Q\rangle_{in}\ . \tag{4.1.40}$$

This is the *ansatz state* that we will use to measure the expectation value of the cost Hamiltonian in equation (4.1.22),

$$\langle Q(\gamma,\beta)|\,\hat{H}_C\,|Q(\gamma,\beta)\rangle\ . \tag{4.1.41}$$

Since the expectation value of the cost Hamiltonian is a real-valued function of both parameters $\gamma$ and $\beta$, we can simply write

$$F(\gamma,\beta) = \langle Q(\gamma,\beta)|\,\hat{H}_C\,|Q(\gamma,\beta)\rangle\ . \tag{4.1.42}$$

We could then send the measurements of the expectation value of the cost Hamiltonian $F(\gamma,\beta)$ to a classical optimizer to propose better values for $\gamma$ and $\beta$, repeating this process as many times as necessary until we reach a good approximation $(\gamma_*,\beta_*)$. However, the QAOA suggests that, rather than optimizing a single pair of parameters $(\gamma,\beta)$, a better approximation can be found by creating a sequence of unitaries $U_C(\gamma)$ and $U_M(\beta)$, each with its own parameter. In other words, the QAOA prescribes that we allow the initial state $|Q\rangle_{in}$ to enter the following parameterized sequence of gates:

$$\begin{aligned}|Q\rangle_{in} &\longmapsto U_M(\beta_p)\,U_C(\gamma_p)\dots U_M(\beta_k)\,U_C(\gamma_k)\dots U_M(\beta_1)\,U_C(\gamma_1)\,|Q\rangle_{in}\\ &= |Q(\gamma_1,\dots,\gamma_k,\dots,\gamma_p,\beta_1,\dots,\beta_k,\dots,\beta_p)\rangle\ .\end{aligned} \tag{4.1.43}$$

Each pair of unitaries $U_M(\beta_k)U_C(\gamma_k)$, for $k = 1, 2, \ldots, p$, is called a *layer*. Specifically, $U_M(\beta_k)U_C(\gamma_k)$ is the $k$th layer, with $U_C(\gamma_k)$ referred to as the $k$th *cost layer* and $U_M(\beta_k)$ as the $k$th *mixer layer*. It has been shown that the larger the number of layers, the better the approximation of the optimization problem. Needless to say, though, the number of layers must be kept within a reasonable limit to ensure that the calculation is not adversely affected by excessive noise.

In total, there are $2p$ parameters to be optimized variationally: $p$ angles $\gamma_k$ and $p$ angles $\beta_k$. We can collect all these variational parameters in a more compact notation: $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$. The *ansatz state* is thus

$$|Q_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{k=1}^{p} U_M(\beta_k)\, U_C(\gamma_k)\, |Q\rangle_{in} \ . \tag{4.1.44}$$

The function in $2p$ variables that the classical computer must optimize is

$$F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle Q_p(\boldsymbol{\gamma}, \boldsymbol{\beta})|\, \hat{H}_C\, |Q_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle \ . \tag{4.1.45}$$

The optimal portfolio is the solution to the problem:

$$\min_{\boldsymbol{\gamma},\boldsymbol{\beta}} \ F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \min_{\boldsymbol{\gamma},\boldsymbol{\beta}} \ \langle Q_p(\boldsymbol{\gamma}, \boldsymbol{\beta})|\, \hat{H}_C\, |Q_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle \ . \tag{4.1.46}$$

## 4.2 Quantum Machine Learning

We have already reviewed the main ideas and mathematical foundations of classical machine learning. In this section, we explore how quantum computing can enhance these classical approaches. We begin by introducing the basics of quantum machine learning, including several key algorithms, and then examine how they can be applied in financial contexts.

### 4.2.1 QML Algorithms

If we want to analyze classical data using a quantum computer, the first step is to encode that information into a quantum system that the quantum computer can process—in other words, into qubits. We begin with a brief review of several methods developed to achieve this.[6]

#### Data Encoding

Suppose we have a set of samples, $n = 1, \ldots, N$, characterized by two features, $x_1$ and $x_2$. For simplicity, assume that each feature can take one of the following four values: 0, 1, 2, or 3. Using simple binary notation, we can represent these values as 00, 01, 10, and 11, respectively. The feature vector of the $n$-th sample can be expressed in binary notation as follows:

$$\begin{aligned} \mathbf{x}^{(n)} = \begin{bmatrix} x_1^{(n)} & x_2^{(n)} \end{bmatrix}^T \longleftrightarrow \mathbf{x}_b^{(n)} &= \begin{bmatrix} x_{1b}^{(n)} & x_{2b}^{(n)} \end{bmatrix}^T \\ &= \begin{bmatrix} (b_1)_1^{(n)} (b_1)_2^{(n)} & (b_2)_1^{(n)} (b_2)_2^{(n)} \end{bmatrix}^T \\ &= \begin{bmatrix} b_{11}^{(n)} b_{12}^{(n)} & b_{21}^{(n)} b_{22}^{(n)} \end{bmatrix}^T . \end{aligned} \tag{4.2.1}$$

[6] For an introduction, see D. Pastorello, *Concise Guide to Quantum Machine Learning*, and M. Schuld & F. Petruccione, *Machine Learning with Quantum Computers*.

*Basis encoding* associates the binary feature vector $\mathbf{x}_b^{(n)}$ with a computational basis vector of a $2^4$-dimensional qubit space:

$$\mathbf{x}_b^{(n)} \longmapsto |\mathbf{x}_b^{(n)}\rangle = |b_{11}^{(n)} b_{12}^{(n)} b_{21}^{(n)} b_{22}^{(n)}\rangle \ . \tag{4.2.2}$$

For any other sample, say $n'$, we have a similar expression,

$$\mathbf{x}_b^{(n')} \longmapsto |\mathbf{x}_b^{(n')}\rangle = |b_{11}^{(n')} b_{12}^{(n')} b_{21}^{(n')} b_{22}^{(n')}\rangle \ . \tag{4.2.3}$$

If the samples $n$ and $n'$ are characterized by the two features $x_1$ and $x_2$, the entire dataset, $X$, is associated with the following state vector:

$$X = \begin{bmatrix} x_1^{(n)} & x_2^{(n)} \\ x_1^{(n')} & x_2^{(n')} \end{bmatrix} \longmapsto |X_b\rangle = \frac{1}{\sqrt{2}} |\mathbf{x}_b^{(n)}\rangle + \frac{1}{\sqrt{2}} |\mathbf{x}_b^{(n')}\rangle \ . \tag{4.2.4}$$

The generalization to more complex cases is straightforward. For simplicity, suppose that $x$ is a positive integer number expressed in the usual decimal system. The corresponding bit string is given by[7]

$$x = 2^{n-1} b_1 + 2^{n-2} b_2 + \ldots + 2^0 b_n = \sum_{j=1}^{n} 2^{n-j} b_j \longleftrightarrow x_b = b_1 b_2 \ldots b_n \, , \tag{4.2.5}$$

where $b_j \in \{0, 1\}$, and $n$ depends on the value of $x$. (Note that the subscript $n$ here should not be confused with the superscript $(n)$ in $x^{(n)}$, which denotes the $n$th sample.) If there are two features, say $i$ and $i'$, the feature vector for the $n$th sample is given by

$$\mathbf{x}^{(n)} = \begin{bmatrix} x_i^{(n)} & x_{i'}^{(n)} \end{bmatrix}^T \longleftrightarrow \mathbf{x}_b^{(n)} = \begin{bmatrix} b_{i1}^{(n)} \ldots b_{in}^{(n)} & b_{i'1}^{(n)} \ldots b_{i'n'}^{(n)} \end{bmatrix}^T , \tag{4.2.6}$$

and the computational basis vector associated with it is

$$\mathbf{x}_b^{(n)} \longmapsto |\mathbf{x}_b^{(n)}\rangle = |b_{i1}^{(n)} \ldots b_{in}^{(n)} b_{i'1}^{(n)} \ldots b_{i'n'}^{(n)}\rangle \ . \tag{4.2.7}$$

In general, for $I$ features:

$$\mathbf{x}^{(n)} = \begin{bmatrix} x_1^{(n)} \ldots x_I^{(n)} \end{bmatrix}^T \longmapsto |b_{11}^{(n)} \ldots b_{1n_1}^{(n)} \ldots b_{I1}^{(n)} \ldots b_{In_I}^{(n)}\rangle \ . \tag{4.2.8}$$

The ket associated with the entire classical dataset $X = \begin{bmatrix} x_i^{(n)} \end{bmatrix}$ is then given by

$$X \longmapsto |X\rangle = \frac{1}{\sqrt{N}} \sum_{n=1}^{N} |\mathbf{x}^{(n)}\rangle \ . \tag{4.2.9}$$

For decimal numbers, the procedure is similar.

Another common method of mapping classical data into quantum states is *amplitude encoding*. We start with the feature vector of the $n$th sample, $\mathbf{x}^{(n)} = \begin{bmatrix} x_1^{(n)} \ldots x_i^{(n)} \ldots x_I^{(N)} \end{bmatrix}^T$, where $n = 1, \ldots, N$, and normalize it,

$$\mathbf{x}^{(n)} \longmapsto \frac{\mathbf{x}^{(n)}}{\|\mathbf{x}^{(n)}\|} = \bar{\mathbf{x}}^{(n)} = \begin{bmatrix} \bar{x}_1^{(n)} \ldots \bar{x}_i^{(n)} \ldots \bar{x}_I^{(n)} \end{bmatrix}^T , \tag{4.2.10}$$

[7] See equation (2.1) of QC1.

where,

$$\|\mathbf{x}^{(n)}\|^2 = \sum_{i=1}^{I} \left(x_i^{(n)}\right)^2, \qquad \|\bar{\mathbf{x}}^{(n)}\|^2 = \sum_{i=1}^{I} \left(\bar{x}_i^{(n)}\right)^2 = 1\,. \tag{4.2.11}$$

We then associate the normalized feature vector $\bar{\mathbf{x}}^{(n)}$ with a quantum state as follows:

$$\bar{\mathbf{x}}^{(n)} \longmapsto |\mathbf{x}^{(n)}\rangle = \sum_{i=1}^{I} \bar{x}_i^{(n)} \, |i\rangle_n \ . \tag{4.2.12}$$

Note that for each feature value $\bar{x}_i^{(n)}$, there is an associated basis vector $|i\rangle_n$. It follows that the dimension of the qubit space must satisfy $2^Q \geq I$, where $Q$ is the number of qubits in $|i\rangle_n$. More specifically,

$$|\mathbf{x}^{(n)}\rangle = \sum_{i=1}^{2^Q} \bar{x}_i^{(n)} \, |i\rangle_n = \sum_{i=1}^{I} \bar{x}_i^{(n)} \, |i\rangle_n + \sum_{i=I+1}^{2^Q} \bar{x}_i^{(n)} \, |i\rangle_n \ , \tag{4.2.13}$$

where $\bar{x}_i^{(n)} = 0$ for $i = I+1, \ldots, 2^Q$. The same idea can be extended to the entire dataset $X = \left[x_i^{(n)}\right]$, where $i = 1, \ldots, I$, and $n = 1, \ldots, N$. In this case, we normalize the elements of the matrix $X$ by setting $\bar{x}_i^{(n)} = x_i^{(n)} / \|x_i^{(n)}\|$, where

$$\|x_i^{(n)}\|^2 = \sum_{i=1}^{I} \sum_{n=1}^{N} \left(x_i^{(n)}\right)^2 . \tag{4.2.14}$$

The corresponding quantum state is given by

$$X \longmapsto |X\rangle = \sum_{i=1}^{I} \sum_{n=1}^{N} \bar{x}_i^{(n)} \, |n\rangle \, |i\rangle = \sum_{i=1}^{2^{Q_I}} \sum_{n=1}^{2^{Q_N}} \bar{x}_i^{(n)} \, |n\rangle \, |i\rangle \ , \tag{4.2.15}$$

where $2^{Q_I} \geq I$, $2^{Q_N} \geq N$, and $\bar{x}_i^{(n)} = 0$ for $n = N+1, \ldots, 2^{Q_N}$ and $i = I+1, \ldots, 2^{Q_I}$.

The final approach we wish to mention is *angle encoding* or *rotational encoding*. Recall that a qubit in the Bloch sphere is described by the vector

$$|q(\vartheta, \phi)\rangle = \cos(\vartheta/2)|0\rangle + e^{i\phi} \sin(\vartheta/2)|1\rangle \, . \tag{4.2.16}$$

See equation (2.1.5). The parameters $\vartheta$ and $\phi$ lie in the ranges $\vartheta \in [0, \pi]$ and $\phi \in [0, 2\pi)$, respectively. The rotation operator $R_a(\theta_a)$, given in equation (2.1.13), rotates a qubit by $\theta_a$ radians about the $a$-axis, where $a = x, y, z$. For example, a rotation of $\theta$ radians about the $y$-axis is given by

$$R_y(\theta) = \cos(\theta/2) I - i \sin(\theta/2) Y \, . \tag{4.2.17}$$

Applying this operator to the basis vector $|0\rangle$, we get

$$R_y(\theta) \, |0\rangle = \cos(\theta/2) \, |0\rangle + \sin(\theta/2) \, |1\rangle \ ; \tag{4.2.18}$$

where we have used that $Y \, |0\rangle = i \, |1\rangle$. Note that this vector has $\phi = 0$.

Suppose we have a single sample and want to encode a feature value, $x_i^{(n)}$, into a quantum state. To do this, we first rescale $x_i^{(n)}$ so that its new value, $\theta_i^{(n)}$, lies within the range $[0, \pi]$. After that, we associate this value with a vector state as follows:

$$\begin{aligned} x_i^{(n)} \longmapsto \theta_i^{(n)} \longmapsto |\theta_i^{(n)}\rangle &= R_y(\theta_i^{(n)}) \, |0\rangle \\ &= \cos(\theta_i^{(n)}/2) \, |0\rangle + \sin(\theta_i^{(n)}/2) \, |1\rangle \ . \end{aligned} \tag{4.2.19}$$

Consider now all the feature values of the $n$th sample, $x_1^{(n)}, \ldots, x_I^{(n)}$. To use a similar angle parametrization, we can associate the smallest feature value, $x_{i,\min}^{(n)}$, with 0 radians and the largest value, $x_{i,\max}^{(n)}$, with $\pi$ radians. The remaining angles will lie between these two extremes,

$$\theta_i^{(n)} = \frac{x_i^{(n)} - x_{i,\min}^{(n)}}{x_{i,\max}^{(n)} - x_{i,\min}^{(n)}} \, \pi \, . \tag{4.2.20}$$

After applying this transformation to all features and replacing $\mathbf{x}^{(n)} \mapsto \boldsymbol{\theta}^{(n)} = \left[\theta_1^{(n)} \ldots \theta_I^{(n)}\right]^T$, we associate the following ket with this column vector:

$$\mathbf{x}^{(n)} \longmapsto \boldsymbol{\theta}^{(n)} \longmapsto |\boldsymbol{\theta}^{(n)}\rangle = \bigotimes_{i=1}^{I} R_y(\theta_i^{(n)}) \, |0\rangle \, . \tag{4.2.21}$$

The rotation of the single-qubit state vector $|0\rangle$ about the $y$-axis allows us to use the angle $\theta$ to encode one feature value. If there are $I$ features, we can encode them in $I$ angles, with one angle per qubit $|0\rangle$. In general, though, a qubit requires two real values (angles in the Bloch sphere) to be fully described. We can therefore use the angle $\phi$, associated with a rotation about the $z$-axis, to encode another piece of classical data. Using the general formula above for a rotation operator, a $z$-rotation acts as:

$$R_z(\phi) = \cos(\phi/2) I - i \sin(\phi/2) Z \, . \tag{4.2.22}$$

When applied to the qubit state $R_y(\theta) \, |0\rangle$, given in (4.2.17), it produces

$$R_z(\phi) R_y(\theta) \, |0\rangle = e^{-\phi/2} \big( \cos(\theta/2) \, |0\rangle + e^{\phi} \sin(\theta/2) \, |1\rangle \big) \, , \tag{4.2.23}$$

where we have used that $Z \, |b\rangle = (-1)^b \, |b\rangle$. Thus, we can encode $2I$ features in the angles of rotation of $I$ qubits. For example, we can associate the odd-indexed features with rotations about the $y$-axis,

$$x_{2i-1}^{(n)} \mapsto \bar{x}_{2i-1}^{(n)} \mapsto \theta_{2i-1}^{(n)} \, , \tag{4.2.24}$$

and the even-indexed features with rotations about the $z$-axis,

$$x_{2i}^{(n)} \mapsto \bar{x}_{2i}^{(n)} \mapsto \phi_{2i}^{(n)} \, . \tag{4.2.25}$$

Ignoring the global phase, we can encode two classical data points as follows:

$$\begin{aligned} \left[x_{2i-1}^{(n)} \; x_{2i}^{(n)}\right]^T \longmapsto & R_z(\phi_{2i}^{(n)}) R_y(\theta_{2i-1}^{(n)}) \, |0\rangle \\ & = \cos(\theta_{2i-1}^{(n)}/2) \, |0\rangle + e^{\phi_{2i}^{(n)}} \sin(\theta_{2i-1}^{(n)}/2) \, |1\rangle \, . \end{aligned} \tag{4.2.26}$$

Finally, the entire set of features corresponding to the $n$th sample is encoded in the following ket:

$$\begin{aligned} \mathbf{x}^{(n)} \longmapsto & \bigotimes_{i=1}^{I} R_z(\phi_{2i}^{(n)}) R_y(\theta_{2i-1}^{(n)}) \, |0\rangle \, . \\ & = \bigotimes_{i=1}^{I} \big( \cos(\theta_{2i-1}^{(n)}/2) \, |0\rangle + e^{\phi_{2i}^{(n)}} \sin(\theta_{2i-1}^{(n)}/2) \, |1\rangle \big) \, . \end{aligned} \tag{4.2.27}$$

Now that we understand how classical data can be encoded into quantum information, let us examine how this quantum information can be processed into the analog of classical neural networks.

### Quantum Neural Networks

Suppose a dataset with $2I$ features per sample point. As we saw above, if we use angle encoding, we can encode each sample in an $I$-qubit system. That is, the $2I$ classical features of the $n$th sample, where $n = 1, \ldots, N$, can be encoded in a quantum state schematically indicated as follows:

$$\mathbf{x}^{(n)} \in \mathbb{R}^{2I} \longmapsto |\boldsymbol{\varphi}^{(n)}\rangle \in \mathbb{C}^{2^I} \, . \tag{4.2.28}$$

We now let pass the quantum state $|\boldsymbol{\varphi}^{(n)}\rangle$ through a parameterized quantum circuit $U(\theta)$,

$$|\boldsymbol{\varphi}^{(n)}\rangle \longmapsto U(\theta)\, |\boldsymbol{\varphi}^{(n)}\rangle \, . \tag{4.2.29}$$

Finally, we perform the measurement. If we denote by $M$ the observable, the expectation value is then

$$\langle \boldsymbol{\varphi}^{(n)}|\, U^\dagger(\theta) M U(\theta)\, |\boldsymbol{\varphi}^{(n)}\rangle \, . \tag{4.2.30}$$

This is not something new. In fact, it is the standard parameterized quantum circuit approach. What differs in a *Quantum Neural Networks* (*QNN*) is the architecture of the circuit. Suppose that, as we did for the classical neural network, we introduce $H$ hidden layers, each with $N_n$ nodes, where $h = 1, \ldots, H$. Each layer consists of single-qubit gates and two-qubit gates (to create entanglement). These are the analogs of nodes in a classical NN. For simplicity, we fix the two-qubit gates (e.g., CNOT gates) and allow the single-qubit gates to be rotational gates. Given that there are $H$ hidden layers, each acting on $I$ qubits, we have at most $I \times H$ angle parameters to train.

Let us close this brief review of quantum machine learning with some general considerations.

The algorithms we have presented above are usually called *native quantum machine learning algorithms*. This terminology arises from the fact that every step—from encoding classical data to the optimization process—is purely quantum. Another approach in quantum machine learning involves incorporating quantum subroutines into the pipeline of a classical machine learning process. For example, the *Quantum Basic Linear Algebra Subroutines* (*QBLAS*) accelerate basic linear algebra computations. These subroutines can be integrated into classical machine learning procedures to speed up certain operations, such as matrix inversion or inner product estimation. However, it has been shown that neither native quantum machine learning algorithms nor QBLAS necessarily provide an exponential speedup over classical machine learning algorithms in general settings. While certain quantum algorithms—such as those leveraging quantum linear algebra techniques—can offer polynomial or, in some cases, exponential improvements for specific problem instances, these speedups rely on strong assumptions about data access and problem structure. Furthermore, given that practical, large-scale hybrid quantum-classical computing architectures are not yet available, realizing meaningful quantum advantages for machine learning remains an open challenge.

Although much of the discussion about quantum computing revolves around the potential speedup of quantum algorithms compared to classical ones, quantum machine learning highlights other advantages beyond mere computational acceleration.

For example, in certain tasks, QML techniques may provide a better approximation than purely classical ML techniques. Even if the computation takes longer, this advantage can be valuable. This contrasts with the traditional perspective, where quantum algorithms are primarily expected to outperform their classical counterparts in terms of speed. The relevance of this shift is clear: in some cases, accuracy is more desirable than faster computation, such as in privacy-sensitive applications or scenarios requiring higher precision in modeling complex systems. For instance, in financial modeling, precise risk estimation, market trend prediction, or portfolio optimization may be more valuable than rapid execution. Another example outside of finance is drug discovery, where accurately modeling quantum interactions at the molecular level may be more crucial for identifying promising compounds, even at the expense of a longer computational time.

As seen in the previous paragraph, evaluating the advantage of quantum machine learning techniques in real-world applications can be quite complex. In addition to this, there are the technical challenges such as the efficiency and robustness of encoding classical data into quantum states, the impact of qubit and quantum gate fidelity on the training process, the use of quantum optimizers instead of the classical ones usually considered, among others.

### 4.2.2 QML in Finance

The use of the algorithms discussed in the previous paragraphs in finance follows directly from our earlier discussion on the applications of classical machine learning in finance. These quantum-enhanced algorithms aim to accelerate the performance of their classical counterparts. For example, *Quantum Principal Component Analysis* (*QPCA*) is used in portfolio management, and *Quantum k-Means Clustering* enhances clustering techniques for applications such as fraud detection and money laundering.

Let us discuss a few more advanced algorithms that have been proposed for use in finance.[8]

#### Quantum Adversarial Models

The first approach we want to discuss is *Generative Adversarial Networks* (*GANs*). They are called networks because they consist of two neural networks: a *generator*—hence the term *generative*—and a *discriminator*. The generator creates data that resembles real data as closely as possible, which is then sent to the discriminator. The discriminator evaluates how likely this data is to be real or generated. This evaluation is then sent to the generator. These models are called *adversarial* precisely because of this cyclic interaction between the two neural networks. They engage in an opposing game, where the generator aims to create synthetic data that is increasingly difficult for the discriminator to distinguish from real data. Thanks to this interaction, GANs are capable of generating highly realistic data. In finance,

[8]In addition to the references mentioned in the footnote 1, the review paper by M. Pistoia et al., "Quantum Machine Learning for Finance" (2021), offers a concise overview of the subject as it stood a couple of years ago. The book by A. Jacquier and O. Kondratyev, *Quantum Machine Learning and Optimization in Finance*, although it covers many of the subjects already discussed in this guide, provides additional insights into the application of these concepts specifically to finance.

GANs are used to generate synthetic financial data, such as stock prices or market conditions, helping to test the resilience of financial strategies and create more robust models. They can also be employed to generate realistic fraudulent transaction data, thereby improving fraud detection systems.

*Quantum Generative Adversarial Networks*

The second adversarial paradigm we want to discuss is *Adversarial Reinforcement Learning* (*ARL*). *Reinforcement Learning* (*RL*) is a machine learning approach that is neither supervised nor unsupervised. The agent that learns in RL is called the *learning agent*. ARL extends reinforcement learning by introducing an *adversarial agent* that aims to challenge or deceive the learning agent. The key advantage of ARL is its ability to improve the robustness of the learning agent, especially in complex, dynamic environments. In finance, ARL can be used to optimize trading strategies by simulating adversarial market conditions, allowing agents to learn and adapt to unpredictable or hostile environments. It can also enhance risk management by training agents to handle worst-case scenarios and improve portfolio management by making strategies more resilient to market manipulation or unexpected shocks. By incorporating adversarial elements, ARL helps develop strategies that are better suited for real-world challenges.

## 4.3 Programming Quantum Computers

Some of the leading firms working toward building the first practical quantum computer (see Section 5.2) have also made their hardware partially available in the cloud to individuals, startups, and large organizations. To use these platforms, though, users must be familiar with the programming languages developed for them. We will discuss some of them here. In addition to the software provided by hardware developers, independent firms have created their own tools to interface with quantum cloud platforms.

### Qiskit

*Qiskit* is the most popular quantum software framework at the moment. Its success can be attributed to several factors. First, it was developed by IBM, one of the leading quantum hardware companies in the world today, and the company provides free access to their quantum computers through the cloud. Second, Qiskit has a Python-based interface that integrates seamlessly with popular data science tools like NumPy and Pandas, making it easy for students and professionals from various fields who are familiar with Python to get started. As a result, Qiskit is a prominent choice for both educational purposes and professional research. Its success is also attributed to the fact that it is open-source software, benefiting from an active and collaborative community, much like Python, that contributes to the framework's continuous development. Fourth, Qiskit enables users to develop quantum algorithms, simulate quantum circuits, and run experiments on quantum computers and simulators (classical computers that simulate the behavior of quantum computers). Finally, Qiskit supports a growing quantum ecosystem, including integration

with other quantum platforms and hardware.[9]

### Cirq

Similar to IBM's Qiskit, Google has developed its own quantum computing framework called *Cirq*. Cirq shares many of the advantages found in Qiskit.[10] First, it is backed by one of the largest tech companies in the world, which has a proven track record of breakthroughs in quantum computing. Second, it offers a Python-based interface that integrates seamlessly with Google's quantum hardware, making it—at least in principle—appealing to students and professionals from diverse fields. Third, Cirq is open-source, allowing users to contribute to its development and utilize it for creating and running quantum circuits, whether on quantum computers or simulators. Like Qiskit, Cirq also integrates with other quantum computing platforms and tools. However, compared to Qiskit, Cirq lacks a large, established, and collaborative community of researchers and developers. While it's difficult to quantify the exact reasons for Qiskit's broader popularity, it is likely that Qiskit's expansive ecosystem, strong educational resources, and greater community support have made it a more widely adopted choice in the quantum computing developer community.[11]

### PyQuil

The *PyQuil* quantum software framework was developed by Rigetti, a quantum hardware company specializing in superconducting qubits. While IBM and Google also focus on superconducting quantum processors, Rigetti is a smaller company with a distinct emphasis on hybrid quantum-classical computing. PyQuil serves as Rigetti's counterpart to Qiskit and Cirq, designed for seamless integration with its quantum hardware. Like the two frameworks discussed above, PyQuil allows users to design and deploy quantum applications on both simulators and real quantum processors. While Qiskit provides a broader ecosystem covering areas such as quantum circuits, machine learning, and quantum chemistry, PyQuil is specifically designed to facilitate hybrid quantum-classical computing. Although Qiskit and Cirq also support hybrid execution, PyQuil is the most explicitly hybrid-focused framework among the three, making it particularly well-suited for variational quantum algorithms and real-time classical processing. That said, PyQuil lacks some of Qiskit's advantages, such as a larger user base and extensive community resources. IBM's strong investment in educational materials, cloud-based quantum access, and an active global community gives Qiskit a more extensive ecosystem. Nonetheless, PyQuil remains a powerful tool for users working within Rigetti's quantum computing stack, particularly those interested in hybrid computing paradigms. [12]

### Amazon Braket

Although Amazon is developing its own quantum hardware, publicly available information suggests that it is less advanced than that of IBM, Google, and Rigetti. In fact, Amazon is best known for its quantum software framework: *Amazon Braket*.

[9] `https://www.ibm.com/quantum/qiskit`

[10] For an insightful discussion on the differences between Qiskit and Cirq, read my LinkedIn post and the comments at https://bit.ly/QiskitvsCirq.

[11] `https://quantumai.google/cirq`

[12] `https://www.rigetti.com`

Amazon Braket does not use Amazon's own quantum hardware but instead provides access to quantum processors from multiple external providers, including Rigetti (superconducting qubits), IonQ (trapped-ion qubits), QuEra (neutral-atom qubits), and D-Wave (a quantum annealer). In addition to providing a Python Software Development Kit (SDK) to build and run quantum algorithms on quantum computers, it also offers classical quantum simulators to test algorithms before running them on quantum hardware. Finally, Amazon Braket's integration with other classical computing resources from the Amazon ecosystem enables users to create seamless hybrid quantum-classical workflows, which are ideal for solving complex problems, such as those involving variational quantum algorithms.[13]

### PennyLane

*PennyLane* is a hardware-agnostic quantum machine learning framework designed to integrate seamlessly with classical machine learning libraries. It connects to quantum computers through APIs from platforms such as Qiskit (IBM), Cirq (Google), Rigetti's Quantum Cloud Services (QCS), and Braket (Amazon). One of its key strengths is quantum differentiation, which allows for efficient gradient computation in quantum circuits, making it particularly valuable for quantum machine learning and AI research. Additionally, PennyLane works seamlessly with classical machine learning frameworks like TensorFlow and PyTorch, enabling users to develop quantum-enhanced machine learning models and optimize complex tasks by incorporating quantum circuits into hybrid workflows.

[13] `https://aws.amazon.com/braket`

# Chapter 5

# How to Get Quantum-Ready

In this Chapter, we focus on the broader context of the adoption of quantum computing by financial institutions. Before delving into the details, however, it is essential to understand the current status of the quantum computing ecosystem at large.

Recognizing the near-future impact of quantum technologies on society, the governments of the most developed countries have begun funding research centers and educational programs. Given the nascent and evolving status of the field, where cutting-edge research is crucial to securing a competitive position, the primary focus of governments has been on supporting research institutions. Research centers have been established in the United States, China, Europe, Singapore, and the Gulf countries, to name just a few.

To address the present and future needs of research centers and industries already engaged in quantum computing, advanced educational programs tailored specifically to these requirements, such as MSc and PhD programs, have been created. These programs cover diverse topics, ranging from quantum optics and machine learning to the funding of startups. They are available in most countries with significant quantum computing initiatives. Governments have played an active role in funding these programs.

Due to the relatively low cost of educational initiatives compared to advanced research—especially if it involves experimental research—quantum computing educational programs are being proposed globally, including in developing countries. Online certifications are also offered by leading educational institutions such as MIT and by private companies.

The private sector, represented by major technology companies such as IBM and Google, has also developed a strong presence in quantum research and education. Companies not only focus on their internal needs and strategies for the future but also contribute to the collective advancement of the field. In addition to this engagement by major corporations, particularly in hardware and software research and development, dozens of startups have emerged to address specific needs, including those related to financial services, as discussed below.

Collaborations between these three sectors—governments, educational institutions, and the private sector—are frequent.[1] For instance, research partnerships often

[1]The seminal book *The Triple Helix: University-Industry-Government Innovation in Action*, by H. Etzkowitz & L. Leydesdorff, remains the basic reference for explaining how the interactions between governments, universities, and private companies contribute to innovation, economic development, and technological progress. Etzkowitz's most recent book, published under the same title, adopts a more practical approach and is well worth reading.

involve banks, public institutions such as universities, and companies focused on hardware and software development. Similar collaborations are also seen in other fields, such as pharmaceuticals, advanced materials, and engineering research.

## 5.1 Adopting Technological Innovations

As history shows, failing to embrace technological innovations can have severe consequences for businesses, particularly in industries where competition and customer expectations evolve rapidly.

Well-known studies have shown that every time a new technology is developed and introduced to the market, there are always groups ready to adopt it, even at the early stages when the product is incomplete and the future uncertain.[2] On the other hand, there are those who, for various reasons, vehemently reject the idea. These are two extreme cases: the *early adopters* (13.5%) and the *laggards* (16%). Most people, however, fall somewhere in between these extremes. In fact, the majority of people will adopt the technology at an intermediate stage of its development. Some will adopt it earlier—referred to as the *early majority* (34%)—and others later, known as the *late majority* (34%). Before these large groups, there is a minority, the *innovators* (2.5%), who create and initially use the new technology.[3]

Regarding quantum computing, it is important to clearly discern whether it is still in the innovators stage or if it has already moved past to the stage where more people are beginning to believe in its potential and adopt it. Among the innovators, we can include companies like IBM, Quantinuum, D-Wave, and IonQ, among many others, which are developing quantum hardware. In the early adopters group, we can mention financial institutions such as JPMorgan and HSBC. Despite this, evidence shows that the vast majority of businesses are still reluctant to take action and start investing in quantum computing initiatives.

This widespread hesitation explains why many leaders in the financial sector—both individuals and small and large companies—devote significant time, energy, and budgets to promoting the potential benefits of quantum computing for businesses.

### 5.1.1 Consequences of Delaying Technology Adoption

Each market is unique, and the way businesses adopt technological innovations must be assessed on a case-by-case basis. The financial services industry is no exception. However, there are some general considerations that are common to most cases. Broadly speaking, companies that resist adopting new technologies risk falling behind and, ultimately, failing.

A key factor to consider is that in today's fast-paced, technology-driven societies, customers have become increasingly demanding and competition has grown fiercer, leaving companies struggling to acquire and retain customers. This dynamic has led

[2]The subject of innovation is vast. Some recommended readings on the subject are C. M. Christensen, *The Innovator's Dilemma*; C. A. O'Reilly III & M. L. Tushman, *Lead and Disrupt*; the classic text by E. M. Rogers, *Diffusion of Innovations*; and J. Tidd & J. Bessant, *Managing Innovation*.

[3]For an insightful discussion on technological innovation, adoption, and quantum computing, read my LinkedIn post and the comments at https://bit.ly/TechInnovationandAdoption.

to a situation where companies that fail to adapt quickly to technological advancements deliver mediocre services, prompting customers to migrate to more tech-savvy alternatives. We must also remember that technological innovations often enable businesses to improve operational efficiency and economic performance. Consequently, companies that delay adopting new technologies risk incurring financial losses, which can translate into higher costs or inferior services for customers. This, in turn, drives customers toward competitors offering better value and convenience.

In sectors where technology plays a pivotal role and where the stakes are exceptionally high, such as in the financial industry, these considerations become even more critical. Adapting to and embracing technological change is not merely an option but a necessity for survival and growth in this increasingly competitive landscape.

Let us briefly mention two well-known examples of companies, outside the financial industry, that failed due to their reluctance to adopt innovative technologies.

As most of you certainly remember, Kodak was once a leader in the photography industry, dominating the market with its iconic film products and memorable advertising campaigns, such as the "Kodak Moments" slogan, which became synonymous with capturing cherished memories. However, despite inventing the first digital camera in 1975, Kodak chose to focus on its profitable film business, fearing that digital cameras would cannibalize its core revenue stream. The company delayed digital adoption, allowing competitors to seize the opportunity. It is said that Steve Sasson, the engineer who invented Kodak's digital camera, after facing several rejections within the company, saw his invention overlooked while competitors like SONY embraced the technology. SONY launched its first digital camera in the 1980s, leveraging the potential Kodak had ignored. As digital cameras became mainstream, produced notably by Japanese companies such as SONY and CANON, Kodak's market share began to plummet. By the 2000s, the once-dominant company was struggling to stay relevant in a rapidly changing market. In 2012, Kodak filed for bankruptcy, becoming a cautionary tale of technological complacency and the risks of failing to adapt to innovation. Today, Kodak's story is often cited as a lesson for businesses in the importance of embracing change and future-proofing strategies.

Another well-known example of a successful company that failed due to its reluctance to embrace new technologies was Blockbuster. For those who may not remember, Blockbuster was once the global leader in video rental services, with thousands of stores around the world. In the 1990s, it was a staple of weekend entertainment. I fondly recall that during my high school and university years, most Friday nights after class, my friends and I would head to one of the many Blockbuster stores to pick out a couple of movies for the weekend. The vibrant stores, with rows of VHS tapes and DVDs, were part of the cultural fabric of the time. However, Blockbuster ignored the rapid rise of digital streaming and dismissed opportunities to adapt its business model. Most famously, it declined an offer to purchase Netflix for just $50 million in 2000, underestimating the disruptive potential of online streaming services. As consumer preferences shifted toward the convenience of digital platforms, Blockbuster's physical rental model quickly became outdated. In 2010, Blockbuster filed for bankruptcy, leaving behind a cautionary tale of what happens when businesses fail to recognize and adapt to technological change.

Like Kodak and Blockbuster, banks that fail to adopt emerging technologies risk becoming irrelevant in a fast-changing market. Let us recall two such cases. Of course, we do not suggest that these banks collapsed solely due to their lack of

technology adoption; however, it played a crucial role.

The first example is Barings Bank, a bank founded in 1762 in London. After a long and prestigious reputation, Barings collapsed when a trader accumulated massive losses through unauthorized trades. The bank's outdated systems and lack of proper internal controls or automated monitoring tools allowed the trader to exploit weaknesses, leading to catastrophic losses. The second example is Lehman Brothers, a global American bank founded in 1850. One of the most famous bank failures in financial history, Lehman Brothers' downfall was, in part, due to its outdated risk management systems and inability to leverage emerging financial technologies. The bank lacked advanced tools for real-time risk assessment and data analysis, which could have identified the growing risks in its portfolio earlier.

We have just seen two examples of banks whose demise was, to some extent, due to a lack of technological adoption. In the competitive and fast-paced modern financial services sector, banks must prioritize customer satisfaction, efficiency and, as evidenced by the case of Lehman Brothers, effective risk management. As we will discuss next, technological innovations play a crucial role in achieving these goals, in addition to helping banks stay compliant with evolving industry standards.

### 5.1.2 The Big-Data Revolution in Fintech

By *fintech*, some experts broadly refer to the transformative impact of technology on the financial sector. If by finance we mean the processes and tools for managing money, investments, and transactions, then, in fact, the history of fintech spans a long timeline. Our main interest, though, is specifically in information and data technology. For completeness, let us briefly recall some of these interactions.

Businesses in the financial sector have been collecting and utilizing data since the inception of these institutions. For instance, pawn shops, dating back to antiquity, indirectly gathered data about their borrowers to assess creditworthiness and manage risk. Similarly, currency exchange houses often sent employees to survey competitors, evaluate exchange rates, and determine competitive pricing to retain customers. In addition, they kept informed of the political and economic situations of the countries issuing the currencies they traded—whether by reading newspapers, exchanging information by word of mouth, or other means. What has changed in modern times is not the reliance on data itself but the sheer volume of data collected, the systematic ways it is stored and analyzed, and its purposeful utilization for decision-making and competitive advantage.

The invention of the telegraph, and later the telephone, revolutionized the flow of information from the late 19th century, creating an explosion in data and a growing need for efficient storage solutions. By the 1940s, the collection and processing of data were further transformed by the invention and subsequent adoption of electronic computers by financial institutions. A landmark example of this technological advancement was the American company Visa, which used these early computers to launch the first credit card system. For the first time, financial institutions could electronically store, manage, and access the data of cardholders, setting the stage for the data-driven financial systems we see today.

From the advent of the internet in the 1990s, the amount of data collected and processed by Big Tech companies has grown at an unprecedented pace. An important fact is that financial institutions are among the greatest buyers of this information.

Artificial intelligence, and more specifically machine learning, are among the tools banks use to analyze these staggering amounts of data and make sound decisions, enabling better risk management, personalized services, and improved operational efficiency.

In the previous chapters, we explored how machine learning has transformed the financial industry, with applications in portfolio optimization, fraud detection, credit risk analysis, anti-money laundering, and enhanced security. Beyond these, there are other notable innovations, such as the delivery of more personalized services and the use of AI-driven chatbots for instant customer support. However, it is important to remember that the shift toward adopting machine learning was not always universally accepted. Just a few years ago, many professionals and financial institutions were hesitant to embrace it, often due to a lack of understanding, concerns about data privacy, or skepticism regarding its effectiveness. Today, however, there is broader recognition of its potential to revolutionize the industry. It is now widely accepted that banks that fail to adopt machine learning, or artificial intelligence more broadly, risk becoming obsolete, much like institutions such as Lehman Brothers and Barings Bank, which failed due to their inability to adapt. There is now a consensus that AI is essential for enhancing customer experience, reducing operational costs, managing risks, and staying competitive in a rapidly evolving financial landscape.[4]

A final word on quantum technologies: If the historical relationship between finance and technology has been defined by the collection, storage, and transmission of information, it is highly likely that the physical nature of that information will play a crucial role in shaping future developments. Just as the transition from analog to digital information revolutionized the financial services industry in the late 20th century, the shift from digital to quantum information promises to be equally, if not more, transformative.[5]

## 5.2 The Race for the First Quantum Computer

The race to build the first practical quantum computer is a global competition, with tech giants, startups, and research institutions vying for breakthroughs. Companies like IBM, Google, and others mentioned below are advancing superconducting qubit technologies, pushing qubit counts and improving error correction. Meanwhile, IonQ and Pasqal are focusing on trapped-ion and neutral-atom quantum computing, respectively. D-Wave, on the other hand, continues to refine quantum annealing, offering specialized solutions for optimization problems. Another key factor is the global race for quantum dominance. The United States, China, the European Union, and other emerging technological powers are heavily investing in national quantum strategies, recognizing the transformative potential of this field.

Despite rapid progress, though, fundamental challenges remain—qubit decoherence, error correction, scalability, and hardware reliability still limit quantum systems from achieving large-scale practical use. Quantum computing is evolving daily

[4]For an insightful discussion on the adoption of quantum computing by financial institutions, read my LinkedIn post and the comments at https://bit.ly/QCBanksAdoption.

[5]An interesting periodization of the evolution of fintech is provided by D. W. Arner et al., "The Evolution of FinTech." For a detailed discussion on the physics of digital and quantum information, refer to my second course on quantum computing, cited above.

and it remains uncertain which approach will ultimately prove the most transformative. Let us explore some of these technologies and the key players shaping the field.

### Superconducting Qubits

*Superconducting qubits* use superconducting circuits with Josephson junctions to create and manipulate quantum states. While they perform well in gate operations, they require extremely cooling systems that are difficult to achieve. Leading companies in this field include IBM, Google, Rigetti, and Alibaba. Since this is currently the most mature quantum computing technology, let us take some time to discuss it.

*IBM* is one of the leading companies in superconducting qubits. Its technology provides qubits with long coherence times and incorporates advanced quantum error mitigation techniques. The company has developed quantum processors such as Eagle in 2021, with 127 qubits; Osprey in 2022, with 433 qubits; and Condor in 2023, with 1,121 qubits. Despite the tenfold increase in qubit count, maintaining high qubit quality as systems scale remains a significant challenge. Recognizing this challenge, in December 2023, IBM announced the Heron processor, featuring 133 qubits. It was the first in IBM's new generation of error-mitigated quantum processors, focusing on improved qubit connectivity and reduced noise rather than simply increasing the number of qubits. In 2024, the company unveiled the IBM Quantum Heron R2, a 156-qubit processor. This chip builds upon the Heron architecture, increasing the qubit count from 133 to 156 and introducing two-level system mitigation to further reduce noise. By the end of the decade, IBM plans to deliver a fully error-corrected system with 200 qubits capable of running 100 million gates.[6]

*Google*, one of the major tech companies heavily investing in quantum computing developments, employs a similar type of superconducting qubits but with a different architecture. Google's quantum circuits use a planar layout with nearest-neighbor coupling, enabling efficient two-qubit gate operations. They focus on quantum error correction, aiming for logical qubits that can sustain long computations. Google's Sycamore processor, which achieved *quantum supremacy* in 2019, used a 54-qubit chip to outperform classical supercomputers in a random circuit sampling task (actually, only 53 qubits were used because one was faulty). In 2024, Google unveiled its latest quantum processor, Willow, featuring 105 qubits. It achieved in less than five minutes a task that would take today's fastest classical supercomputers millions and millions of years to complete. This was the second time Google achieved quantum supremacy—the only Western company to have reached this goal. Google's roadmap includes building a 1-million-physical-qubit system with error correction by the 2030s.[7]

*Rigetti* is a much smaller quantum hardware company, with approximately 150 employees, that also focuses on developing superconducting qubit processors. Its Aspen-9 processor was announced in 2021 with 32 qubits, and the Aspen-M was announced in 2023 with up to 80 qubits. Their high gate fidelity for single-qubit and two-qubit gates has the potential for scalability and improved performance. Rigetti plans to build quantum processors with up to 1,000 qubits by the end of the decade, with fidelity above 99% and increasing coherence times. Due to their

[6] `https://www.ibm.com/quantum/technology`

[7] `https://quantumai.google/quantumcomputer`

practical-oriented objectives, their roadmap highlights scalable, high-fidelity quantum processors capable of tackling more practical and complex problems by 2030. Additionally, Rigetti is focusing on hybrid quantum-classical integrations. Their approach involves combining quantum processors with classical computing resources to tackle practical problems.[8]

*Alibaba* is another major tech company that focuses on developing quantum hardware using superconducting qubits. They have made significant groundbreaking contributions with their quantum processors. For example, the Zuchongzhi processor, with 56 qubits, was able to achieve *quantum supremacy* in 2020. According to available information, Alibaba's quantum processors have reached up to 60 qubits as of the time of writing. Their roadmap also aims to enhance the fidelity of quantum gates (with goals above 99%) and extend the coherence times of qubits for more complex computations. Their goal is to achieve practical quantum advantage by the end of the decade, with applications in areas like optimization and artificial intelligence.

### Trapped-Ion Qubits

In this approach, individual qubits are encoded in the internal energy states, hyperfine states, or other quantum states of the ions, and precision lasers are used to manipulate these states. The most commonly used ions for trapped ion quantum computing are barium (Ba), calcium (Ca), magnesium (Mg), and ytterbium (Yb). The concept of using ions trapped in electromagnetic fields for quantum computing began to take shape by the mid-1990s. However, it wasn't until the 2010s that *trapped-ion qubits* were scaled up to handle multiple qubits, with advancements in fidelity, coherence, and error correction. Today, trapped-ion quantum technology is regarded as one of the most promising approaches in quantum computing.

*IonQ*[9] and *Quantinuum*[10] are two prominent companies in the field of trapped-ion quantum computing, each employing different hardware strategies. IonQ has developed quantum computers with up to 32 physical qubits, while Quantinuum has achieved up to 20. IonQ's roadmap aims to reach 64 algorithmic qubits (error-corrected qubits) by the end of 2025, with a target logical two-qubit gate fidelity of 99.999%. In contrast, Quantinuum plans to develop a system with 96 physical qubits and a physical error rate below $5 \times 10^{-4}$, and by 2029, they intend to introduce a processor with thousands of physical qubits and hundreds of algorithmic qubits, targeting a logical error rate between $1 \times 10^{-5}$ and $1 \times 10^{-10}$. Both companies face the challenge of scaling their systems while maintaining error correction, qubit coherence, and performance as they increase qubit numbers.

### Photonic Qubits

In addition to the three cases of quantum supremacy achieved using superconducting technologies, *photonic qubits* (also called *optical qubits*) have also achieved quantum supremacy.

Photonic qubits encode quantum information using the intrinsic properties of individual photons, such as polarization. These qubits are generated by single-photon

[8] `https://www.rigetti.com`
[9] `https://ionq.com`
[10] `https://www.quantinuum.com`

sources, manipulated using beam splitters and phase shifters, and measured with single-photon detectors. The key advantages of this technology are its low decoherence, fast transmission, and room-temperature operation, making photonic qubits particularly suited for quantum communication and networking. However, challenges such as weak photon-photon interactions, photon loss, and reliance on probabilistic gates hinder scalability. Despite these obstacles, companies like *PsiQuantum*[11] and *Xanadu*[12] are making significant strides toward large-scale, fault-tolerant photonic quantum computing systems.

As mentioned above, quantum supremacy using photonic qubits was first demonstrated in 2020 with the *Jiuzhang experiment* at the University of Science and Technology of China. The group was able to solve a problem in minutes that would take millions of years on the most powerful classical computers. The follow-up Jiuzhang 2.0 experiment in 2021 further validated these results. While groundbreaking, these experiments are task-specific and, as previously indicated, face ongoing challenges for potential future applications.

### Neutral Atom Qubits

*Neutral atom qubits* use individual neutral atoms, whose states are manipulated by highly focused laser beams to perform quantum computations. The most commonly used atoms for neutral atom quantum computing are alkali metals, such as rubidium (Rb), cesium (Cs), and sodium (Na). One of the key advantages of this approach is scalability, as large numbers of atoms can be controlled simultaneously. Achievements in the field include high-fidelity operations, long coherence times, and successful multi-qubit entanglement. Despite these advancements, challenges remain, such as slow gate speeds, laser precision issues, and the need for error correction. A leading company in this technology is *Pasqal*, founded in 2019 in France by a group of physicists, including the renowned quantum physicist and Nobel laureate Alain Aspect. Pasqal has achieved control over more than 100 qubits in its systems (specific fidelity metrics for this system have not been publicly disclosed). While challenges persist, neutral atom qubits hold significant promise for future large-scale quantum computing. According to the official roadmap, they plan to develop a fault-tolerant quantum computing processor with 128 or more logical qubits by 2028.[13]

### Topological Qubits

In *topological qubits*, quantum information is stored in the global properties of quantum states within certain materials. The key advantage of topological qubits is their enhanced robustness to errors, as the information is encoded in the global properties of the quantum states—specifically in the braiding of exotic particles called *anyons*, which are neither fermions nor bosons. These qubits possess properties that enable them to be isolated and manipulated in a way that makes them resistant to local disturbances. This error resilience has the potential to significantly enhance the scalability and reliability of quantum computers. The leading company in the development of topological qubits is *Microsoft*, which focuses on a specific type of anyon called *Majorana fermions*. The main challenges involve not only the theoretical un-

[11] `https://www.psiquantum.com`
[12] `https://xanadu.ai`
[13] `https://www.pasqal.com`

derstanding of these objects but also experimental difficulties, such as isolating the anyons and performing the necessary braiding operations. Despite these challenges, Microsoft is fully committed to the development of topological qubits, investing heavily in research to make topological quantum computing a practical technology.[14]

### Quantum Annealers

This is the first time we mention *quantum annealers*. We have not discussed them earlier because quantum annealers are not gate-based quantum computers (they are not structured as quantum circuits). In quantum annealing, a problem is encoded into the system's Hamiltonian and the system evolves towards its lowest energy state, which represents the optimal solution. The underlying physics involves a quantum phenomenon called the *adiabatic process*. Quantum annealers are considered non-universal because they are specifically designed to solve optimization problems. They excel at solving optimization tasks in fields like logistics, material science, and finance. *D-Wave* is a leading company in the development of quantum annealers, with their latest processor containing 5,000 qubits. However, the fidelity remains much lower than that of gate-based quantum systems due to noise and decoherence. While quantum annealers face the same challenges as other quantum computing approaches, such as noise, decoherence, and error correction, achieving reliable results for complex problems remains difficult. Some quantum computing experts are skeptical about the results obtained by quantum annealers and the potential of the technology for the future.[15]

## 5.3 Financial Industry Leaders

If we want to understand the potential impact that quantum computers may have in the financial sector, what a better place to look at than the bigger actors in the industry. They want to be sure that when the technology matures, the financial sector will likely be among the first to adopt quantum-enhanced solutions for these high-impact applications.

JPMorgan, HSBC, Goldman Sachs, all of them multinational corporations with valuations of trillions of dollars, are well aware of the catastrophic consequences can have on their future not tacking the upfront position and taking actions regarding the new quantum technology. All of them have created their own quantum research groups and they publish their research. They have invested tens of millions of dollars and have created research groups with dozens of members from complementary fields such as finance, physics, engineering, mathematics, and other technical disciplines.[16]

### JPMorgan Chase

JPMorgan is perhaps the most prominent financial institution actively promoting quantum computing. Its Managing Director and Head of Global Technology Applied Research has helped establish the firm as a leader in quantum computing for financial services. He has not only built a strong team of researchers and published

[14] https://quantum.microsoft.com

[15] https://www.dwavesys.com

[16] For an insightful discussion on how small and large banks are adopting quantum computing, read my LinkedIn post and the comments at https://bit.ly/QCBigvsSmallBanks.

papers and surveys reviewing the state of the art for the expert community but is also actively engaged in promoting quantum computing to non-experts in the financial sector, such as quants and executives. The group's research has been particularly focused on quantum computing for portfolio optimization, option pricing, risk analysis, and machine learning applications such as fraud detection and natural language processing.[17]

### HSBC

HSBC is also a well-known bank heavily investing in quantum computing for finance. However, according to available information, the bank focuses more on cybersecurity, particularly on using quantum technologies to ensure the secure storage and communication of sensitive information, such as customer transactions. Another area of active research at the bank is fraud detection. Thus, although JPMorgan and HSBC are two leaders in the quantum computing for finance industry and have areas of overlap, JPMorgan focuses more on quantum computing solutions for financial services, such as portfolio optimization and quantum machine learning, while HSBC appears to prioritize cybersecurity and fraud detection.[18]

### Wells Fargo

Wells Fargo is another leader in quantum computing for financial services and security. Its areas of research are similar to those of JPMorgan and HSBC, with its own unique strengths. However, the reason we want to mention it here is different. The Managing Director and Chief Technology Officer of Advanced Technology at Wells Fargo has been particularly vocal about the hype surrounding quantum computing and its impact on how financial institutions, from small to large, approach this technology. He believes that the hardware is still not developed enough, and that making large investments in quantum computing for financial services at this stage is not the right decision. He advocates that banks should refrain from making excessive investments, limit their spending to stay informed about the technology and new developments, and only make the leap into quantum once there are practical demonstrations of quantum technology in finance. In his view, banks should remain frugal in their investments and avoid falling into the trap of hype around the supposed practical advantages of quantum computing, which, as of today, remain largely speculative.[19]

Big financial companies, known as *incumbents* in financial jargon, strive to protect themselves not only from their direct competitors but also from Big Tech companies and the emergence of startups that can disrupt the system. These startups have the potential to capture a portion of the market share or make incumbents reliant on their innovations, whether in hardware, software, or specialized services. Early collaborations between big financial firms and startups have become a common strategy for the former to integrate new technologies and mitigate risks.

Before delving into the startup landscape, it is important to address the potential risks that financial incumbents face regarding Big Tech companies, which are

[17] `https://www.jpmorgan.com/technology/applied-research`
[18] `https://www.hsbc.com/who-we-are/hsbc-and-digital/hsbc-and-quantum`
[19] Refer to the comments at https://bit.ly/QCBigvsSmallBanks.

among the largest collectors of information in the world and leaders in quantum computing research and development. Companies like Google, Amazon, and Meta (formerly Facebook) have already begun entering the financial services industry, and the possibility of them expanding their presence more decisively is a real concern for traditional financial institutions.

The Chinese multinational company Alibaba, actively involved in quantum computing research, is particularly noteworthy. This platform has seamlessly integrated financial services into its ecosystem, offering payment solutions, lending, and investment products, thereby transforming how millions of users manage their finances. Such developments illustrate the disruptive potential Big Tech companies could bring to the global financial industry.

To conclude, it is worth noting that some major banks have yet to start—or at least have not made it publicly known—that they are adopting quantum computing in their research or hiring personnel to work on it. Perhaps these institutions believe, as historical precedent has sometimes proven, that it is more strategic to wait until the technology has matured before making the substantial intellectual and financial investments required to catch up with competitors. However, this strategy is not without its risks. Delaying adoption could mean falling significantly behind early adopters who have already established expertise and infrastructure. On the other hand, investing heavily in a technology that is still several years away from widespread practical implementation also carries inherent risks. The decision requires a careful balance between foresight and pragmatism.

## 5.4 The Startup Landscape

In recent years, the financial industry has seen the emergence of several startups dedicated to addressing financial challenges, such as portfolio optimization and option pricing. Due to the confidential nature of the services they provide, specific details are often not available. However, let us mention a few that have published their work.

### Multiverse Computing

Multiverse Computing was founded in 2019 by a team of financial experts and physicists. Originally established in San Sebastián, Spain, the company has expanded its presence with offices in Paris, Munich, London, among others. Initially, Multiverse Computing focused on providing quantum software solutions for the finance industry. Recognizing that many finance professionals are not well-versed in quantum computing, the company developed Singularity, a software platform that allows users to implement quantum algorithms through familiar tools like Microsoft Excel, without requiring prior quantum computing knowledge. Multiverse Computing has collaborated with major financial institutions, including BBVA and Crédit Agricole CIB, to explore the application of quantum computing in finance. These collaborations have led to advancements in areas such as portfolio optimization and risk analysis. The company actively publishes its findings in research papers, contributing to the broader scientific and financial communities. More recently, Multiverse Computing has emphasized the development of quantum-inspired algorithms to enhance artificial intelligence capabilities. This strategic shift aims to leverage quantum tech-

nologies to improve AI processes, making them more efficient and effective across various applications. Over time, Multiverse Computing has broadened its scope to offer quantum-inspired solutions across various sectors, including energy, manufacturing, logistics, biotech, pharmaceuticals, and aerospace.[20]

### Quantum Signals

Much smaller than Multiverse Computing, Quantum Signals focuses exclusively on quantum computing solutions for the financial sector. Founded in 2024 by two quantum computing experts in finance, the company has, from the beginning, focused on quantum-inspired solutions aimed at speeding up and improving the accuracy of artificial intelligence applications.[21]

### AbaQus

AbaQus was founded in 2021 in Vancouver. While detailed information about the company remains limited, online sources suggest that the company focuses on the practical implementation of quantum algorithms, providing access to cloud-based quantum services, experimenting with quantum-inspired optimization, and integrating hybrid quantum-classical approaches into existing financial workflows.[22]

### QuantFi

Founded in 2019 by American and French business partners, QuantFi is a startup that offers quantum computing solutions for financial institutions. They engage in training initiatives to help financial professionals understand quantum computing and its potential impact—similar to what we saw with AbaQus. Additionally, they conduct joint research projects with institutions to explore quantum computing applications in finance.[23]

It is important to exercise caution when evaluating the achievements of software developments in quantum computing for finance. In addition to the well-known lack of transparency in machine learning results (where many claimed outcomes are presented without adequate data or processing details), the financial sector faces the additional challenge that large companies, as well as startups, are often unwilling to disclose their complete methodologies for strategic and competitive reasons. Furthermore, regulations frequently prohibit the disclosure of sensitive information about customers, adding another layer of complexity. These factors make it difficult—and sometimes purely speculative—to assess the actual level of development of software solutions proposed in the market for addressing financial sector problems. This lack of transparency can hinder informed evaluations and the broader adoption of promising technologies.

As in almost any area of scientific research, achieving progress requires a balance among the interests of companies, the ethical handling of customer data, and adherence to scientific principles. Such a balance would allow the scientific community to critically assess and validate the factual advancements in quantum computing for

[20] https://multiversecomputing.com
[21] https://www.quantumsignals.ai
[22] https://www.abaqus.dev
[23] https://www.quantfi.com

finance.

## 5.5 A Safe Roadmap to the Quantum Future

Quantum computing is expected to be a game-changer for industries like finance in the near to mid-term future. While the development of this technology is still in its early stages and significant work remains, as quantum technology matures, it will enable faster and more accurate solutions that classical computers cannot match. In this context, financial institutions that have already adopted or plan to adopt quantum solutions early are likely to gain a significant competitive edge, particularly in areas where computational speed and precision are critical. Examples include optimizing financial portfolios, enhancing Monte Carlo simulations, and identifying patterns in the vast data sets collected by financial institutions.

While it is true that the few quantum computers available today are expensive and largely experimental, the assumption that small and mid-sized financial institutions are destined to fall behind the larger players is inaccurate. In fact, several actionable steps can be taken by smaller institutions to remain competitive—and in some cases, even gain an advantage over bigger financial incumbents.

The first strategy that comes to mind, and can be implemented today, is to educate the workforce on quantum computing and shift the mindset of the entire team toward embracing the upcoming revolution of quantum computing in finance. This transition will undoubtedly require specialized skills that are not commonly found in financial organizations. Therefore, hiring personnel who can bridge the gap between traditional and quantum approaches to solving financial problems (as discussed above) is essential. Early preparation will allow institutions to build a team of experts, even if small, to align their strategies with quantum technologies, particularly focusing on algorithms tailored to financial challenges. This proactive approach will ensure a smoother transition as fully quantum solutions become available.

Given the high cost of the limited quantum computers available, several technology companies offer cloud-based quantum computing services.[24] These platforms enable individuals and institutions to experiment with quantum algorithms without requiring an investment in expensive quantum hardware. By leveraging these services, small and mid-sized financial institutions can begin exploring quantum applications tailored to their unique needs.

A strategy commonly employed by large financial companies, but also accessible to smaller institutions, is partnering with universities, research centers, and fintech startups that are open to collaborations for research purposes. By engaging in these initiatives, small institutions can gain early access to emerging quantum tools and expertise, which can enhance their understanding of both research and the market.

In addition to the direct financial applications of quantum computing, small and mid-sized financial institutions can also leverage it for logistical purposes. The speedup gained from quantum machine learning in optimization problems can help these institutions streamline operations and reduce costs.

In summary, by upskilling and hiring a quantum computing workforce, utilizing cloud-based quantum services, and collaborating in quantum research efforts with

[24] See details above, 4.3.

educational institutions or fintech startups, small financial organizations can benefit from quantum computing without the resources of a major bank. This approach will enable them to prepare for a future in finance shaped by quantum technologies.

# Chapter 6

# Conclusion

In this guide, I have sought to strike a balance between technical and practical aspects, emphasizing the complexities of applying quantum computation techniques to the financial services industry. An inherently interdisciplinary subject, it encompasses quantum mechanics, computer science, mathematical finance, programming, technological research and business innovation.

Before concluding with actionable steps you can take to prepare for the next major technological revolution in the financial industry, let us recap some of the main points covered in this short book.

Quantum computation, by leveraging the principles of quantum mechanics that govern nature, aims to build quantum computers whose efficiency and precision will surpass current digital computers by hundreds or even millions of orders of magnitude.

Among the many applications this groundbreaking technology will have in society, the financial industry is expected to be one of the first to benefit from it. By accelerating and improving conventional computational techniques—such as optimization, Monte Carlo simulations, and machine learning—quantum computers promise to transform the future of finance. They will enhance the way financial organizations provide services, refine predictive strategies, and optimize investment methods, all while ensuring compliance with increasingly stringent regulations imposed by governments and regulatory bodies.

To become "quantum ready," small and mid-sized financial institutions must act proactively before it is too late. Although workable quantum computers are likely several years away, this time should be used to prepare staff and familiarize them with the most pressing advancements in the field. This preparation will not only ensure a smoother transition but also provide a competitive edge in a rapidly evolving technological landscape.

If you are interested in learning how to implement quantum technologies in your own organization, please do not hesitate to contact me. I would be glad to help you navigate this exciting frontier and position your institution for success in the quantum era.

# Index