UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Quantum algorithms for function optimization

Autor: Gerard Ortega Ballesteros

Director:	Dra. Joana Cirici
Realitzat a:	Departament de Matemàtiques i Informàtica

Barcelona, January 23, 2021

Abstract

Quantum computing has started to become a reality, as many big and small companies are building state of the art quantum computers, with some even offering quantum services to the public, as quantum computers have been proved to offer a significant computational advantage in comparison to classical computers. Often, though, mathematicians and computer scientists have difficulties getting into the field of quantum computing because of their lack of knowledge about quantum physics. In this work, we take on the more applied side of quantum computing using an abstracted mathematical framework - the gate-based quantum computing model, completely separated from the physics behind.

We first introduce the intuition behind quantum computing and present the mathematical basis of the aforementioned gate-based model. We review and implement some elemental quantum algorithms and after that then dive into the topic of function optimization. For discrete function optimization, we review Grover Adaptive Search, devise an efficient implementation of the algorithm, run it on a simulator and analyze its computational cost. For continuous function optimization, we summarize the most relevant work regarding the topic and pave the way for others that want to implement the algorithms described.

Resum

La computació quàntica és ara part de la realitat, en la qual diverses empreses estan construint les seves computadores quàntiques i oferint serveis de computació quàntica, motivats per la supremacia quàntica respecte la computació clàssica. Malauradament, els matemàtics i científics de la computació que volen entrar dins d'aquest món sovint es troben obstaculitzats pel fet de no saber de física quàntica. En aquest treball, ens centrem en la part aplicada de la computació quàntica fent servir una abstracció matemàtica totalment separada de la part física - la computació quàntica basada en portes quàntiques.

Primer ens introduïm en les nocions bàsiques de la computació quàntica de manera intuïtiva i presentem la base matemàtica del model de portes quàntiques mencionat. Expliquem i implementem alguns algorismes quàntics fonamentals i després ens llencem de cap al tema d'optimització de funcions. Per funcions discretes, detallem el candidat Grover Adaptive Search, ideem una implementació eficient de l'algorisme, l'executem en un simulador i analitzem el seu cost computacional. Per funcions contínues, resumim els articles més relevants que tracten el tema i fem una anàlisi general del contingut i la possible implementació per a facilitar la feina a qui en el futur vulgui realitzar implementacions dels algorismes.

2020 Mathematics Subject Classification. 81P68, 68Q12

Acknowledgements

Before anything else, I would like to thank my advisor Dr. Joana Cirici for her patience, her trust, and her help in correcting the contents of the project, significantly improving its quality.

Furthermore, I would like to thank my father for inspiring me to get into quantum computing, my mother for emotional support throughout the entire degree, and any of my friends that endured my rambles about the topic during the project.

Contents

1 Introduction

In quantum computing, a qubit (**qu**antum **bit**) is the equivalent to the usual classical computation bit. It is the minimal unit of data that we can work with. In a classical bit we can store and read the values 0 and 1, and this is achieved physically, for instance, by magnetizing a film of ferromagnetic material in a HDD or by storing electric pulses in memory cells of a SSD. Usually, the value 1 corresponds to the presence of the magnetization or the electric pulse, while the value 0 is associated with its absence.

A qubit stores the state of a two-level quantum system, which can be in one of two different elemental states by which we refer to $|0\rangle$ and $|1\rangle$, similar to the classical bit. We will not get into quantum systems and will strictly talk about qubits. The key difference lies in the fact that a qubit does not need to be just in one state or the other, but can be in a **superposition** of these two states, a kind of mixture of these. This is mathematically modeled as a complex linear combination of the two basic states.

A **pure qubit state** is described by an expression of the form:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix};\ \alpha_0, \alpha_1 \in \mathbb{C};\ |\alpha_0|^2 + |\alpha_1|^2 = 1,$$

where $|\alpha| = \alpha \cdot \bar{\alpha}$ denotes the modulus of α.

The complex numbers α_0 and α_1 are called the **amplitudes** of the states $|0\rangle$ and $|1\rangle$. When both $\alpha_0, \alpha_1 \neq 0$, the qubit is said to be in a superposition of the states $|0\rangle$ and $|1\rangle$, and the values $|\alpha_0|^2$ and $|\alpha_1|^2$ reflect how much each elemental state contributes to the superposition. A qubit then stores these two amplitudes α_0 and α_1. The concept of superposition is exemplified in section 1.6 of [KLM07].

In contrast to the classical bit, a qubit holds a greater quantity of data and allows for special techniques that can be used to execute algorithms in exponentially less steps compared to a classical computer (see for instance section 3.2 of [NC00] and section 1.2 of [KLM07]). This, though, does not mean that qubits do not have their limitations. For example, the basic operations of *copy* and *read* that can be applied to classical bits cannot be applied to qubits, meaning that we cannot copy a state of one qubit to another nor we can directly read the values α_0 and α_1 stored. On the other hand, current classical memory storage lasts for an indefinite amount of time, can be quickly read and written repeatedly, is error-proof and considerably cheap, while qubits, unless kept isolated and in extreme physical conditions, such as near absolute zero temperatures, are incredibly volatile and hold information for only ephemeral quantities of time, as phenomena like **quantum decoherence and dissipation** come into play, processes by which a the information stored in a qubit becomes corrupted and therefore lost (see for instance section 8 of [NC00]).

Different physical implementations of qubits are, for instance, electron spins (states encoded into spins of electrons), ion traps (states encoded into energy levels of electrons), and superconducting structures (very small electrical circuits at low temperatures). Some physical implementations are outlined in [Zyg18]. This matter is currently an active research field.

The previous facts are reasons for which, compared to bits, qubits are harder to handle and work with. While we can read the value stored in a bit, we cannot simply read the amplitudes α_0, α_1 stored in a qubit. What is done instead is a **measurement** of the qubit. Because of how quantum measurements work, when measuring the qubit, we will either read the state $|0\rangle$ or the state $|1\rangle$ at random with probabilities $|\alpha_0|^2$ and $|\alpha_1|^2$ respectively (hence it makes sense that $|\alpha_0|^2 + |\alpha_1|^2 = 1$). After the quantum measurement, the qubit will collapse onto the state $|0\rangle$ if we read $|0\rangle$, or onto the state $|1\rangle$ otherwise, meaning that if we then repeat the measurement we will read the same state we read the first time over and over, having destroyed the information it held.

Conceptually, we can understand a classical bit as a light bulb that is either turned off (state 0) or turned on (state 1). In this fashion, a qubit would be understood as a light bulb rapidly shifting between off (state $|0\rangle$) and on (state $|1\rangle$). When we try to take a picture of the light bulb (we measure the qubit), we will capture an image of the bulb either turned on or off, and after that, the bulb will stop blinking and will remain turned on or off depending on how the bulb appeared on the picture. This last fact rules out the possibility of measuring a qubit, say, 1000 times, and approximating α_0, α_1 by counting how many times we read $|0\rangle$ and $|1\rangle$. This is more formally explained in Box 2.4 on [NC00]

Another idea one could devise would be to copy the state of a qubit onto another qubit (to then measure the copy), but on the framework of quantum mechanics, the **No-cloning theorem** (see Box 12.1 of [NC00]) states that perfect copies of qubit states cannot be created; only imperfect or dependent copies from the original state can be made, so cloning the qubit state on 1000 other qubits and measuring all of them is not possible as well.

Harnessing the advantages that qubits and quantum computing provide relies on performing special calculations or manipulations that would not be possible with bits on classical computers. There exist many models of quantum computing that define and study manipulations over qubits in different ways, but the most physics-free and natural for mathematicians and computer scientists is the **gate model**. Amongst other more physics-centered models lies the adiabatic model, which we do not discuss but is actually equivalent to the gate model [Aha+08]. Quantum algorithms are often easier to devise or explain in some quantum computing models more than others.

In the gate model, the counterpart to logic gates that act on classical bits (AND, OR, NOT, ...) are **quantum gates**, which act on qubits. A 1-qubit quantum gate is an atomic process that operates on a single qubit and modifies its state. Mathematically, a quantum gate is modeled as a unitary matrix $U \in \mathcal{M}_2(\mathbb{C})$.

Some basic and well-known gates are the Pauli gates X, Y, Z, the Hadamard gate H and the $\pi/4$ gate T:

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},\ Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},\ H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},\ T := \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix}$$

Applying a quantum gate U to a qubit $|\psi\rangle$ will leave the qubit in the state $U|\psi\rangle$. For example, applying the Hadamard gate H to a qubit $|\psi\rangle$ in the state $|1\rangle$ would leave the qubit in the state:

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

and then, measuring this qubit would yield one of the elemental states $|0\rangle$ and $|1\rangle$ with equal probability, thus creating a superposition of states. The reason behind this unitarity constraint on these matrices is that if the amplitudes α_0, α_1 of a pure state $|\psi\rangle$ satisfy $|\alpha_0|^2 + |\alpha_1|^2 = 1$, then the same condition will hold for the amplitudes of the result state $U|\psi\rangle$. Just like with classical gates, there are quantum gates that act on more than one qubit, but they will be addressed in section 2.

The X gate is often called the *NOT* gate, because its action on qubits is $|0\rangle \longmapsto |1\rangle$ and $|1\rangle \longmapsto |0\rangle$, just like what a classical *NOT* gate would do.

The T gate is often called the $\pi/4$ gate because, up to an unimportant global phase factor of $e^{i\frac{\pi}{8}}$, the T gate is equivalent to:

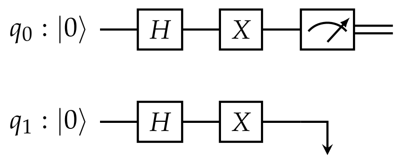
$$e^{i\frac{\pi}{8}}T = e^{i\frac{\pi}{8}} \begin{pmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$

The reason for this fact will be explained later in section 2.2.

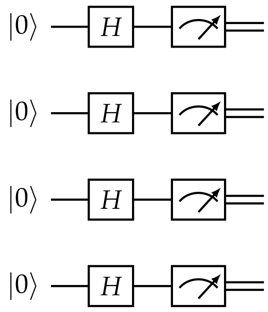
Physical implementations of quantum gates depend on the chosen physical implementation of qubits. Currently, applying quantum gates is not a perfect process and errors may arise. Precision of quantum gates is measured by their **error** (formally defined in section 2.7), which quantifies how much the result of applying the gate differs from the expected result.

With the basic elements of quantum computing: qubits and quantum gates, we can start designing **quantum circuits** to perform operations and execute algorithms, which are just like classical logical circuits, but with quantum gates and qubits instead of logical gates and bits.

Each qubit is drawn as a wire, and to perform operations on them we set the desired gates on the wires in succession. After the calculations, qubits may be measured or discarded. The following example depicts two qubits q_0, q_1, both on initial state $|0\rangle$, undergoing quantum gates H and X, in order. At the end, q_0 is measured, obtaining a classical result, and q_1 is not measured and therefore discarded.



Here is a very simple example of a circuit that uses a 4-qubit register (4 qubits) and outputs a random integer ranging from 0 to 15 in the form of a 4-bit string, provided by the measured qubits in the state $1/\sqrt{2}\,|0\rangle + 1/\sqrt{2}\,|1\rangle$:



The Hadamard gate performs the following transformation:

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

If we initialize all qubits to $|0\rangle$, after the Hadamard gates they will be in an equal superposition of the states $|0\rangle$ and $|1\rangle$, and measuring each of them will yield one of those elemental states with equal probability, effectively outputting 4 random bits. This is a simple example to illustrate quantum circuits, but with this we barely scratch the surface of how deep the techniques for designing them really are, as most quantum algorithms rely on **superposition** and **entanglement** (which will be formally defined in section 2, often requiring a purely algebraic analysis to use effectively. In the succeeding sections we may see these techniques and algebraic analysis when reviewing or designing certain quantum algorithms, where we will use other quantum gates other than the five basic gates presented before.

Quantum computing is based on quantum mechanics, which is a relatively modern field of physics. Though the term "quantum mechanics" has been around since 1920, the term "quantum computing" did not appear until 1982, which was when the theoretical physicist Richard Feynman observed that simulations of quantum systems took exponential time in classical computers. Other physicists took interest on the topic a few years before 1982, though, and studied the possibility of quantum computers.

Still, quantum computing did not begin sparking interest until 1994, when Peter Shor developed the famous Shor's algorithm, a quantum algorithm capable of factoring large primes exponentially faster than classical computers. The steep computational cost of large prime factorization is precisely what makes cryptosystems safe, meaning that Shor's algorithm could theoretically break many cryptography protocols currently in use, hence the sudden interest on quantum computing.

Quantum algorithms are executed in **quantum computers**, which permit physical realizations of quantum circuits. There exist some quantum computers built by big companies such as Google, IBM or Amazon, as well as some other small companies. But because quantum computing has existed outside of the theoretical frame for so short, quantum computers are still in an experimental phase, and are not available to the general public like classical computers.

Some companies, more notably IBM, offer public usage of some of their quantum computers, but the number of qubits available is very low, and one has to account for quantum noise and errors caused by physical imperfections. Therefore, to test that quantum algorithms work in ideal conditions, they are executed in simulators that model qubits and quantum gates instead.

Simulators can help in designing and empirically testing quantum algorithms, but simulations become very expensive computationally as the number of qubits n increases. Simulating the state of a qubit requires storage of 2 complex scalars, and quantum gates require 2×2 complex scalars. This scales exponentially for n qubits, as we will see in the next section that simulating the state of n qubits requires 2^n complex scalars, and quantum gates gates can take as much as $2^n \times 2^n$ complex scalars to simulate. Quantum gate application simulations require matrix products, and with these sizes, they will take $O(2^{3n})$ operations to simulate, an absurdly high computational cost.

Therefore, running quantum computer simulators locally on personal computers eventually becomes too much for classical computers as the number of qubits increase. This is why the simulations and tests on [Ort21] use at maximum 16 qubits, otherwise, they take too much time to perform.

There exist open-source quantum computing frameworks that include simulators that can be run locally in classical computers. The most prominent one is **Qiskit** [Abr+19; 21a], founded by IBM, but another instance is **Grove** [SCZ16; 21b], founded by Rigetti Computing. Still, one can implement their own quantum computer simulator based on the mathematical abstraction of gate-based quantum computing presented along this section, if they wish to do so. An example, along with some executions and source code, can be found in [Fin19].

We next briefly summarize the contents of this thesis.

In section 2 we summarise and review the mathematical framework of gate-based quantum computing. After that, in section 3 we present some of the most basic and relevant quantum algorithms, and provide custom implementations and interactive test runs of Phase Estimation and Grover Search to showcase how they work.

In section 4 we present the discrete version of the optimization problem that we want to solve, and thoroughly analyze Grover Adaptive Search as a candidate. In particular, we analyze the computational cost of Quantum Counting usage and the trial and error method, which we end up discarding. Using the analysis made, we ultimately devise a novel methodology for Grover Adaptive Search and demonstrate empirically that it preserves more than half of Grover Search's quantum advantage when using a large number of qubits. At the end, we provide a custom implementation of Grover Adaptive Search using the devised methodology with the source code and interactive test runs. The custom implementation is a general, unconstrained version of Grover Adaptive Search that the one Qiskit offers under the name of Grover Optimizer [Qisb]. It is a prerequisite to go over sections 2 and 3 before reading section 4.

Finally, in section 5, we present the continuous version of the optimization problem, and review some other works that propose quantum versions of classical algorithms such as Gradient Descent and Newton's Method. The section's purpose is to mention the most notable and interesting works on continuous function optimization to help whoever is interested on the topic and wants to implement any of the algorithms.

2 Gate-based quantum computing

This section explains the basic mathematical background behind gate-based quantum computing, and is mostly based on [KLM07]. Other basic references for quantum computing include [NC00] and [Zyg18].

We first review n-qubit systems as well as their geometric interpretation. Then we formally define entanglement and explain what n-qubit quantum gates and controlled gates are. After that, we expand on measurements, introduce the notion of mixed states and overview universality for quantum gates.

2.1 Dirac notation and n-qubit systems

We will begin by explaining this notation $|\cdot\rangle$ used in the previous chapter.

Let $\psi = (\alpha_0, \dots, \alpha_{n-1})$ be a point on the complex vector space $\mathbb{C}^n$. The **ket** $|\psi\rangle$ and the **bra** $\langle\psi|$ represent ψ and its adjoint $\overline{\psi}$ as a column and row vector, respectively:

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \end{pmatrix}, \quad \langle\psi| = \left(\overline{\alpha_0}, \dots, \overline{\alpha_{n-1}}\right).$$

Thogether with another point $\varphi = (\beta_0, \dots, \beta_{n-1})$, the **bra-ket** then is the inner product of the two vectors:

$$\langle\psi|\varphi\rangle = (\overline{\alpha_0}, \dots, \overline{\alpha_{n-1}}) \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{n-1} \end{pmatrix} = \overline{\alpha_0}\beta_0 + \cdots + \overline{\alpha_{n-1}}\beta_{n-1}.$$

This notation is called the **Dirac notation**, and is the standard notation for describing qubit states. In the previous chapter, we used it as a model for pure states of one qubit, but the general definition is for any number of qubits.

Remark 2.1. The vector space $\mathbb{C}^n$ together with the inner product $\langle \cdot | \cdot \rangle$ forms a Hilbert space, which is an Euclidean space that is also a complete metric space with respect to the distance induced by the inner product.

Definition 2.2. A **pure n-qubit state** is described by a ket $|\psi\rangle \in \mathbb{C}^N$ such that $\langle\psi|\psi\rangle = 1$, where $N = 2^n$.

Remark 2.3. Note that the previous condition is equivalent to

$$\langle\psi|\psi\rangle = (\overline{\alpha_0}, \dots, \overline{\alpha_{N-1}}) \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} = \overline{\alpha_0}\alpha_0 + \cdots + \overline{\alpha_{N-1}}\alpha_{N-1} = |\alpha_0|^2 + \cdots + |\alpha_{N-1}|^2 = 1.$$

These values $|\alpha_0|^2, \dots, |\alpha_{N-1}|^2$ correspond to the probabilities of reading each elementary state of the system after a measurement. We describe each of the elemental states using the vectors of the canonical basis of $\mathbb{C}^N$, which are denoted as

$$|0\rangle, |1\rangle, \dots, |N-1\rangle = \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}, \dots, \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix}.$$

Quantum computing makes use of multiple qubits (not just one) to implement algorithms. As each qubit can be on 2 different elemental states ($|0\rangle$ or $|1\rangle$), two qubits together can be in 4 different combinations of elemental states ($|0\rangle|0\rangle$, $|0\rangle|1\rangle$, $|1\rangle|0\rangle$ or $|1\rangle|1\rangle$). It follows naturally that n qubits can form a total of N combinations of the elemental states.

This is suitably described by the **tensor product**.

Definition 2.4. Let $\mathcal{H}_1, \mathcal{H}_2$ be two vector spaces with bases

$$\{|\psi_i\rangle\,; i=1,...,n\}, \{|\varphi_1\rangle\,; j=1,...,m\},$$

respectively. The **tensor product** $\mathcal{H}_1 \otimes \mathcal{H}_2$ is another vector space with basis

$$\{|\psi_i\rangle \otimes |\varphi_j\rangle\,; i=1,...,n; j=1,...,m\}.$$

Remark 2.5. The map $\otimes : \mathcal{H}_1 \times \mathcal{H}_2 \longrightarrow \mathcal{H}_1 \otimes \mathcal{H}_2$ is bilinear. It is special because any other bilinear map $\mathcal{B} : H_1 \times H_2 \longrightarrow H_3$ factors through it via a unique linear map $\mathcal{L} : H_1 \otimes H_2 \longrightarrow H_3$ (that is, for every bilinear map $\mathcal{B}$ there exists a unique linear map $\mathcal{L}$ such that $\mathcal{B} = \mathcal{L} \circ \otimes$).

Between all the possible concrete choices for the tensor product, the one that reflects the probabilistic nature of states and measurements is the **Kronecker product**.

Definition 2.6. Let $\mathcal{M}_1, \mathcal{M}_2$ be two matrix spaces. The **Kronecker product** $\otimes : \mathcal{M}_1 \times \mathcal{M}_2 \longrightarrow \mathcal{M}_1 \otimes \mathcal{M}_2$ is defined by:

$$A \otimes B = \begin{pmatrix} a_{1,1} \cdot B & \dots & a_{1,n} \cdot B \\ \vdots & \ddots & \vdots \\ a_{m,1} \cdot B & \dots & a_{m,n} \cdot B \end{pmatrix}.$$

From now on, the symbol $\otimes$ will refer to the Kronecker product and not just any tensor product, however, most of the time it is omitted and $|\psi_1\rangle \otimes |\psi_2\rangle$ is just written as $|\psi_1\psi_2\rangle$ for simplicity. The definition of Kronecker product for vector spaces is the same if we see vectors as $1 \times n$ matrices.

Example 2.7. Let $|\psi_1\rangle = \sqrt{3/11}\,|0\rangle + \sqrt{8/11}\,|1\rangle, |\psi_2\rangle = \sqrt{7/13}\,|0\rangle - \sqrt{6/13}\,|1\rangle$. Then:

$$|\psi_1\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = \begin{pmatrix}\sqrt{3/11}\\ \sqrt{8/11}\end{pmatrix} \otimes \begin{pmatrix}\sqrt{7/13}\\ -\sqrt{6/13}\end{pmatrix} = \begin{pmatrix}\sqrt{3/11}\sqrt{7/13}\\ -\sqrt{3/11}\sqrt{6/13}\\ \sqrt{8/11}\sqrt{7/13}\\ -\sqrt{8/11}\sqrt{6/13}\end{pmatrix} = \begin{pmatrix}\sqrt{21/143}\\ -\sqrt{18/143}\\ \sqrt{56/143}\\ -\sqrt{48/431}\end{pmatrix}.$$

We can see that when measuring the qubits, in order to read the state $|00\rangle$ both qubits must output $|0\rangle$ when measured. The chances of this happening are $3/11 \cdot 7/13 = 21/143$, which is precisely the modulus of the first element of $|\psi_1\psi_2\rangle$ squared. The same is true for the other states $|01\rangle, |10\rangle$ and $|11\rangle$.

The state $|\psi_1\psi_2\rangle$ is called the **joint state** of the two qubits.

During the rest of this work we will only consider the qubit as our minimal data unit for computations, which recall store the states of 2-level quantum systems. It is worth noting that in quantum physics not only particles that form 2-level quantum systems exist. Some others form 3-level quantum systems for instance, and can store amplitudes for three elemental states $|0\rangle, |1\rangle$ and $|2\rangle$. In this case, they are called **qutrits**, instead of qubits. The generalized notion for this are **qudits**, which can be in d elemental states $|0\rangle, \dots, |d-1\rangle$.

While the vast majority of quantum algorithms are designed only with qubit usage in mind, the mathematical framework for quantum computing is entirely compatible with qudits.

2.2 Geometric interpretation of qubits

Geometrically, a qubit can be visualized as a single point on the surface of a sphere known as the **Bloch Sphere**. This representation can help understand their algebraic expression, how measurements work and what quantum gates do to them.

Consider the pure state $|\psi\rangle$ of a qubit:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle; \ \alpha_0, \alpha_1 \in \mathbb{C}; \ |\alpha_0|^2 + |\alpha_1|^2 = 1.$$

If we write α_0, α_1 in polar form, we get:

$$|\psi\rangle = r_0 e^{i\gamma_0} |0\rangle + r_1 e^{i\gamma_1} |1\rangle; \ r_0^2 + r_1^2 = 1; \ \gamma_0, \gamma_1 \in [0, 2\pi].$$

Now consider the state $e^{i\varphi}|\psi\rangle$, where $\varphi \in [0, 2\pi]$. We can say that both states $|\psi\rangle$ and $e^{i\varphi}|\psi\rangle$ are equivalent, because:

- Measuring $|\psi\rangle$ or $e^{i\varphi}|\psi\rangle$ will yield the same results, because the values r_0, r_1 are not modified by the factor $e^{i\varphi}$.
- The result of applying a quantum gate U to $|\psi\rangle$ or $e^{i\varphi}|\psi\rangle$ will not be affected by the constant $e^{i\varphi}$, as $U(e^{i\varphi}|\psi\rangle) = e^{i\varphi}(U|\psi\rangle)$.

Even though $|\psi\rangle$ and $e^{i\varphi}|\psi\rangle$ are not exactly the same, they behave in the same way, both when measuring or manipulating them. We refer to this by saying that $|\psi\rangle$ and $e^{i\varphi}|\psi\rangle$ are equivalent and differ from a global phase factor of $e^{i\varphi}$. This is known as **global phase invariance** and this fact allows us to simplify $|\psi\rangle$ by multiplying by $e^{-i\gamma_0}$. For convenience, we define $\varphi = \gamma_1 - \gamma_0$ and so we get:

$$|\psi\rangle = r_0 |0\rangle + r_1 e^{i\varphi} |1\rangle; \ r_0^2 + r_1^2 = 1; \ \varphi \in [0, 2\pi].$$

By defining θ such that $r_0 = \cos(\theta/2)$ (and so $r_1 = \sin(\theta/2)$) we obtain:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + \sin\left(\frac{\theta}{2}\right) e^{i\varphi} |1\rangle; \ \theta \in [0, \pi]; \ \varphi \in [0, 2\pi].$$

With this, we reach the conclusion that a state has these 2 degrees of freedom θ and φ. If we assign it to a point on the surface of a sphere with the map

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)e^{i\varphi}|1\rangle \longmapsto (x,y,z) = (\sin(\theta)\cos(\varphi), \sin(\theta)\sin(\varphi), \cos(\theta)),$$

we get the picture on Figure 1.

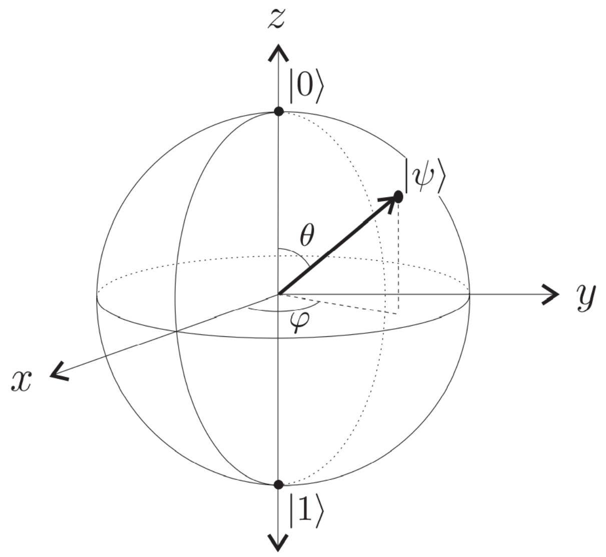


Figure 1. A pure state mapped to a point on the surface of a sphere - The **Bloch sphere**

With this model in mind, we can understand a measurement as placing two magnets on the north and south pole of the Bloch sphere with our eyes closed. The nearer the point is to the north magnet, the higher the chance it gets attracted to it, and the same can be said about the south magnet. When the point reaches one of the two magnets, we will feel it on our hands without having to look. Any successive measurements will yield the same result, because after the first measurement, we left the point on the pole of the magnet it got attracted to.

Applying quantum gates to a state can be seen as rotating the Bloch sphere. For example, applying the NOT gate X would be equivalent to a rotation of π radians on the y-axis, moving the point from one pole to another.

This fact of global phase invariance seen in the previous process explains why the T gate introduced in section 1 is also called the $\pi/4$ gate even though the rotations depicted in it are of $\pi/8$ radians. Just like the states $|\psi\rangle$ and $e^{i\varphi}|\psi\rangle$ are equivalent, the quantum gates U and $e^{i\varphi}U$ are equivalent as well because of the linearity of matrices. As shown before, the T gate has its name due to the fact that it introduces a **relative phase factor** of $\frac{\pi}{4}$, which in contrary to global phase factors, is indeed very relevant to computations, though not to measurements.

Although the Bloch Sphere is a useful visualization of pure states, the 3D restriction that it implies makes it difficult to generalize to n-qubit states. This is easily solved if we view the Bloch sphere as the complex projective line.

Recall that the **complex projective space** $\mathbb{P}^{N-1}$ is the set of lines in the complex space $\mathbb{C}^N$ passing through the origin. It may be described as the quotient

$$\mathbb{P}^{N-1} := \frac{\mathbb{C}^N - \{0\}}{z \sim \lambda z}, \ \lambda \in \mathbb{C}^*.$$

A point $z \in \mathbb{P}^{N-1}$ will be denoted by its homogeneous coordinates $z = [z_0 : \cdots : z_{N-1}]$ where, by definition, there is always at least an integer i such that $z_i \neq 0$, and for any $\lambda \in \mathbb{C}^*$ we have

$$[z_0 : \cdots : z_{N-1}] = [\lambda z_0 : \cdots : \lambda z_{N-1}].$$

One can see that there is a one-to-one correspondence between pure states of n qubits and points on the space $\mathbb{P}^{N-1}$. For any n-qubit state with amplitudes $\alpha_0, \ldots, \alpha_{N-1}$, we can associate it to the point with homogeneous coordinates $[\alpha_0 : \cdots : \alpha_{N-1}]$. On the other hand, for any point with homogeneous coordinates $[z_0 : \cdots : z_{N-1}]$, we can associate it to the state with amplitudes

$$\frac{z_0}{\sqrt{\sum_{k=0}^{N-1} |z_k|^2}}, \ldots, \frac{z_{N-1}}{\sqrt{\sum_{k=0}^{N-1} |z_k|^2}}.$$

The intuition behind this correspondence is that the relation $z \sim \lambda z$ that defines $\mathbb{P}^{N-1}$ is the equivalent to global phase invariance between pure states.

We refer to [Bal+19] for a deep exploration of the geometry behind quantum states and quantum physics in general.

2.3 Entangled qubits

Quantum entanglement is at the heart of quantum physics, with crucial roles in quantum information theory, superdense coding and quantum teleportation among others. We next review its main features. We refer to the exhaustive reviews [Hor+09; GT09; PV07] for the basic aspects of entanglement including its history, characterization, measurement, classification and applications.

Let us consider the case of 2 qubits. We can easily calculate the joint state if we know the states of both qubits, but can we do the inverse process?

Given a joint state $|\varphi\rangle = (a, b, c, d) \in \mathbb{C}^4$, we want to find $|\psi_1\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ and $|\psi_2\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle$ such that $|\psi_1 \psi_2\rangle = |\varphi\rangle$. This is equivalent to solving the system

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \iff \begin{cases} \alpha_0 \beta_0 = a \\ \alpha_0 \beta_1 = b \\ \alpha_1 \beta_0 = c \\ \alpha_1 \beta_1 = d \end{cases} \tag{1}$$

for $\alpha_0, \alpha_1, \beta_0$ and β_1, which does not always have a solution (when $(a, b, c, d) = (1/\sqrt{2}, 0, 0, 1/\sqrt{2})$, for instance). When this happens, the joint state of the qubits cannot be expressed as a single product of the states of each qubit as separate systems, as they are codependent. This phenomenon is called **entanglement**.

Definition 2.8. Let $|\varphi\rangle$ be the joint state of n qubits. The system they form is said to be **k-separable** when we can group the qubits into k groups with joint states $|\psi_1\rangle, \dots, |\psi_k\rangle$ such that

$$|\varphi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_k\rangle .$$

When a joint state of some qubits is 1-separable (meaning it cannot be decomposed into independent states of each system), they are said to be **entangled**, or that they form an entangled state.

Non-entangled qubits can be measured separately, and the output result of each qubit will not affect the other. If two qubits are entangled, though, measuring one of them will force the other to collapse onto a new state depending on the obtained result.

Example 2.9. Consider the previous non-separable joint state

$$({}^1\!/\!\sqrt{2}, 0, 0, {}^1\!/\!\sqrt{2}) = {}^1\!/\!\sqrt{2}\,|00\rangle + {}^1\!/\!\sqrt{2}\,|11\rangle .$$

If we measure the first qubit and we read the state $|0\rangle$, the other qubit will collapse onto the state $|0\rangle$ as well, because the probability of the joint state being in the state $|01\rangle$ is 0. The same analysis can be made if we read a $|1\rangle$ to predict that the other qubit will collapse onto the state $|1\rangle$.

This would be an example of a **partial measurement**, which is when we measure only some of the qubits from an entangled system.

One of the main differences between classical and quantum computing is actually the fact that qubits can become entangled, while bits cannot. After two qubits become entangled, no matter the physical distance between them, they will still be entangled (assuming they are kept in stable conditions). For instance, if we have two qubits in the joint state $({}^1\!/\!\sqrt{2}, 0, 0, {}^1\!/\!\sqrt{2})$ and one of them is taken to the Moon, as long as they are kept isolated and stable, by measuring the one on Earth we will know the state of the qubit in the Moon without measuring it, as entanglement does not break over great distances.

This can indeed be used for quantum communication (see Superdense Coding and Quantum Teleportation in section 5 of [KLM07]) or just for local use in our algorithms.

Remark 2.10. Two qubits in the joint state $|\Phi^+\rangle = ({}^1\!/\!\sqrt{2}, 0, 0, {}^1\!/\!\sqrt{2})$ of example 2.9 are called an **EPR pair**, which corresponds to a joint state of two qubits with maximum entanglement.

2.4 n-qubit quantum gates

Recall that a quantum gate acting on a 1-qubit state is modeled as a unitary matrix $U \in \mathcal{M}_2(\mathbb{C})$.

Definition 2.11. A **unitary matrix** U is a matrix with elements in $\mathbb{C}$ that satisfies $UU^\dagger = U^\dagger U = I$, where $U^\dagger$ is the Hermitean conjugate of U, which is sometimes written as U^H or $\overline{U}^*$.

Some useful properties of unitary matrices that we will use are:

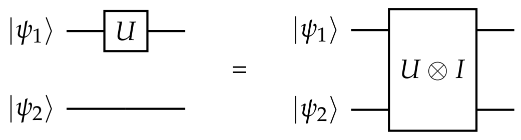
- U is a unitary matrix $\iff$ All eigenvalues λ of U satisfy $|\lambda| = 1$.
- U is a unitary matrix $\implies$ U is diagonalizable.
- U is a unitary matrix $\iff$ All eigenvectors of different eigenvalues are orthonormal.

When two qubits are entangled, we can no longer express the joint state as two separate states. We can still apply quantum gates to each of them independently, but because they are entangled, both states will evolve together. Just like joint states are described by the tensor product of separated states, the action of quantum gates on a joint state is also described by it as well.

Suppose we have a joint state $|\psi_1\rangle \otimes |\psi_2\rangle$ and we apply a quantum gate U to the first qubit. Implicitly, we are applying the identity gate I to the second qubit, so the joint state gets mapped to

$$(U|\psi_1\rangle) \otimes (I|\psi_2\rangle) = (U \otimes I)(|\psi_1\rangle \otimes |\psi_2\rangle) = (U \otimes I)|\psi_1\psi_2\rangle .$$

This transformation can be written as a circuit in the two following equivalent manners (but the first one is obviously much simpler than the second, and therefore better overall):



Example 2.12. Following from example 2.9, consider again the joint state $|\psi_1\psi_2\rangle = {}^1\!/\!\sqrt{2}\,|00\rangle + {}^1\!/\!\sqrt{2}\,|11\rangle$. Applying the Hadamard gate H to the first qubit will leave the system in the joint state

$$(H|\psi_1\rangle) \otimes (I|\psi_2\rangle) = (H \otimes I)|\psi_1\psi_2\rangle = \left(\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \otimes \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\right)\begin{pmatrix}1/\sqrt{2}\\ 0\\ 0\\ 1/\sqrt{2}\end{pmatrix} =$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}\\ 0\\ 0\\ 1/\sqrt{2}\end{pmatrix} = \begin{pmatrix}1/2\\ 1/2\\ 1/2\\ -1/2\end{pmatrix}.$$

It can easily be checked that after the Hadamard gate, the qubits are still entangled.

When a quantum gate transforms the states of non-entangled qubits into a joint entangled state, the quantum gate is said to be an **entangling gate**.

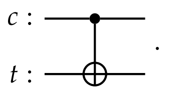
Definition 2.13. Let $q_1, ..., q_n$ be n qubits. An n-qubit quantum gate U is an **entangling gate** when there exists an n-separable joint state $|\psi\rangle$ of the qubits such that $U|\psi\rangle$ is not n-separable.

When a quantum gate U is an entangling gate, it cannot be decomposed into single-qubit quantum gates $A_1, \dots, A_n$ such that $U = A_1 \otimes \cdots \otimes A_n$, meaning that its action on the states of the qubits cannot be separated into independent transformations, one for each qubit. This does not mean that it entangles all qubits. Suppose U is a 4-qubit entangling gate. The gate could be 3-separable but only entangle two of the four qubits. It would still be an entangling gate, but at least we could rewrite it as $U = A_1 \otimes A_2 \otimes A_3$ where one of these three is a 2-qubit entangling gate.

One such notable example of $2-$qubit entangling gate is the **Controlled-Not** gate $CNOT$, with matrix form

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

and circuit representation



This gate performs the following transformation on a 2-qubit joint state:

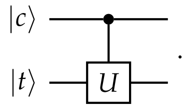
$$|00\rangle \longmapsto |00\rangle\,, \quad |01\rangle \longmapsto |01\rangle\,, \quad |10\rangle \longmapsto |11\rangle\,, \quad |11\rangle \longmapsto |10\rangle$$

We see that if the first qubit is in the state $|0\rangle$, nothing happens, but if the first qubit is in the state $|1\rangle$, the result is the same state but with the second qubit flipped. Flipping a qubit is the same as applying the NOT gate X, so the CNOT applies the X gate to the second qubit whenever the first qubit is in the state $|0\rangle$ (hence the name "Controlled Not"). This first qubit that decides if the X gate is applied is called the **control** qubit, and the second qubit, the one to apply the gate X to, is called the **target** qubit (labelled c and t on the circuit representation from above).

The $CNOT$ gate belongs to an important family of gates - the **controlled gates**. Given a quantum gate U, the controlled U gate $c{-}U$ is the gate that performs the following map:

$$|00\rangle \longmapsto |00\rangle\,, \quad |01\rangle \longmapsto |01\rangle\,, \quad |10\rangle \longmapsto (I \otimes U)\,|10\rangle\,, \quad |11\rangle \longmapsto (I \otimes U)\,|11\rangle\,,$$

that is, it applies a U gate onto the target qubit if the control qubit is in state $|1\rangle$, and does nothing if the control qubit is in state $|0\rangle$. The matrix form of the gate is $c{-}U = \begin{pmatrix} I & \mathbf{0} \\ \mathbf{0} & U \end{pmatrix}$, and its circuit representation is:



A controlled gate may have more than one control and target qubits. Constructing its matrix form follows an analogous process to the one for one control and target qubits. The resulting matrix will have $I_{2\times 2}$ blocks for the control qubits and a big block with the quantum gate to apply to the target qubits.

Example 2.14. The Controlled H gate $c-H$ has matrix form $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 1/\sqrt{2} \\ 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$

Remark 2.15. When the qubits that we are working with have labels to distinguish them (for example, suppose the control qubit is called c and the target qubit t), instead of "apply the gate $c - U$ with control qubit c and target qubit t", we will write "apply gate U on t controlled on c" or "apply the gate $U_{c \to t}$".

Remark 2.16. The fact that quantum gates are modeled as unitary matrices allows us to use properties of unitary matrices for designing circuits. A few notable properties are:

- Both the matrix and tensor products of unitary matrices return unitary matrices. Therefore, every quantum circuit can be summarized into a single quantum gate (to be reused for other quantum circuits).
- Unitary matrices are invertible, so for every quantum gate there exists another quantum gate that reverses its performed operations.
- Roots of unitary matrices (the n-th root of a matrix B is another matrix A such that $A^n = B$) are unitary matrices. That means that for each quantum gate, there exist quantum gates that perform a fraction of the operation it performs.

2.5 Von Neumann and projective measurements

In the previous chapter, the notion of measurement was explained conceptually as a probabilistic reading of a state with possible results $|0\rangle$ and $|1\rangle$. In the same way, multiple qubits can be measured at the same time to read a joint state consisting of multiple single $|0\rangle$ and $|1\rangle$ states. This is the standard version of measurement, but it is not the only version that can be implemented. In fact, it is a specific case of a generalized notion of measurement, the **Von Neumann measurement**.

Definition 2.17. Let $|\psi\rangle$ be an n-qubit state. A **Von Neumann measurement** of $|\psi\rangle$ with respect to an orthonormal basis $B = \{|\varphi_1\rangle, ..., |\varphi_N\rangle\}$ of C^N reads the state $|i\rangle$ with probability $p(i) = |\langle\varphi_i|\psi\rangle|^2$ and collapses the system onto the state $|\varphi_i\rangle$.

If we write the state as a linear combination of the basis B we get $|\psi\rangle = \sum_{j=1}^{N} \alpha_j |\varphi_j\rangle$ with $\alpha_j \in \mathbb{C}$. Recall that because B is an orthonormal basis, then:

$$\langle\varphi_i|\varphi_i\rangle = 1 \; \forall \; 1 \leq i \leq N; \;\; \langle\varphi_i|\varphi_j\rangle = 0 \; \forall \; 1 \leq i \neq j \leq N$$

and so, when we calculate $p(i)$ we find that:

$$p(i) = |\langle\varphi_i|\psi\rangle|^2 = \left| \langle\varphi_i| \sum_{j=1}^{N} \alpha_j |\varphi_j\rangle \right|^2 = \left| \sum_{j=1}^{N} \alpha_j \langle\varphi_i|\varphi_j\rangle \right|^2 = |\alpha_i|^2.$$

Remark 2.18. The standard version of measurement mentioned before is called **measurement in the computational basis** or standard measurement, because it is the specific case in which $B = \{|0\rangle, ..., |N-1\rangle\}$, that is, when B is the computational basis. Note that when measuring a single qubit in the state $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, then

$$\begin{aligned} p(0) &= |\langle 0|\psi\rangle|^2 = |\langle 0| (\alpha_0 |0\rangle + \alpha_1 |1\rangle)|^2 = |\alpha_0 \langle 0|0\rangle + \alpha_1 \langle 0|1\rangle|^2 = |\alpha_0|^2, \\ p(1) &= |\langle 1|\psi\rangle|^2 = |\langle 1| (\alpha_0 |0\rangle + \alpha_1 |1\rangle)|^2 = |\alpha_0 \langle 1|0\rangle + \alpha_1 \langle 1|1\rangle|^2 = |\alpha_1|^2. \end{aligned}$$

which coincides with the previous definition. Recall that $\{|0\rangle, |1\rangle\}$ is an orthonormal basis, so

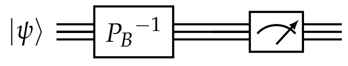
$$\begin{aligned} \langle 0|0\rangle = \langle 1|1\rangle = 1, \\ \langle 0|1\rangle = \langle 1|0\rangle = 0. \end{aligned}$$

Von Neumann measurements can be implemented using a device that performs standard measurements and a quantum gate P_B that performs a basis change to the basis B. Such quantum gate (and its inverse) can be implemented because the matrix form of P_B is unitary, as both the computational basis and B are orthonormal.

Let $|\psi\rangle = \sum_{j=1}^{N} \alpha_j |\varphi_j\rangle$. Applying P_B^{-1} to the state before the standard measurement maps the state to

$$|\psi\rangle \longmapsto P_B^{-1} |\psi\rangle = P_B^{-1} \sum_{j=1}^{N} \alpha_j |\varphi_j\rangle = \sum_{j=1}^{N} \alpha_j P_B^{-1} |\varphi_j\rangle = \sum_{j=1}^{N} \alpha_j |j\rangle .$$

Measuring this state will output $|i\rangle$ with probability $|\alpha_i|^2$, which is associated with the state $|\varphi_i\rangle$ on the basis B, meaning that we have indeed performed a measurement on the basis B. The following circuit implements such measurement:



After the measurement, the state will collapse onto $|i\rangle$. We can then optionally apply P_B to map the collapsed state to:

$$|i\rangle \longmapsto P_B |i\rangle = P_B |i\rangle = |\varphi_i\rangle$$

for extra computations if desired. Most of the time this step is skipped because after a measurement, those qubits are seldom used as the measurement is often the last operation performed on quantum algorithms.

The concept of measurement can be generalized even further to a more general version of measurements - **projective measurements**.

Definition 2.19. Let $|\psi\rangle$ be an n-qubit state. Let $P_1, ..., P_m \in \mathcal{M}_N(\mathbb{C})$ with $m \leq N$ be orthogonal projectors such that $I = \sum_{i=1}^{m} P_i$. A **projective measurement** of $|\psi\rangle$ with respect to $P_1, ..., P_m$ reads the state $|i\rangle$ with probability $p(i) = \langle\psi| P_i |\psi\rangle$ and collapses the system onto the state

$$\frac{P_i |\psi\rangle}{\sqrt{p(i)}}.$$

Recall that the condition of $P_1, ..., P_m$ being orthogonal projectors translates to $P_i^2 = P_i$ (projector) and $P_i^\dagger = P_i,\ P_iP_j = \mathbf{0}\ \forall\ 1 \leq i \neq j \leq m$ (orthogonal). Also, the condition $I = \sum_{i=1}^m P_i$ ensures that $\sum_{i=1}^m p(i) = 1$, because

$$\sum_{i=1}^m p(i) = \sum_{i=1}^m \langle\psi| P_i |\psi\rangle = \langle\psi| \left(\sum_{i=1}^m P_i\right) |\psi\rangle = \langle\psi| I |\psi\rangle = \langle\psi|\psi\rangle = 1.$$

Remark 2.20. The Von Neumann measurement is a special case of projective measurement where $P_i = |\varphi_i\rangle\langle\varphi_i|$. This fact is seen through the following equalities:

$$p(i) = \langle\psi| P_i |\psi\rangle = \langle\psi|\varphi_i\rangle \langle\varphi_i|\psi\rangle = \langle\varphi_i|\psi\rangle^* \langle\varphi_i|\psi\rangle = |\langle\varphi_i|\psi\rangle|^2,$$

which coincide with the previous definition of Von Neumann measurements. In this case, we have (for n qubits) N projectors with $\operatorname{rank}(P_i) = 1$, one for each vector of the orthonormal basis.

A projective measurement is more general in the way that there is no need to have as many orthogonal projectors as different elemental states and provide one of the N possible readings. As long as the condition $I = \sum_{i=1}^m P_i$ is satisfied, and our projectors are orthogonal, we can choose as many as we want and however we want, even with different ranks.

Physically implementing specific projective measurements is not a trivial matter and requires proper analysis. The notion of projective measurements is more interesting on a theoretical/simulation level rather than physical/implementation.

2.6 Pure states and mixed states

Until now we have referred to the states of qubits as pure states, and this particular distinction hints to the existence of some kind of non-pure - mixed states.

Definition 2.21. An n-qubit **mixed state** is an ensemble

$$|\psi\rangle = \{(|\psi_1\rangle, p_1), \dots, (|\psi_r\rangle, p_r)\},$$

where $|\psi_1\rangle, \dots, |\psi_r\rangle$ are n-qubit pure states and $p_1, \dots, p_r \in [0, 1]$ are probabilities such that $\sum_{j=1}^r p_j = 1$.

Indeed, qubits can sometimes be in what are called **mixed states**, which is a probabilistic mix of pure states. Mixed states are a tool that we can use to analyze entangled qubits separately.

Example 2.22. Consider two qubits with joint state

$$|\psi_1\psi_2\rangle = (1/\sqrt{2}, 0, 1/2, 1/2) = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{2} |10\rangle + \frac{1}{2} |11\rangle .$$

Measuring them together will output:

$$\begin{cases} |00\rangle \text{ with probability } 1/2 \\ |10\rangle \text{ with probability } 1/4 \\ |11\rangle \text{ with probability } 1/4 \end{cases} .$$

If we only only measured the first qubit, we would read $|0\rangle$ or $|1\rangle$ with a $50/50$ chance. If we read $|0\rangle$ on the first qubit, the second would collapse onto the state $|0\rangle$, and if we read $|1\rangle$ instead, the second would collapse onto state $1/\sqrt{2}\,|0\rangle + 1/\sqrt{2}\,|1\rangle$. Without measuring the first qubit, we have no idea of which of the two options is the correct for the second qubit, but we know the probabilities of each option.

We can therefore think of the second qubit being in the mixed state

$$|\psi_2\rangle = \left\{ \left(|0\rangle, \frac{1}{2} \right), \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}, \frac{1}{2} \right) \right\}.$$

Remark 2.23. 1-qubit mixed states can be geometrically interpreted as points on the interior of the Block Sphere.

Mixed states give us a way of working separately with sub-systems of an entangled system, as we can perform unitary operations and measurements on the ensembles. If we consider the mixed state $|\psi\rangle = \{(|\psi_1\rangle, p_1), \dots, (|\psi_r\rangle, p_r)\}$:

- Applying a quantum gate to the mixed state amounts to applying the gate to each pure state of the ensemble, so
$$U|\psi\rangle = \{(U|\psi_1\rangle, p_1), \dots, (U|\psi_r\rangle, p_r)\}.$$
- Recall in subsection 2.5 that a standard measurement of a pure state $|\varphi\rangle$ reads $|i\rangle$ with probability $p(i) = |\langle i|\varphi\rangle|^2$ for $i = 0, \dots, n-1$. To factor all pure states in the ensemble, we calculate the weighted sum of all these probabilities:
$$p(i) = p_1|\langle i|\psi_1\rangle|^2 + \cdots + p_r|\langle i|\psi_r\rangle|^2.$$

While it is a valid solution, if the system that we are studying is entangled to many other different systems, (when r is large), we will have a lot of pure states and probabilities in our ensemble and calculations will become cumbersome. For this reason, an alternative is used.

Definition 2.24. Let $|\psi\rangle = \{(|\psi_1\rangle, p_1), \dots, (|\psi_r\rangle, p_r)\}$ be an n-qubit mixed state. We define its **density operator** is:

$$\rho = \sum_{j=1}^{r} p_j\, |\psi_j\rangle\langle\psi_j|\,.$$

For n-qubit mixed states, the density operator ρ will be an $N \times N$ matrix, so we will refer to it as the **density matrix**.

Example 2.25. In example 2.22, the density matrix of $|\psi_2\rangle$ would be:

$$\rho = \frac{1}{2}\begin{pmatrix} 1 \\ 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix} = \begin{pmatrix} 3/4 & 1/4 \\ 1/4 & 1/4 \end{pmatrix}.$$

Instead of working with the ensemble of pure states, the density matrix is calculated and used to compute transformations with quantum gates and measurements, as the computations are far simpler.

- After a quantum gate U, the density matrix of the mixed state will be

$$\sum_{i=1}^{r} p_i(U\left|\psi_i\right\rangle)(\left\langle\psi_i\right| U^\dagger) = \sum_{i=1}^{r} p_i U \left|\psi_i\right\rangle\!\left\langle\psi_i\right| U^\dagger = U\left(\sum_{i=1}^{r} p_i \left|\psi_i\right\rangle\!\left\langle\psi_i\right|\right) U^\dagger = U\rho U^\dagger.$$

- The previous formula for $p(i)$ when measuring the mixed state can be restated as:

$$\begin{gathered}
p(i) = \\
p_1|\langle i|\psi_1\rangle|^2 + \cdots + p_r|\langle i|\psi_r\rangle|^2 = \\
p_1\,\langle i|\psi_1\rangle\,\langle i|\psi_1\rangle^* + \cdots + p_r\,\langle i|\psi_r\rangle\,\langle i|\psi_r\rangle^* = \\
p_1\,\langle i|\psi_1\rangle\,\langle \psi_1|i\rangle + \cdots + p_r\,\langle i|\psi_r\rangle\,\langle \psi_r|i\rangle = \\
|i\rangle\,(p_1\,|\psi_1\rangle\!\langle\psi_1|)\,\langle i| + \cdots + |i\rangle\,(p_r\,|\psi_r\rangle\!\langle\psi_r|)\,\langle i| = \\
|i\rangle\,(p_1\,|\psi_1\rangle\!\langle\psi_1| + \cdots + p_r\,|\psi_r\rangle\!\langle\psi_r|)\,\langle i| = \\
|i\rangle\,\rho\,\langle i|\,.
\end{gathered}$$

Example 2.26. Let us finish example 2.25 by performing a standard measurement with the density matrix:

$$\begin{aligned}
p(0) &= \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}3/4 & 1/4\\ 1/4 & 1/4\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{3}{4},\\
p(1) &= \begin{pmatrix}0 & 1\end{pmatrix}\begin{pmatrix}3/4 & 1/4\\ 1/4 & 1/4\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{4}.
\end{aligned}$$

Remark 2.27. Note that $|i\rangle\,\rho\,\langle i|$ is a real number, so we can restate it as $\mathrm{Tr}\,(|i\rangle\,\rho\,\langle i|)$, where Tr denotes the trace of a matrix. From there, using the properties of the matrix trace we get:

$$\mathrm{Tr}\,(|i\rangle\,\rho\,\langle i|) = \mathrm{Tr}\,(|i\rangle\!\langle i|\,\rho)\,.$$

The density matrix also gives us a way to compute measurements on some of the qubits from an entangled system. This is known as performing a **partial measurement**. From the entire system's density matrix, we obtain a smaller density matrix of the qubits we want to measure, and then use said smaller density matrix for measurements. The smaller density matrix is obtained through the **partial trace**.

Definition 2.28. Let ρ be the density matrix of the composite system consisting of system A in state $|a\rangle$ and system B in state $|b\rangle$, that is, $\rho = |a\rangle\!\langle a| \otimes |b\rangle\!\langle b|$. The **partial trace** over system A is defined by:

$$\mathrm{Tr}_A\,(\rho) = |b\rangle\!\langle b|\,\mathrm{Tr}\,(|a\rangle\!\langle a|)\,,$$

which is the density matrix of the qubits from system B isolated from the qubits from system A. The partial trace over system B is defined analogously.

We provide an example of this procedure in appendix A.1.

Calculating the density matrix of a mixed state is a direct process, but going back to recover the ensemble is not a trivial matter. For any density matrix, the existence of a corresponding mixed state is guaranteed, but it does not necessarily have to be unique. Computation and measurement-wise, those correspondent mixed states will be indistinguishable.

2.7 Universal set of quantum gates

In classical computation, most of the circuits that perform operations on many bits are not manufactured from scratch and are instead created using smaller circuits or gates (take for instance multiplexors, which are implemented using *AND* gates). A set of logic gates with which all the other logic gates (and therefore any other bigger circuit) can be implemented is called a **universal set of logic gates**.

It is of much interest to find small and simple universal sets of logic gates because understanding them allows us to minimize the number of gates that will have to be physically implemented in order to build any logic circuit. In quantum computing this interest is emphasized when we consider the fact that for any given number of qubits in a quantum system there exist an infinite number of unitary operators, way more than the finite number of logic gates.

But can these unitary operators be physically implemented with perfect accuracy? Until now, we have made this assumption without giving it much thought, but the infinite number of them puts it on doubt. Indeed, in real life, physical implementations of unitary operators carry a small accuracy error.

Definition 2.29. Let U be a desired unitary transformation, V be an approximation. The **error** in the approximation is defined as:

$$E(U,V) = \max_{|\psi\rangle} \|(U-V)\,|\psi\rangle\| \,.$$

If in classical computation the requirement for a universal set of logic gates was to simply implement any other gate, in quantum computing, the requirement will be to implement any unitary operation with bounded error or desired accuracy.

Definition 2.30. A **universal set of quantum gates** is a set of quantum gates with which any unitary transformation can be approximated with error bounded by $\epsilon \geq 1$ for any arbitrary small value of ϵ.

For 1-qubit unitary operations, we have the following:

Theorem 2.31. *The set $\{H,T\}$ is universal for 1-qubit gates.*

Example 2.32. The Pauli gates X,Y,Z have the following decomposition, up to a global phase factor:

$$X = HT^4H, \quad Y = HT^4HT^4, \quad Z = T^4.$$

And for n-qubit unitary operations, by using the following universality:

Theorem 2.33. *A set composed of any 2-qubit entangling gate, together with all 1-qubit gates, is universal.*

We can finally deduce that:

Theorem 2.34. *The set $\mathcal{G} = \{H,T,CNOT\}$ is universal for n-qubit gates.*

Example 2.35. The **Toffoli** gates are the n-qubit generalization of $CNOT$ gates. An n-qubit Toffoli gate has $n-1$ control qubits and 1 target qubit. If all the control qubits are in state $|1\rangle$, the Toffoli gate flips the state of the target qubit, and does nothing otherwise. In page 182 of [NC00] we can see how it can be implemented using only H,T and $CNOT$ gates.

So this means only accurate implementations of these 3 gates are necessary to build any quantum circuit with any desired precision, though this does not mean that only using these gates is the most efficient procedure. Using pre-made and more accurate combinations of H, T and $CNOT$ as extra gates in $\mathcal{G}$ (as long as they can be physically implemented) can reduce the total number of used gates for approximation, which is interesting due to the fact that there is a small error chance with each use of a quantum gate.

It is very important to be assured that any quantum circuit that we theoretically devise can be brought to reality through an indefinitely accurate physical implementation, but it is also important to somehow quantify how expensive this implementation will be (and by expensive we mean how many gates we will need). For 1-qubit gates, the **Solovay-Kitaev Theorem** quantifies this cost, provided a specific condition is satisfied.

Theorem 2.36 (Solovay-Kitaev Theorem)**.** *If $\mathcal{G}$ is a universal set of 1-qubit quantum gates and for any gate $g \in \mathcal{G}$ its inverse g^{-1} can be exactly implemented with a finite number of gates from $\mathcal{G}$, then it is assured that any 1-qubit gate can be implemented with error at most ϵ with $\mathcal{O}(\log^c(1/\epsilon))$ gates from $\mathcal{G}$, where c is a positive constant.*

Remark 2.37. Most of the time the constant c lies in the interval $[2,4]$, depending on the implementation, but it has been proven that with an optimal gate set, it can get as low as 1. More information about the Solovay-Kitaev Theorem can be found in Appendix 3 of [NC00].

It is easy to check that

$$H^{-1} = H; \quad T^{-1} = T^7,$$

so our universal set from before satisfies the theorem's condition and therefore we can use this asymptotic quantification to gauge the trade-off between number of gates and accuracy.

Having made this analysis for quantum gates, we now turn to quantum circuits. To implement quantum circuits designed on paper, for each gate we will use an approximation of error lower or equal than ϵ. To calculate the total error of the circuit, we will use the following fact:

$$E(U_m \cdots U_1, V_m \cdots V_1) \leq E(U_1, V_1) + \cdots + E(U_m, V_m),$$

deduced from definition 2.29, which tells us that if a quantum circuit requires m quantum gates and we use approximations of error bounded by ϵ, then the error of the circuit will be bounded by $m \cdot \epsilon$.

3 Basic quantum algorithms

In this section we will focus on a few basic quantum algorithms that will be needed for the next section. These are the Phase Estimation, Eigenvalue Estimation and Grover Search algorithms. For each algorithm, we will give a detailed explanation of the algorithm, a construction of the quantum circuit, analyze their output, and then showcase a few executions with custom implementations running on Qiskit's QASM Simulator.

We will assume that we have absolutely accurate implementations of all the quantum gates used, so no errors associated to physical imperfections can occur, and that the quantum computer that the algorithms will run on is ideal and has no noise.

Also, until now the following notation was used to describe the elemental states from the computational basis:

$$|0\rangle, |1\rangle, \ldots, |n-1\rangle = \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}, \ldots, \begin{pmatrix}0\\0\\\vdots\\1\end{pmatrix}.$$

From now on, this notation will be used to describe the decimal representation of the n-bit string states. For instance, if $n = 8$, we will refer to the state

$$|00001101\rangle = |0\rangle |0\rangle |0\rangle |0\rangle |1\rangle |1\rangle |0\rangle |1\rangle$$

as $|13\rangle$, and vice versa.

3.1 The Phase Estimation algorithm

The **Phase Estimation algorithm** is a quantum algorithm that outputs an estimation of the phase of a quantum register. It is a fundamental and crucial algorithm as it is used as a subroutine in many other quantum algorithms that work with phases (for instance, the famous Shor's algorithm). To begin, let us detail more what we refer to as the "phase of a quantum register".

Recall that the 1-qubit Hadamard gate H performs the following transformation:

$$|0\rangle \longmapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \quad |1\rangle \longmapsto \frac{|0\rangle - |1\rangle}{\sqrt{2}},$$

which can be abbreviated as:

$$|x\rangle \longmapsto \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}.$$

What it does is essentially encode the value x as the phase of the qubit, which is the value $\omega \in [0, 1)$ in the expression:

$$\frac{|0\rangle + e^{2\pi i\omega} |1\rangle}{\sqrt{2}}.$$

If $x = 0$, the value $\omega = 0$ will be encoded, and if $x = 1$, the value $\omega = 1/2$ will be encoded instead. To go back and recover the original value x we can just apply H again, as the gate satisfies $H^{-1} = H$.

We can take two qubits instead of one and apply an H gate to each of them, which will result in the following map:

$$|x_1 x_2\rangle \longmapsto H|x_1\rangle \otimes H|x_2\rangle = \frac{|0\rangle + (-1)^{x_1}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + (-1)^{x_2}|1\rangle}{\sqrt{2}} =$$

$$\frac{|00\rangle + (-1)^{x_2}|01\rangle + (-1)^{x_1}|10\rangle + (-1)^{x_1 \cdot x_2}|11\rangle}{2} =$$

$$\frac{(-1)^{(0\,0)\cdot\binom{x_1}{x_2}}|00\rangle + (-1)^{(0\,1)\cdot\binom{x_1}{x_2}}|01\rangle + (-1)^{(1\,0)\cdot\binom{x_1}{x_2}}|10\rangle + (-1)^{(1\,1)\cdot\binom{x_1}{x_2}}|11\rangle}{2},$$

where the symbol $\cdot$ denotes the usual scalar product. By defining $\mathbf{x} := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, the result can be elegantly expressed as:

$$\sum_{\mathbf{y} \in \{0,1\}^2} (-1)^{\mathbf{y}\cdot\mathbf{x}} |y\rangle .$$

By similar calculations it can be shown that n Hadamard gates perform the following map on the joint state of n qubits:

$$|x_1 x_2 \dots x_n\rangle = |\mathbf{x}\rangle \longmapsto H^{\otimes n}|\mathbf{x}\rangle = \sum_{\mathbf{y} \in \{0,1\}^n} (-1)^{\mathbf{y}\cdot\mathbf{x}} |\mathbf{y}\rangle ; \; \mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

The phases $\mathbf{y} \cdot \mathbf{x}$ are a specific subset of phases, but in general, an encoded phase ω will produce a state of the form

$$\sum_{y=0}^{2^n-1} e^{2\pi i \omega y} |y\rangle ; \; \omega \in [0,1),$$

which coincides with the 1-qubit case. When the register is on this state, we say that is has the phase ω encoded. The problem of retrieving an estimation of ω is called the **Phase Estimation problem**, the solution of which is the **Quantum Fourier Transform**, abbreviated as the QFT. By giving the solution to the problem we will construct the QFT.

The algorithm makes use of an n-qubit quantum register with labelled qubits $q_1, \dots, q_n$, starting with the phase encoded and from which we will measure an estimation of the phase at the end.

Note that ω can be expressed in binary form as $\omega = 0.x_1 x_2 \dots$ with an indefinite number of digits $x_1, x_2, \dots \in \{0,1\}$. It can be shown that:

$$\sum_{\mathbf{y}\in\{0,1\}^n} e^{2\pi i(0.x_1x_2\ldots)y}\,|y\rangle =$$

$$\frac{|0\rangle + e^{2\pi i 2^{n-1}(0.x_1x_2\ldots)}\,|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i 2^{n-2}(0.x_1x_2\ldots)}\,|1\rangle}{\sqrt{2}} \otimes\cdots\otimes \frac{|0\rangle + e^{2\pi i(0.x_1x_2\ldots)}\,|1\rangle}{\sqrt{2}} =$$

$$\frac{|0\rangle + e^{2\pi i(0.x_nx_{n+1}\ldots)}\,|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i(0.x_{n-1}x_n\ldots)}\,|1\rangle}{\sqrt{2}} \otimes\cdots\otimes \frac{|0\rangle + e^{2\pi i(0.x_1x_2\ldots)}\,|1\rangle}{\sqrt{2}}$$

Now suppose that the phase ω could be expressed exactly with n digits, meaning $\omega = 0.x_1x_2\ldots x_n$. If that were the case, then the previous expression would amount to:

$$\frac{|0\rangle + e^{2\pi i(0.x_n)}\,|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i(0.x_{n-1}x_n)}\,|1\rangle}{\sqrt{2}} \otimes\cdots\otimes \frac{|0\rangle + e^{2\pi i(0.x_1x_2\ldots x_n)}\,|1\rangle}{\sqrt{2}}$$

By applying an H gate on the first qubit, we would obtain x_n. To obtain x_{n-1}, we would have to eliminate x_n from the second qubit before applying an H gate. To achieve this, consider the following gate:

$$R_k = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{2\pi i}{2^k}}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{2\pi i(0.0\ldots1)}\end{pmatrix},$$

where this number $0.0\ldots1$ with a 1 on the k-th digit is just $1/2^k$ expressed in binary form. We are interested in its inverse, which is:

$$R_k^{-1} = \begin{pmatrix}1 & 0\\ 0 & e^{-\frac{2\pi i}{2^k}}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{2\pi i(-0.0\ldots1)}\end{pmatrix}.$$

If $x_n = 0$, then we do not need to eliminate x_n, as $0.x_{n-1}x_n = 0.x_{n-1}$. But if $x_n = 1$, by applying the gate R_2^{-1}, we would get:

$$R_2^{-1}\frac{|0\rangle + e^{2\pi i(0.x_{n-1}1)}\,|1\rangle}{\sqrt{2}} = \frac{|0\rangle + e^{2\pi i(0.x_{n-1}1-0.01)}\,|1\rangle}{\sqrt{2}} = \frac{|0\rangle + e^{2\pi i(0.x_{n-1})}\,|1\rangle}{\sqrt{2}},$$

from which we can now retrieve x_{n-1} by applying an H gate. Basically, we want to apply R_2^{-1} when the first qubit is $|1\rangle$, and do nothing otherwise. To do that, we can apply the controlled gate ${R_2^{-1}}_{1\to2}$.

Retrieving x_{n-2} from the third qubit after x_n and x_{n-1} have been retrieved from the two first qubits would follow an analogous procedure: apply ${R_3^{-1}}_{1\to3}$ and ${R_2^{-1}}_{2\to3}$, and so on for the rest of the qubits. We show the associated quantum circuit in Figure 2.

The circuit that performs the phase encoding process is the QFT. The phase decoding circuit, the one that solves this problem, is the inverse of the QFT (QFT^{-1}). After applying the QFT^{-1}, we can simply measure the qubits in the computational basis to obtain the digits $x_1, x_2, \ldots, x_n$ from the phase $\omega = 0.x_1x_2\ldots x_n$.

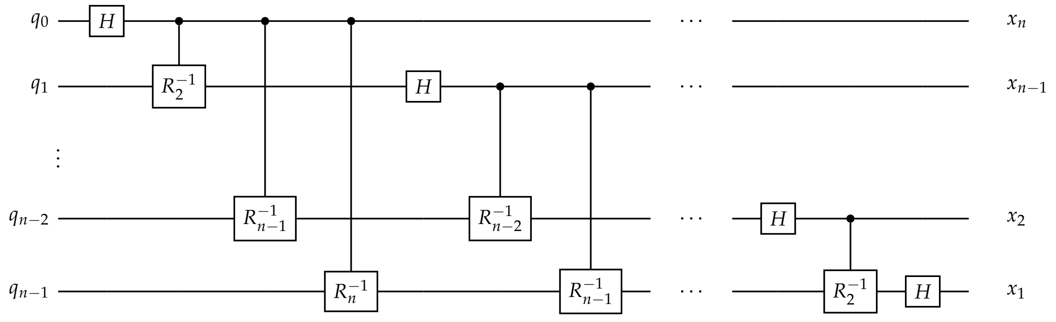


Figure 2. The solution circuit to the Phase Estimation problem: the QFT^{-1}

Most of the time, the phase ω will not exactly be $0.x_1x_2 \dots x_n$. What we will obtain then is a superposition of n-digit binary approximations of ω, with increasing amplitude the closer the approximation is to ω. After the measurement, we will obtain one of them at random. One can visualize such probabilities in Figure 3.

Lemma 3.1. *The probability of obtaining each of the n-digit binary approximations x for the phase ω after running the QFT^{-1} and performing a measurement in the computational basis is:*

$$p(x) = \begin{cases} \frac{1}{2^{2n}} \frac{sin^2(\pi(2^n\omega - x))}{sin^2(\pi(\omega - x/2^n))}, & if\, x \neq 2^n\omega \\ 1, & if\, x = 2^n\omega \end{cases}.$$

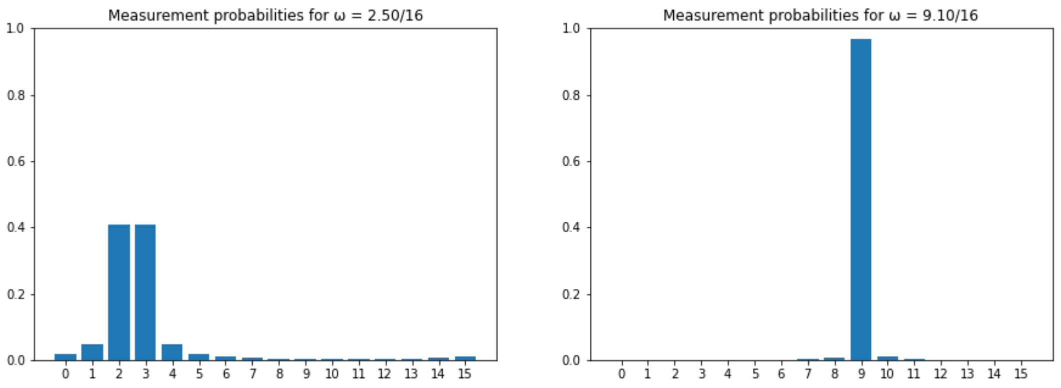


Figure 3. Some examples of measurement probabilities of the QFT^{-1}

Theorem 3.2. *Let $\omega \in [0,1)$ be the phase to estimate and x the closest n-digit binary approximation of ω. After applying the Phase Estimation algorithm, the probability of measuring x is lower-bounded by $4/\pi^2 \approx 0.405284$.*

Theorem 3.3. *Let $\omega \in [0,1)$ be the phase to estimate and x, x' the two closest n-digit binary approximations of ω. After applying the Phase Estimation algorithm, the probability of measuring x or x' is lower-bounded by $8/\pi^2 \approx 0.810569$.*

Two possible workarounds to ensure a better approximation of the phase are:

- Increasing the number of qubits n, which will sometimes not be possible for physical reasons.
- Running the QFT^{-1} more than once to sample many approximations, which may be non-viable when preparing the state that we want to read the phase from is an expensive process.

In summary, for the elemental states $|x\rangle = |0\rangle, \ldots, |2^n - 1\rangle$, we can summarise the action of the QFT and the QFT^{-1} as:

$$|x\rangle \xmapsto{\mathrm{QFT}} \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i y \frac{x}{2^n}} |y\rangle \xmapsto{\mathrm{QFT}^{-1}} |x\rangle$$

Encoding and decoding states as phases into qubits can prove very useful for many algorithms, in particular for those that rely on some classical arithmetic operations, like additions, substractions or multiplications.

The QFT uses the same register q in state $|x\rangle$ as control and target of the rotation gates R_k and R_k^{-1} to encode the phase $\omega = x/2^n$. With two different registers a and b on states $|x_a\rangle, |x_b\rangle$, respectively, by using one of them for control and the other as the target, the phase $\omega_a \pm \omega_b = x_a \pm x_b/2^n$ can be encoded. Using the QFT^{-1} on the target register will then yield the state $|x_a \pm x_b\rangle$.

This can be used to devise circuits that can perform arithmetic operations. This is explored in more depth on [Şah20]. In section 4, we will see a very similar technique used on the Grover Adaptive Search algorithm for cost function computation.

Also, one can see that the rotation gates R_k used for the QFT have very small angles. This poses the question: Can these gates be actually implemented physically? Not only in the construction of the QFT, but in many other quantum circuits, the availability of these precise rotational gates is crucial and a necessity. One can refer to [McK+16] where implementations of these rotational gates are studied.

The Phase Estimation algorithm with a custom version of the QFT can be tested on `PE_Tester.ipynb`, hosted on the GitHub repository at [Ort21].

3.2 Grover's Quantum Search algorithm

To begin, let $f : \{0,1\}^n \longrightarrow \{0,1\}$ be an unknown function and consider the problem of finding a string $x \in \{0,1\}^n$ such that $f(x) = 1$. Suppose we cannot abuse the structure of f. If this is the case, a classical computer has no option but to try all the different n-bit strings one by one until one of them gives out a 1. The computational cost of this problem scales exponentially as n grows large enough (in terms of the times f has to be evaluated).

Grover Search is a quantum algorithm that can solve this problem evaluating f a reduced number of times. It works by setting all the n-bit string states in superposition on a register, evaluating f on all of them at the same time, and then amplifying the amplitude of the string states that give out a 1 to maximize the probability of reading them when measuring the register. In this section we will thoroughly explain and analyze this algorithm.

The algorithm makes use of two quantum registers:

- An n-qubit register of labelled qubits $q_1, \ldots, q_n$ initially set in the state $|0\rangle_n$, that can store the superposition of all the n-bit string states $|0\rangle_n, |1\rangle_n, \ldots, |2^n - 1\rangle_n$.
- An ancillary 1-qubit register with label a set in the state $|-\rangle := {}^{1}/_{\sqrt{2}}\,|0\rangle - {}^{1}/_{\sqrt{2}}\,|1\rangle)$ throughout the entire algorithm. This can easily be achieved with initial state $|0\rangle$ and X and H gates:

$$|0\rangle \xmapsto{X} |1\rangle \xmapsto{H} \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle$$

The algorithm also requires the construction of the following gates:

- A **state preparation operator** A that produces an equal superposition of all possible n-bit string states. Note that this does not necessarily mean that we want all the string states in superposition, as some of them may be unneeded due to some problem specific reasons.

  Let $I \subseteq \{0, 1, ..., 2^n - 1\}$ be all the possible states. The action of operator A must be:

$$A : |0\rangle_n \longmapsto \frac{1}{\sqrt{|I|}} \sum_{x \in I} |x\rangle_n .$$

  Note that when $I = \{0, 1, ..., 2^n - 1\}$, which is when we want all the states, we can simply choose $A = H^{\otimes n}$. $H^{\otimes n}$ is often chosen for simplicity. We recall the action of $H^{\otimes n}$:

$$H^{\otimes n} : |0\rangle_n \longmapsto \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n - 1} |x\rangle_n .$$

  We will study this concrete case, as we will make some assumptions that require that $I = \{0, 1, ..., 2^n - 1\}$, so from now on, $A = H^{\otimes n}$, which has the particularity that $H^{\otimes n \dagger} = H^{\otimes n}$.

- An **oracle** U_f that applies the function f from the problem. The action of U_f must be:

$$U_f : |x\rangle_n |b\rangle \mapsto |x\rangle_n |b \oplus f(x)\rangle .$$

  The construction of this operator is not trivial and depends on the concrete problem at hand. Note that this operator acts on both registers. The reason the ancillary register is set to state $|-\rangle$ is because of the action performed by U_f:

$$U_f : |x\rangle_n |-\rangle \longmapsto U_f |x\rangle_n |-\rangle = U_f |x\rangle_n \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |x\rangle_n \frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} =$$
$$\begin{cases} |x\rangle_n \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |x\rangle_n |-\rangle & \text{if } f(x) = 0 \\ |x\rangle_n \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -|x\rangle_n |-\rangle & \text{if } f(x) = 1 \end{cases} .$$

If $f(x) = 0$, it does nothing, and if $f(x) = 1$, it performs a phase shift of -1 on the state.

- A **phase shift operator** D, the action of which is:

$$D : \begin{cases} |x\rangle_n \mapsto - |x\rangle_n & \text{if } x \neq 0 \\ |0\rangle_n \mapsto |0\rangle_n \end{cases},$$

  performing a phase shift of -1 on all states except $|0\rangle_n$. This is equivalent to shifting the phase of only the $|0\rangle_n$ state and leaving the rest untouched. This operator is often called the **Grover diffusion operator** or the **Grover diffuser**. The matrix corresponding to this n-qubit operator is a $2^n \times 2^n$ diagonal matrix where all elements are 1s except the first, which has value -1. One can refer to [SBM04] to see how these operators can be implemented without the usage of n-qubit gates.

With these, we construct the **Grover Iterate** $G = ADA^\dagger U_f$. Starting with initial state

$$(A \otimes HX) |0\rangle_n |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in I} |x\rangle_n |-\rangle ,$$

we repeatedly apply G a number of times. After that, by measuring the first register on the computational basis, we will read one of the strings x that satisfies $f(x) = 1$ with high probability (a "good" string), if there is any. The following analysis will show why that is true.

When applying the Grover Iterate, first U_f is applied. Let I_g, I_b ("g" for "good", "b" for "bad") be a partition of the possible states I, where:

$$I_g = \{x \in I \mid f(x) = 1\}, \quad I_b = \{x \in I \mid f(x) = 0\}.$$

For convenience, we denote $t_g, t_b := |I_g|, |I_b|$, respectively. We can separate the superposition of all possible states into two superpositions of good and bad states:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in I} |x\rangle_n |-\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n |-\rangle + \frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n |-\rangle .$$

U_f will only affect the superposition of good states:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n |-\rangle + \frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n |-\rangle \overset{U_f}{\longmapsto} \frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n |-\rangle - \frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n |-\rangle .$$

From now on we will ignore the ancilla register on state $|-\rangle$ as the remaining operators do not act on such register, so the previous result reduces to:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n - \frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n = \sum_{x \in I} \alpha_x |x\rangle_n , \quad \begin{cases} \alpha_x = \frac{1}{\sqrt{2^n}} & \text{if } f(x) = 0 \\ \alpha_x = -\frac{1}{\sqrt{2^n}} & \text{if } f(x) = 1 \end{cases}$$

Now out of all 2^n different states we have t_g good states with amplitude $-1/\sqrt{2^n}$ and t_b bad states with amplitude $1/\sqrt{2^n}$. On this state, the operators $ADA^\dagger$ perform a special kind of symmetry operation over the amplitudes of the states about their mean, given by the following lemma:

Lemma 3.4. *Let*

$$\mu = \frac{1}{t}\sum_{x\in I}\alpha_x$$

be the mean of all amplitudes of the state

$$\sum_{x\in I}\alpha_x \left|x\right\rangle_n .$$

The operators $ADA^\dagger$ act on the amplitudes α_x performing the following map:

$$\alpha_x \longmapsto 2\mu - \alpha_x.$$

Lemma 3.4 can be visualized in Figure 4. The red line marks the mean of the amplitudes of the states after applying the oracle U_f, which is the value through which the symmetry has been performed.

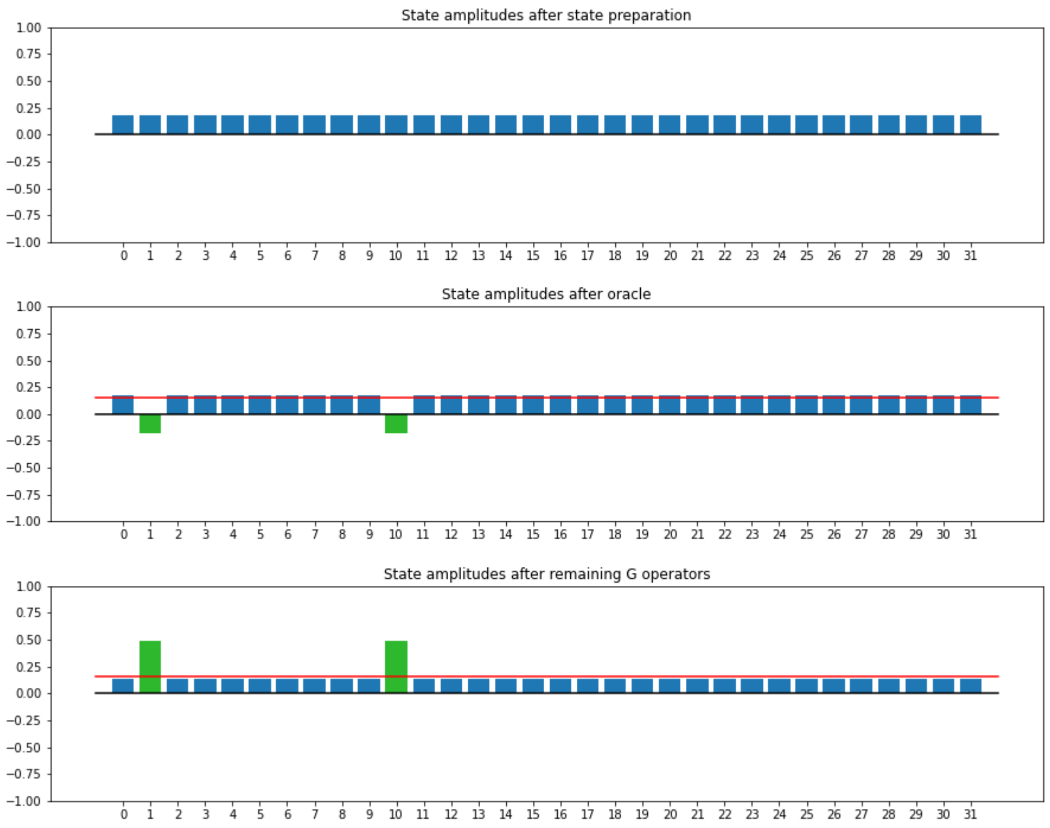


Figure 4. Amplitudes during the process of one G application ($n = 5, t_g = 2, t_b = 30$)

Contrary to what one might intuitively think, as this it what happens with most classical iterative numerical methods, it is not true that the more times we apply G then the higher the chances of reading a good string x. For each problem, there is an optimal number of times G must be applied. If that number is surpassed or not reached, the probability of reading a good string will be reduced.

The number of times the Grover Iterate G must be applied depends on t_g, the number of good strings. Further analysis shows that:

Theorem 3.5. *Let*

$$\sin^2(\theta) = \frac{t_g}{2^n} \implies \theta = \arcsin\left(\sqrt{\frac{t_g}{2^n}}\right).$$

Then the probability of reading a good string after k applications of the Grover Iterate G is:

$$P(k) = \sin^2((2k+1)\theta).$$

It is very important to point out that theorem 3.5 requires the assumption $I = \{0, ..., 2^n - 1\}$.

To maximize our success chances, we must choose k such that $\sin^2((2k+1)\theta)$ is closest to 1, which is the same as $(2k+1)\theta$ being close to $\pi/2, 3\pi/2, \ldots$. If we choose to get close to $\pi/2$, then:

$$(2k+1)\theta \approx \frac{\pi}{2} \implies k \approx \frac{\pi}{4\theta} - \frac{1}{2},$$

so we can simply round to the nearest integer and choose:

$$k = \left\lfloor \frac{\pi}{4\theta} - \frac{1}{2} \right\rceil = \left\lfloor \frac{\pi}{4\theta} \right\rfloor = \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(\sqrt{t_g/2^n}\right)} \right\rfloor$$

Example 3.6. In our previous visualization, with $n = 5$ and $t_g = 2$, the number of times to apply G would be:

$$k = \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(\sqrt{2/32}\right)} \right\rfloor = \lfloor 3.108269 \rfloor = 3.$$

In Figure 5 we can see how the amplitudes of the bad strings are very close to 0.

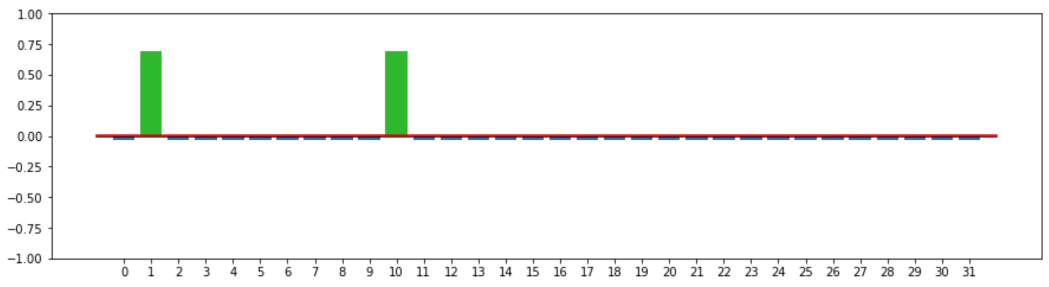


Figure 5. Amplitudes after 3 G applications ($n = 5, t_g = 2, t_b = 30$)

One key problem about the algorithm though is that sometimes we might not know which is the number of good strings t_g. The problem of calculating t_g beforehand is known as the **Quantum Counting problem**, which we will see in section 3.3. There also are some modified versions of the algorithm that output a good string with lower probability and that must be executed more than once, but which do not require prior knowledge of the value t_g. In the standard case when $t_g = 1$, then the best number of Grover iterations is:

$$k = \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} \right\rfloor .$$

We present in Figures 6 and 7 the complete quantum circuit of Grover Search and the construction of the iterator G.

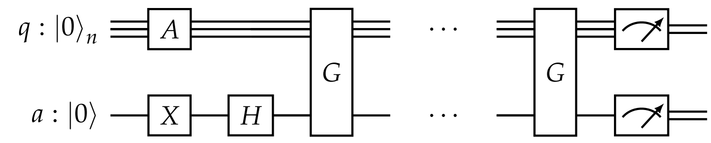


Figure 6. The solution circuit to the search problem

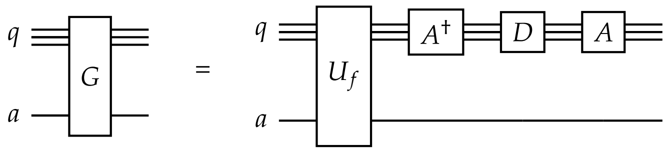


Figure 7. The Grover iterate $G = ADA^\dagger U_f$

When measuring the result, we might not always obtain an optimal answer, but as n grows, it has been proved that the probability of a bad read decreases exponentially.

Theorem 3.7. *Let $f : \{0,1\}^n \longrightarrow \{0,1\}$ with only one string $x \in \{0,1\}^n$ such that $f(x) = 1$, and I chosen to be $\{0,1,\ldots,2^n - 1\}$. Then the probability of obtaining x after applying the Grover Search algorithm is lower bounded by $1 - O(1/2^n)$.*

The Grover Search algorithm achieves a polynomial improvement on the number of times the function to optimize has to be evaluated. Suppose, on the worst case, that only one of the 2^n possible strings is an optimal string. We have seen that the number of quantum applications of f (that is, the number of applications of the Grover iterator G) is:

$$\frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} - \frac{1}{2}.$$

On the other hand, a classical computer has to try them all. On average, it will have to evaluate half the possible strings until it finds the optimal one, meaning the number of classical applications of f is

$$2^{n-1} \in O(2^n),$$

where $O(\cdot)$ is the big-O notation.

When studying the asymptotic growth of the number of quantum applications, when x is close to 0, we can simply substitute $\arcsin(x)$ by the first term of its Taylor series x. It is clear that $\lim_{n\to\infty} 1/\sqrt{2^n} = 0$, and therefore under these conditions:

$$\frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} - \frac{1}{2} \approx \frac{\pi}{4}\sqrt{2^n} - \frac{1}{2} \in O(2^{n/2}).$$

The Grover Search algorithm with a custom implementation can be tested on `GS_Tester.ipynb`, hosted on the GitHub repository at [Ort21].

3.3 The Eigenvalue Estimation algorithm and Quantum Counting

The **Eigenvalue Estimation algorithm** makes use of the Phase Estimation algorithm explained previously to output an estimate of the eigenvalue associated with a specific eigenvector of a unitary matrix. We will review this algorithm and how it can be used to solve the Quantum Counting problem mentioned in section 3.2.

Recall from section 2.4 that unitary matrices have eigenvalues and eigenvectors of modulus 1. This means that its eigenvalues have form $e^{2\pi i\omega}$ with $\omega \in [0,1]$, and that its eigenvectors can be represented as pure states on a quantum register. This allows us to work with the unitary matrix, the eigenvalues and the eigenvectors on a quantum circuit. A primordial requisite of the algorithm then is that we have the unitary matrix encoded as a quantum gate to be used in the algorithm.

Let U denote both the unitary matrix and the n-qubit quantum gate with such unitary matrix encoded, and let $|\psi\rangle$ be an eigenstate (the eigenvector encoded as a pure state) of U with eigenvalue $e^{2\pi i\omega}$. Consider the action of U over the eigenstate controlled on one extra qubit:

$$|0\rangle\,|\psi\rangle \overset{c\text{–}U}{\longmapsto} |0\rangle\,|\psi\rangle$$

$$|1\rangle\,|\psi\rangle \overset{c\text{–}U}{\longmapsto} |1\rangle\,U\,|\psi\rangle = |1\rangle\,e^{2\pi i\omega}\,|\psi\rangle = e^{2\pi i\omega}\,|1\rangle\,|\psi\rangle$$

When the control qubit is in state $|0\rangle$, then nothing happens to the system, but when the control qubit is state $|1\rangle$, then the eigenvalue adds to the global phase factor of the state. If we superpose both $|0\rangle$ and $|1\rangle$ states on the control, applying $c{-}U$ will result in:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}\,|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle\,|\psi\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\,|\psi\rangle \overset{c\text{–}U}{\longmapsto} \frac{1}{\sqrt{2}}\,|0\rangle\,|\psi\rangle + e^{2\pi i\omega}\frac{1}{\sqrt{2}}\,|1\rangle\,|\psi\rangle = \frac{|0\rangle + e^{2\pi i\omega}\,|1\rangle}{\sqrt{2}}\,|\psi\rangle\,,$$

so the phase of the eigenvalue will get encoded on the control qubit. From there we could apply the QFT^{-1} as in Phase Estimation to obtain the first digit x_1 of the decimal expression $0.x_1x_2\ldots$ of the phase ω. If we want more digits $x_2, x_3, \ldots, x_m$, we will have to generate the states

$$\frac{|0\rangle + e^{2\pi i(2\omega)}\,|1\rangle}{\sqrt{2}}\,|\psi\rangle\,, \frac{|0\rangle + e^{2\pi i(2^2\omega)}\,|1\rangle}{\sqrt{2}}\,|\psi\rangle\,, \ldots, \frac{|0\rangle + e^{2\pi i(2^{m-1}\omega)}\,|1\rangle}{\sqrt{2}}\,|\psi\rangle\,.$$

This can be easily achieved by applying $c{-}U$ a total of k times (abbreviated as applying $c{-}U^k$). The action of such application on the state $|1\rangle\,|\psi\rangle$ is:

$$|1\rangle\,|\psi\rangle \overset{c\text{–}U^k}{\longmapsto} |1\rangle\,U^k\,|\psi\rangle = |1\rangle\left(e^{2\pi i\omega}\right)^k|\psi\rangle = e^{2\pi ik\omega}\,|1\rangle\,|\psi\rangle\,,$$

therefore, to obtain digits $x_1, x_2, \ldots, x_m$, we apply $c{-}U, c{-}U^2, c{-}U^4, \ldots, c{-}U^{2^{m-1}}$, each controlled on a qubit in the state $1/\sqrt{2}\,|0\rangle + 1/\sqrt{2}\,|1\rangle$, apply the QFT^{-1} on the control qubits and then measure the result. This process is summarized in the circuit on Figure 8. The output returned by the Phase Estimation algorithm (the QFT^{-1}) is already analyzed in section 3.1.

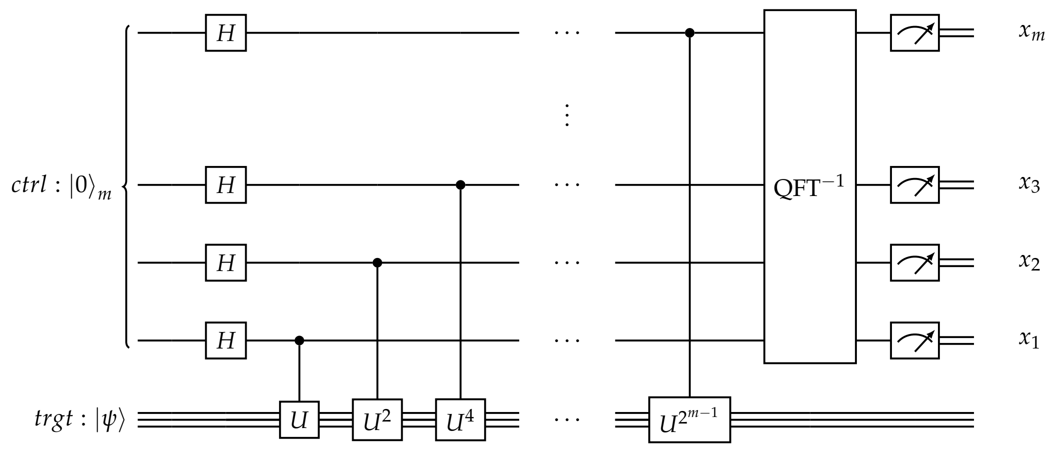


Figure 8. Circuit of Eigenvalue Estimation

We have made the assumption that the target register already begins in an eigenstate $|\psi\rangle$. When this is not the case (for instance, when we do not know the eigenstates of U) and the register starts on some other state $|\varphi\rangle$, what will be the output after the algorithm?

Recall from section 2.4 again that unitary matrices are always diagonalizable, meaning that for our unitary U, there exists an eigenstate basis $\{|\psi_1\rangle, \dots, |\psi_{2^n}\rangle\}$ formed by all eigenstates of U. Furthermore, the eigenstates of different eigenvalues are orthonormal, so an orthonormal basis can be chosen. This means that whatever the initial state $|\varphi\rangle$ was, we can decompose it as:

$$|\varphi\rangle = \mu_1 |\psi_1\rangle + \cdots + \mu_{2^n} |\psi_{2^n}\rangle, \; |\mu_1|^2 + \cdots + |\mu_{2^n}|^2 = 1,$$

so any state $|\varphi\rangle$ can be understood as a superposition of all the eigenstates of U. Let $e^{2\pi i \omega_1}, \dots, e^{2\pi i \omega_{2^m}}$ be all the eigenvalues of the aforementioned eigenstates. The output will be the m-digit estimation of ω_j given by the QFT^{-1} with probability $|\mu_j|^2$.

The Eigenvalue Estimation algorithm is of interest because it can help us solve the Quantum Counting problem. By estimating the eigenvalues of the Grover iterator G we can obtain approximations for t_g. Using the same notation as in section 3.2, consider the state preparation operator $H^{\otimes n}$ and its action:

$$|0\rangle_n \overset{H^{\otimes n}}{\longmapsto} \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n + \frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n .$$

By defining

$$\left|\psi_g\right\rangle = \frac{1}{\sqrt{t_g}} \sum_{x \in I_g} |x\rangle_n ,$$
$$\left|\psi_g\right\rangle = \frac{1}{\sqrt{t_b}} \sum_{x \in I_b} |x\rangle_n = \frac{1}{\sqrt{2^n - t_g}} \sum_{x \in I_b} |x\rangle_n ,$$

we can rewrite the previous expression as:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in I_g} |x\rangle_n + \frac{1}{\sqrt{2^n}} \sum_{x \in I_b} |x\rangle_n = \sqrt{\frac{t_g}{2^n}} \left|\psi_g\right\rangle + \sqrt{\frac{2^n - t_g}{2^n}} |\psi_b\rangle = \sin(\theta) \left|\psi_g\right\rangle + \cos(\theta) |\psi_b\rangle .$$

As the action of the Grover iterator G over the previous state is:

$$\sin(\theta) \left|\psi_g\right\rangle + \cos(\theta) |\psi_b\rangle \xmapsto{G^k} \sin((2k+1)\theta) \left|\psi_g\right\rangle + \cos((2k+1)\theta) |\psi_b\rangle ,$$

then it can be understood as a rotation matrix of angle 2θ on the basis $\mathcal{B} = \{|\psi_b\rangle , \left|\psi_g\right\rangle\}$, which has matrix:

$$\begin{pmatrix} \cos(2\theta) & -\sin(2\theta) \\ \sin(2\theta) & \cos(2\theta) \end{pmatrix} .$$

Some analysis shows that the eigenvalues and eigenvectors of this matrix are

$$e^{2i\theta}, e^{-2i\theta},$$

and

$$|\psi_+\rangle := \begin{pmatrix} \frac{i}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}_{\mathcal{B}} = \frac{i}{\sqrt{2}} |\psi_b\rangle + \frac{1}{\sqrt{2}} \left|\psi_g\right\rangle ,$$
$$|\psi_-\rangle := \begin{pmatrix} \frac{-i}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}_{\mathcal{B}} = \frac{-i}{\sqrt{2}} |\psi_b\rangle + \frac{1}{\sqrt{2}} \left|\psi_g\right\rangle ,$$

respectively. We can isolate the states $|\psi_b\rangle$ and $\left|\psi_g\right\rangle$ to obtain the change of basis

$$|\psi_b\rangle = \frac{-i}{\sqrt{2}} |\psi_+\rangle + \frac{i}{\sqrt{2}} |\psi_-\rangle ,$$
$$\left|\psi_g\right\rangle = \frac{1}{\sqrt{2}} |\psi_+\rangle + \frac{1}{\sqrt{2}} |\psi_-\rangle ,$$

which we can use to follow up on the previous expression:

$$\sin(\theta) \left|\psi_g\right\rangle + \cos(\theta) |\psi_b\rangle =$$
$$\sin(\theta) \left(\frac{1}{\sqrt{2}} |\psi_+\rangle + \frac{1}{\sqrt{2}} |\psi_-\rangle \right) + \cos(\theta) \left(\frac{-i}{\sqrt{2}} |\psi_+\rangle + \frac{i}{\sqrt{2}} |\psi_-\rangle \right) =$$
$$(\sin(\theta) - i\cos(\theta)) \frac{1}{\sqrt{2}} |\psi_+\rangle + (\sin(\theta) + i\cos(\theta)) \frac{1}{\sqrt{2}} |\psi_-\rangle =$$
$$e^{-i\theta} \frac{1}{\sqrt{2}} |\psi_+\rangle + e^{i\theta} \frac{1}{\sqrt{2}} |\psi_-\rangle .$$

To summarize, we have seen that

$$H^{\otimes n} |0\rangle_n = e^{-i\theta} \frac{1}{\sqrt{2}} |\psi_+\rangle + e^{i\theta} \frac{1}{\sqrt{2}} |\psi_-\rangle ,$$

which is a superposition of the two eigenvectors of the Grover iterator G. Running the Eigenvalue Estimation algorithm with $H^{\otimes n} |0\rangle_n$ on the target state will then yield 2θ or -2θ with probabilities $\left|e^{\pm i\theta} 1/\sqrt{2}\right|^2 = 1/2$. The following circuit in Figure 9 showcases the process.

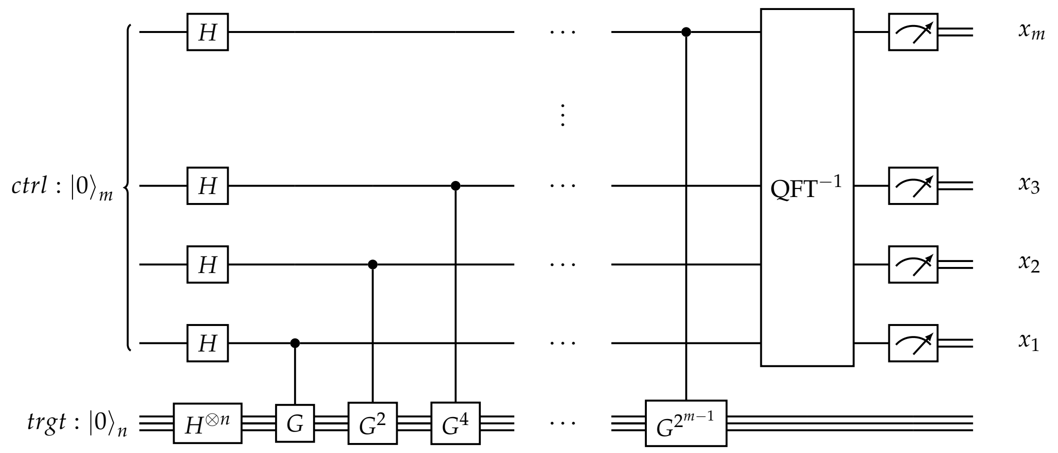


Figure 9. Circuit for Quantum Counting in Grover Search

Although this algorithm can be useful when the optimal number of G applications is not known, one has to note that running this algorithm requires $2^m - 1$ applications of G. To obtain accurate enough estimates of t_g, we need to set $m \approx n$, and this means that the number of G applications for this algorithm lies in $O(2^n)$, which is as expensive as an exhaustive search.

There exist alternative versions of Quantum Counting that manage to reduce the number of qubits m and the number of G applications in exchange for multiple runs and lower accuracy, some of which are detailed in section 8.3 of [KLM07].

4 Function optimization - Discrete version

In this section we computationally analyze **Grover Adaptive Search** [GWG20] as a candidate algorithm for discrete function optimization. This is based on Grover Search, an algorithm that, as we have seen, provides quantum advantage.

Grover Adaptive Search is already implemented in IBM's Qiskit, but restricted to QUBOs - Quadratic Unconstrained Binary Optimization (problems). A QUBO is a quadratic, discrete function over the bits of a string that can be turned into a general cost function through the process in [Qisa]. We implement it for any binary cost function. To test the results and analysis, we provide a custom implementation for general discrete cost functions.

4.1 Problem statement

Let $\mathcal{C} : \{0,1\}^n \to \mathbb{R}$ be a function, often called the **cost function** (binary cost function in our case, as the input is a binary string), that takes an n-bit string $x = x_1 \dots x_n$ and outputs a real number $\mathcal{C}(x)$. The goal of this optimization problem version is to find the optimal string x_{opt} that gives out the highest (or lowest) value of $c_{opt} = \mathcal{C}(x_{opt})$. Any cost function can be decomposed in the following manner:

$$\mathcal{C} = \sum_{S \in \{\texttt{0},\texttt{1},\texttt{X}\}^n} \omega_S \mathcal{C}_S, \ \mathcal{C}_S : \{0,1\}^n \to \{0,1\}.$$

The functions $\mathcal{C}_S$ are called **clauses**, which are equality conditions on some of the bits of the input string that return 1 or 0 whether they are all satisfied or not. A `0` or a `1` denotes a bit that must equal 0 or 1, respectively, and a `X` denotes a free bit with irrelevant value to the clause. That is, they are functions of the form:

$$\mathcal{C}_S(x) = \prod_{i \,|\, s_i = 1} x_i \prod_{j \,|\, s_j = 0} (1 - x_j).$$

When the input string x produces the result 1, we say that x satisfies clause C_S. For example, the string `0010` satisfies clause $\mathcal{C}_{\texttt{X0X0}}$, but not clause $\mathcal{C}_{\texttt{X1X1}}$. The clause $\mathcal{C}_{\texttt{XXXX}}$ is satisfied by any 4-bit string. The values $\omega_S \in \mathbb{R}$ are the **weights** of the clauses, which indicate how important each clause is. We will use the notation $\mathcal{C}_{\omega_S, S}$ to denote clause $\mathcal{C}_S$ with value ω_S.

A simpler but more well-known version of this problem is when $\omega_S = 1$ or 0 for all S, called the **MAX-SAT** problem. In this case, finding the maximum of $\mathcal{C}$ is equivalent of finding the maximum number of simultaneously satisfiable clauses from the cost function (together with the optimal string).

Algorithms that run on classical computers try to make use of the specific problem's structure to devise faster methods with less steep computational costs. In some cases, an approximation close to the optimal solution is given instead to reduce computation time.

4.2 Grover Adaptive Search

Grover Adaptive Search is an algorithm that uses a special version of Grover Search multiple times iteratively to optimize a cost function. We will focus on finding the minimum of the cost function because with a few changes on the cost function, any other optimization problem can be transformed into a minimization problem.

Given a cost function $\mathcal{C} : \{0,1\}^n \longrightarrow \mathbb{Z}$, the idea is to codify the function $f_y : \{0,1\}^n \longrightarrow \{0,1\}$ defined by:

$$f_y(x) = \begin{cases} 1 & \text{if } \mathcal{C}(x) < y \\ 0 & \text{if } \mathcal{C}(x) \geq y \end{cases},$$

where $y \in \mathbb{Z}$ is a chosen threshold and then run the Grover Search algorithm. Suppose we have previously evaluated f on a string $x^{(0)}$, obtaining $f(x^{(0)}) = y^{(0)}$. Now, by choosing the threshold $y = y^{(0)}$ and codifying $f_{y^{(0)}}$ on a quantum circuit, if we run the Grover Search algorithm we will obtain with high probability a string $x^{(1)}$ such that:

$$f_{y^{(0)}}(x^{(1)}) = 1 \iff \mathcal{C}(x^{(1)}) < y^{(0)} \iff y^{(1)} < y^{(0)}.$$

If we change the threshold by choosing $y = y^{(1)}$ and run Grover Search again, we will read with high probability a better string $x^{(2)}$, so by repeating the process iteratively, provided we apply the Grover iterator G the adequate number of times for each Grover Search execution, we will read better and better solutions $x^{(3)}, x^{(4)}, \ldots$, until we eventually find an optimal solution. In some cases though, after running Grover Search, we will not read a better string than the previous one. The cause may be one of the following:

- The Grover Search algorithm failed, as it gives out good strings with probability close to 1, but not exactly 1. In this case, we do not update the threshold and re-run the iteration.
- There are no strings x such that $f_y(x) = 1$ because we already have an optimal solution. In this case, we want to stop the algorithm and return said solution.

To discern between these two cases, we need a way to count how many good strings there are for an arbitrary threshold y. This problem is known as the **Quantum Counting problem**, which has an algorithm to solve it that involves controlled applications of the Grover iterator G.

When Quantum Counting is not available, we will have no way of distinguishing between the two cases, nor a way to determine how many times to apply the Grover iterator G and maximize the probability of a good read. An alternative to circumvent this obstacle is through a process of trial and error by running Grover Search with $1, 2, 3, \ldots$ applications of the Grover iterator G until a better string is read or until we give up and eventually stop the algorithm.

4.2.1 Computational cost analysis

Recall the Quantum Counting algorithm from section 3.3. We can use this algorithm to obtain an estimate for t_g and choose an optimal number of G applications. The process would be, before each run of Grover Search, to run Quantum Counting, read the result and compute with it the optimal number of G applications. If the read result is the string $0\dots0$, denoting $\theta = 0$ or 2π and therefore $t_g = 0$, we stop the algorithm.

With the Counting algorithm, each iteration will require an extra 2^m applications of G, on top of the applications required for running Grover Search. Provided Grover Search does not fail, as with large values of n the chances of failure are minuscule, after every iteration we will read a better string at random out of all the good strings. On average, the read string will be better than half the good strings, so after updating the threshold, only half of them will still be good on the next iteration.

he first threshold is computed by choosing a string $x^{(0)}$ at random and evaluating it. On average, we will start with a threshold $y^{(0)} = f(x^{(0)})$ better than half the 2^n possible strings, meaning that there will initially be 2^{n-1} good strings. As each run of Grover Search halves this number, we will have to run Grover Search n times with $t_g \approx 2^{n-1}, 2^{n-2}, \dots, 1$. We know that the optimal number of applications of G in Grover Search is:

$$\left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(\sqrt{t_g/2^n}\right)} \right\rfloor \approx \frac{\pi}{4} \frac{1}{\arcsin\left(\sqrt{t_g/2^n}\right)} - \frac{1}{2} \lessapprox \frac{\pi}{4}\sqrt{\frac{2^n}{t_g}} - \frac{1}{2},$$

so an estimate of the total number of applications of G in the Grover Search runs will be:

$$\sum_{k=1}^{n} \left(\frac{\pi}{4}\sqrt{\frac{2^n}{2^k}} - \frac{1}{2} \right) = \frac{\pi}{4}\sqrt{2^n} \sum_{k=1}^{n} \frac{1}{\sqrt{2^k}} - \sum_{k=1}^{n} \frac{1}{2} = \frac{\pi}{4}\sqrt{2^n} \frac{1}{\sqrt{2}} \frac{1 - \frac{1}{\sqrt{2^n}}}{1 - \frac{1}{\sqrt{2}}} - \frac{n}{2} =$$

$$\frac{\pi}{4} \frac{\sqrt{2^n} - 1}{\sqrt{2} - 1} - \frac{n}{2}.$$

This added to the $n+1$ Quantum Counting runs (one for each Grover Search run and one extra run at the end to confirm that $t_g = 0$), gives us a total of

$$\frac{\pi}{4} \frac{\sqrt{2^n} - 1}{\sqrt{2} - 1} - \frac{n}{2} + (n+1)2^m$$

G iterator applications. Keep in mind that as n increases, we must also increase the precision of 2θ (or $2\pi - 2\theta$) returned by Quantum Counting, and therefore raise the number of qubits m, as the range of values that t_g lies on gets wider and wider. What can be deduced from the previous analysis is that if $m \geq n/2$, then the term that grows the fastest is $(n+1)2^m$, which is very likely to happen, as the fact that $t_g \in [0, 2^n]$ forces $m \approx n$ for enough output accuracy.

We see then that Quantum Counting introduces an overhead in Grover Adaptive Search on top of the cost of Grover Search that can be toggled by changing m. If we cannot use the algorithm to estimate the number of optimal G applications, a possible solution would be to try them all sequentially until a good string is read, and otherwise stop the algorithm.

One can see that at maximum, this number will be:

$$r_{max} := \max_{t_g=1}^{2^n} \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(\sqrt{t_g/2^n}\right)} \right\rfloor = \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} \right\rfloor,$$

so if we try $r = 1, 2, \ldots, r_{max}$ applications and no good string has been read, we can stop as we can be sure we already have an optimal string, as when n is sufficiently large, with the optimal number of G applications Grover Search is almost sure to succeed. Another way we can shave off more G applications is by considering the fact that the number of applications increases as t_g decreases, as shown in the graph from Figure 10.

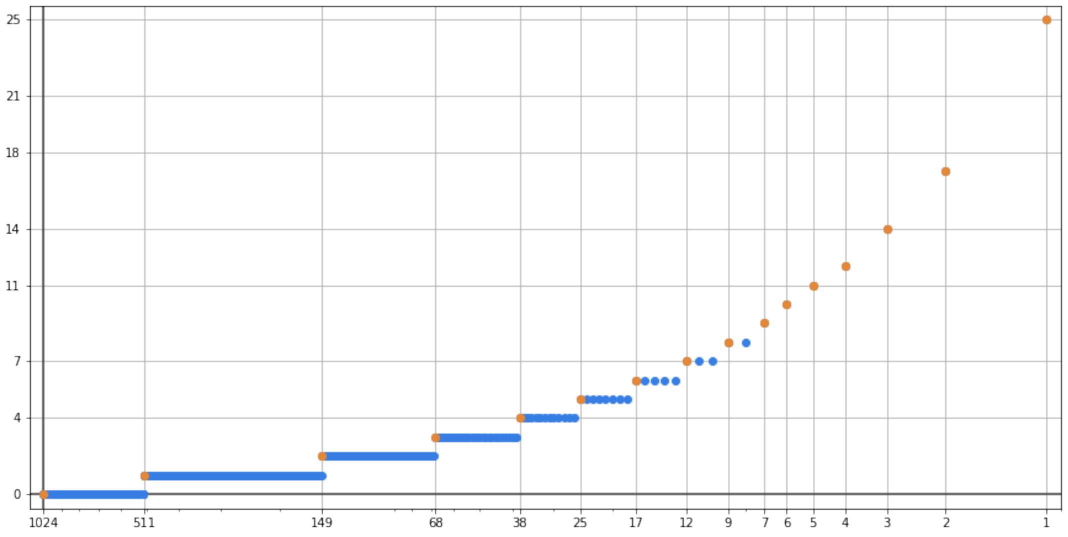


Figure 10. Number of optimal G applications for each number t_g of remaining good strings ($n = 10$)

In a Grover Search run, we know that if we choose the optimal number of G applications, we will almost surely obtain a better solution. In turn, if we obtain a better solution, it may be because we chose such optimal number or because we chose a lower number but had a stroke of luck. In either case, our chosen number will be equal or lower than the optimal number.

Now, obtaining a better solution will lower the number of remaining good strings, meaning that the new optimal number will be greater or equal than the actual optimal number, and so, greater or equal than our chosen number. Therefore, we do not need to start again from $1, 2, 3, \ldots$ G iterations when obtaining a better solution, we just need to re-run Grover Search with the same number we chose.

Regarding the total number of G applications, let us consider again the average case when the remaining number of good strings is $t_g \approx 2^{n-1}, 2^{n-2}, \ldots, 1$. Suppose, in the worst case, that if the optimal number of G applications is not chosen, Grover Search fails, and when such number is chosen, Grover Search succeeds. This means that we will run Grover Search with $1, 2, 3, \ldots, r_{max}$ G applications, but we will have to repeat runs for $t_g = 2^{n-1}, 2^{n-2}, \ldots, 1$. The total number of G applications amounts to:

$$\sum_{k=1}^{n} \left(\frac{\pi}{4} \sqrt{\frac{2^n}{2^k}} - \frac{1}{2} \right) + \sum_{r=1}^{r_{max}} r = \frac{\pi}{4} \frac{\sqrt{2^n} - 1}{\sqrt{2} - 1} - \frac{n}{2} + \frac{(r_{max} + 1) r_{max}}{2}.$$

We can consider r_{max} to be:

$$r_{max} = \left\lfloor \frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} \right\rfloor \approx \frac{\pi}{4} \frac{1}{\arcsin\left(1/\sqrt{2^n}\right)} - \frac{1}{2} \lessapprox \frac{\pi}{4}\sqrt{2^n} - \frac{1}{2},$$

and by substituting in the previous expression, we get:

$$\frac{\pi}{4}\frac{\sqrt{2^n}-1}{\sqrt{2}-1} - \frac{n}{2} + \frac{\left(\frac{\pi}{4}\sqrt{2^n}+\frac{1}{2}\right)\left(\frac{\pi}{4}\sqrt{2^n}-\frac{1}{2}\right)}{2} =$$
$$\frac{\pi}{4}\frac{\sqrt{2^n}-1}{\sqrt{2}-1} - \frac{n}{2} + \frac{\pi^2}{8}2^n - \frac{1}{2} \in O(2^n).$$

This iterative rule, though it does not require usage of Quantum Counting, is no better than an exhaustive search, as the number of G applications scales in $O(2^n)$, just like in a classical computer, so the overhead it introduces is too expensive and negates any quantum advantage.

This methodology, though, can be improved even more, as there are some shortcuts that can be used to reduce the number of G applications. Consider again Figure 10, and note that when t_g gets close to 1, the number of G applications ends up increasing by more than one unit as t_g decreases. Specifically, for $n = 10$, we have:

Remaining good strings (t_g)	5	4	3	2	1
Number of G applications	11	12	14	17	25

We could theoretically skip the Grover Search runs with $13, 15, 16, 18, 19, 20, 21, 22, 23$ and 24 applications, which account for 191 out of the 325 added G applications in the overhead. The only requisite for this would be to have the list of number of G applications precalculated, which only needs to be done once. For $n = 10$, the iterations are reduced by 58.77%, and the reduction percentage increases even more as n grows, asymptotically getting close to 100%. The reduction percentage can be visualized in Figure 11, and the comparison between the original and reduced numbers of G applications is plotted on Figure 12.

Proper formal quantitative analysis should be made to determine how much exactly this shortcut reduces the number of G applications, but we can make some empirical analysis alternatively.

On the graph in Figure 12 we can appreciate some linear pattern in both the original and reduced G applications, so we will assume that both grow linearly in the graph. The y-axis is scaled logarithmically, so these growths are actually exponential. If we plot the ratio of their logarithms, we can see in Figure 13 that it slowly decreases as n grows.

We assume that this ratio eventually converges to an unknown value. When $n = 64$, the ratio is approximately 0.6974469780, just slightly under 0.7, so we can think of it converging to some value $r < 0.7$. Finally, let $G(n), \overline{G}(n)$ be the original and reduced numbers of G applications, respectively. The following equivalence is true:

$$\frac{\log\left(\overline{G}(n)\right)}{\log\left(G(n)\right)} = r \iff \overline{G}(n) = G(n)^r.$$

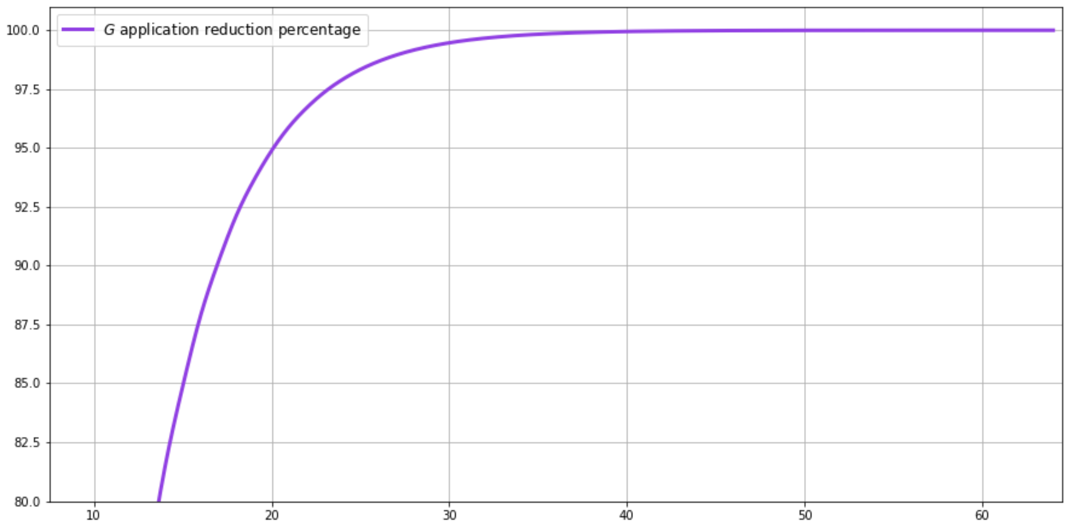


Figure 11. Reduction percentage of G applications (up to $n = 64$)

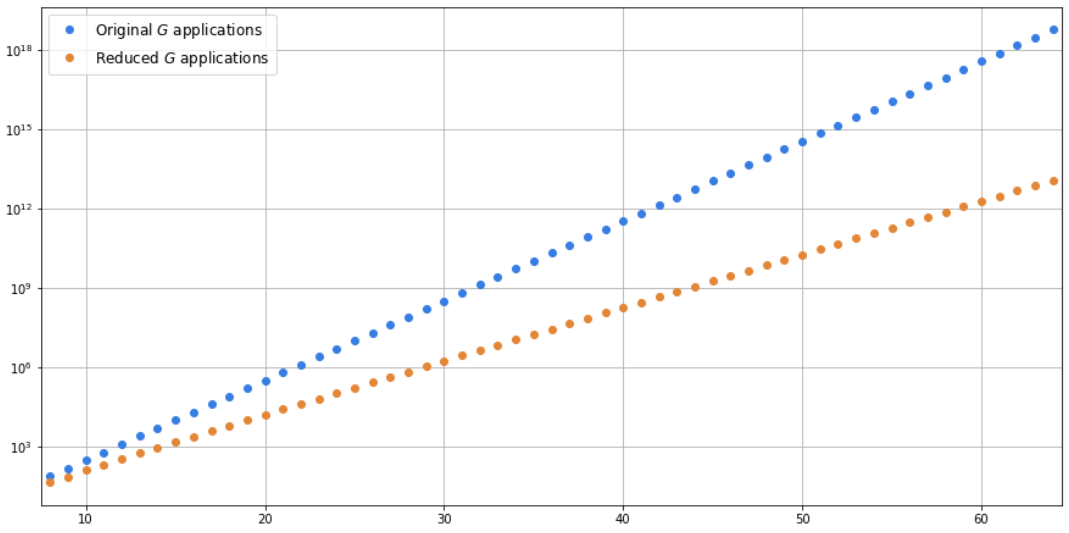


Figure 12. Original and reduced G applications (up to $n = 64$)

So if all the assumptions made were valid, as we had $G(n) \in O(2^n)$, then we would have $\overline{G}(n) \in O(2^{rn})$ with $r < 0.7$, and therefore, as $r < 1$, some of Grover Search's quantum advantage with respect to the classical exhaustive search would be preserved. Ideally for very large values of n, if it were true that $r \leq 0.5$, then the overhead would not waste any of Grover's Search quantum advantage.

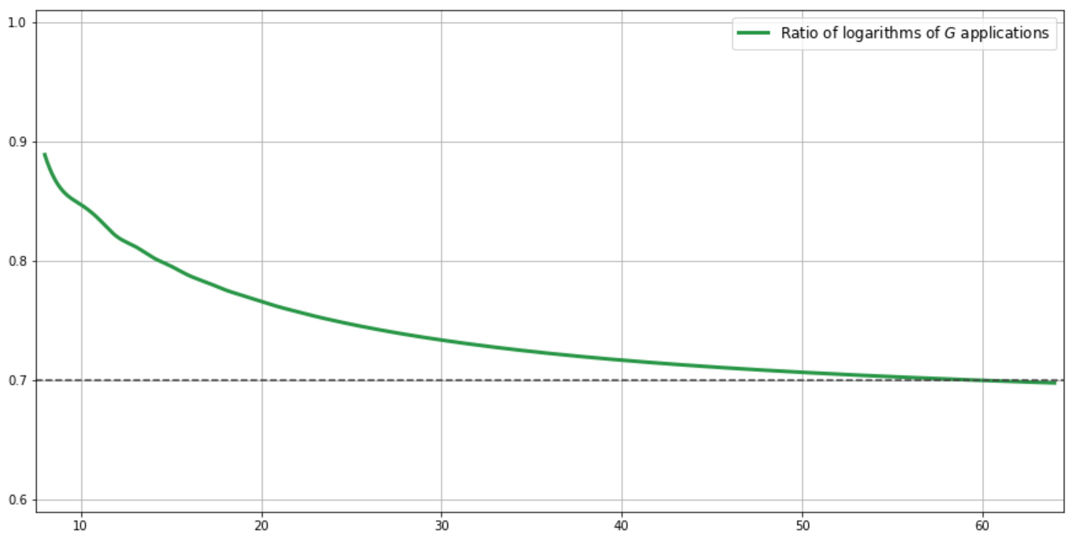


Figure 13. Ratio of logarithms of original and reduced G applications (up to $n = 64$)

4.2.2 Quantum circuit construction

Here we thoroughly explain the construction of the quantum circuit for the implementation of Grover Adaptive Search. The circuit used is an adaptation of Grover Search, so we have an iterator G composed of a preparer A, a diffuser D and an oracle O. The diffuser and oracle are fairly simple, and very similar to the ones used in Grover Search, while the preparer is the most complex, as we now need to encode evaluations of the cost function in the quantum circuit.

In Grover Search we made use of two registers: q, an n-qubit register to which the preparer A is applied to hold a superposition of all strings in $\{0,1\}^n$, and a, an ancillary 1-qubit register. This version of Grover Search will use the registers s, an n-qubit register with the same purpose as q, and v, an m-qubit register that will hold evaluations of the cost function (s for "strings" and v for "values"). We will detail the value m further in the construction.

Let us start constructing the preparer A. Let $R(\theta)$ be the phase gate with matrix form:

$$R(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, \theta \in [0, 2\pi).$$

Its action on a qubit in the state $H|0\rangle = {}^1\!/\!\sqrt{2}\,|0\rangle + {}^1\!/\!\sqrt{2}\,|1\rangle$ is:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \xmapsto{R(\theta)} \frac{|0\rangle + e^{i\theta}|1\rangle}{\sqrt{2}}.$$

Using this gate, we define the m-qubit gate $U_G(\theta) = R(2^{m-1}\theta) \otimes \cdots \otimes R(\theta)$, which can be visualized in Figure 14.

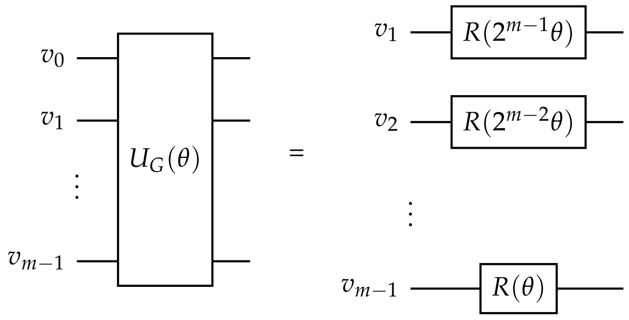


Figure 14. Implementation of gate $U_G(\theta)$

When the v register is in state $H^{\otimes m}\left|0\right\rangle_m$, the action of $U_G(\theta)$ is:

$$H^{\otimes m}\left|0\right\rangle_m = \frac{1}{\sqrt{2^m}}\sum_{y=0}^{2^m-1}\left|y\right\rangle = \frac{\left|0\right\rangle+\left|1\right\rangle}{\sqrt{2}}\otimes\cdots\otimes\frac{\left|0\right\rangle+\left|1\right\rangle}{\sqrt{2}} \xmapsto{U_G(\theta)}$$
$$\frac{\left|0\right\rangle+e^{i2^{m-1}\theta}\left|1\right\rangle}{\sqrt{2}}\otimes\cdots\otimes\frac{\left|0\right\rangle+e^{i\theta}\left|1\right\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2^m}}\sum_{y=0}^{2^m-1}e^{iy\theta}\left|y\right\rangle.$$

Now recall that the action of the QFT^{-1} is:

$$\frac{1}{\sqrt{2^m}}\sum_{y=0}^{2^m-1}e^{2\pi iy\frac{x}{2^m}}\left|y\right\rangle \xmapsto{\text{QFT}^{-1}} \left|x\right\rangle.$$

Given an integer $x\in[0,2^m-1]$, we can choose

$$\theta = 2\pi\frac{x}{2^m} = \frac{\pi x}{2^{m-1}},$$

and then if we apply $U_G(\theta)$, followed by QFT^{-1}, we will end up with state $\left|x\right\rangle$ on the v register. Multiple applications of different gates $U_G(\theta_1),\ldots,U_G(\theta_l)$, with $\theta_i = \pi x_i/2^{m-1}$, will yield:

$$H^{\otimes m}\left|0\right\rangle_m \xmapsto{U_G(\theta_l)\cdots U_G(\theta_1)} \frac{1}{\sqrt{2^m}}\sum_{y=0}^{2^m-1}e^{iy(\theta_1+\cdots+\theta_l)}\left|y\right\rangle \xmapsto{\text{QFT}^{-1}} \left|x_1+\cdots+x_l \pmod{2^m}\right\rangle.$$

In a way, the gate $U_G(\theta_i)$ adds x_i to the resulting state after applying QFT^{-1}. If the integers $x_1,\ldots,x_l$ add up to more than 2^m, then overflow will occur. A solution is to simply increase the number m of qubits on register v.

This only accounts for addition of positive integers, so if we want to enable substraction and signed integers, then we need to work in Two's Complement representation, which uses the last bit to store the sign of the integer. For this, we only need to restrict the integers x to the interval $[-2^{m-1}, 2^{m-1}-1]$.

Using the n-qubit register s that stores the strings in $\{0,1\}^n$, this gives us a way to evaluate the cost function: For each clause $\mathcal{C}_\alpha$ with weight ω_α, add a $U_G(\pi\omega_\alpha/2^{m-1})$ gate controlled on the qubits from register s corresponding to the bits of $\mathcal{C}_\alpha$. If a bit has to be 1 to satisfy the clause, we will control the gate on its corresponding qubit. If it has to be 0, then we will add an X before and after the control, as in Figure 15. If the bit is free, then we do not place a control on the corresponding qubit.

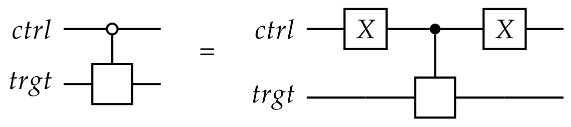


Figure 15. Implementation of a quantum gate controlled on $|0\rangle$

An example is depicted on Figure 16 showing how the 7-bit clause $\mathcal{C}_{6,\texttt{XX00X11}}$ would be encoded.

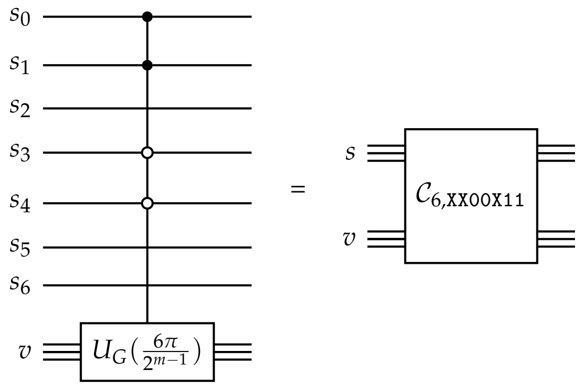

s_0
s_1
s_2
s_3
s_4
s_5
s_6
v
$U_G(\frac{6\pi}{2^{m-1}})$
=
s
v
$\mathcal{C}_{6,\texttt{XX00X11}}$

Figure 16. Clause gate $\mathcal{C}_{6,\texttt{XX00X11}}$

Now we have a way to evaluate the cost function. What we want now is to identify strings x that output values $\mathcal{C}(x) < y$. Consider the fact that $\mathcal{C}(x) < y \iff \mathcal{C}(x) - y < 0$.

We will check the second condition instead of the first. Identifying a negative or positive output in the Two's Complement notation is very simple, as we just need to check the sign bit. Moreover, implementing the cost function $\mathcal{C} - y$ is easily done by adding the free clause gate $\mathcal{C}_{-y,\texttt{X}\cdots\texttt{X}}$ to the clause gates of $\mathcal{C}$.

With all this, then we can construct the preparer A. We begin in states $|0\rangle_n$ and $|0\rangle_m$ on both registers. First, like in Grover Search, we set all strings from $\{0,1\}^n$ in superposition on register s by applying $H^{\otimes n}$ to it. We also apply $H^{\otimes m}$ to register v in order to store the phases encoded by the clause gates. After that, we apply all the clause gates on v controlled by s, and at the end, we apply the QFT^{-1} to register v to retrieve the cost function evaluation from its phase. The result will be a superposition of all strings in s and a superposition of all evaluations in v.

The preparer A is visualized in the circuit depicted on Figure 17.

The oracle U_f is very easy to implement. It will suffice to place a Z gate on the last qubit of register v, qubit q_{m-1}. Given a string x, if $\mathcal{C}(x) \geq y$, the qubit will be in state $|0\rangle$ and the Z gate will do nothing. On the other hand, if $\mathcal{C}(x) < y$, then the qubit will be in

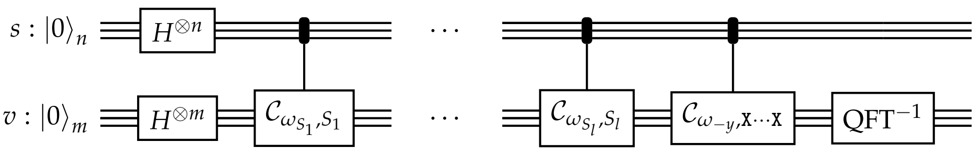

$s : |0\rangle_n$
$H^{\otimes n}$
$v : |0\rangle_m$
$H^{\otimes m}$
$\mathcal{C}_{\omega_{S_1}, S_1}$
$\cdots$
$\mathcal{C}_{\omega_{S_l}, S_l}$
$\mathcal{C}_{\omega_{-y}, \mathtt{X} \cdots \mathtt{X}}$
QFT^{-1}

Figure 17. Circuit for preparer A

state $|1\rangle$ and the Z gate will add a phase factor of -1 that will carry on to the circuit's global phase, effectively distinguishing between positive and negative evaluations.

And finally, the diffuser D does not change and will be the same as in Grover Search, a diagonal gate with all 1s except the first element, which will have a -1. This gate is to be applied to register s.

For preparer A we also have to construct its inverse, as the Grover iterator is $G = ADA^{\dagger}U_f$. Luckily, all gates that form the preparer are easily invertible:

- H is its own inverse, and so are $H^{\otimes n}$ and $H^{\otimes m}$.
- QFT can be constructed by placing all gates forming QFT^{-1} in reversed order and using R_k gates instead of R_k^{-1} (see section 3.1).
- The inverse of the clause gate $\mathcal{C}_{\omega_S, S}$ is the clause gate $\mathcal{C}_{-\omega_S, S}$, easily achieved substituting gate $U_G\left({}^{\omega_S \pi}/_{2^{m-1}}\right)$ by $U_G\left({}^{-\omega_S \pi}/_{2^{m-1}}\right)$.

The inverse preparer $A^{\dagger}$ is visualized in the circuit depicted on Figure 18.

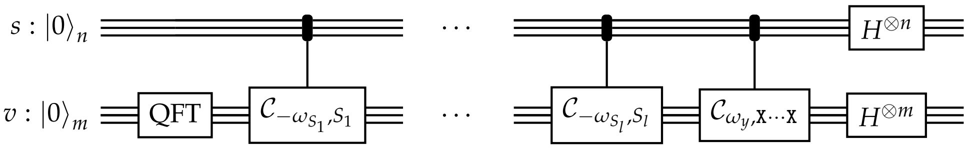

$s : |0\rangle_n$
$H^{\otimes n}$
$v : |0\rangle_m$
QFT
$\mathcal{C}_{-\omega_{S_1}, S_1}$
$\cdots$
$\mathcal{C}_{-\omega_{S_l}, S_l}$
$\mathcal{C}_{\omega_y, \mathtt{X} \cdots \mathtt{X}}$
$H^{\otimes m}$

Figure 18. Circuit for the inverse preparer $A^{\dagger}$

Remark 4.1. All the clause gates are commute between each other, so they can be placed in any order.

Lastly, we formalize the steps to take in order to perform the Grover Adaptive Search. As prerequisites, the states of both s, v registers must be $|0\rangle^{\otimes n}$, $|0\rangle^{\otimes m}$ respectively.

Grover Adaptive Search algorithm

```
Result: Binary string $x_{opt}$ and value $y_{opt}$ such that
        $\mathcal{C}(x_{opt}) = y_{opt} = \min_{x \in \{0,1\}^n} \mathcal{C}(x)$
Precalculate optimal numbers of $G$ applications $r_1, r_2, \dots, r_{j_{max}}$
Set index $j \leftarrow 1$
Set number of Grover Search iterations $r \leftarrow r_j$
Choose $x^{(0)} \in \{0,1\}^n$ randomly
Update best string $x_{opt} \leftarrow x^{(0)}$
Calculate $y^{(0)} \leftarrow \mathcal{C}(x^{(0)})$
Update threshold $y_{opt} \leftarrow y^{(0)}$
Encode threshold $y_{opt}$ as the clause gate $\mathcal{C}_{-y_{opt},\mathtt{X}\cdots\mathtt{X}}$
Set $finish \leftarrow$ FALSE
while $finish =$ FALSE do
    Run Grover Search with $r$ iterations to obtain $x^{(k+1)}, y^{(k+1)}$
    if $y^{(k+1)} < y_{opt}$ then
        Update best string $x_{opt} \leftarrow x^{(k+1)}$
        Update threshold $y_{opt} \leftarrow y^{(k+1)}$
        Encode threshold $y_{opt}$ as the clause gate $\mathcal{C}_{-y_{opt},\mathtt{X}\cdots\mathtt{X}}$
    else if $j \neq j_{max}$ then
        Increment index $j \leftarrow j + 1$
        Set number of Grover Search iterations $r \leftarrow r_j$
    else
        Set $finish \leftarrow$ TRUE
end
Output best string and threshold $x_{opt}, y_{opt}$
```

The Grover Adaptive Search algorithm with a custom implementation can be tested on `GAS_Tester_1.ipynb`, hosted on the GitHub repository at [Ort21].

As the algorithm has a probabilistic nature, where it can take many different paths with a different success chance on each one, extensive tests can help give us an overall view on the performance of the algorithm and provide initial insight on the cases in which the algorithm fails. Extensive tests can be run and viewed on `GAS_Tester_2.ipynb`, hosted on the GitHub repository at [Ort21].

5 Function optimization - Continuous version

In this section we analyze existing algorithms for continuous function optimization ([Reb+16], [KP17], [Jor04]) and evaluate the possibility, and if possible, the difficulty of gate-based implementations for testing. These and more are mentioned in [Bia+16], which serves as a compilation of quantum versions of algorithms needed for machine learning.

One must note that in these works the proposed algorithms are explained, and the results of some tests are analyzed and visualized, but no gate-based implementation nor quantum circuit diagrams are included.

5.1 Problem statement

Let $\mathcal{C} : X \to \mathbb{R}$ be a continuous function where $X \subseteq \mathbb{R}$ is a compact set. The goal of this optimization problem version is to find the optimal point $x_{opt} \in X$ that gives out the optimal (highest or lowest) value of $c_{opt} = \mathcal{C}(x_{opt})$. Most classical methods that solve this problem perform an iterative operation that sequentially outputs points $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ with increasingly optimal evaluations $\mathcal{C}(x^{(1)}), \mathcal{C}(x^{(2)}), \mathcal{C}(x^{(3)}), \ldots$. Contrary to the discrete version, there is no exhaustive procedure to solve this problem, as there are infinite possible points in X to try out.

The most famous iterative algorithms for this kind of problem are Gradient Descent and Newton's Method, which take as input a starting point $x^{(0)}$ close to the optimal point and make use of the first and second derivatives of $\mathcal{C}$ to output points closer and closer to said optimal point. These derivatives are calculated numerically most of the time, as we may not always have a formal expression for $\mathcal{C}$.

The methods assume the function has a global optimum to converge to. The goal is to output the global optimum, but when there are multiple local optima, the methods will converge to one of them, often the closest to the starting point $x^{(0)}$. A simple but expensive way to avoid converging to local optima is to run the methods many times with different starting points and return the best optima found.

Function optimization plays a key role in fields such as artificial intelligence, machine learning and computer vision, to name a few, which in turn have a myriad of real-world applications in data science, economics, psychology, medicine... Quantum algorithms that achieve any kind of speedup, whether polynomial or exponential, can have a massive impact in any of these areas.

5.2 Numerical Gradient Estimation

In [Jor04], the author proposes an algorithm to estimate the gradient of a continuous function. He details the algorithm for numerical estimation of the first order partial derivatives of f through pages 1 and 2, and then in the subsequent pages he analyzes the accuracy of the algorithm given the number of qubits used for the registers, and also comments how to numerically estimate partial derivatives of higher order.

We will detail a few very important prerequisites that the author lightly comments at the beginning of page 1 that will help understanding the algorithm he proposes.

We begin with a continuous function $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ that we want to estimate the gradient of. The method estimates the gradient at point 0, so if we want to estimate the gradient of f at point $a \in \mathbb{R}^d$, we redefine f through the rule $f(x) \to f(x-a)$.

Once we have f redefined, we need to make a few changes to it for use in the algorithm. In order to adapt f, the domain is restricted to a chosen compact set $X \subseteq \mathbb{R}^d$. We also select a closed interval $Y \subseteq \mathbb{R}$ such that $f(X) \subseteq Y$. Then we scale X to $[0,1]^d$ and Y to $[0,1]$. The result of these restrictions is a scaled version of f, a new function $\tilde{f} : [0,1]^d \longrightarrow [0,1]$.

Using the adapted function $\tilde{f}$, we define an approximation $\bar{f} : \mathbb{Z}_{2^n}^d \longrightarrow \mathbb{Z}_{2^{n_o}}$ defined by

$$\bar{f}(x_1, \dots, x_d) = \left\lfloor 2^{n_o} \tilde{f}\left(\frac{x_1}{2^n}, \dots, \frac{x_d}{2^n}\right) \right\rceil.$$

The algorithm makes use of a quantum gate (referred to "black box" by the author) to evaluate $\bar{f}$ that we will denote by $U_{\bar{f}}$. This gate must perform the following operation:

$$|x_1\rangle \cdots |x_d\rangle |a\rangle \xmapsto{U_{\bar{f}}} |x_1\rangle \cdots |x_d\rangle \left|a + \bar{f}(x_1, \dots, x_d) \pmod{2^{n_o}}\right\rangle.$$

In the previous definitions, n and n_0 are the number of qubits of the input registers in states $|x_1\rangle, \dots, |x_d\rangle$ and the output register that begins in state $|a\rangle$, respectively. The number of qubits for the input and the output can be tuned at will. The more qubits assigned to the registers, the higher the accuracy of the results. The total number of qubits used will be $nd + n_o$.

The appeal of the algorithm is that it only requires one application of $U_{\bar{f}}$ to estimate the gradient, whereas the classical numerical estimation of the gradient requires $2d$ applications of f, as it uses the the approximate formula

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, \dots, x_i + l/2, \dots, x_d) - f(x_1, \dots, x_i - l/2, \dots, x_d)}{l}, \quad l > 0,$$

which requires 2 f applications for each partial derivative.

Aside from the lack of prerequisite explanation, the procedure to perform algorithm is detailed, along with the expected output after each set of steps, though there are no formal proofs that verify the outputs included in the article. If analyzed with care, writing an implementation of this algorithm is a feasible task.

5.3 Gradient Descent and Newton's Method on homogenous polynomials

In [Reb+16], a quantum algorithm is proposed for Gradient Descent and Newton's Method restricted to homogenous polynomials of even order, with only a few monomials.

Definition 5.1. An homogenous polynomial p of order d is a polynomial where all the monomials have degree d.

Example 5.2. The polynomial $x_1x_3^3 + x_2^3x_3 - x_2^4 + 8x_1x_2x_3^2$ is homogenous of order 4.

The authors state this restriction because an homogenous polynomial of even order can be as a tensor product, a natural and convenient translation to the quantum computing framework. Let x, A be:

$$x := \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}, A = \begin{pmatrix} a_1^1 & \cdots & a_1^{N^{2d}} \\ \vdots & \ddots & \vdots \\ a_{N^{2d}}^1 & \cdots & a_{N^{2d}}^{N^{2d}} \end{pmatrix}.$$

Then the expression

$$x^T \otimes \cdots \otimes x^T \otimes A \otimes x \otimes \cdots \otimes x$$

is a homogenous polynomial of degree $2d$ on variables $x_1, \ldots, x_N$.

Example 5.3. The previous polynomial $x_1x_3^3 + x_2^3x_3 - x_2^4 + 8x_1x_2x_3^2$ corresponds to the expression:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^T \otimes \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^T \otimes \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 2/3 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & -1 & 1/4 & 0 & 1/4 & 0 \\ 0 & 0 & 2/3 & 0 & 1/4 & 0 & 2/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 2/3 & 1/4 \\ 0 & 0 & 2/3 & 0 & 1/4 & 0 & 2/3 & 0 & 0 \\ 0 & 2/3 & 1/4 & 2/3 & 0 & 0 & 1/4 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \otimes \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Note that the matrix of coefficients A can be chosen symmetric. The matrix A can be decomposed in the following manner:

$$A = \sum_{\alpha=1}^{K} A_1^\alpha \otimes \cdots \otimes A_d^\alpha,$$

where A_i^α are symmetric matrices of size $N \times N$. We provide the exact decomposition of the polynomial from examples 5.2 and 5.3 in appendix A.2.

The authors rely on this decomposition to define operators $\mathbb{D}$ and $\mathbb{H}$ in pages 6 and 7, which correspond to the gradient and the hessian of the polynomial as an N-variable function. Then they assume that these operators can be simulated with a Hamiltonian (in the framework of adiabatic quantum computing) and explain the steps of their quantum version of Gradient Descent and Newton's Method in sections 3 and 4, respectively. The quantum algorithms are pretty similar, just like their classical counterparts. In section 5 the authors consider the case where some monomials are not homogenous by adding a linear term and slightly modifying their algorithms.

The authors claim that both algorithms output solutions $(x_1, \ldots, x_N)$ spherically constrained, which is another way of saying $|x_1|^2 + \ldots |x_N|^2 = 1$. The solution ends up encoded in a quantum register of n qubits (hence the spherical constraint), where $2^n = N$.

While the first sections can be understood without the need of physics knowledge, no gates nor circuit diagrams are provided to the reader. The computations from the algorithms can be translated to quantum gates, but the authors do not provide a way of implementing the operators $\mathbb{D}$ and $\mathbb{H}$ as quantum gates, instead they just assume that their simulations can be efficient from the adiabatic side of quantum computing.

Though the article gives some insight on how one could perform quantum Gradient Descent and Newton's Method, it is not enough to devise a gate-based implementation.

5.4 Gradient Descent for quadratic functions

In [KP17], the authors devise an iterative quantum method to perform Gradient Descent for quadratic functions.

Definition 5.4. A quadratic function f on variables $x_1, \dots, x_n$ is a polynomial of degree 2 on $x_1, \dots, x_n$.

These kind of functions can be expressed in matrix form. Let x be

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

A quadratic form has matrix form $x^T A x + x^T b + c$, with $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, c \in \mathbb{R}$, where a can be chosen symmetric.

Example 5.5. The quadratic function $f(x_1, x_2, x_3) = 3x_1x_2 - 4x_3^2 + 6x_1 - x_2 - 3$ has matrix form

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^T \begin{pmatrix} 0 & 3/2 & 0 \\ 3/2 & 0 & 0 \\ 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 6 \\ -1 \\ 0 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - 3.$$

When this is the function that we want to optimize, the gradient has matrix form $Ax + b$.

Example 5.6. In the previous example, we have:

$$\nabla f(x) = \begin{pmatrix} 0 & 3/2 & 0 \\ 3/2 & 0 & 0 \\ 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 6 \\ -1 \\ 0 \end{pmatrix}.$$

In page 7 the authors remind the reader the update rule of Gradient Descent and use the previous properties to modify the rule to a more convenient expression. To summarize, let r_0 be an initial point with $\|r_0\| = 1$. The resulting point θ_τ after τ iterations of Gradient Descent is:

$$\theta_\tau = r_0 + \alpha \sum_{t=1}^{\tau} S^{t-1}\left(L(r_0)\right),$$

where $S(x) = (I + \alpha A)x$, $L(x) = Ax + b$. Now define $r_i = S^{t-i}\left(L(r_0)\right)$. Then the previous expression can be rewritten as:

$$\theta_\tau = r_0 + \alpha r_1 + \cdots + r_\tau.$$

The goal is to calculate the sum of all these values $r_0, \ldots, r_\tau$, which is what the proposed algorithm does. The authors define unitaries V, U on page 13, given by their actions:

$$\begin{aligned}
&V : \left|0\right\rangle \left|r_0\right\rangle \left|0\right\rangle \longmapsto \left|1\right\rangle \left(\alpha \left\|\tilde{L}(r_o)\right\| \left|\tilde{L}(r_o)\right\rangle \left|0\right\rangle + \left|G_1\right\rangle \left|1\right\rangle\right) \\
&V : \left|t\right\rangle \left\|r_t\right\| \left|r_t\right\rangle \left|0\right\rangle \longmapsto \left|t+1\right\rangle \left(\alpha \left\|\tilde{S}(r_t)\right\| \left|\tilde{S}(r_t)\right\rangle \left|0\right\rangle + \left|G_{t+1}\right\rangle \left|1\right\rangle\right)
\end{aligned}$$

$$\begin{aligned}
&U : \left|0\right\rangle \left|0\right\rangle \left|0\right\rangle \left|r_0\right\rangle \left|0\right\rangle \longmapsto \left|0\right\rangle \left|0\right\rangle \left|0\right\rangle \left|r_0\right\rangle \left|0\right\rangle \\
&U : \left|0\right\rangle \left|0\right\rangle \left|0\right\rangle \left|r_0\right\rangle \left|0\right\rangle \longmapsto \left|t\right\rangle \left|0\right\rangle \left(\alpha \left\|\tilde{S}^{t-1}(\tilde{L}(r_0))\right\| \left|t\right\rangle \left|\tilde{S}^{t-1}(\tilde{L}(r_0))\right\rangle \left|0\right\rangle + \left|G'_t\right\rangle \left|1\right\rangle\right)
\end{aligned}$$

where G_i, G'_i are uninteresting garbage states, and $\tilde{S}, \tilde{T}$ are approximations of S, T, respectively.

Then, the authors detail the steps of the algorithm in section 3.2. They offer some computational cost analysis in the following sections and introduce other algorithms for linear algebra. The steps from section 3.2 are very detailed, but can be a bit hard to follow as the authors swap the positions of registers on some steps, and also, no circuit diagram is provided to the reader. Nevertheless, one can write a list of all the gates and register swaps that must be applied and generate a quantum circuit. It only remains to implement unitary U used in the process.

For an implementation of unitary U, the authors, the authors claim that given an implementation for V, unitary U is simply V controlled on the first register (page 8).

The implementation of unitary V is explained in Proposition 4.9 (pages 25, 26). It uses their Singular Value Estimation algorithm explained in section 4.2, which in turn depends on efficient implementation of two other unitaries also named U, V specified in Theorem 4.2 (page 17), relying in a quantum data structure called the **QRAM** (Quantum Random Access Memory). For more information about this data structure, see [GLM07].

Currently, Qiskit does not offer implementations of QRAM, so simulations of this algorithm cannot theoretically be performed. A dirty but working alternative would be to use the modules `UnitaryGate` or `Initialize`.

A Additional examples

A.1 Density matrix partial trace

To showcase how the partial trace (section 2.6) is applied, we will start with a 2-qubit mixed state, calculate the joint density matrix and then we will extract the density matrix of the first qubit. Let q_A, q_B be the first and second qubits of our system. Consider the 2-qubit mixed state given by the next ensemble:

$$\left\{ \left(\frac{3}{4}, {}^{1}\!/\!\sqrt{2}\,|00\rangle + {}^{1}\!/\!{}_{2}\,|10\rangle + {}^{1}\!/\!{}_{2}\,|11\rangle \right), \left(\frac{1}{4}, {}^{1}\!/\!\sqrt{2}\,|00\rangle + {}^{1}\!/\!\sqrt{2}\,|11\rangle \right) \right\}.$$

We calculate its density matrix:

$$\frac{3}{4}\begin{pmatrix} 1/\sqrt{2} & 0 & 1/2 & 1/2 \end{pmatrix}\begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 1/2 \\ 1/2 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \end{pmatrix}\begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{pmatrix} =$$

$$\frac{3}{4}\begin{pmatrix} 1/2 & 0 & 1/2\sqrt{2} & 1/2\sqrt{2} \\ 0 & 0 & 0 & 0 \\ 1/2\sqrt{2} & 0 & 1/4 & 1/4 \\ 1/2\sqrt{2} & 0 & 1/4 & 1/4 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 \end{pmatrix} =$$

$$\begin{pmatrix} 1/2 & 0 & 3/8\sqrt{2} & 3+\sqrt{2}/8\sqrt{2} \\ 0 & 0 & 0 & 0 \\ 3/8\sqrt{2} & 0 & 3/16 & 3/16 \\ 3+\sqrt{2}/8\sqrt{2} & 0 & 3/16 & 5/16 \end{pmatrix}.$$

The density matrix can be expressed as a linear combination of one-element matrices $|00\rangle\langle 00|, \ldots, |11\rangle\langle 11|$. For our case, this linear combination is:

$$\begin{gathered} 1/2\,|00\rangle\langle 00| + 3/8\sqrt{2}\,|00\rangle\langle 10| + 3+\sqrt{2}/8\sqrt{2}\,|00\rangle\langle 11| + \\ 3/8\sqrt{2}\,|10\rangle\langle 00| + 3/16\,|10\rangle\langle 10| + 3/16\,|10\rangle\langle 11| + \\ 3+\sqrt{2}/8\sqrt{2}\,|11\rangle\langle 00| + 3/16\,|11\rangle\langle 10| + 5/16\,|11\rangle\langle 11| = \end{gathered}$$

$$\begin{gathered} 1/2\,|0\rangle\langle 0| \otimes |0\rangle\langle 0| + 3/8\sqrt{2}\,|0\rangle\langle 1| \otimes |0\rangle\langle 0| + 3+\sqrt{2}/8\sqrt{2}\,|0\rangle\langle 1| \otimes |0\rangle\langle 1| + \\ 3/8\sqrt{2}\,|1\rangle\langle 0| \otimes |0\rangle\langle 0| + 3/16\,|1\rangle\langle 1| \otimes |0\rangle\langle 0| + 3/16\,|1\rangle\langle 1| \otimes |0\rangle\langle 1| + \\ 3+\sqrt{2}/8\sqrt{2}\,|1\rangle\langle 0| \otimes |1\rangle\langle 0| + 3/16\,|1\rangle\langle 1| \otimes |1\rangle\langle 0| + 5/16\,|1\rangle\langle 1| \otimes |1\rangle\langle 1|. \end{gathered}$$

To obtain the density matrix of qubit q_A, we trace out the qubit q_B (apply the partial trace over the second qubit):

$$\begin{gathered}
{}^{1}/_{2}\,\mathrm{Tr}\left(|0\rangle\langle 0|\otimes|0\rangle\langle 0|\right)_{q_B} + {}^{3}/_{8\sqrt{2}}\,\mathrm{Tr}\left(|0\rangle\langle 1|\otimes|0\rangle\langle 0|\right)_{q_B} + {}^{3+\sqrt{2}}/_{8\sqrt{2}}\,\mathrm{Tr}\left(|0\rangle\langle 1|\otimes|0\rangle\langle 1|\right)_{q_B} + \\
{}^{3}/_{8\sqrt{2}}\,\mathrm{Tr}\left(|1\rangle\langle 0|\otimes|0\rangle\langle 0|\right)_{q_B} + {}^{3}/_{16}\,\mathrm{Tr}\left(|1\rangle\langle 1|\otimes|0\rangle\langle 0|\right)_{q_B} + {}^{3}/_{16}\,\mathrm{Tr}\left(|1\rangle\langle 1|\otimes|0\rangle\langle 1|\right)_{q_B} + \\
{}^{3+\sqrt{2}}/_{8\sqrt{2}}\,\mathrm{Tr}\left(|1\rangle\langle 0|\otimes|1\rangle\langle 0|\right)_{q_B} + {}^{3}/_{16}\,\mathrm{Tr}\left(|1\rangle\langle 1|\otimes|1\rangle\langle 0|\right)_{q_B} + {}^{5}/_{16}\,\mathrm{Tr}\left(|1\rangle\langle 1|\otimes|1\rangle\langle 1|\right)_{q_B} =
\end{gathered}$$

$$\begin{gathered}
{}^{1}/_{2}\,|0\rangle\langle 0|\,\mathrm{Tr}\left(|0\rangle\langle 0|\right) + {}^{3}/_{8\sqrt{2}}\,|0\rangle\langle 1|\,\mathrm{Tr}\left(|0\rangle\langle 0|\right) + {}^{3+\sqrt{2}}/_{8\sqrt{2}}\,|0\rangle\langle 1|\,\mathrm{Tr}\left(|0\rangle\langle 1|\right) + \\
{}^{3}/_{8\sqrt{2}}\,|1\rangle\langle 0|\,\mathrm{Tr}\left(|0\rangle\langle 0|\right) + {}^{3}/_{16}\,|1\rangle\langle 1|\,\mathrm{Tr}\left(|0\rangle\langle 0|\right) + {}^{3}/_{16}\,|1\rangle\langle 1|\,\mathrm{Tr}\left(|0\rangle\langle 1|\right) + \\
{}^{3+\sqrt{2}}/_{8\sqrt{2}}\,|1\rangle\langle 0|\,\mathrm{Tr}\left(|1\rangle\langle 0|\right) + {}^{3}/_{16}\,|1\rangle\langle 1|\,\mathrm{Tr}\left(|1\rangle\langle 0|\right) + {}^{5}/_{16}\,|1\rangle\langle 1|\,\mathrm{Tr}\left(|1\rangle\langle 1|\right) =
\end{gathered}$$

$$ {}^{1}/_{2}\,|0\rangle\langle 0| + {}^{3}/_{8\sqrt{2}}\,|0\rangle\langle 1| + {}^{3}/_{8\sqrt{2}}\,|1\rangle\langle 0| + {}^{3}/_{16}\,|1\rangle\langle 1| + {}^{5}/_{16}\,|1\rangle\langle 1| = $$

$$ {}^{1}/_{2}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix} + {}^{3}/_{8\sqrt{2}}\begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix} + {}^{3}/_{8\sqrt{2}}\begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix} + {}^{3}/_{16}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix} + {}^{5}/_{16}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix} = $$

$$ \begin{pmatrix} {}^{1}/_{2} & {}^{3}/_{8\sqrt{2}} \\ {}^{3}/_{8\sqrt{2}} & {}^{1}/_{2} \end{pmatrix}. $$

Now that we have the density matrix of the first qubit, we can compute the probabilities of reading any state when measuring the qubit. We calculate the probabilities of reading $|0\rangle$ and $|1\rangle$:

$$\begin{aligned}
|0\rangle &: \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix} {}^{1}/_{2} & {}^{3}/_{8\sqrt{2}} \\ {}^{3}/_{8\sqrt{2}} & {}^{1}/_{2} \end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{2} \\
|1\rangle &: \begin{pmatrix}0 & 1\end{pmatrix}\begin{pmatrix} {}^{1}/_{2} & {}^{3}/_{8\sqrt{2}} \\ {}^{3}/_{8\sqrt{2}} & {}^{1}/_{2} \end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{2}
\end{aligned}$$

A.2 Homogenous polynomial decomposition

Recall the matrix from example 5.3:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\
0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 2/3 & 1/4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\
0 & 0 & 0 & 0 & -1 & 1/4 & 0 & 1/4 & 0 \\
0 & 0 & 2/3 & 0 & 1/4 & 0 & 2/3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 2/3 & 1/4 \\
0 & 0 & 2/3 & 0 & 1/4 & 0 & 2/3 & 0 & 0 \\
0 & 2/3 & 1/4 & 2/3 & 0 & 0 & 1/4 & 0 & 0
\end{pmatrix}$$

We see that each of the following tensor products of symmetric matrices generate all the elements when added together:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2/3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} = \sqrt{2/3}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \sqrt{2/3}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} = 1/2\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes 1/2\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \sqrt{2/3} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \otimes \sqrt{2/3} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = 1/2 \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \otimes 1/2 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = 1/2 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \otimes 1/2 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2/3 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \sqrt{2/3} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \otimes \sqrt{2/3} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \sqrt{2/3} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \sqrt{2/3} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \end{pmatrix} = 1/2 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes 1/2 \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

References

[21a] *Qiskit/qiskit*. `https://github.com/Qiskit/qiskit`. Accessed: 2021-01-23. 2021.

[21b] *rigetti/grove*. `https://github.com/rigetti/grove`. Accessed: 2021-01-23. 2021.

[Abr+19] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: `10.5281/zenodo.2562110`.

[Aha+08] D. Aharonov et al. "Adiabatic quantum computation is equivalent to standard quantum computation". In: *SIAM Rev.* 50.4 (2008), pp. 755–787.

[Bal+19] E. Ballico et al., eds. *Quantum physics and geometry*. Vol. 25. Lecture Notes of the Unione Matematica Italiana. Springer, Cham, 2019, pp. vii+172.

[Bia+16] J. Biamonte et al. "Quantum Machine Learning". In: (2016). arXiv: `1611.09347 [quant-ph]`.

[Fin19] M. G. Find. "Algorithms For Quantum Computers". In: (2019).

[GLM07] V. Giovannetti, S. Lloyd, and L. Maccone. "Quantum random access memory". In: (2007). arXiv: `0708.1879 [quant-ph]`.

[GT09] Otfried Gühne and Géza Tóth. "Entanglement detection". In: *PhysRep* 474.1-6 (Apr. 2009), pp. 1–75.

[GWG20] A. Gilliam, S. Woerner, and C. Gonciulea. "Grover Adaptive Search for Constrained Polynomial Binary Optimization". In: (2020). arXiv: `1912.04088 [quant-ph]`.

[Hor+09] R. Horodecki et al. "Quantum entanglement". In: *Rev. Mod. Phys.* 81 (2009), pp. 865–942.

[Jor04] S. P. Jordan. "Fast quantum algorithm for numerical gradient estimation". In: (2004). arXiv: `0405146 [quant-ph]`.

[KLM07] P. Kaye, R. Laflamme, and M. Mosca. *An introduction to quantum computing*. Oxford University Press, Oxford, 2007.

[KP17] I. Kerenidis and A. Prakash. "Quantum gradient descent for linear systems and least squares". In: (2017). arXiv: `1704.04992 [quant-ph]`.

[McK+16] D. C. McKay et al. "Efficient Z-Gates for Quantum Computing". In: (June 2016). arXiv: `1612.00858 [quant-ph]`.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000, pp. xxvi+676.

[Ort21] G. Ortega. *TFG-QuantumFunctionOptimization*. `https://github.com/RedReservoir/TFG-QuantumFunctionOptimization`. 2021.

[PV07] M. B. Plenio and Sh. Virmani. "An Introduction to entanglement measures". In: *Quant. Inf. Comput.* 7 (2007), pp. 1–51.

[Qisa] Qiskit. *Converters for Quadratic Programs*. `https://qiskit.org/documentation/tutorials/optimization/2_converters_for_quadratic_programs.html`. Accessed: 2021-01-23.

[Qisb] Qiskit. *Grover Optimizer*. `https://qiskit.org/documentation/tutorials/optimization/4_grover_optimizer.html`. Accessed: 2021-01-20.

[Reb+16] P. Rebentrost et al. "Quantum gradient descent and Newton's method for constrained polynomial optimization". In: (2016). arXiv: `1612.01789 [quant-ph]`.

[Şah20] E. Şahin. "Quantum arithmetic operations based on quantum Fourier transform on signed integers". In: (2020). arXiv: `2005.00443`.

[SBM04] V. V. Shende, S. S. Bullock, and I. L. Markov. "Synthesis of Quantum Logic Circuits". In: (2004). arXiv: `1912.04088 [quant-ph]`.

[SCZ16] Robert S Smith, Michael J Curtis, and William J Zeng. *A Practical Quantum Instruction Set Architecture*. 2016.

[Zyg18] B. Zygelman. *A first introduction to quantum computing and information*. Springer, Cham, 2018.