Quantum gradient estimation and its application to quantum reinforcement learning

A.J. Cornelissen

August 21, 2018

Master thesis

Student number:	4322231
Master's program:	Applied Mathematics
Specialization:	Analysis
To be defended on:	September 4th, 2018
Assessment committee:	Prof. Dr. R.M. de Wolf
	Prof. Dr. J.M.A.M. van Neerven
	Dr. M.P.T. Caspers
Research institute:	Delft University of Technology
	in collaboration with Centrum Wiskunde & Informatica
Faculty:	Electrical Engineering, Mathematics and Computer Science
Contact:	ajcornelissen@outlook.com

Abstract

In 2005, Jordan showed how to estimate the gradient of a real-valued function with a high-dimensional domain on a quantum computer. Subsequently, in 2017, it was shown by Gilyén et al. how to do this with a different input model. They also proved optimality of their algorithm for ℓ^∞-approximations of functions satisfying some smoothness conditions.

In this text, we expand the ideas of Gilyén et al., and extend their algorithm such that functions with fewer regularity constraints can be used as input. Moreover, we show that their algorithm is essentially optimal in the query complexity to the phase oracle even for classes of functions that have more stringent smoothness conditions. Finally, we also prove that their algorithm is optimal for approximating gradients with respect to general ℓ^p-norms, where $p \in [1, \infty]$.

Furthermore, we investigate how Gilyén et al.'s algorithm can be used to do reinforcement learning on a quantum computer. We elaborate on Montanaro's ideas for quantum Monte-Carlo simulation, and show how they can be used to implement quantum value estimation of Markov reward processes. We also show essential optimality of this algorithm in the query complexity of all its oracles. Next, we show how we can construct a quantum policy evaluation algorithm, and how we can use these algorithms as subroutines in Gilyén et al.'s quantum gradient estimation algorithm to perform quantum policy optimization.

The most important open questions remain whether it is possible to improve the query complexity of the extension of Gilyén et al.'s algorithm, when function classes containing functions of Gevrey-type 1 are used as input, as at the moment for this specific parameter setting the algorithm is not better than a very simple classical gradient estimation procedure. Improvement of this result would automatically improve the quantum policy optimization routine as well.

Preface

This thesis report is the result of the final project that was part of my master program Applied Mathematics at the Delft University of Technology. The research was done in close collaboration with Centrum Wiskunde & Informatica in Amsterdam, from January to August 2018.

For me, this project was the first real experience with conducting research along the frontiers of the current knowledge in the field of quantum computing. This enthused me a lot, and I hope to continue to try pushing the boundaries of our common knowledge in the years to come.

I owe a debt of gratitude to a lot of people who helped me through this project. First and foremost, I would like to thank Ronald de Wolf, without whom I probably would have never found these interesting problems that lie at the interface between mathematics and quantum computing. I would like to thank him for taking the time and energy to supervise this project, and regularly providing helpful insights and discussions.

Secondly, I would also like to thank Martijn Caspers, who throughout the project was always willing to help me out, and arranged the possibility to present my findings to all that were interested.

Simultaneously, I would like to thank Ronald de Wolf, Martijn Caspers and Jan van Neerven for taking the time to take part in my assessment committee.

Furthermore, I would like to thank András Gilyén for his insightful research paper [GAW17] that I was able to learn from, and for expressing recognition for my work by referencing this master thesis in his paper.

Additionally, I would like to thank my parents and sister, for supporting me not only over the course of the last few months, but also throughout the rest of my studies.

Finally, I would like to thank all my fellow students who showed interest in the research that I was conducting. Especially, I would like to thank Erik Meulman for the many useful discussions we have had over the course of these last nine months, and for inspiring me to think about reinforcement learning in a quantum computing setting.

Arjan Cornelissen,
August 20th, 2018

Contents

1 Introduction

Over the past few decades, the world has undergone dramatic changes as it entered the period collectively referred to as the *information age*. Machines are taking over more and more tasks that previously had to be performed by hand, sharing information all over the globe is becoming easier and less costly every year, and ever more computational power becomes available to the masses at ever decreasing costs. All of these developments were made possible by major advances on the microscopic level, as computer processor manufacturers have been able to develop smaller and smaller chips, capable of performing more and more operations per second.

This development is reaching its physical limit, though. As computer chip manufacturers are creating transistors that measure just a few nanometers across, they face new problems that stem from the laws of quantum mechanics. Most notably, quantum effects like *quantum tunneling* prevent the development of transistors that are much smaller than the ones that are being created today. So, to keep up with the demands of society, researchers are faced with finding new ways to improve the existing technology.

One of the most radical ideas in this respect is to not regard these quantum phenomena as problematic, but to try to utilize them to one's advantage instead. The research field collectively referred to as *quantum computing* is the field that is concerned with these ideas. Specifically, it investigates the development of a device referred to as a *quantum computer*, capable of harnessing the quantum mechanical effects to perform computations. In this text, we provide the reader with an elementary introduction into quantum mechanics, in Chapter 2, and subsequently we show how the fundamental laws of quantum mechanics constitute the basic building blocks of quantum computing, in Chapter 3. We also elaborate on some commonly used techniques in this field there.

A very prominent question in the field of quantum computing is how computations that can be done on a normal computer, can be sped up using techniques that are based on quantum mechanics. A comprehensive list, known as the quantum algorithm zoo, has been compiled by Stephen Jordan [Jor]. It features some of the well-known quantum algorithms that achieve a significant speed-up over classical algorithms, most notably Shor's algorithm and Grover's algorithm, as well as some lesser known problems that can be solved faster on a quantum computer than on a classical computer.

One of these problems is numerically estimating the gradient at some point in the domain of a real-valued function with high-dimensional domain. The initial algorithm was developed, coincidentally, by Stephen Jordan as well [Jor05]. Recently, Jordan's idea was modified and improved upon by Gilyén et al. in [GAW17] and they proved that their algorithm is optimal in some circumstances. We expand on the ideas that were presented in [GAW17]. In Chapter 4, we extend the algorithm they present to broader classes of input functions, and in Chapter 5, we prove optimality of their algorithm in a greater variety of circumstances.

Almost simultaneously with quantum computing, the field of *reinforcement learning* has also advanced tremendously over the last few decades. This field, simply put, is concerned with using trial and error to learn the best way to perform some task. Recent breakthroughs in this field include the defeat of one of the world's leading Go players, Lee Sedol, in March 2016 [SSS+17], and the defeat of one of the best chess engines, Stockfish, in December 2017 [SHS+17].

The algorithms that arise from the field of reinforcement learning generally require a lot of computational power, from which the question naturally arises whether these computations can be sped up using a quantum computer. The area trying to answer this question is known as *quantum reinforcement learning*, and we give an introduction to this field in Chapter 6. Moreover, as gradient estimation is a commonly used technique in the field of reinforcement learning, we investigate whether Gilyén et al.'s ideas can be used in the field of quantum reinforcement learning as well.

This report constitutes the final thesis of the master program Applied Mathematics, at the Delft University of Technology. The research was conducted in close collaboration with Centrum Wiskunde & Informatica.

2 Introduction to quantum mechanics

As the name suggests, quantum computing is performing computations using the laws of quantum mechanics. Hence, if one is to fully understand quantum computing, then one needs to understand the basic principles of quantum mechanics. Within the context of this text, one can think of the field of quantum mechanics as providing the connection between quantum computing and physics, but it must be noted that quantum mechanics itself is much broader.

Following the approach provided by [NC00], the basic principles of quantum mechanics can be summarized in four statements, which are commonly referred to as the *postulates of quantum mechanics*. These postulates provide the framework on top of which the theory of quantum computing is built. Hence, in order to provide the reader with a solid introduction into quantum computing, elaborating on the postulates of quantum mechanics is indispensable.

These postulates are already interesting and complicated enough, that we can infer some very general, though useful results from it. These results often play a very central role in our understanding of the theory of quantum mechanics, and hence they cannot be omitted in any thorough introductory text into the field.

This chapter aims to introduce the reader to the field of quantum mechanics. The main part of this chapter is Section 2.2, in which the postulates of quantum mechanics are elaborated upon. Fully expressing these postulates, though, requires some mathematical background that is beyond undergraduate level. That is why, first of all, Section 2.1 is devoted to developing the necessary mathematical theory. Finally, in Section 2.2 and Section 2.3, we introduce the reader to some of the general results that follow directly from the postulates of quantum mechanics.

2.1 Mathematical background

In this section, we develop the mathematical tools necessary to understand the very basic constructs in quantum mechanics. Specifically, we will introduce the notion of Hilbert spaces and highlight some of its properties in Subsection 2.1.2, and subsequently we will consider tensor products of Hilbert spaces in Subsection 2.1.3.

We will assume that the reader is familiar with undergraduate level mathematical constructs in the field of real analysis, such as metric spaces, vector spaces, normed spaces, inner product spaces and notions of completeness. An introduction to these concepts can be found in many standard introductory texts into the field, for example [AB98].

To improve readability, we will mainly focus on developing the intuition behind the constructs that we introduce in this chapter. This approach is sufficient to accurately highlight all the aspects that we will need throughout the remainder of this text. However, if one were to try to understand the behavior of these constructs in different settings than the ones in which we will encounter them during this text, one might run into trouble with the introduction presented in this section. Hence, in order to facilitate more curious readers, and in an attempt to avoid any possible ambiguity, we will give a very precise rigorous introduction of these concepts in Appendix A.

2.1.1 Notation

Before we start to introduce these more advanced constructs, though, let's first clear up some notational issues. First of all, throughout this text, we will assume that $\mathbb{N}$ is the set of natural numbers starting from 1, hence $\mathbb{N} = \{1, 2, 3, \dots\}$. In many other texts, one might find that $\mathbb{N}$ also contains the element 0 (which

would make it a half-group under the addition operation), but within this text, we will use a definition of $\mathbb{N}$ such that $0 \notin \mathbb{N}$.

Secondly, and perhaps more importantly, throughout this text we will be using the standing assumption that the inner product is linear in the second variable, and conjugate linear in the first. Mathematicians tend to consider the inner product to be linear in the first variable, and conjugate linear in the second, whereas we will be using the exact opposite convention, which happens to be more common among physicists, and is also in closer connection to matrix-vector multiplications in linear algebra. So, whenever we have elements x and y from some complex inner product space and $c \in \mathbb{C}$, we have:

$$\langle x, cy \rangle = c\langle x, y \rangle = \langle \overline{c}x, y \rangle$$

This concludes our introductory remarks on the notation that we will be using. In the following sections, we will introduce the most important mathematical constructs that we will need to accurately describe the basic building blocks of quantum mechanics.

2.1.2 Hilbert spaces

Arguably the most important and most central construct in the theory of quantum mechanics is the Hilbert space. As we will see in Section 2.2, in order to describe any physical system on a quantum mechanical level, one must resort to using Hilbert spaces. Because of its central role in the remainder of this text, we will attempt to avoid any confusion about this concept, and hence we present a precise mathematical definition below.

Definition 2.1.1: Hilbert space
Let $\mathcal{H}$ be an inner product space over $\mathbb{K}$, where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. Recall that we can define a norm $\|\cdot\| : \mathcal{H} \to \{x \in \mathbb{R} : x \geq 0\}$, by:

$$\forall h \in \mathcal{H}, \qquad \|h\| = \sqrt{\langle h, h \rangle}$$

If $\mathcal{H}$ is complete with respect to this norm (i.e., all Cauchy sequences in $\mathcal{H}$ converge in $\mathcal{H}$), then we refer to $\mathcal{H}$ as a *Hilbert space*. If $\mathbb{K} = \mathbb{C}$, then we say in addition that $\mathcal{H}$ is a *complex Hilbert space*. Finally, we refer to the elements of Hilbert spaces as *vectors*.

In order to build some intuition of the concept of Hilbert spaces, let's consider some examples. First of all, remark that every Hilbert space is an inner product space, and so in order to define a Hilbert space, we need to define a set, and an inner product operation on that set. This, we will do in all the examples below.

Example 2.1.2: Perhaps the simplest example of a Hilbert space is $\mathbb{R}$, where we take the inner product to be ordinary multiplication, so for all $x, y \in \mathbb{R}$, $\langle x, y \rangle = xy$. One can easily show that this inner product obeys all the inner product axioms. In particular, we have that $\langle x, x \rangle = x^2 \geq 0$ for all $x \in \mathbb{R}$, ensuring that the radical that appears in Definition 2.1.1 is well-defined.

Example 2.1.3: Similarly, arguably the simplest example of a complex Hilbert space is $\mathbb{C}$, where we define the inner product to be $\langle x, y \rangle = \overline{x}y$, for all $x, y \in \mathbb{C}$. Again, we see that for all $x \in \mathbb{C}$, $\langle x, x \rangle \geq 0$, as is required for any inner product. Moreover, it is apparent that this inner product is indeed linear in the second variable, and conjugate linear in the first (mathematicians would consider the inner product $\langle x, y \rangle = x\overline{y}$ instead, but according to the physicist view, this is not considered to be an inner product at all, as it is not linear in the second variable).

Example 2.1.4: Other very simple Hilbert spaces are $\mathbb{R}^n$ and $\mathbb{C}^n$, for any $n \in \mathbb{N}$. The inner products on these spaces very easily follow from the inner products of vectors, as prescribed by linear algebra.

Example 2.1.5: Somewhat more interesting complex Hilbert spaces are $\ell^2(\mathbb{N})$ and $L^2(\mathbb{R})$. First of all, we

define $\ell^2(\mathbb{N})$ as such:

$$\ell^2(\mathbb{N}) = \left\{ (a_1, a_2, a_3, \dots) : \forall i \in \mathbb{N}, a_i \in \mathbb{C}, \sum_{i=1}^{\infty} |a_i|^2 < \infty \right\}$$

The inner product is defined as:

$$\langle (a_1, a_2, a_3, \dots), (b_1, b_2, b_3, \dots) \rangle = \sum_{i=1}^{\infty} \overline{a_i} b_i$$

One can easily check using the Cauchy-Schwarz inequality that for any $a, b \in \ell^2(\mathbb{N})$, we have $|\langle a, b \rangle| < \infty$. Again, we easily observe that the inner product is linear in the second variable, and conjugate linear in the first (hence most mathematicians would define the inner product with the conjugate on the b_i's, rather than on the a_i's).

Example 2.1.6: Alternatively, one could consider the subspace of $\ell^2(\mathbb{N})$ of all finitely supported sequences, which I will for now define as $\ell^2_f(\mathbb{N})$. So, we have:

$$\ell^2_f(\mathbb{N}) = \{ (a_1, a_2, a_3, \dots) : \forall i \in \mathbb{N}, a_i \in \mathbb{C}, \exists n \in \mathbb{N} : \forall i \geq n, a_i = 0 \}$$

We can use the same inner product as the one that we defined on $\ell^2(\mathbb{N})$ to define an inner product on $\ell^2_f(\mathbb{N})$. We remark here, though, that the resulting space $\ell^2_f(\mathbb{N})$ with the corresponding inner product is *not* a Hilbert space, as it is not complete with respect to the norm induced by this inner product. In particular, $\ell^2_f(\mathbb{N})$ is merely an inner product space, but it can not be referred to as a Hilbert space.

To prove that it is not complete, we can simply construct a Cauchy sequence in $\ell^2_f(\mathbb{N})$ that does not converge in $\ell^2_f(\mathbb{N})$. To that end, consider the sequence $(a_n)_{n=1}^{\infty} \subseteq \ell^2_f(\mathbb{N})$, $a_n = (1, \frac{1}{2}, \frac{1}{3}, \dots, \frac{1}{n}, 0, 0, \dots)$. We leave it to the reader to check that this is indeed a Cauchy sequence in $\ell^2_f(\mathbb{N})$, but that it does not converge in $\ell^2_f(\mathbb{N})$.

Example 2.1.7: Our final example of a Hilbert space is the space $L^2(\mathbb{R})$. Its precise mathematical definition is already quite involved, but we can intuitively think of this space of containing all integrable functions $f : \mathbb{R} \to \mathbb{C}$, satisfying the relation:[1]

$$\int_{-\infty}^{\infty} \overline{f(x)} f(x) \, \mathrm{d}x < \infty$$

The inner product between two functions $f, g \in L^2(\mathbb{R})$ is defined by:[2]

$$\langle f, g \rangle = \int_{-\infty}^{\infty} \overline{f(x)} g(x) \, \mathrm{d}x$$

Again, we can see the linearity in the second variable pop up.

The space $L^2(\mathbb{R})$ in particular is interesting in quantum mechanics, as it can be used to describe the behavior of a free particle that is free to move in one direction. We will encounter this space again when we develop some intuition about the postulates of quantum mechanics, in Section 2.2.

The above examples can be used as canonical examples as Hilbert spaces, and together they highlight most of the important properties of Hilbert spaces that we will need. This means that whenever some statement is true for all of the spaces introduced in the examples above, it is generally true for all Hilbert spaces that we will encounter throughout this text, indicating that they are well-suited for developing intuition about the Hilbert spaces we will be using.

Let's now step back a little and see what structure the Hilbert spaces have on an abstract level. If a vector x of a Hilbert space satisfies $\langle x, x \rangle = 1$, we say that the element has *unit length*. If two vectors x, y of a

[1] In the formal definition of $L^2(\mathbb{R})$, this integral must be interpreted as a Lebesgue integral, but we will not go into details about that here. The interested reader is referred to [AB98].

[2] Idem.

Hilbert space satisfy $\langle x, y\rangle = 0$, then we say that x and y are *orthogonal*. If we have a (finite or infinite) set of vectors $\{x_i\}_{i\in I}$ of a Hilbert space, we say that the set is orthogonal if any two vectors of the set are orthogonal. If in addition all of the elements of the set $\{x_i\}_{i\in I}$ are of unit length, we say that the set is *orthonormal*, and that any two vectors are orthonormal. At this point, the reader should check that the set of vectors $\{e_i\}_{i=1}^{\infty} \subseteq \ell^2(\mathbb{N})$ is orthonormal, where for $i \in \mathbb{N}$, e_i is defined as the sequence being identically 0 except for just a single 1, $(0, 0, 0, \dots, 0, 0, 1, 0, 0, \dots)$, where the 1 is in the ith position.

A *basis* of a Hilbert space is a (finite or infinite) set of linearly independent vectors such that the linear subspace of finite linear combination of these vectors is dense in the Hilbert space. In particular, this means that every vector in the Hilbert space can be written as a convergent sum of these vectors. If the set of vectors that comprises the basis is in addition orthogonal or orthonormal, we speak of an *orthogonal* or *orthonormal basis*, respectively. The reader should check that the set $\{e_i\}_{i=1}^{\infty}$ is indeed an orthonormal basis of $\ell^2(\mathbb{N})$.

We know that two bases of a Hilbert space will always have equal cardinality (the proof can for example be found in any introductory text on linear algebra). Hence we can unambiguously define the dimension of a Hilbert space to be the cardinality of any basis of it.

The question now naturally arises whether there exist Hilbert spaces for which it is impossible to construct a basis, and hence that do not have a well-defined dimension. The answer is more difficult than one would expect. If one assumes the axiom of choice, one can always construct a basis of a Hilbert space. One proves this using Zorn's lemma. Otherwise, one can in general only construct bases of Hilbert spaces where the dimension is at most the cardinality of $\mathbb{N}$. These spaces are known as *separable Hilbert spaces*.

Definition 2.1.8: Separability of Hilbert spaces
A Hilbert space is called *separable* if it contains a countable dense subset.

Theorem 2.1.9: Separability and the existence of bases
A countable basis of a Hilbert space exists if and only if the Hilbert space is separable.

Sketch of the proof. Separability implies the existence of a countable basis through the constructive process of Gramm-Schmidt orthogonalisation. Conversely, we can construct a countable dense subset from a countable basis by considering all linear combinations with rational coefficients of the basis vectors. □

As the existence of a countable basis implies that a countable subset of linearly independent vectors spans a dense subset of the Hilbert space, we can intuitively think of separable Hilbert spaces as having a countable dimension. In other words, from any point in a separable Hilbert space, one can travel in a number of dimensions that is no greater than the cardinality of $\mathbb{N}$.

One might ask why the equivalent characterization of Theorem 2.1.9 is not used as the defining property of separable Hilbert spaces. The reason for this is because the definition as presented in Definition 2.1.8 readily generalizes to a Banach space setting, whereas the equivalent characterization does not. As we will not be concerned with Banach spaces throughout the remainder of this text, for all intents and purposes, the reader can consider the equivalent characterization presented in Theorem 2.1.9 as the definition of separability, throughout the rest of this text.

We remark that all the spaces that we considered in the example earlier in this section, are in fact separable. In $\mathbb{R}$, we have the subset $\mathbb{Q}$, which can easily be proven to be countable by enumerating all possible fractions and hereby constructing a bijection with $\mathbb{N}$, and moreover it is dense in $\mathbb{R}$. Similarly, in $\mathbb{C}$, we have the dense subset $\mathbb{Q} + i\mathbb{Q}$. From there, it is not difficult to see that for every $n \in \mathbb{N}$, we have that $(\mathbb{Q} + i\mathbb{Q})^n$ is dense in $\mathbb{C}^n$. Furthermore, for $\ell^2(\mathbb{N})$, we have already seen that a basis exists, and hence it is separable as well.

Constructing a countable dense subset of $L^2(\mathbb{R})$ is a little bit more involved. Here, we will just state that one can do so by considering linear combinations with rational coefficients of identity functions on finite

intervals of $\mathbb{R}$ with rational endpoints. We leave it to the reader to check that this set is countable and dense in $L^2(\mathbb{R})$.

Finally, we consider the question when two Hilbert spaces can be considered equal. Formally speaking, we can only say that two Hilbert spaces are equal if they are equal as sets, and if the inner products defined on them have equal action. In general, though, we are mainly focused on whether the Hilbert spaces have equal properties. This is where the following definition and theorem come into play.

Definition 2.1.10: Hilbert space isomorphisms
Let A and B be Hilbert spaces over $\mathbb{K}$, where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, with corresponding inner products $\langle \cdot, \cdot \rangle_A$ and $\langle \cdot, \cdot \rangle_B$, respectively. Let ψ be a bijective linear map $\psi : A \to B$, such that:

$$\langle a_1, a_2 \rangle_A = \langle \psi(a_1), \psi(a_2) \rangle_B$$

Then we say that ψ is a *Hilbert space isomorphism from A to B*. We say that A and B are *isomorphic as Hilbert spaces*, if there exists a Hilbert space isomorphism from A to B. If this is the case, we denote this by $A \simeq B$.

Theorem 2.1.11: Isomorphism characterization of separable Hilbert spaces
Let A and B be Hilbert spaces over $\mathbb{K}$, where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$. Then A and B are isomorphic as Hilbert spaces if and only if they have equal dimension.

Sketch of the proof. Suppose that A and B are isomorphic as Hilbert spaces. Let $\{a_i\}_{i \in I}$ be an orthonormal basis of A, and let ψ be a Hilbert space isomorphism from A to B. Then, we find that $\{\psi(a_i)\}_{i \in I}$ is an orthonormal set in B. Moreover, we have $\|a\|^2 = \langle a, a \rangle_A = \langle \psi(a), \psi(a) \rangle_B = \|\psi(a)\|_B^2$, and so ψ is bounded. By linearity of ψ, we find that:

$$\sum_{i \in I} c_i \psi(a_i) = \psi \left(\sum_{i \in I} c_i a_i \right)$$

where by the boundedness of ψ, we trivially check that convergence of one sum implies convergence of the other. As ψ is bijective, we find that every element can be written as a convergent sum of the elements $\{\psi(a_i)\}_{i \in I}$, hence it is a basis. As we have $|\{a_i\}_{i \in I}| = |I| = |\{b_i\}_{i \in i}|$, we find that A and B have equal dimension.

Conversely, if A and B have equal dimension, we can construct equinumerous orthonormal bases $\{a_i\}_{i \in I}$ and $\{b_i\}_{i \in I}$ of A and B, respectively. But then we can find a bijection between the bases, i.e., for all $i \in I$, $\psi(a_i) = b_i$. Now, we extend this map linearly to a map from A to B. We easily check that this is an isomorphism of Hilbert spaces, and hence that A and B are isomorphic as Hilbert spaces. □

Note that if we are willing to use the axiom of choice, then we can indeed define the dimension of any Hilbert space. Otherwise, the above statement only holds for separable Hilbert spaces (as we cannot say that two Hilbert spaces have equal dimension if we cannot define the dimension in the first place).

So, to recap, we can consider two Hilbert spaces isomorphic if and only if their dimensions are equal. This has the remarkable implication that for every finite-dimensional complex Hilbert space $\mathcal{H}$, we can find an $n \in N$ such that $\mathcal{H} \simeq \mathbb{C}^n$. Moreover, rather unexpectedly, we obtain $\ell^2(\mathbb{N}) \simeq L^2(\mathbb{R})$, but here any isomorphism is highly non-trivial.

2.1.3 Tensor products

In this section, we will introduce the reader to the mathematical concept known as the tensor product. The way we do this is somewhat odd, because we will refrain from giving a precise mathematical definition, but we will describe all of its relevant properties instead. The reason for omitting a precise construction in the

main body of this text is because it is rather involved, and does not provide us with any additional insight. However, we must not underestimate the importance of careful rigorous constructions of the mathematical objects that we want to use, so we do present the precise construction in Appendix A. The curious reader can find all the details there.

The concept that we will introduce in this chapter, we will refer to as the *tensor product*. However, in literature, one can find several different forms of tensor products, e.g. vector space tensor products, Banach space tensor products and Hilbert space tensor products. We will be using the term *tensor product* synonymously with *Hilbert space tensor product* throughout the remainder of this text, and we will not encounter any of the others (except for in Appendix A where we introduce the concept in a rigorous way).

So, let's cut to the chase and give the most important properties of the Hilbert space tensor product.

Theorem 2.1.12: Properties of the Hilbert space tensor product space
Let A and B be Hilbert spaces over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Then, we find:

1. $A \otimes B$ is a Hilbert space.
2. For all $a \in A$ and $b \in B$, $a \otimes b \in A \otimes B$.
3. For all $a_1, a_2 \in A$ and $b_1, b_2 \in B$, $\langle a_1 \otimes b_1, a_2 \otimes b_2 \rangle_{A\otimes B} = \langle a_1, a_2 \rangle_A \cdot \langle b_1, b_2 \rangle_B$.
4. For all $a \in A$ and $b \in B$, we have $\|a \otimes b\|_{A\otimes B} = \|a\|_A \cdot \|b\|_B$.
5. The map $A \times B \to A \otimes B$, given by $(a, b) \mapsto a \otimes b$ is jointly continuous.
6. For all $a_1, a_2 \in A$ and $b \in B$, we have $a_1 \otimes b + a_2 \otimes b = (a_1 + a_2) \otimes b$.
7. For all $a \in A$ and $b_1, b_2 \in B$, we have $a \otimes b_1 + a \otimes b_2 = a \otimes (b_1 + b_2)$.
8. For all $a \in A$, $b \in B$ and $k \in \mathbb{K}$, we have $k(a \otimes b) = (ka) \otimes b = a \otimes (kb)$.
9. The map $A \times B \to A \otimes B$, given by $(a, b) \mapsto a \otimes b$, is bilinear.
10. Let $\mathcal{B}_A$ and $\mathcal{B}_B$ be orthonormal bases of A and B, respectively. Then, $\mathcal{B}_{A\otimes B} = \{a \otimes b : a \in \mathcal{B}_A, b \in \mathcal{B}_B\}$ is an orthonormal basis of $A \otimes B$.
11. Whenever the dimensions of A and B are defined, $\dim(A \otimes B) = \dim(A) \dim(B)$.

Proof. For the proof, we refer the reader to Appendix A. □

Note that in Theorem 2.1.12, we specify that whenever we have $a \in A$ and $b \in B$, where A and B are Hilbert spaces, then $a \otimes b \in A \otimes B$. We do not specify *what* $a \otimes b$ is, though, we will just consider it to be a mathematical object, which happens to be an element of the Hilbert space $A \otimes B$.

As $A \otimes B$ is a Hilbert space, it is closed under taking linear combinations. Hence, for any $a_1, a_2 \in A$ and $b_1, b_2 \in B$, we have that $a_1 \otimes b_1 + a_2 \otimes b_2 \in A \otimes B$. If we choose a_1 and a_2, and b_1 and b_2 to be linearly independent, we can show that we cannot write this element as a simple tensor, i.e., there do not exist $a \in A$ and $b \in B$ such that $a_1 \otimes b_1 + a_2 \otimes b_2 = a \otimes b$. So, we define simple tensors to be the elements of $A \otimes B$ that can be written as $a \otimes b$ for some $a \in A$ and $b \in B$, and using property 10 of Theorem 2.1.12, we observe that these elements span a dense subspace of $A \otimes B$.

The bilinearity property, i.e. properties 6 through 9 in Theorem 2.1.12, can lead to some confusion at first sight. For example, let 0_B denote the additive identity of B. Then we have for any vector $a \in A$ that $a \otimes 0_B = a \otimes (0 \cdot 0_B) = 0(a \otimes 0_B) = 0_{A\otimes B}$, the additive identity element of $A \otimes B$. Hence, we cannot view the set $\{a \otimes 0_B : a \in A\}$ as a set of elements lying along an axis of the tensor product space, analogously to the Cartesian product. Instead, this set only contains one element, the 0-vector in the resulting space $A \otimes B$.

Another source for confusion is that scalar multiples can be distributed over the left and right part of the tensors freely. For all $c \in \mathbb{K}$, $a \in A$ and $b \in B$, for example, we have:

$$c(a \otimes b) = (ca) \otimes b = a \otimes (cb) = \left(\sqrt{c}a\right) \otimes \left(\sqrt{c}b\right)$$

where the last equality holds whenever $\sqrt{c}$ is defined. This can be especially confusing when working with minus signs:

$$(-a) \otimes b = -(a \otimes b) = a \otimes (-b) = (ia) \otimes (ib) = (-ia) \otimes (-ib) \qquad \text{and} \qquad (-a) \otimes (-b) = a \otimes b$$

where the last two equalities on the left hand side only hold whenever $\mathbb{K} = \mathbb{C}$.

From Theorem 2.1.12, we observe that if A and B are Hilbert spaces over $\mathbb{K} = \{\mathbb{R}, \mathbb{C}\}$, then $A \otimes B$ is again a Hilbert space. Hence, if C is another Hilbert space over $\mathbb{K}$, we can tensor it again, this time with C, and obtain the new space $A \otimes B \otimes C$. Again, for any $a \in A$, $b \in B$ and $c \in C$, we obtain that $a \otimes b \otimes c$ is an element of $A \otimes B \otimes C$, and all properties in the above theorem generalize to $A \otimes B \otimes C$ in the obvious way. For example, observe that the map $(a, b, c) \mapsto a \otimes b \otimes b$ becomes *trilinear*, where $a \in A$, $b \in B$ and $c \in C$. Continuing in this manner, we can define any tensor product space of a finite number of Hilbert spaces.

Now, we introduce some shorthand notation. Let $(A_n)_{n=1}^N$ be a finite sequence of Hilbert spaces, and for every $i \in \{1, \ldots, N\}$, let $a_i \in A_i$. Then, we can write:

$$\bigotimes_{n=1}^{N} A_n = A_1 \otimes \cdots \otimes A_N \qquad \text{and} \qquad \bigotimes_{n=1}^{N} a_n = a_1 \otimes \cdots \otimes a_N$$

Moreover, if A is any Hilbert space and $a \in A$, then we can write:

$$A^{\otimes N} = \bigotimes_{n=1}^{N} A = \underbrace{A \otimes \cdots \otimes A}_{N \text{ times}} \qquad \text{and} \qquad a^{\otimes N} = \bigotimes_{n=1}^{N} a = \underbrace{a \otimes \cdots \otimes a}_{N \text{ times}}$$

Note that in this text, we will only be using a finite number of tensor products. It appears to be possible to generalize this to countable tensor products, and interestingly, the resulting infinite tensor product space is not separable for any infinite sequence of non-trivial Hilbert spaces. More about this can be found in [Ng13].

Next, let's consider operators on Hilbert space tensor product spaces. To that end, we introduce the following definition of the tensor product of two operators.

Definition 2.1.13: Tensor product of bounded linear operators on Hilbert spaces
Let A and B be two Hilbert spaces over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$, and let L_A and L_B be bounded linear operators acting on A and B, respectively. Then, we define the operator $L_A \otimes L_B$ to act on $A \otimes B$, and we define it to be the linear extension of the following mapping:

$$L_A \otimes L_B : a \otimes b \mapsto L_A a \otimes L_B b$$

Proof ot the well-definedness of $L_A \otimes L_B$. A very similar proof is given in the appendix, where the well-definedness of Definition A.8 is verified. The interested reader is referred there. □

Let's derive some properties of these tensor product operators:

Theorem 2.1.14: Properties of tensor products of bounded linear operators on Hilbert spaces
Let A and B be two Hilbert spaces over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Then we have the following statements concerning tensor product operators:

1. Let $L_A \in \mathcal{B}(A)$ and $L_B \in \mathcal{B}(B)$. Then $L_A \otimes L_B \in \mathcal{B}(A \otimes B)$ and $\|L_A \otimes L_B\|_{\mathcal{B}(A\otimes B)} = \|L_A\|_{\mathcal{B}(A)} \cdot \|L_B\|_{\mathcal{B}(B)}$.
2. The map $\mathcal{B}(A) \times \mathcal{B}(B) \to \mathcal{B}(A \otimes B)$ defined by $(L_A, L_B) \mapsto L_A \otimes L_B$ is jointly continuous.
3. Let $L_{A,1}, L_{A,2} \in \mathcal{B}(A)$ and $L_B \in \mathcal{B}(B)$. Then $L_{A,1} \otimes L_B + L_{A,2} \otimes L_B = (L_{A,1} + L_{A,2}) \otimes L_B$.
4. Let $L_A \in \mathcal{B}(A)$ and $L_{B,1}, L_{B,2} \in \mathcal{B}(B)$. Then $L_A \otimes L_{B,1} + L_A \otimes L_{B,2} = L_A \otimes (L_{B,1} + L_{B,2})$.
5. Let $L_A \in \mathcal{B}(A)$, $L_B \in \mathcal{B}(B)$ and $k \in \mathbb{K}$. Then, $k(L_A \otimes L_B) = (kL_A) \otimes L_B = L_A \otimes (kL_B)$.
6. The map $\mathcal{B}(A) \times \mathcal{B}(B) \to \mathcal{B}(A \otimes B)$ defined by $(L_A, L_B) \mapsto L_A \otimes L_B$ is bilinear.
7. Let $L_{A,1}, L_{A,2} \in \mathcal{B}(A)$ and $L_{B,1}, L_{B,2} \in \mathcal{B}(B)$. Then $(L_{A,1} \otimes L_{B,1})(L_{A,2} \otimes L_{B,2}) = (L_{A,1}L_{A,2}) \otimes (L_{B,1}L_{B,2})$.
8. Let $L_A \in \mathcal{B}(A)$ and $L_B \in \mathcal{B}(B)$. Then $(L_A \otimes L_B)^* = L_A^* \otimes L_B^*$.
9. Let $S_A \in \mathcal{B}(A)$ and $S_B \in \mathcal{B}(B)$ be self-adjoint. Then $S_A \otimes S_B$ is self-adjoint as well.
10. Let $U_A \in \mathcal{B}(A)$ and $U_B \in \mathcal{B}(B)$ be unitary. Then $U_A \otimes U_B$ is unitary as well.
11. Let $N_A \in \mathcal{B}(A)$ and $N_B \in \mathcal{B}(B)$ be normal. Then $N_A \otimes N_B$ is normal as well.

Proof. We will postpone the proof of statement 1 to the end of this proof. First of all, we remark that statement 2 follows trivially from statement 1. Furthermore, we observe that statements 3 through 5 can be easily checked by looking at the action of these tensor product operators on the simple tensors. Then we can infer the general statement by linear extension (where we use the well-definedness). Statement 6 is just a restatement of statements 3 through 5. Statement 7 follows directly from linear extensions of the action of the composite operator on the simple tensors.

For statement 8, let $a_1, a_2 \in A$ and $b_1, b_2 \in B$. Then, we find:

$$\begin{aligned}\langle (L_A \otimes L_B)^*(a_1 \otimes b_1), a_2 \otimes b_2\rangle_{A\otimes B} &= \langle a_1 \otimes b_1, (L_A \otimes L_B)(a_2 \otimes b_2)\rangle_{A\otimes B} = \langle a_1 \otimes b_1, L_A a_2 \otimes L_B b_2\rangle_{A\otimes B} \\ &= \langle a_1, L_A a_2\rangle_A \cdot \langle b_1, L_B b_2\rangle_B = \langle L_A^* a_1, a_2\rangle_A \cdot \langle L_B^* b_1, b_2\rangle_B \\ &= \langle L_A^* a_1 \otimes L_B^* b_1, a_2 \otimes b_2\rangle_{A\otimes B}\end{aligned}$$

As the simple tensors span a dense subset of $A \otimes B$, we find that $(L_A \otimes L_B)^* = L_A^* \otimes L_B^*$ on a dense subspace of $A \otimes B$. By continuity, we can extend this to all of $A \otimes B$.

Statement 9 follows directly from 8. Statements 10 and 11 follow directly from 7 and 8.

So, now it remains to prove statement 1 without making use of any of the other statements. To that end, let $a \in A$ and $b \in B$. We find, using property 4 of Theorem 2.1.12:

$$\begin{aligned}\sup_{\substack{a\in A, b\in B\\ a\neq 0, b\neq 0}} \frac{\|(L_A \otimes L_B)(a \otimes b)\|_{A\otimes B}}{\|a \otimes b\|_{A\otimes B}} &= \sup_{\substack{a\in A, b\in B\\ a\neq 0, b\neq 0}} \frac{\|L_A a \otimes L_B b\|_{A\otimes B}}{\|a \otimes b\|_{A\otimes B}} = \sup_{\substack{a\neq 0\\ a\in A}} \frac{\|L_A a\|_A}{\|a\|_A} \cdot \sup_{\substack{b\in B\\ b\neq 0}} \frac{\|L_B b\|_B}{\|b\|_B} \\ &= \|L_A\|_{\mathcal{B}(A)} \cdot \|L_B\|_{\mathcal{B}(B)}\end{aligned}$$

Hence, we directly obtain:

$$\begin{aligned}\|L_A \otimes L_B\|_{\mathcal{B}(A\otimes B)} &= \sup_{\substack{x\in A\otimes B\\ x\neq 0}} \frac{\|(L_A \otimes L_B)x\|_{A\otimes B}}{\|x\|_{A\otimes B}} \geq \sup_{\substack{a\in A, b\in B\\ a\neq 0, b\neq 0}} \frac{\|(L_A \otimes L_B)(a \otimes b)\|_{A\otimes B}}{\|a \otimes b\|_{a\otimes b}} \\ &= \|L_A\|_{\mathcal{B}(A)} \cdot \|L_B\|_{\mathcal{B}(B)}\end{aligned}$$

This proves one inequality. For the other, we take $N \in \mathbb{N}$ and finite sequences $(a_j)_{j=1}^N$ in A and $(b_j)_{j=1}^N$ in B, such that:

$$x = \sum_{j=1}^{N} a_j \otimes b_j$$

Note that we can apply Gramm-Schmidt orthogonalisation to the a_j's, and absorb all the scalar factors into the b_j's by bilinearity. So, without loss of generality, we can assume that the sequence $(a_j)_{j=1}^N$ is orthogonal in A. Then, we find that the sequences $(a_j \otimes b_j)_{j=1}^N$ and $(a_j \otimes L_B b_j)_{j=1}^N$ are orthogonal in $A \otimes B$. Hence, we obtain:

$$\begin{aligned}\|(I \otimes L_B)x\|_{A\otimes B}^2 &= \left\|\sum_{j=1}^N a_j \otimes L_B b_j\right\|_{A\otimes B}^2 = \sum_{j=1}^N \|a_j \otimes L_B b_j\|_{A\otimes B}^2 \leq \sum_{j=1}^N \|a_j\|_A^2 \cdot \|L_B\|_{\mathcal{B}(B)}^2 \cdot \|b_j\|_B^2 \\ &= \|L_B\|_{\mathcal{B}(B)}^2 \cdot \sum_{j=1}^N \|a_j \otimes b_j\|_{A\otimes B}^2 = \|L_B\|_{\mathcal{B}(B)}^2 \cdot \left\|\sum_{j=1}^N a_j \otimes b_j\right\|_{A\otimes B}^2 = \|L_B\|_{\mathcal{B}(B)}^2 \cdot \|x\|_{A\otimes B}^2\end{aligned}$$

As we took x arbitrarily among the linear combinations of simple tensors in $A \otimes B$, which form a dense subspace of $A \otimes B$ by construction, we obtain:

$$\|I \otimes L_B\|_{\mathcal{B}(A\otimes B)} \leq \|L_B\|_{\mathcal{B}(B)}$$

Similarly, we prove that $\|L_A \otimes I\|_{\mathcal{B}(A\otimes B)} \leq \|L_A\|_{\mathcal{B}(A)}$. But then, we find that $(L_A \otimes L_B)$ is a bounded operator on the linear subspace of $A \otimes B$ spanned by all simple tensors. This space is dense in $A \otimes B$ by construction, and hence we obtain that $L_A \otimes L_B$ is a bounded operator acting on all of $A \otimes B$. This allows us to make the following norm estimate:

$$\begin{aligned}\|L_A \otimes L_B\|_{\mathcal{B}(A\otimes B)} &= \|(L_A \otimes I)(I \otimes L_B)\|_{\mathcal{B}(A\otimes B)} \leq \|L_A \otimes I\|_{\mathcal{B}(A\otimes B)} \cdot \|I \otimes L_B\|_{\mathcal{B}(A\otimes B)} \\ &\leq \|L_A\|_{\mathcal{B}(A)} \cdot \|L_B\|_{\mathcal{B}(B)}\end{aligned}$$

This completes the proof. □

Finally, we have a look at the tensor product of the finite-dimensional complex Hilbert spaces $\mathbb{C}^n$ and $\mathbb{C}^m$, for $n, m \in \mathbb{N}$. Obviously, we have $\dim(\mathbb{C}^n) = n$ and $\dim(\mathbb{C}^m) = m$. Hence, employing property 11 of Theorem 2.1.12 allows us to conclude that $\dim(\mathbb{C}^n \otimes \mathbb{C}^m) = nm$. Now notice that $\dim(\mathbb{C}^{nm}) = nm$. Hence, by employing Theorem 2.1.11, we find $\mathbb{C}^n \otimes \mathbb{C}^m \simeq \mathbb{C}^{nm}$.

Theorem 2.1.11 does not supply us with an explicit isomorphism, though. However, we can explicitly construct one, as follows:

$$\psi : \mathbb{C}^n \otimes \mathbb{C}^m \to \mathbb{C}^{nm}, \qquad \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ b_m \end{bmatrix} \mapsto \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ \vdots \\ a_1 b_m \\ a_2 b_1 \\ \vdots \\ a_n b_m \end{bmatrix}$$

This map is known as the *Kronecker product of vectors*. This map is so frequently used that we usually replace the map symbol with an equality. That is, we identify the spaces $\mathbb{C}^n \otimes \mathbb{C}^m$ with the space $\mathbb{C}^{nm}$, in the following manner:

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ \vdots \\ a_1 b_m \\ a_2 b_1 \\ \vdots \\ a_n b_m \end{bmatrix}$$

This has important implications when we consider the operators on these spaces. Suppose, namely, that we have matrices $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{m\times m}$. Using the above identification, one can easily deduce from sparse matrix considerations that we must make the following identification as well:

$$
\begin{aligned}
A \otimes B &= \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \otimes \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1m} \\ B_{21} & B_{22} & \cdots & B_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mm} \end{bmatrix} \\
&= \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} & \cdots & A_{11}B_{1m} & A_{12}B_{11} & \cdots & A_{1n}B_{1m} \\ A_{11}B_{21} & A_{11}B_{22} & \cdots & A_{11}B_{2m} & A_{12}B_{21} & \cdots & A_{1n}B_{2m} \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ A_{11}B_{m1} & A_{11}B_{m2} & \cdots & A_{11}B_{mm} & A_{12}B_{m1} & \cdots & A_{1n}B_{mm} \\ A_{21}B_{11} & A_{21}B_{12} & \cdots & A_{21}B_{1m} & A_{22}B_{11} & \cdots & A_{2n}B_{1m} \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ A_{n1}B_{m1} & A_{n1}B_{m2} & \cdots & A_{n1}B_{mm} & A_{n2}B_{m1} & \cdots & A_{nn}B_{mm} \end{bmatrix}
\end{aligned}
$$

This is known as the *Kronecker product of matrices*. Hence, we have constructed a very concrete way to think about tensor products on finite-dimensional spaces. This matrix-view of things will help us in the chapters to come, but we will not encounter it again in the remainder of this one.

This completes our discussion about the mathematical background necessary that one needs to understand the postulates of quantum mechanics. Thus, we are now in good position to introduce the postulates. This is what the next section is all about.

2.2 The postulates of quantum mechanics

Now that we have developed the necessary mathematical background, we are ready to turn our attention to the basics of quantum mechanics. Similarly as in [NC00], we will summarize the basic principles of quantum mechanics into 4 postulates (though we will permute the order). We will go through these 4 postulates one by one, and following every postulate, we will try to develop some intuition.

There are points in the theory where there is still some ambiguity, i.e. where no general consensus on the *correct* formulation of the theory seems to have been reached. Whenever we encounter such instances, we will briefly elaborate on the discrepancy that exists between the different formulations. Luckily, none of these instances are relevant for the field of quantum computing, so we will not be concerned with attempting to resolve these issues. On the contrary, we will generally take the easiest option, leaving the options that require more involved mathematics for more complete introductory texts on quantum mechanics from a mathematical perspective.

So, let's start with the first postulate.

Postulate 2.2.1: State space
Associated to any isolated physical system, there is a separable[a] complex Hilbert space $\mathcal{H}$, referred to as the *state space* of the system. The state of the physical system is completely described by a unit vector in this state space, which we refer to as a *state vector*, or simply a *state*.

[a] The literature is not completely clear about whether the state space is required to be separable. [NC00] makes no reference to the separability condition whatsoever, whereas in the Dirac-Von Neumann axioms, it is a necessary condition. In this text, we will mainly be concerned with finite-dimensional Hilbert spaces, hence separability will be a requirement that is automatically fulfilled. Hence, assuming that the state space is separable does not lead to any problems for the remainder of this text.

First of all, we will elaborate a little bit on the notation that we use to indicate vectors in state spaces. Within the realm of quantum mechanics, it is commonplace to denote states, i.e., unit vectors in the state space, by the ket-symbol $|\cdot\rangle$, where the dot is replaced by some unique symbol, used to distinguish between states. Moreover, let's say that we have two such states, $|s\rangle$ and $|t\rangle$. Then, we denote their inner product by $\langle s|t\rangle$. Finally, we let $\langle s|$ be the element of the dual of the Hilbert space corresponding to $|s\rangle$. Hence, formally, we can write $\langle s| : \mathcal{H} \to \mathbb{C}, |t\rangle \mapsto \langle s|t\rangle$. Note that as the inner product is conjugate linear in the first variable, we obtain that $|s\rangle \mapsto \langle s|$ is a conjugate linear map, so in particular if $|\psi\rangle = \alpha\,|\phi_1\rangle + \beta\,|\phi_2\rangle$, we have $\langle\psi| = \overline{\alpha}\,\langle\phi_1| + \overline{\beta}\,\langle\phi_2|$.

Note that the postulate does not tell us what state space is related to what isolated physical system. Instead, it just tells us that *every* isolated physical system has a Hilbert space associated to it. In a sense, it defines a framework in which we can describe nature, but it does not tell us how we should use it.

In order to develop some intuition, let's specialize to an exemplary state space. For example, consider a particle that is free to move along a one-dimensional infinite axis. Let's say, for the sake of simplicity, that the state space of this particle is $L^2(\mathbb{R})$. Then, all states the particle can be in, are precisely the functions $f : \mathbb{R} \to \mathbb{C}$, such that the square of the absolute value of this function integrates to 1, i.e.:

$$\int_{-\infty}^{\infty} |f(x)|^2 \,\mathrm{d}x = 1$$

For future reference, let's introduce two possible states of the free particle:

$$|s_+\rangle : x \mapsto e^{-\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_+}(x) \qquad \text{and} \qquad |s_-\rangle : x \mapsto e^{\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_-}(x)$$

Note that these vectors in $L^2(\mathbb{R})$ indeed have unit length, which we can easily check:

$$\langle s_+|s_+\rangle = \int_{-\infty}^{\infty} \left(e^{-\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_+}(x)\right)^2 \,\mathrm{d}x = \int_{-\infty}^{\infty} e^{-x}\,\mathbb{1}_{\mathbb{R}_+}(x)\,\mathrm{d}x = \int_0^{\infty} e^{-x}\,\mathrm{d}x = 0 - (-1) = 1$$
$$\langle s_-|s_-\rangle = \int_{-\infty}^{\infty} \left(e^{\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_-}(x)\right)^2 \,\mathrm{d}x = \int_{-\infty}^{\infty} e^{x}\,\mathbb{1}_{\mathbb{R}_-}(x)\,\mathrm{d}x = \int_{-\infty}^{0} e^{x}\,\mathrm{d}x = 1 - 0 = 1$$

By itself, it is not at all clear how we should interpret the statement that "the particle is in a state $|s_+\rangle$". We will elaborate on this a little bit more after having introduced the next postulate.

Postulate 2.2.2: Measurement
Associated to any measurement of a physical system with corresponding state space $\mathcal{H}$, is a set of operators $\{M_i\}_{m\in I}$ acting on $\mathcal{H}$ which satisfies the equation (known as the completion relation):

$$\sum_{m\in I} M_m^* M_m = \mathbb{1}_{\mathcal{H}}$$

where the set of measurement outcomes, I, can be any set.[a] If the system is in state $|\psi\rangle \in \mathcal{H}$, the probability that one measures the outcome $m \in I$, is given by:

$$\mathbb{P}(m) = \langle\psi|\,M_m^* M_m\,|\psi\rangle$$

If prior to measurement, the physical system was in state $|\psi\rangle \in \mathcal{H}$, and the measurement outcome was $m \in I$, then the resulting state of the system, directly after measurement, is given by:

$$\frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|\,M_m^* M_m\,|\psi\rangle}}$$

[a] Note that if I is uncountable it is not directly obvious how this summation is to be interpreted. One can make this explicit using *limits of nets*. As in quantum computing one only encounters finite sums of measurement operators, we will not be concerned with this technicality here.

Let's attempt to develop some intuition behind this postulate by looking at it from the example of the particle moving along an infinitely long straight axis, parametrized by $\mathbb{R}$. Recall that we said that its state space is given by $L^2(\mathbb{R})$.

As we said that the particle is free to move along the one-dimensional straight axis, it is natural to ask where the particle is. To that end, suppose that we want to figure out whether the particle is located in some interval $J = (a, b) \subseteq \mathbb{R}$. Then, we can devise a measurement operation, which will tell us whether the particle is located in this interval, or outside of it. Hence, we define our set of measurement outcomes to be $\{i, o\}$, where i denotes that the particle is inside the interval J, and o denotes that the particle is outside of it.

Next, we must define the corresponding measurement operators M_i and M_o, which act on $L^2(\mathbb{R})$. These, we will define as follows:

$$(M_i f)(x) = f(x) \cdot \mathbb{1}_J(x) \qquad \text{and} \qquad (M_o f)(x) = f(x) \cdot \mathbb{1}_{\mathbb{R}\setminus J}(x)$$

First of all, let's check that these measurement operators satisfy the completeness relation. To that end, first of all observe that $M_i^* = M_i$ and $M_o^* = M_o$, as we are dealing with multiplication operators on $L^2(\mathbb{R})$. Hence, we find, for all $f \in L^2(\mathbb{R})$:

$$(M_i^* M_i + M_o^* M_o) f(x) = M_i^2 f(x) + M_o^2 f(x) = \mathbb{1}_J(x)^2 f(x) + \mathbb{1}_{\mathbb{R}\setminus J}(x)^2 f(x) = f(x)$$

So, indeed, $M_i^* M_i + M_o^* M_o = \mathbb{1}_{\mathcal{H}}$, hence the completeness relation is satisfied.

Now, we can employ Postulate 2.2.2 and calculate the probability that our measurement outcome indicates the particle is located somewhere in the interval J, whenever its state is $|\psi\rangle = f : \mathbb{R} \to \mathbb{C}$. Hence, we calculate:

$$\mathbb{P}(i) = \langle\psi| M_i^* M_i |\psi\rangle = \int_{-\infty}^{\infty} \overline{f(x)} \cdot (M_i^* M_i f)(x)\, \mathrm{d}x = \int_{-\infty}^{\infty} \overline{f(x)} \cdot f(x) \cdot \mathbb{1}_J(x)^2\, \mathrm{d}x = \int_J |f(x)|^2\, \mathrm{d}x$$

Similarly, the probability that the particle is located outside of the interval J, is given by:

$$\mathbb{P}(o) = \langle\psi| M_o^* M_o |\psi\rangle = \int_{-\infty}^{\infty} \overline{f(x)} (M_o^* M_o f)(x)\, \mathrm{d}x = \int_{-\infty}^{\infty} \overline{f(x)} \cdot f(x) \cdot \mathbb{1}_{\mathbb{R}\setminus J}(x)^2\, \mathrm{d}x = \int_{\mathbb{R}\setminus J} |f(x)|^2\, \mathrm{d}x$$

Note that both these probabilities indeed sum to 1, as they should:

$$\mathbb{P}(i) + \mathbb{P}(o) = \int_J |f(x)|^2\, \mathrm{d}x + \int_{\mathbb{R}\setminus J} |f(x)|^2\, \mathrm{d}x = \int_{-\infty}^{\infty} |f(x)|^2\, \mathrm{d}x = 1$$

Now, we are ready to make a crucial observation. If the state of the particle is a function whose function is identically 0 in the interval J, then the probability of measuring that the particle is located in the interval J is 0. However, the more "mass" of the function is located in the interval J, the higher the probability of measuring that the particle is located in that interval. So, the amplitude of the function tells us something about how likely it is that we will measure the particle to be in that location.

Let's make this a little bit more concrete. Suppose that the particle is in the state $|s_+\rangle$. If we perform a measurement to see whether the particle is located in the interval $(1, \infty)$, we obtain:

$$\mathbb{P}(\text{inside } (1,\infty)) = \int_1^{\infty} \left| e^{-\frac{1}{2}x} \mathbb{1}_{\mathbb{R}_+}(x) \right|^2 \mathrm{d}x = \int_1^{\infty} e^{-x}\, \mathrm{d}x = 0 - (-e^{-1}) = \frac{1}{e}$$

On the other hand, if we perform a measurement to figure out whether the particle is in the interval $(-\infty, 0)$, we obtain:

$$\mathbb{P}(\text{inside } (-\infty,0)) = \int_{-\infty}^{0} \left| e^{\frac{1}{2}x} \mathbb{1}_{\mathbb{R}_+}(x) \right|^2 \mathrm{d}x = \int_{-\infty}^{0} 0\, \mathrm{d}x = 0$$

So, before we perform any measurement, can we answer the question whether the particle is in the interval $(-\infty, 0)$? Yes, we can answer this question with *definitely not*, because the probability that our measurement will indicate that it is, is 0. Similarly, can we answer the question whether the particle is in the interval $(1, \infty)$? *No, we cannot answer this question.* All we can say is that *if* we were to measure whether the particle is in the interval $(1, \infty)$, we would get an affirmative answer with probability $1/e$, and a negative answer with probability $1 - 1/e$. In other words, the position of the particle is not a well-defined number. Instead, it is given by a probability distribution, induced by the state vector.

Now, let's say that the particle was in the state $|s_+\rangle$, that we performed a measurement to see whether it was in the interval $(1, \infty)$, and that we obtained an affirmative answer. Postulate 2.2.2 now prescribes what happens to the state of the particle. The state the particle is in after the measurement is, is given by:

$$\frac{M_{\text{inside } (1,\infty)} |s_+\rangle}{\sqrt{\langle s_+| M^*_{\text{inside } (1,\infty)} M_{\text{inside } (1,\infty)} |s_+\rangle}} = \frac{M_{\text{inside } (1,\infty)} |s_+\rangle}{\sqrt{\mathbb{P}(\text{inside } (1,\infty))}} = \frac{\mathbb{1}_{(1,\infty)}(x)\mathbb{1}_{\mathbb{R}_+}(x)e^{-\frac{1}{2}x}}{\sqrt{\frac{1}{e}}} = e^{-\frac{1}{2}(x-1)}\mathbb{1}_{(1,\infty)}(x)$$

So, we can see that the state is altered by our measurement. This is a very important concept in quantum mechanics: *in general one cannot perform a measurement without influencing the state of the physical system that is being measured.* In the particular example of the particle moving along the infinite axis, this means that we can never recover the entire state of the particle by subsequent measurement, because the state is already altered after our first measurement.

Finally, suppose that our particle starts out in the state $|s_+\rangle$, and we measure whether it is located in the interval $(1, \infty)$ twice directly subsequently, i.e., there is no time in between the two measurements. Will we get the same answer? We already know that if after the first measurement, we obtain that it is in the interval $(1, \infty)$, then the state after the first measurement is:

$$e^{-\frac{1}{2}(x-1)}\mathbb{1}_{(1,\infty)}(x)$$

So, now the probability that our second measurement yields that the particle is located somewhere in the interval $(1, \infty)$ is given by:

$$\mathbb{P}(\text{inside } (1,\infty)) = \int_1^\infty |e^{-\frac{1}{2}(x-1)}\mathbb{1}_{(1,\infty)}(x)|^2 \, \mathrm{d}x = \int_1^\infty e^{-(x-1)} \, \mathrm{d}x = 0 - (-1) = 1$$

And hence *for sure*, our second measurement will reaffirm what the first measurement already told us, namely that the particle is indeed located in the interval $(1, \infty)$. Hence, we can think about this as if *any measurement alters the state in such a way that any subsequent measurement performed instantly thereafter will always reaffirm the outcome of the first measurement.*

All the concepts highlighted in the discussion lie at the heart of quantum mechanics, and hence are vital if one wants to grasp the implications of the theory. Because these observations play such a central role in the field, we have conveniently enumerated the most important implications of the first two postulates of quantum mechanics in the theorem below.

Theorem 2.2.3: Implications of the first two postulates

1. Suppose that we perform any measurement of any system. Then the probabilities of the outcomes sum to 1, affirming that we indeed always get exactly one outcome.
2. Suppose that we have two isolated physical systems with identical state spaces, which we will refer to as $\mathcal{H}$. Suppose that one is in a state $|\psi\rangle$, and the other is in a state $e^{i\phi}|\psi\rangle$, where $\phi \in [0, 2\pi)$. Then, identical measurements will yield identical probabilities on the outcomes, when applied to either of the systems. In particular, we are not able to distinguish between two states that differ by a global phase factor.
3. Suppose we have a set of measurement operators $\{M_m\}_{m\in I}$ such that for all $m \in I$, we have $M_m^2 = M_m$. Suppose we were to perform this measurement twice directly after each other, i.e. with no time in between. Then the outcome of the second measurement will always affirm the outcome of the first one.

Proof. We start with statement 1. To that end, suppose we have a set of measurement operators $\{M_m\}_{m\in I}$ which we use to measure a system that is in state $|\psi\rangle \in \mathcal{H}$. Then we obtain, using linearity and the completeness relation:

$$\sum_{m\in I} \mathbb{P}(m) = \sum_{m\in I} \langle\psi| M_m^* M_m |\psi\rangle = \langle\psi| \left(\sum_{m\in I} M_m^* M_m\right) |\psi\rangle = \langle\psi| \mathbb{1}_{\mathcal{H}} |\psi\rangle = \langle\psi|\psi\rangle = 1$$

This proves statement 1.

Secondly, we focus on statement 2. To that end, suppose that $\{M_m\}_{i\in I}$ is a set of measurement operators acting on $\mathcal{H}$. Suppose that the first system is in state $|\psi\rangle \in \mathcal{H}$, and that the second system is in state $e^{i\phi}|\psi\rangle$. Let $m \in I$. The probability that we obtain outcome m when measuring the first system is give by:

$$\mathbb{P}(\text{measurement of system 1 yields } m) = \langle\psi| M_m^* M_m |\psi\rangle$$

Similarly, we find that the probability of obtaining measurement outcome m when measuring the second system is:

$$\begin{aligned}\mathbb{P}(\text{measurement of system 2 yields } m) &= \langle\psi| e^{-i\phi} M_m^* M_m e^{i\phi} |\psi\rangle = e^{-i\phi} e^{i\phi} \langle\psi| M_m^* M_m |\psi\rangle \\ &= \langle\psi| M_m^* M_m |\psi\rangle\end{aligned}$$

Here, the minus sign in the exponential comes from the fact that the map $|s\rangle \mapsto \langle s|$ is conjugate linear. Hence, we find that both probabilities are indeed equal. As this is the case for all measurement outcomes and for all measurement operations, we obtain that we can never tell the state of the two systems apart.

Finally, we prove statement 3. To that end, suppose that the set of measurement operators acts on a state space $\mathcal{H}$, and suppose that the system starts out in state $|\psi\rangle$. Next, suppose that we apply the measurement corresponding to the measurement operators twice, in such a way that there is no time in between the measurements. If the first measurement yielded outcome $m \in I$, then the state directly after the first measurement is given by:

$$|\psi_m\rangle = \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^* M_m |\psi\rangle}}$$

Now, we obtain that the probability of obtaining m as outcome of the second measurement can be calculated to be:

$$\frac{\langle\psi| M_m^*}{\sqrt{\langle\psi| M_m^* M_m |\psi\rangle}} M_m^* M_m \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^* M_m |\psi\rangle}} = \frac{\langle\psi| (M_m^2)^* M_m^2 |\psi\rangle}{\langle\psi| M_m^* M_m |\psi\rangle} = \frac{\langle\psi| M_m^* M_m |\psi\rangle}{\langle\psi| M_m^* M_m |\psi\rangle} = 1$$

And hence for sure, the second measurement will affirm the outcome of the first measurement. □

The requirement that we impose in the third statement of this theorem seems rather arbitrary. We will see in Section 2.3, though, that the measurements that obey this relation actually form a very important class of measurements.

Now that we have developed some intuition about what it means for an isolated physical system to be in a particular state, we are ready to consider what happens when we have multiple isolated physical systems.

Postulate 2.2.4: Composition
Suppose that we have two isolated physical systems, having corresponding state spaces $\mathcal{H}_1$ and $\mathcal{H}_2$. Then the combined physical system is automatically isolated, and its state space is given by $\mathcal{H}_1 \otimes \mathcal{H}_2$. Moreover, if the first physical system is in state $|h_1\rangle$ and the second physical system is in state $|h_2\rangle$, then the state of the composite system is $|h_1\rangle \otimes |h_2\rangle$.

Often, the notation $|h_1\rangle \otimes |h_2\rangle$ is abbreviated to $|h_1\rangle |h_2\rangle$ or even $|h_1, h_2\rangle$. We will use the above forms interchangeably throughout the remainder of this text.

Definition 2.2.5: Entanglement
Suppose that we have a composite system of two isolated physical systems. Then, we say that the isolated physical systems are *entangled*, when we cannot write the state of the composite system as a simple tensor.

Let's consider an example of a composite system. Suppose that we have two axes (one might picture them as parallel lines extending off to infinity), along which two particles move independently. Then, the corresponding state space of the combined system of these particles is $L^2(\mathbb{R}) \otimes L^2(\mathbb{R})$. Next, suppose that the first particle is in state $|s_+\rangle$, and the second particle is in state $|s_-\rangle$, which we defined earlier. Then, the state of the composite system is $|s_+\rangle \otimes |s_-\rangle = |s_+\rangle\,|s_-\rangle = |s_+, s_-\rangle$. Note that this is again a vector of unit length in the vector space $L^2(\mathbb{R}) \otimes L^2(\mathbb{R})$, as:

$$(\langle s_+| \otimes \langle s_-|)(|s_+\rangle \otimes |s_-\rangle) = \langle s_+|s_+\rangle \cdot \langle s_-|s_-\rangle = 1 \cdot 1 = 1$$

So, in this situation, the state of the composite system is a simple tensor. This implies that the composite system is not entangled, i.e., that the state of the composite system is not an entangled state.

Let's now consider another state that the composite system can be in. We consider the following state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|s_+, s_+\rangle + |s_-, s_-\rangle\right)$$

Let's first of all check that this is indeed a unit vector in $L^2(\mathbb{R}) \otimes L^2(\mathbb{R})$. To that end, we first of all calculate:

$$\langle s_-|s_+\rangle = \int_{-\infty}^{\infty} \overline{e^{\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_-}(x)} \cdot e^{-\frac{1}{2}x}\mathbb{1}_{\mathbb{R}_+}(x)\,\mathrm{d}x = 0$$

Thus, we can now calculate the length of the vector $|\psi\rangle$ in $L^2(\mathbb{R}) \otimes L^2(\mathbb{R})$:

$$\begin{aligned}
\langle\psi|\psi\rangle &= \frac{1}{2}\left(\langle s_+, s_+| + \langle s_-, s_-|\right)\left(|s_+, s_+\rangle + |s_-, s_-\rangle\right) \\
&= \frac{1}{2}\left(\langle s_+, s_+|s_+, s_+\rangle + \langle s_+, s_+|s_-, s_-\rangle + \langle s_-, s_-|s_+, s_+\rangle + \langle s_-, s_-|s_-, s_-\rangle\right) \\
&= \frac{1}{2}\left(\langle s_+|s_+\rangle\,\langle s_+|s_+\rangle = \langle s_+|s_-\rangle\,\langle s_+|s_-\rangle + \langle s_-|s_+\rangle\,\langle s_-|s_+\rangle + \langle s_-|s_-\rangle\,\langle s_-|s_-\rangle\right) \\
&= \frac{1}{2}\left(1 + 0 + 0 + 1\right) = 1
\end{aligned}$$

Hence, we find that $|\psi\rangle$ indeed is a state that the composite system of two particles moving along infinitely long straight axes can be in. As $|s_+\rangle$ and $|s_-\rangle$ are not scalar multiples of each other, though, $|\psi\rangle$ is not a simple tensor. Hence, we find that $|\psi\rangle$ is an entangled state, and that if the composite system is in state $|\psi\rangle$, then it is entangled.

An interesting question that now arises is the following. Suppose that the composite system is in the state $|\psi\rangle$ defined above, what is the state of the first particle? The only correct answer to this question is that the question is invalid. If a composite system is in an entangled state, it no longer makes sense to talk about the states of the individual components. This can only be done when the system is not in an entangled state.[3]

Measurements on composite systems obey the same rules that are prescribed in Postulate 2.2.2. Let's consider an example. Again, suppose that the composite system of two freely moving particles along two axes starts out in the state $|\psi\rangle$. Let us measure whether the second particle is located somewhere in the

[3]Sometimes, somewhat sloppily, reference is made to the state of the first particle of an entangled system. What is meant here is the *partial state*, which is found by taking a partial trace of the density operator. We will not go into detail about this here, but we refer the interested reader to [NC00], section 2.4.3.

interval $(0,\infty)$. Again, the measurement outcomes are $\{\text{inside } (0,\infty), \text{outside } (0,\infty)\}$. The corresponding measurement operators are:

$$M_{\text{inside }(0,\infty)} = \mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+} \qquad \text{and} \qquad M_{\text{outside }(0,\infty)} = \mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_-}$$

Note that indeed these satisfy the completeness relation:

$$M_{\text{inside }(0,\infty)} + M_{\text{outside }(0,\infty)} = \mathbb{1}_{\mathbb{R}} \otimes (\mathbb{1}_{\mathbb{R}_+} + \mathbb{1}_{\mathbb{R}_-}) = \mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}} = \mathbb{1}_{L^2(\mathbb{R})\otimes L^2(\mathbb{R})}$$

Now, we can calculate the probability that we obtain that the second particle is located in the interval $(0,\infty)$:

$$\begin{aligned}
\mathbb{P}(\text{inside } (0,\infty)) &= \langle\psi| M^*_{\text{inside}(0,\infty)} M_{\text{inside}(0,\infty)} |\psi\rangle \\
&= \frac{1}{\sqrt{2}}(\langle s_+, s_+| + \langle s_-, s_-|)(\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+})^*(\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) \cdot \frac{1}{\sqrt{2}}(|s_+, s_+\rangle + |s_-, s_-\rangle) \\
&= \frac{1}{2}(\langle s_+, s_+| + \langle s_-, s_-|)(\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+})(|s_+, s_+\rangle + |s_-, s_-\rangle) \\
&= \frac{1}{2}\big(\langle s_+, s_+| (\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_+, s_+\rangle + \langle s_+, s_+| (\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_-, s_-\rangle \\
&\quad + \langle s_-, s_-| (\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_+, s_+\rangle + \langle s_-, s_-| (\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_-, s_-\rangle\big) \\
&= \frac{1}{2}\big(\langle s_+| \mathbb{1}_{\mathbb{R}} |s_+\rangle \langle s_+| \mathbb{1}_{\mathbb{R}_+} |s_+\rangle + \langle s_+| \mathbb{1}_{\mathbb{R}} |s_-\rangle \langle s_+| \mathbb{1}_{\mathbb{R}_+} |s_-\rangle \\
&\quad + \langle s_-| \mathbb{1}_{\mathbb{R}} |s_+\rangle \langle s_-| \mathbb{1}_{\mathbb{R}_+} |s_+\rangle + \langle s_-| \mathbb{1}_{\mathbb{R}} |s_-\rangle \langle s_-| \mathbb{1}_{\mathbb{R}_+} |s_-\rangle\big) \\
&= \frac{1}{2}\left(\langle s_+|s_+\rangle \cdot \langle s_+|s_+\rangle + \langle s_+|s_-\rangle \cdot \langle s_+| 0 + \langle s_+|s_+\rangle \cdot \langle s_-|s_+\rangle + \langle s_-|s_-\rangle \cdot \langle s_-| 0\right) \\
&= \frac{1}{2}(1 + 0 + 0 + 0) = \frac{1}{2}
\end{aligned}$$

So, we find that with probability $\frac{1}{2}$, we measure the second particle to be located in the positive part of the axis. As the probabilities must sum to 1, we find that the probability that it is located in the negative part of the axis must be equal to $\frac{1}{2}$ as well.

Now, suppose that the measurement yields that the second particle is indeed located in the positive part of the axis along which it is free to move. What will be the state of the composite system after the measurement? Postulate 2.2.2 provides us with the tools to calculate this, and we find that the resulting state of the composite system can be calculated as follows:

$$\begin{aligned}
\frac{M_{\text{inside }(0,\infty)} |\psi\rangle}{\sqrt{\langle\psi| M^*_{\text{inside }(0,\infty)} M_{\text{inside }(0,\infty)} |\psi\rangle}} &= \frac{(\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |\psi\rangle}{\sqrt{\mathbb{P}(\text{inside } (0,\infty))}} = \frac{1}{\sqrt{\frac{1}{2}}} \cdot \big(\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}\big) \cdot \frac{1}{\sqrt{2}}(|s_+, s_+\rangle + |s_-, s_-\rangle) \\
&= \sqrt{2} \cdot \frac{1}{\sqrt{2}}\big((\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_+, s_+\rangle + (\mathbb{1}_{\mathbb{R}} \otimes \mathbb{1}_{\mathbb{R}_+}) |s_-, s_-\rangle\big) \\
&= \mathbb{1}_{\mathbb{R}} |s_+\rangle \otimes \mathbb{1}_{\mathbb{R}_+} |s_+\rangle + \mathbb{1}_{\mathbb{R}} |s_-\rangle \otimes \mathbb{1}_{\mathbb{R}_+} |s_-\rangle \\
&= |s_+\rangle \otimes |s_+\rangle + |s_-\rangle \otimes 0 = |s_+, s_+\rangle
\end{aligned}$$

So, we find that the resulting state is a simple tensor. Hence, the measurement has completely eradicated the entanglement. This is a common observation in quantum mechanics: *measurement can break entanglement.*

Even more interestingly, before we performed the measurement, the composite system was in an entangled state, and hence it did not make sense to talk about the state of the first particle. Now, however, even though we only measured the second system and did not touch the first one, we are all of a sudden able to talk about the state of the first system, as the composite system is no longer entangled. Moreover, if we had measured whether the first particle was in the interval $(0,\infty)$ before we performed the measurement we analyzed above, we would have obtained the result inside $(0,\infty)$ with probability $\frac{1}{2}$. Now, though, if

the second particle turned out to be inside $(0, \infty)$, we would measure the first particle to be in the interval $(0, \infty)$ with probability 1. So, the measurement of the second system has influenced the probability of the measurement outcomes of measurements performed on the first system.

The above postulate allows us to describe combined states of multiple systems. So, we can think of Postulate 2.2.4 as allowing us to scale up the descriptive power of quantum mechanics in the spatial dimensions. With this view, the next postulate is a natural counterpart, as it allows us to scale up our theory in time.

Postulate 2.2.6: Evolution
The *evolution* of a closed physical system is described by a unitary transformation of its state space. In other words, let $|\psi_t\rangle$ be the state of some closed physical system at time t. Then we have that for any times t_1 and t_2, the states $|\psi_{t_1}\rangle$ and $|\psi_{t_2}\rangle$ are related by:

$$|\psi_{t_1}\rangle = U\,|\psi_{t_2}\rangle$$

where U only depends on t_1 and t_2.

Again, suppose that we consider one particle that is moving along an infinitely long straight axis, whose state space is $L^2(\mathbb{R})$. An example of a unitary transformation on this state space is τ_1, the operation that shifts a given function $f : \mathbb{R} \to \mathbb{C}$ to the right by 1, so $(\tau_1 f)(x) = f(x-1)$. Hence, it could happen that if the free particle is in state f at some time t_1, then at time t_2 the particle is in state $\tau_1 f$, i.e., its state has moved along the infinite horizontal axis one unit to the right. As the operator τ_1 is indeed a unitary operator, we obtain that the new, shifted function has the same norm as the original one. Hence, it still has unit length, which ensures that it is again a state.

Observe that Postulate 2.2.6 does not tell us *how* the states at times t_1 and t_2 are related, but only that they are related by *some* unitary operator. Figuring out what physical systems correspond to what evolution operators is a very interesting and very delicate problem indeed. We will only hint at how one can solve this problem here. Associated to every closed physical system is what is known as a *Hamiltonian*, which is an operator acting on the state space of the closed physical system. Once one know this Hamiltonian, one can use *Schrödinger's equation* to obtain the evolution operator between two given times. In practice, figuring out the Hamiltonian is a very difficult problem, and then obtaining the evolution operator is not at all trivial either. More on this can e.g. be found in [NC00], or [Gri16].

This completes our discussion about the postulates of quantum mechanics. In the subsequent sections, we will build some theory around these postulates such that we can conveniently refer to it later on in this text.

2.3 Projective measurements

In the previous section, we have defined how we can describe a measurement of a physical system, using the theory of quantum mechanics. It turns out that throughout this text, we are mainly concerned with a special subclass of these measurement operations, known as the projective measurements. These, we will define here, and subsequently, we will prove some of its properties.

Definition 2.3.1: Projective measurement
Let $\{M_m\}_{m \in I}$ be a set of measurement operators acting on a state space $\mathcal{H}$. Suppose that for all $m \in I$, we have that M_m is self-adjoint, and that for all $m, n \in I$, we have:

$$M_m M_n = \delta_{m,n} M_m$$

Then, we say that these are *orthogonal projective measurement operators*, and the corresponding measurement is called a *projective measurement*.

Such orthogonal projective measurement operators map every element of the state space onto some linear subspace of the state space. Suppose that the states $\{|\psi_1\rangle, \ldots, |\psi_n\rangle\}$ span this orthogonal subspace. Then, we can write the corresponding orthogonal projective measurement operator as:

$$\sum_{k=1}^{n} |\psi_k\rangle \langle\psi_k|$$

This is especially convenient if the orthogonal projective measurement operator has one-dimensional range, because then we can simply write it as

$$|\psi\rangle \langle\psi|$$

where $|\psi\rangle$ is a state (i.e., a unit vector) in its range.

Recall that we already encountered projective measurements before, in Theorem 2.2.3. There we proved the remarkable property that two consecutively executed identical measurements on the same system will always yield the same outcome, provided that there is no time in between the measurements.

A projective measurement where the outcomes are real numbers can be concisely represented into one operator acting on the state space of the physical system being measured. This operator is known as the observable, and it is defined as follows.

Definition 2.3.2: Observable
Suppose that we have an isolated physical system with corresponding state space $\mathcal{H}$, and a set of pairwise orthogonal projective measurement operators $\{M_m\}_{m\in I}$, where $I \subseteq \mathbb{R}$. Then, the *observable* corresponding to this measurement, $\mathcal{A}$, which is a (not necessarily bounded) operator on the state space $\mathcal{H}$, is defined as:

$$\mathcal{A} = \sum_{m\in I} mM_m$$

It is instructive to note that as all the M_m's are projection operators, we have effectively written $\mathcal{A}$ in its spectral decomposition.

Observables have some very convenient properties, which we highlight in the following theorem.

Theorem 2.3.3: Properties of observables
Let $\{M_m\}_{m\in I}$ be a set of orthogonal projective measurement operators, where $I \subseteq \mathbb{R}$, and let $\mathcal{A}$ be the corresponding observable. Then, we find:

1. $\mathcal{A}$ is self-adjoint.
2. The probability that we obtain measurement outcome m when we measure a physical system in state $|\psi\rangle$ is given by:
$$\mathbb{P}(m) = \langle\psi| M_m |\psi\rangle$$
3. Suppose that the physical system being measured is in state $|\psi\rangle$, and suppose that $|\psi\rangle$ is in the domain of $\mathcal{A}$. We denote the expected value of the measurement by $\mathbb{E}(\mathcal{A})$. Then we find that $\mathbb{E}(\mathcal{A})$ is finite, and:
$$\mathbb{E}(\mathcal{A}) = \langle\psi| \mathcal{A} |\psi\rangle$$

Proof. The first statement is not very difficult to check. We asserted in Definition 2.3.1 that for all $m \in I$, M_m is self-adjoint. Hence, we obtain:

$$\mathcal{A}^* = \left(\sum_{m\in I} mM_m\right)^* = \sum_{m\in I} mM_m^* = \sum_{m\in I} mM_m = \mathcal{A}$$

This proves 1. To prove 2, we simply use the properties that we imposed on M_m:

$$\mathbb{P}(m) = \langle\psi| M_m^* M_m |\psi\rangle = \langle\psi| M_m^2 |\psi\rangle = \langle\psi| M_m |\psi\rangle$$

Hence, we have proven statement 2, and we can now readily use it to prove statement 3:

$$\mathbb{E}(\mathcal{A}) = \sum_{m \in I} m\mathbb{P}(m) = \sum_{m \in I} m \langle\psi| M_m |\psi\rangle = \langle\psi| \left(\sum_{m \in I} m M_m \right) |\psi\rangle = \langle\psi| A |\psi\rangle$$

Here, we needed that $|\psi\rangle$ is in the domain of $\mathcal{A}$, because otherwise the right-hand side is not defined. This completes the proof. □

This completes our discussion about projective measurements. There is yet another way of looking at quantum measurements, by introducing what are known as POVMs, or *positive operator-valued measures*. This we will not need in the remainder of this text, and hence we will not cover it here. The interested reader, we refer to [NC00], section 2.2.6.

3 Quantum computing

In the previous chapter, we have introduced quantum mechanics in a very general setting. This means that we have provided the reader with some basic tools that can be used to understand the field of quantum mechanics as a whole.

In this chapter, we will specialize towards the field of quantum computing, which is the subfield of quantum mechanics concerned with performing calculations using the postulates described in the previous chapter. In this subfield, only finite-dimensional Hilbert spaces are of interest to us, so we shall abandon the idea of infinite-dimensional Hilbert spaces altogether from now on.

First of all, we will introduce the most fundamental notion in quantum computing, known as the qubit, in Section 3.1. Next, we will introduce the concept of quantum gates, which can be used to manipulate qubits, in Section 3.2. Afterwards, we will elaborate on how we can combine these quantum gates to form quantum circuits and quantum algorithms, in Section 3.3 and Section 3.4. Finally, in Section 3.5, we will cover the most important quantum circuits and quantum algorithms, which form the basic building blocks of the field of study.

3.1 Qubits

The most fundamental notion in the field of quantum computing is the *qubit*. First of all, we will cover single-qubit systems in Subsection 3.1.1. Then, we will generalize this to multiple-qubit systems in Subsection 3.1.2.

3.1.1 Single-qubit systems

We will start with its formal definition. Afterwards, we will attempt to develop some intuition behind this concept.

Definition 3.1.1: Qubit
Suppose that we have a physical system whose state space is 2-dimensional. Then, we refer to this physical system as a *single-qubit system*. The amount of information it can store, we refer to as a *qubit*. Moreover, to the state space of any single-qubit system, we associate an orthonormal basis of the state space, which we denote by $\{|0\rangle, |1\rangle\}$. This, we refer to as the *computational basis*, and its elements we refer to as the *computational basis states*. All states that are not a scalar multiple of any of the computational basis states, we refer to as *superposition states*.

The above definition can sound pretty abstract at first, so let's attempt to develop some intuition behind this concept. To that end, we will first of all consider the classical analogue to the qubit, known as the *bit*. The term *bit* is shorthand for *binary digit*, and recall that a binary digit is a number that is either 0 or 1.

We can use bits to store information. For instance, suppose that we have a door that can be either open or closed. We can associate a 0 with the fact that the door is closed, and similarly let's write a 1 whenever the door is open. Now, we have found a way to represent a certain amount of information, namely whether this door is open or closed, in a bit. Hence, storing whether a door is open or closed, amounts to storing one bit of information.

Now, let's relate this back to qubits. The term *qubit* is shorthand for *quantum bit*, indicating that it is indeed the quantum analogue of a classical bit.

Let's investigate how we would store whether a door is open or closed in a single-qubit system. For example, we could bring the single-qubit system in the state $|0\rangle$ whenever the door is closed, and similarly we could

ensure that the system is in state $|1\rangle$ if the door is open. Hence, we can store whether a door is open or closed in a qubit as well as in a classical bit.

However, there are many more states we can bring a single-qubit system into, aside from the two computational basis states $|0\rangle$ and $|1\rangle$ we discussed above. More precisely, as all states of a single-qubit system are unit vectors in the state space, which is spanned by the set of computational basis states, $\{|0\rangle, |1\rangle\}$, all states that a single-qubit system can be in, can be described by the following expression, with suitable choices for $\alpha, \beta \in \mathbb{C}$:

$$\alpha |0\rangle + \beta |1\rangle \qquad \text{with} \qquad |\alpha|^2 + |\beta|^2 = 1$$

Whenever both α and β are non-zero, the state $\alpha |0\rangle + \beta |1\rangle$ is referred to as a superposition state. On the other hand, if either α or β is zero, then one of the terms vanishes and hence the resulting state is either $\alpha |0\rangle$ and $\beta |1\rangle$. So, we refer to these states as scalar multiples of computational basis states.

Suppose now that we want to figure out whether our single-qubit system is in state $|0\rangle$ or $|1\rangle$. This is done by performing a measurement in the computational basis. We formally define this in the following definition.

Definition 3.1.2: Measurement in the computational basis
Let $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. These form a family of measurement operators $\{M_i\}_{i\in\{0,1\}}$ in the sense of Postulate 2.2.2. These measurement operators, we refer to as *computational basis measurement operators*. The corresponding measurement, we refer to as the *measurement in the computational basis*.

It is easily checked that the set of computational basis measurement operators satisfies the completeness relation, as specified in Postulate 2.2.2. That is, we have that $|0\rangle\langle 0| + |1\rangle\langle 1| = I$, the identity operator on the state space of the corresponding single-qubit system.

Let's attempt to develop some intuition behind this measurement in the computational basis. Suppose that we prepare a single-qubit system in the state $|0\rangle$, and perform a measurement in the computational basis. Postulate 2.2.2 now provides us with a way to evaluate the probabilities of the possible outcomes, where we use that $\{|0\rangle, |1\rangle\}$ is an orthonormal set of vectors:

$$\mathbb{P}(0) = \langle 0| M_0^* M_0 |0\rangle = \langle 0|0\rangle \langle 0|0\rangle \langle 0|0\rangle = 1 \qquad \text{and} \qquad \mathbb{P}(1) = \langle 0| M_1^* M_1 |0\rangle = \langle 0|1\rangle \langle 1|1\rangle \langle 1|0\rangle = 0$$

So, for sure, we will obtain the measurement outcome 0. If, in contrast, we prepare the single-qubit system in the state $|1\rangle$, we can evaluate the measurement outcome probabilities of the measurement in the computational basis in a similar manner:

$$\mathbb{P}(0) = \langle 1| M_0^* M_0 |1\rangle = \langle 1|0\rangle \langle 0|0\rangle \langle 0|1\rangle = 0 \qquad \text{and} \qquad \mathbb{P}(1) = \langle 1| M_1^* M_1 |1\rangle = \langle 1|1\rangle \langle 1|1\rangle \langle 1|1\rangle = 1$$

Hence, this time, we obtain the measurement outcome 1 with certainty. So, in conclusion, performing a measurement in the computational basis tells us with certainty, i.e., with success probability 1, in which computational basis state a single-qubit system is, if we know beforehand that its state is indeed a computational basis state. In this sense, a qubit behaves similarly to a bit, where we can usually simply read the state of the bit with success probability 1.

More interesting things happen when we measure a single-qubit system in the computational basis if the state of this system is not a computational basis state itself. In particular, suppose that the state of our single-qubit system is given by $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$, with $\alpha, \beta \in \mathbb{C}$. Of course, the state must be a unit vector, so we must have $|\alpha|^2 + |\beta|^2 = 1$. Now, observe that we can, similarly as before, evaluate the probability distribution over the outcomes of a measurement in the computational basis. Here, we use that $\langle\psi| = \overline{\alpha}\langle 0| + \overline{\beta}\langle 1|$.

$$\mathbb{P}(0) = \langle\psi| M_0^* M_0 |\psi\rangle = \langle\psi|0\rangle \langle 0|0\rangle \langle 0|\psi\rangle = \langle\psi|0\rangle \langle 0|\psi\rangle = (\overline{\alpha}\langle 0|0\rangle + \overline{\beta}\langle 1|0\rangle)(\alpha\langle 0|0\rangle + \beta\langle 0|1\rangle) = \overline{\alpha}\alpha = |\alpha|^2$$
$$\mathbb{P}(1) = \langle\psi| M_1^* M_1 |\psi\rangle = \langle\psi|1\rangle \langle 1|1\rangle \langle 1|\psi\rangle = \langle\psi|1\rangle \langle 1|\psi\rangle = (\overline{\alpha}\langle 0|1\rangle + \overline{\beta}\langle 1|1\rangle)(\alpha\langle 1|0\rangle + \beta\langle 1|1\rangle) = \overline{\beta}\beta = |\beta|^2$$

Thus, interestingly, if we measure a single-qubit system that is in the state $\alpha |0\rangle + \beta |1\rangle$ in the computational basis, then we obtain 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$.

Next, recall that a measurement in general influences the very state of the system being measured. So, let's investigate how the state of a single-qubit system is altered by performing a measurement in the computational basis. Again, we assume that the single-qubit system, prior to measurement, was in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, and suppose that the measurement in the computational basis yielded the measurement outcome 0. Then, Postulate 2.2.2 prescribes the resulting state of the single-qubit system directly after the measurement:

$$\frac{M_0|\psi\rangle}{\sqrt{\langle\psi|M_0^* M_0|\psi\rangle}} = \frac{|0\rangle\langle 0|\psi\rangle}{\sqrt{\mathbb{P}(0)}} = \frac{\alpha|0\rangle}{|\alpha|} = \frac{\alpha}{|\alpha|}|0\rangle$$

This resulting state is a scalar multiple of the computational basis state $|0\rangle$, and hence is no longer a superposition state. Similarly, if the result of the measurement had been 1, the resulting state directly after the measurement would be:

$$\frac{M_1|\psi\rangle}{\sqrt{\langle\psi|M_1^* M_1|\psi\rangle}} = \frac{|1\rangle\langle 1|\psi\rangle}{\sqrt{\mathbb{P}(1)}} = \frac{\beta|1\rangle}{|\beta|} = \frac{\beta}{|\beta|}|1\rangle$$

And hence in this case, the resulting state is a scalar multiple of the computational basis state $|1\rangle$. So, we conclude that after performing a measurement in the computational basis, the resulting state is always a scalar multiple of a computational basis state, and hence cannot be a superposition state. This is an important conclusion: *a measurement in the computational basis will always break the superposition.* In particular, this means that if we have a "quantum door" that was closed with probability $|\alpha|^2$ and open with probability $|\beta|^2$ prior to measurement, then, based on the measurement outcome, we can tell for sure whether it is closed or open after the measurement in the computational basis.[1]

This concludes our discussion on the development of intuition for single-qubit systems. The final piece of this subsection will be devoted to clearing up some notational issues, which will make life a lot easier in the sections to come.

First of all, we will identify the state space of any single-qubit system with $\mathbb{C}^2$. The existence of such an identification is justified as we can easily deduce from Theorem 2.1.11 that both spaces are isomorphic. Explicitly, observe that every state of a single-qubit system can be written as a 2-dimensional vector with respect to the computational basis introduced in Definition 3.1.1. That is, we make the following identification, for all $\alpha, \beta \in \mathbb{C}$:

$$\alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \alpha\mathbf{e}_0 + \beta\mathbf{e}_1$$

Here, $\{\mathbf{e}_0, \mathbf{e}_1\}$ is the Cartesian basis of $\mathbb{C}^2$. The following relations are special cases of this identification:

$$|0\rangle = \begin{bmatrix}1\\ 0\end{bmatrix} = \mathbf{e}_0 \qquad \text{and} \qquad |1\rangle = \begin{bmatrix}0\\ 1\end{bmatrix} = \mathbf{e}_1$$

In particular, observe that $|0\rangle$ is *not* the 0-vector. Rather, it is a vector of unit length in the state space of a single-qubit system. Observe that, with this identification, both computational basis state vectors of any single-qubit system are in one-to-one correspondence with the Cartesian basis vectors of $\mathbb{C}^2$.

Finally, we introduce the following special superposition states, which will be of use later on:

Definition 3.1.3: Special single-qubit superposition states
We define the following shorthand notation:

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

This concludes our discussion on single-qubit systems. Next, we will generalize the ideas that we developed in this subsection to multiple-qubit systems.

[1] Standard introductory texts use the example of "Schrödinger's cat" which is in a superposition of being dead or alive, rather than a quantum door that can be open or closed.

3.1.2 Multiple-qubit systems

In this section, we will look at multiple-qubit systems. Similar to the previous subsection, we will first of all provide a definition and some properties, and subsequently we will attempt to develop some intuition for them.

Definition 3.1.4: Multiple-qubit systems
Let $n \in \mathbb{N}$ and suppose that we have n single-qubit systems. Then the composition of all these n physical systems, we refer to as an *n-qubit system*. We refer to the states of this system as *n-qubit states*. Simple tensor products of computational basis states of the single-qubit systems, we refer to as *computational basis states*, and all states that are not a scalar multiple of any of the computational basis states, we refer to as *superposition states*.

So, recall that to each of the n single-qubit systems, we have associated a set of computational basis states $\{|0\rangle, |1\rangle\}$. The above definition now tells us that the computational basis states of the composite n-qubit system are all the states $|b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$, where $b_1, \ldots, b_n \in \{0, 1\}$.

Next, let's prove some properties of multiple-qubit systems.

Theorem 3.1.5: Properties of multiple-qubit systems
Let $n \in \mathbb{N}$ and suppose that we have n single-qubit systems, with corresponding state spaces $\mathcal{H}_1, \ldots, \mathcal{H}_n$. Then, the corresponding n-qubit system has the following properties:

1. The state space of the n-qubit system is $\mathcal{H} = \bigotimes_{k=1}^{n} \mathcal{H}_k$.
2. The set $\{|b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle : b_1, \ldots, b_n \in \{0, 1\}\}$ forms an orthonormal basis of $\mathcal{H}$.
3. $\dim(\mathcal{H}) = 2^n$ and $\mathcal{H} \simeq \mathbb{C}^{2^n}$.

Proof. The first statement follows directly from Postulate 2.2.4. The second statement is a direct consequence of Theorem 2.1.12, property 10. Statement 3 follows from statement 2, the definition of the dimension of a Hilbert space, and Theorem 2.1.11. □

At this point it is instructive to note that the dimension of the state space grows exponentially in the number of single-qubit systems that comprise the composite system. Indeed, if one has an n-qubit system, then the dimension of its state space is 2^n. This means that a state of an n-qubit system can be a superposition state of 2^n computational basis states. This fact forms the core idea in the exponential speedups that can be achieved with quantum computing.

There exists a variety of different notations for the computational basis states of multiple-qubit systems. These, we cover in the next definition.

Definition 3.1.6: Computational basis states of multiple-qubit systems
Let $n \in \mathbb{N}$ and $b_1, \ldots, b_n \in \{0, 1\}$. Define $j = (b_1 b_2 \cdots b_n)_2$, i.e. the number $j \in \{0, 1, \ldots, 2^n - 1\}$ whose binary expansion is given by $b_1 b_2 \cdots b_n$. Then, we identify the following notations to denote n-qubit computational basis states:

$$|j\rangle = |b_1 b_2 \cdots b_n\rangle = |b_1\rangle |b_2\rangle \cdots |b_n\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$$

Next, let $k \in \mathbb{Z}$, and suppose that $k \equiv j \mod 2^n$, where $0 \leq j < 2^n$. Then we identify the following notation for n-qubit states: $|k\rangle = |j\rangle$.

So, for example, by the 3-qubit state $|13\rangle$, we denote the following state:

$$|13\rangle = |5\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle$$

With the above definition, several ambiguities are possible. For example, the 2-qubit computational basis state $|-1\rangle$ is equal to $|3\rangle = |11\rangle = |1\rangle \otimes |1\rangle$, whereas the 3-qubit computational basi state $|-1\rangle$ is equal to

$|7\rangle = |111\rangle = |1\rangle \otimes |1\rangle \otimes |1\rangle$. Furthermore, consider the 4-qubit state $|11\rangle$. Then, we must interpret 11 as a decimal number, so we obtain $|11\rangle = |1011\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle$. However, if we are talking about the 2-qubit state $|11\rangle$, we must interpret 11 as a binary number, and hence we simply mean $|1\rangle \otimes |1\rangle$. In practice, these ambiguities very rarely occur, but whenever they do, we will explicitly state which of the states is meant.

Now, let's attempt to build some intuition for multiple-qubit systems. To that end, we will again draw a parallel with storing information classically. Suppose that we want to store what season it is, i.e., we want to store whether it is spring, summer, autumn or winter. To do so, we could take two bits, and assign the values 00, 01, 10 and 11 to spring, summer, autumn and winter, respectively. Hence, by concatenating two bits, we enabled ourselves to distinguish between 4 different states. It is not difficult to see that in general, taking n bits allows us to distinguish between 2^n states.

Similarly, suppose that we want to store the season in a 2-qubit quantum system. Again, we could associate $|00\rangle$ to spring, $|01\rangle$ to summer, $|10\rangle$ to autumn and $|11\rangle$ to winter. Now, our 2-qubit system can be in some superposition of these four states. More precisely, all states of our 2-qubit system can be written as follows, where $\alpha, \beta, \gamma, \delta \in \mathbb{C}$:

$$\alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle + \delta |3\rangle \qquad \text{with} \qquad |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

Again, we can think of our system as being in some kind of weighted average of the four basis states representing the four seasons.

Similarly to the single-qubit case, we can measure in what computational basis state an n-qubit system is, by performing a measurement in the computational basis. We provide its formal definition below.

Definition 3.1.7: Measurement of a multiple-qubit system in the computational basis
Let $n \in \mathbb{N}$ and suppose that we have an n-qubit system. Then, we define, for all $j \in \{0, 1, \ldots, 2^n - 1\}$, the following measurement operator:

$$M_j = |j\rangle \langle j|$$

where $|j\rangle$ is an n-qubit state vector and $\langle j|$ is its Hilbert space dual vector. These measurement operators, we refer to as *computational basis measurement operators*, and they form the family $\{M_j\}_{j=0}^{2^n-1}$. The measurement corresponding to this family of measurement operators is referred to as the *measurement in the computational basis*.

Analyzing the probability distribution of a measurement in the computational basis of a multiple-qubit system is analogous to the single-qubit case. Suppose that we have a 2-qubit system in the following state:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle + \delta |3\rangle$$

Then, we obtain the following probability distribution on the possible measurement outcomes:

$$\mathbb{P}(0) = |\alpha|^2, \qquad \mathbb{P}(1) = |\beta|^2, \qquad \mathbb{P}(2) = |\gamma|^2, \qquad \mathbb{P}(3) = |\delta|^2$$

Hence, if a two-qubit system starts out in a state that is a scalar multiple of a computational basis state, we will recover this computational basis state with probability 1 upon performing a measurement in the computational basis. On the other hand, if for instance we have a two-qubit system in the state $\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |3\rangle$, then a measurement in the computational basis yields the outcome 0 with probability $1/2$, and similarly 3 is obtained with probability $1/2$ as well. These observations extend to more general n-qubit systems in the natural way.

There is something more we can do with multiple-qubit systems, though. For example, for a two-qubit system, we could decide to perform a *partial measurement*, e.g. we could decide to just measure the first qubit in the computational basis. The corresponding family of measurement operators is $\{M_0 \otimes I, M_1 \otimes I\}$, where M_0 and M_1 are the single-qubit computational basis measurement operators.

Let's analyze the consequences of such a measurement by means of an example. Suppose that we have a two-qubit system in the following state:

$$|\psi\rangle = \frac{1}{\sqrt{3}}|00\rangle + \frac{1}{\sqrt{3}}|01\rangle + \frac{1}{\sqrt{3}}|11\rangle$$

Then, employing the methods prescribed by Postulate 2.2.2, we can evaluate the probability distribution over the measurement outcomes:

$$\mathbb{P}(0) = \langle\psi|\,(M_0^* \otimes I)(M_0 \otimes I)\,|\psi\rangle = \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}(1) = \langle\psi|\,(M_1^* \otimes I)(M_1 \otimes I)\,|\psi\rangle = \frac{1}{3}$$

We can observe that we can simply quadratically sum the amplitudes of the computational states that have for instance a 0 in the first qubit, in order to obtain the probability of obtaining the corresponding measurement outcome 0 when performing the partial measurement on the first qubit.

If we relate this to our example of storing the seasons in a 2-qubit system, suppose that we just want to know whether it is either of summer or spring, or either of autumn or winter. Recall that we associated $|00\rangle$ with summer, $|01\rangle$ with spring, $|10\rangle$ with autumn, and $|11\rangle$ with winter. Both summer and spring correspond to a state in which the first qubit is in state $|0\rangle$, whereas both autumn and winter correspond to a state in which the first qubit has state $|1\rangle$. Hence, we can simply apply a partial measurement on the first qubit to obtain our answer.

This completes our attempt at developing some intuition for multiple-qubit systems. In the remainder of this subsection, we will introduce some notational conveniences, and elaborate on why they do not give rise to any ambiguities.

First of all, we can again identify the state space of an n-qubit system with $\mathbb{C}^{2^n}$. To that end, observe that we can write every n-qubit state with appropriate choices for $(\alpha_j)_{j=0}^{2^n-1} \subseteq \mathbb{C}$, which is justified by Theorem 3.1.5, property 2:

$$\sum_{j=0}^{2^n-1} \alpha_j |j\rangle$$

This allows for the following identification of the state space of an n-qubit system with $\mathbb{C}^{2^n}$, where we denote the Cartesian basis of $\mathbb{C}^{2^n}$ by $\{\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{2^n-1}\}$:

$$\sum_{j=0}^{2^n-1} \alpha_j |j\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-1} \end{bmatrix} = \sum_{j=0}^{2^n-1} \alpha_j \mathbf{e}_j$$

This implies that for all $j \in \{0, 1, \ldots, 2^n - 1\}$, we have the following simple relation:

$$|j\rangle = \mathbf{e}_j$$

Hence, we again find a one-to-one correspondence between the computational basis states of the state space of an n-qubit system and the Cartesian basis states of $\mathbb{C}^{2^n}$.

Next, we check here that this identification is compatible with the Kronecker product, as defined in Section 2.1. To that end, we take any $b_1, b_2, \ldots b_n \in \{0, 1\}$ and we define $b = (b_1 \ldots b_n)_2$ to be the integer with corresponding binary representation $b_1 \ldots b_n$. Now, we observe:

$$\begin{array}{ccc} |b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle & = & |b\rangle \\ \| & & \| \\ \mathbf{e}_{b_1} \otimes \mathbf{e}_{b_2} \otimes \cdots \otimes \mathbf{e}_{b_n} & = & \mathbf{e}_b \end{array}$$

where the bottom equality follows from our discussion in Section 2.1. Hence, indeed, by linearity, we have found that we can simultaneously use the identification that we introduced in Section 2.1, and the one that we introduced in this section, without reaching contradictory statements.

Let's consider some examples of two-qubit states, and how the identification with vectors from $\mathbb{C}^4$ can be used in calculations. In particular, let's consider the state $|+\rangle \otimes |+\rangle$. We can simply write out its definition and expand the parentheses to figure out how we can write this state as a superposition of the computational basis states:

$$|+\rangle \otimes |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

We can do the same calculation using the identifications we introduced above, where in the middle equality we calculate the Kronecker product between two two-dimensional vectors:

$$|+\rangle \otimes |+\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\-1\\1\\-1\end{bmatrix} = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

Observe that both calculations yield the same result, as they should. In fact, there is no preferred way of doing such calculations, on the contrary, both are equally valid. Usually, the context dictates which one of the methods is easiest to use, but the reader is free to apply both of these methods interchangeably.

Finally, note that in this section, the concept of *superposition* played a very central role, whereas we never mentioned the word *entanglement*. It is very important to understand the distinction between these two notions, which is why we devote the next remark to this distinction.

Remark 3.1.8: Superposition vs. entanglement
It is important to understand the difference between *superposition* and *entanglement*. We say that an n-qubit system is in superposition, if it is not in one of the computational basis states. On the other hand, we say that an n-qubit system is in an entangled state, or simply that it is entangled, if its state cannot be written as a simple tensor, cf. Postulate 2.2.4. Hence, a 1-qubit system can be in superposition, but it cannot be entangled. Furthermore, as all the computational basis states are simple tensors, we obtain that whenever an n-qubit system is entangled, it is necessarily in superposition.

So, again consider the following state:

$$\frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) = |+\rangle \otimes |-\rangle$$

This is a superposition state, as it is not a scalar multiple of any of the four computational basis states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. However, as it can be written as a simple tensor, namely $|+\rangle \otimes |+\rangle$, it is not an entangled state.

This concludes our discussion on single-qubit and multiple-qubit systems. In the subsequent section, we will consider how we can manipulate these systems, in order to perform calculations.

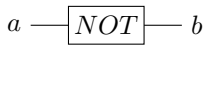
3.2 Quantum gates

This section will be devoted to introducing the concept of quantum gates. In short, these quantum gates provide us with the basic operations that we can perform to manipulate qubits. To understand how they do this, we will first of all draw a parallel with their classical counterparts. Afterwards, we will elaborate on those quantum gates that modify one single qubit, in Subsection 3.2.1, and subsequently, in Subsection 3.2.2, we will cover quantum gates that modify affect qubits.

So first, in order to understand how quantum computers can manipulate qubits, let's first have a look at how classical computers manipulate bits. On hardware level, namely, classical computers manipulate bits by means of *logic gates*. These gates can be thought of as functions, mapping a few input bits to a few output bits. For example, the NOT-gate is a very simple logic gate, which takes one bit as input and outputs one bit as well. If its input bit is 0, it outputs a 1, and similarly if its input bit is 1, it outputs a 0. Hence applying this logic gate to a bit has the effect of flipping the state of this bit.

Some well-known logic gates are listed in Section 3.2, namely the NOT, AND, OR and XOR gates. Here, the gates are also graphically depicted as rectangular blocks. In these schematic pictures, the horizontal lines connected to the left side of the gate indicate the input bits, and the horizontal lines exiting the gate on the right represent the output bits, i.e., time progresses from left to right. Hence, one can observe on the left side in Section 3.2 that the NOT-gate has 1 input and 1 output bit.

NOT gate:

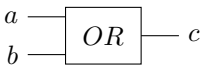


a	b
0	1
1	0

AND gate:

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

OR gate:



a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

XOR gate:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3.1: Some well-known logic gates. In the schematic pictures, the rectangular block represents the gate. The horizontal lines extending to the left of it, represent the input bits, and the lines on the right side correspond to the output bits. The table below it describes how the values of the input and output bits are related. We refer to these tables as the *truth tables* of the corresponding logic gates.

Similarly to the classical case, qubits can also be modified by gates, but these gates are known as *quantum gates*. These quantum gates also have a certain number of qubits as input, and they have a certain number of output qubits as well, just like in the classical case. However, Postulate 2.2.6 tells us that we can only modify physical systems in a unitary manner. In particular, this means that these quantum gates cannot remove any qubits from or add any qubits to the system they are operating on. So, we find that the number of input and output qubits, associated to any quantum gate, must be the same.

The above observation allows us to make a clear distinction between single-qubit gates and multiple-qubit gates. We will first of all consider single-qubit gates, in Subsection 3.2.1, and subsequently we will consider multiple-qubit gates in more detail, in Subsection 3.2.2.

3.2.1 Single-qubit gates

As remarked before, a quantum computer can use quantum gates to manipulate qubits. Whenever a quantum gate operates on one qubit, we refer to this gate as a *single-qubit gate*. This section aims to give a brief overview of the different single-qubit gates that one might encounter in the field of quantum computing.

We already remarked before, as a consequence of Postulate 2.2.6, that every single-qubit gate must perform a unitary transformation on the state space of the single-qubit system. As the state space of a single-qubit system is isomorphic to $\mathbb{C}^2$, and as we by convention associate the computational basis states, $|0\rangle$ and $|1\rangle$, to the Cartesian basis vectors of $\mathbb{C}^2$, we can represent the action of every single-qubit gate as a 2×2 matrix with complex entries. We refer to this matrix as the *matrix representation* of the single-qubit gate, and hence, due to the canonical mapping between the state space and $\mathbb{C}^2$, implicitly we always assume this matrix representation to be with repsect to the computational basis. Generally, we will simply identify the quantum gate with its matrix representation, hence we will refer to both as being the same mathematical object.

One of the easiest operations that we can perform on a single-qubit system is the X-gate, which is in most respects the quantum analogue of the classical NOT-gate. Its matrix representation is given by:

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

Recall that we made the following identification in Section 3.1:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

So, we find that X has the following action on the computational basis states:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \qquad \text{and} \qquad X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Thus, we see that $|0\rangle$ is mapped to $|1\rangle$ and $|1\rangle$ is mapped to $|0\rangle$. Hence, applying the X-gate comes down to flipping the computational basis states of the qubit on which the gate is applied. Here, we see the analogy with the NOT-gate, which flipped the state of the bit on which it was applied.

Whereas the NOT-gate was the only logic gate with one input and output bit that we considered, there are many more single-qubit gates of interest to us. The most well-known, with their corresponding matrix representations, are listed in Figure 3.2. Here, we also provide a graphical depiction of the quantum gate. That is, we represent every gate as a block, with 1 horizontal line extending from it on both the left and right sides. These lines, just as with the logic gates, represent the input and output qubits of the quantum gate, respectively. Similarly to the classical case, we can think of time progressing from left to right in these graphical depictions of single-qubit gates.

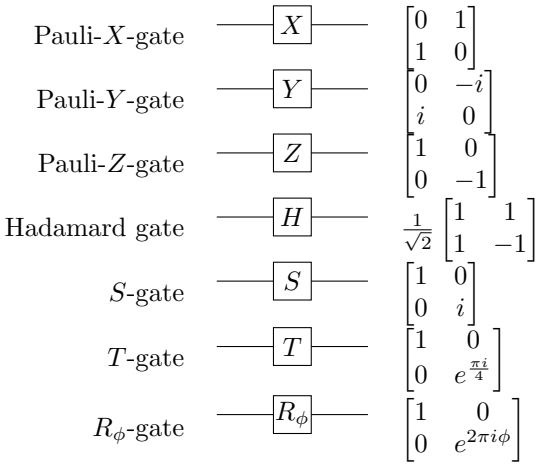


Figure 3.2: Some well-known single-qubit gates. Note that all of them have exactly one input and output qubit. Moreover, in the last one, the R_ϕ-gate, ϕ can be a real number. If we take $\phi = 1/2$, we obtain the Z-gate. Similarly if we take $\phi = 1/4$, we obtain the S-gate, and $\phi = 1/8$ yields the T-gate.

Let's highlight some of the important properties of the single-qubit gates listed in Figure 3.2. We already mentioned that X is just like the NOT-gate in classical computing. Additionally, it is one of the Pauli gates, which are commonly denoted by X, Y and Z. The Z-gate is also called the *phase flip gate*, as it maps $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|0\rangle - \beta|1\rangle$. Hence, it flips the phase of the $|1\rangle$ state, whereas it leaves the phase of the $|0\rangle$ state untouched.

Continuing down the list of single-qubit gates presented in Figure 3.2, we arrive at the Hadamard gate. Observe that it has the following action on the computational basis states:

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \qquad \text{and} \qquad |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$$

Thus, we observe that the Hadamard gate is able to create superposition states from computational basis states. This property of H we will use frequently throughout many quantum algorithms. Moreover, observe that $H^2 = I$, and hence H is self-inverse.

Next, observe that the R_ϕ-gate defines a gate for all real values of ϕ. Moreover, observe that if we choose $\phi = 1/2$, the matrix representation reduces to that of Z. Similarly, $\phi = 1/4$ leaves us with the S-gate, and by choosing $\phi = 1/8$, we obtain the T-gate. Hence, the R_ϕ-gate can be used to model many of the other single-qubit gates, which is something we will be using later on as well.

Finally, recall that all the before-mentioned quantum gates are unitary operators on the state space of a single-qubit system. As such, their corresponding matrix representations are unitary, and hence we obtain for every single-qubit gate with matrix representation U:

$$UU^* = I$$

U^*, the Hilbert space adjoint of U, is again a unitary operator. It acts on the same space as U, i.e., on the state space of a single-qubit system. Hence, U^* again defines a single-qubit gate, and we refer to this gate as the *reverse single-qubit gate of* U, the *reverse gate of* U or simply the *reverse of* U.[2] Moreover, its matrix representation is the Hermitian conjugate of the matrix representation of U. For instance, we easily obtain that the reverse single-qubit gate of R_ϕ is given by: $(R_\phi)^* = R_{-\phi}$, which can be easily checked by calculating the matrix representations of the left- and right-hand side, and observing that both are equal.

This concludes our discussion on single-qubit gates. In the next section, we will generalize these ideas to operations on multiple-qubit systems.

3.2.2 Multiple-qubit gates

In the last section, we elaborated on how quantum computers can manipulate single-qubit systems. In this section, we generalize these ideas to multiple-qubit systems. Particularly, we will consider *n-qubit quantum gates*, which are quantum gates that manipulate n-qubit systems.

Recall that all quantum gates must be unitary operations, with the same number of input and output qubits. In particular, this means that if we have a quantum gate which acts on an n-qubit system, its action is a unitary operator on the state space of this n-qubit system. But from Theorem 3.1.5, we know that this state space is isomorphic to $\mathbb{C}^{2^n}$. Hence, we can represent every n-qubit quantum gate by a $2^n \times 2^n$ matrix with complex entries. This matrix, we refer to as the *matrix representation of the n-qubit quantum gate*. Similarly to the single-qubit case, we will frequently identify the single-qubit gate with its matrix representation.

These quantum gates, we can also graphically depict like we did in the single-qubit case. In Figure 3.3, for example, we have drawn several quantum gates that act on 2, 3 and 4 qubits, respectively. The corresponding matrix representations will be 4×4, 8×8, and 16×16 matrices with complex entries, respectively.

There are several important multiple-qubit gates and classes of them that deserve special mention here. Below, we will cover the most important ones, which are very broadly used throughout the field of quantum computing. Many have some kind of specialized notation associated to them.

[2] In the field of quantum computing, it appears to be more common to denote the reverse gate of U by $U^\dagger$, instead of U^*. We, however, will use the notation U^*, because it is in accordance with the notation found in the mathematical introduction, presented in Chapter 2.

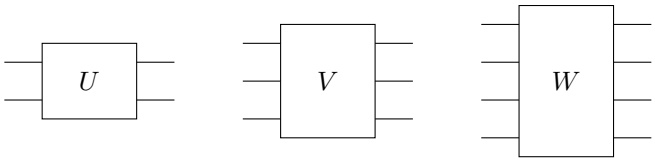


Figure 3.3: Schematic depiction of a 2-qubit gate U, a 3-qubit gate V and a 4-qubit gate W.

First of all, we consider the $CNOT$-gate, which is shorthand for the Controlled-NOT gate. This is a 2-qubit gate, which has the following matrix representation:

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

It is interesting to observe the action that this 2-qubit gate has on the computational basis states. We will show how to evaluate this for the state $|00\rangle$ below, where we use the Kronecker product of vectors as introduced in Section 2.1:

$$CNOT\,|00\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle$$

Similarly, one can evaluate the action of $CNOT$ on the other basis states. Then, we obtain that $CNOT$ has the following action:

$$\begin{array}{ll} |00\rangle \mapsto |00\rangle & |10\rangle \mapsto |11\rangle \\ |01\rangle \mapsto |01\rangle & |11\rangle \mapsto |10\rangle \end{array}$$

Hence, we see that on the second qubit, a NOT operation is performed, but only when the first qubit is 1. This explains the name controlled-NOT, because controlled on the first qubit being 1, a NOT-operation is performed on the second qubit. In the context of the CNOT gate, we call the first qubit the *control qubit*, and the second qubit the *target qubit*.

The following specialized graphical depiction of the CNOT-gate has been invented:

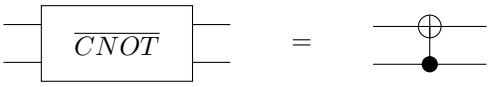
Here, the black dot is placed on the line that represents the control qubit, and the encircled plus sign is placed on the horizontal line that represents the target qubit.

Of course we can also reverse the role of the control and target qubits. That is, we can make the second qubit the control qubit, and the first qubit the target qubit. The resulting 2-qubit gate, we will call $\overline{CNOT}$.[3] It will perform a flip of the first qubit, whenever the second is in state $|1\rangle$, and it will not modify the first qubit if the second qubit is in state $|0\rangle$. The matrix representation and action on the basis states are given as follows:

$$\overline{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad \text{and} \qquad \begin{array}{ll} |00\rangle \mapsto |00\rangle & |10\rangle \mapsto |10\rangle \\ |01\rangle \mapsto |11\rangle & |11\rangle \mapsto |01\rangle \end{array}$$

[3] This is not standard notation.

We now depict the $\overline{CNOT}$-gate in the obvious way: we place the black dot on the control qubit, which is now the second qubit, and we place the encircled plus on the first qubit, which is the target qubit in this case. Thus, we have:



Next, we can extend this notion of the controlled-NOT operation to arbitrary controlled single-qubit operations. To that end, for any single-qubit gate U, we can introduce the controlled-U gate, acting on 2 qubits. This controlled-U gate has the following 4×4 matrix representation:

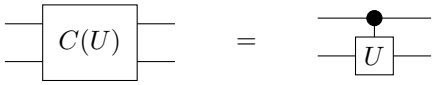
$$C(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \multicolumn{2}{c}{\multirow{2}{*}{U}} \\ 0 & 0 & & \end{bmatrix}$$

Hence, we obtain the following action, where $|\psi\rangle$ is any single-qubit state:

$$C(U)(|0\rangle \otimes |\psi\rangle) = |0\rangle \otimes |\psi\rangle \qquad \text{and} \qquad C(U)(|1\rangle \otimes |\psi\rangle) = |1\rangle \otimes U\,|\psi\rangle$$

So, indeed, we find that the single-qubit operation U is only performed on the second qubit whenever the first qubit is 1. Again, we refer to the first qubit as the control qubit, and the second qubit as the target qubit.

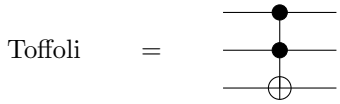
For these conditional operations, we also have some shorthand notation. We denote $C(U)$ as follows:



Similarly to the graphical depiction of the $CNOT$-gate, again the black dot is placed on the horizontal line that denotes the control qubit, and the U-gate is placed on the horizontal line denoting the target qubit.

With controlled single-qubit operations, we can again reverse the roles of the control and target qubits. The resulting gates, we again refer to as $\overline{C(U)}$. The reader is encouraged to check that the controlled-X operation is identical to the $CNOT$-gate, that is, to check that $C(X) = CNOT$. Similarly, we can also check that $\overline{C(X)} = \overline{CNOT}$.

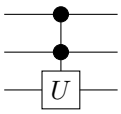
Up until now, we have only considered special 2-qubit gates. We can easily form multiple-qubit gates by adding more control qubits though. For instance, we can consider the following gate, which is known as the Toffoli gate:



Now we have two control qubits. Hence, a NOT-operation is now only applied to the target qubit whenever both of the control qubits are 1. Thus, this 3-qubit quantum gate maps $|000\rangle$ to $|000\rangle$, and also for example $|101\rangle$ to $|101\rangle$. Only when both of the control qubits are 1, the last qubit is flipped, e.g. $|110\rangle$ is mapped to $|111\rangle$. Writing out the action of this gate on all the computational basis states allows us to construct the matrix representation of this Toffoli gate, which becomes:

$$\text{Toffoli} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Similarly, we can create single-qubit gates controlled on n-qubits, yielding $(n+1)$-qubit gates. For instance, we could consider the $C(C(U))$-gate, which implements a single-qubit gate U whenever the first two control qubits are set to $|1\rangle$. This, we would denote as follows:

$C(C(U)) \quad = \quad$ 

And the corresponding matrix representation would look like this:

$$C(C(U)) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \multicolumn{2}{c}{\multirow{2}{*}{U}} \\ 0 & 0 & 0 & 0 & 0 & 0 & & \end{bmatrix}$$

Finally, we can also define the *reverse of multiple-qubit gates*, similarly to how we defined them in the single-qubit case. That is, for every multiple-qubit gate U, we define its reverse gate U^* to be the gate such that $UU^* = I$. For the controlled operations, we can easily check that $(C(U))^* = C(U^*)$. Finally, we also leave it to the reader to check that $C(C(X)) =$ Toffoli.

This completes our discussion on multiple-qubit gates. The next sections will expand on how sequences of quantum gates can be built to obtain quantum circuits, that can subsequently be used in quantum algorithms.

3.3 Quantum circuits

In this section, we will elaborate on how we can combine the quantum gates introduced in the previous section to build quantum circuits. Afterwards, we introduce the complexity measures on quantum circuits, which we refer to as the *elementary gate complexity* and the *query complexity*.

Suppose we have a two-qubit system, and suppose that we want to apply a T-gate to the first qubit, followed by a $CNOT$-gate, where the first qubit is used as the control qubit, and the second qubit is used as the target qubit. The entire recipe for applying this sequence of two quantum gates, we refer to as a *quantum circuit*. A fundamental property of any quantum circuit is the number of qubits it acts on, just as this is a fundamental property of any quantum gate.

We can always depict quantum circuits using the graphical representations of the quantum gates that we introduced in the previous section. We draw a number of horizontal lines, which we refer to as *wires*, equal to the number of qubits the quantum circuit acts on, and we attach the gates that we want to apply to the qubits to the corresponding wires. The order of application is given in the graphical representation from left to right. See Figure 3.4 for the graphical representation of the circuit we discussed above.

We can also always calculate the matrix representation that is associated to the action of any quantum circuit, by simply multiplying the matrix representations corresponding to the individual gates. One thing to keep in mind here is to *always reverse the order of the matrix multiplication*. To see why this is necessary, suppose that A and B are matrices, and that v is a vector. Then, if we calculate ABv, we see that B acts on v first, and only then we calculate the matrix-vector product A with Bv. So, the combined action of AB consists of first of all applying B to the vector, and afterwards A. In a quantum circuit, though, the left-most gate is applied first to the state of the system, and afterwards the gates that are to the right of it. So, in order to make sure that in our matrix representation, the action of the left-most gate is applied to

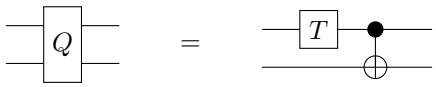


Figure 3.4: Example of a quantum circuit acting on two qubits. First, a T-gate is applied to the first qubit. Afterwards, a $CNOT$-gate is applied, where the first qubit serves the role of the control qubit, and the second qubit is used as the target qubit. We refer to this circuit by Q, and we will use the abbreviated notation shown on the left-hand side.

the state vector first, we must make sure that we put the matrix corresponding to the left-most gate in the right-most position in the matrix multiplication.

To clarify, if we consider the circuit from Figure 3.4, we obtain the following matrix representation:

$$Q = CNOT \cdot (T \otimes I) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{\pi i}{4}} \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{\frac{\pi i}{4}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{\pi i}{4}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{\frac{\pi i}{4}} & 0 & 0 \\ 0 & 0 & 0 & e^{\frac{\pi i}{4}} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In many respects, we can consider quantum circuits just like quantum gates. We already saw that both act on a fixed number of qubits, and both have a fixed matrix representation. In addition to that, we can also *nest* quantum circuits, i.e., we can use quantum circuits in definitions of new quantum circuits. For example, we can incorporate the circuit from Figure 3.4, Q, in a bigger circuit acting on three qubits, R, shown in Figure 3.5. Here, we say that Q is a *subcircuit* in R, and that R *queries* Q.

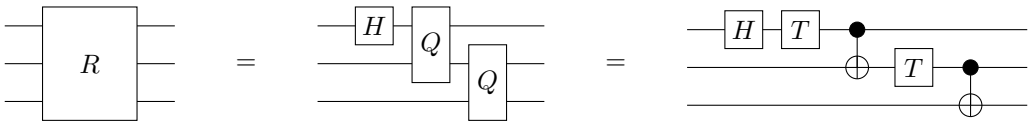


Figure 3.5: Example of a quantum circuit R, defined using nesting of quantum circuits. The Q blocks represents the quantum circuit introduced in Figure 3.4. So, in R, we first of all apply a Hadamard gate to the first qubit, and then apply the circuit from Figure 3.4 twice, once on the first and second qubit, and subsequently on the second and third qubit. We say that R *queries* the quantum circuit Q twice, and that Q is a *subcircuit* in R. The *unwrapped* version of R, i.e., the version in which all subcircuits are expanded into elementary gates, is shown on the right hand side.

Again, we can easily obtain the matrix representation of this larger circuit, similarly to how we determined the matrix representation of Q. In fact, we can use this matrix representation of Q to calculate the matrix representation of the quantum circuit R in Figure 3.5, without having to use the unwrapped version of R, i.e., the version that is shown on the right-hand side. Again, as this is a common mistake, we emphasize that one must reverse the order of the gates and subcircuits before calculating the matrix product.

It sometimes comes in handy to indicate the action of a quantum circuit on some input state in the graphical depiction. We can do this by indicating the input state of each qubit on the left side of its corresponding wire, and similarly the output state of each qubit on the right side of its corresponding wire. An example of such a circuit is shown in Figure 3.6.

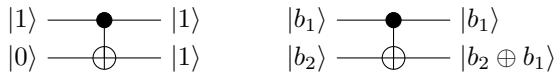


Figure 3.6: Exemplary usage of input and output states on both sides of the circuit. Both circuits in this figure consist only of one $CNOT$-gate. On the left, we show what the action of this circuit is on the initial state $|1\rangle \otimes |0\rangle$. On the right, we show what the action of the $CNOT$-gate is on the state $|b_1 b_2\rangle$, where $b_1, b_2 \in \{0, 1\}$. As we can write all computational basis states of a two-qubit system in this form for some choice of b_1 and b_2, we have concisely described the action of this circuit on all computational basis states of a two-qubit system on the right-hand side of this figure. Here, the $\oplus$-symbol must be interpreted as addition modulo 2.

We can also indicate the input state of multiple qubits at once, by bundling wires on the left with a bracket. Similarly, we can bundle output wires by a bracket on the right of the quantum circuit. Figure 3.7 shows how this works.

Figure 3.7: These two circuits show how one can specify multiple-qubit input and output states. On the left, we can see a two-qubit input state being supplied to a circuit containing just one $CNOT$-gate. The reader is encouraged to check that this circuit indeed turns the entangled input state into the indicated unentangled output state. Similarly, we see on the right how we can indicate the output state of two entangled qubits.

It can alslo be useful to bundle multiple qubits into a single horizontal wire. For instance, this can be used to reduce the apparent size of the circuit, and it allows us to give circuits where the number of qubits used is variable. Whenever we bundle multiple qubits into a single wire, we indicate this by an oblique line segment which crosses the horizontal line, above which we denote the number of qubits that are bundled. An example of such a circuit is shown in Figure 3.8.

Figure 3.8: Example of two qubits bundled into one horizontal wire. Note that the input and output states are now two-qubit states, where we take $b_1, b_2 \in \{0, 1\}$, and again interpret $\oplus$ as addition modulo 2. The circuit shown here is equal to the circuit used in Figure 3.6 and Figure 3.7, even though its graphical representation looks very different at first glance.

Frequently, we will encounter quantum circuits that make use of one or multiple auxiliary qubits, all of which are required to be in state $|0\rangle$ prior to execution of the circuit, and all of which are returned into state $|0\rangle$ at the end of the circuit. These qubits, we refer to as *auxiliary qubits*, or *ancilla qubits*[4]. We allow ourselves to leave out these ancilla qubits in shorthand notations of the quantum circuit. Moreover, when we say that a circuit acts on n qubits, we do not count the ancilla qubits. If necessary, we also mention the number of ancilla qubits that is used by a specific circuit. Figure 3.9 elaborates on the concept of an ancilla qubit by means of an example.

Just as it is possible to reverse any quantum gate, it is also possible to reverse any given quantum circuit. To find the reverse of a quantum circuit, one can replace all the gates by their reverse, and subsequently reverse the order. For example, the reverse of the circuit Q, introduced in Figure 3.4, is shown in Figure 3.10.

It is natural to measure the complexity of a quantum circuit by the number of elementary operations that it implements. To formalize this idea, though, we must first of all establish this set of elementary operations.

[4] *Ancilla* is Latin for slave girl.

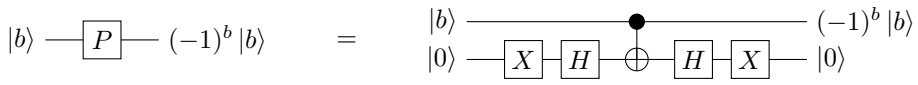


Figure 3.9: Exemplary usage of an ancilla qubit. In the circuit on the right-hand side, we use the second qubit as an ancilla qubit. Hence, we require that the input state of this qubit is $|0\rangle$, and that its state is set back to $|0\rangle$ at the end of the circuit. The reader is encouraged to check that this is indeed the case in the circuit shown on the right-hand side. In the abbreviated notation, shown on the left-hand side, we allow ourselves to omit the wire of the ancilla qubit. So, P does represent a circuit that requires two qubits to run, even though only 1 qubit is shown. We say that P acts on 1 qubit, and that it uses 1 additional ancilla qubit. As a final note, the reader is encouraged to check that the matrix representation of P is equal to that of Z.

Figure 3.10: The reverse of the quantum circuit Q, introduced in Figure 3.4. Note that the gate order is reversed, and that all the gates are replaced by their reversed counterpart. In particular, observe that the $CNOT$-gate is self-inverse (i.e., $CNOT^2 = I$), and so reversing it does not have any effect.

This is what the next definition achieves.

Definition 3.3.1: Elementary gates
An *elementary quantum gate*, or simply an *elementary gate*, is a single-qubit gate, possibly controlled by a single other qubit, whose matrix representation is different from the identity matrix.

For example, the gates T, H, X, and the controlled versions of these gates, $C(T)$, $C(H)$ and $C(X)$, are all elementary quantum gates. In particular, as we remarked before that $CNOT = C(X)$, we find that the $CNOT$-gate is an elementary gate.

Now that we have established what the *elementary operations* are, we can now formalize how we can measure the complexity of a quantum circuit. We refer to this measure as the *elementary gate complexity*.

Definition 3.3.2: Elementary gate complexity of a quantum circuit
The *elementary gate complexity* of a quantum circuit Q is the number of elementary gates it consists of. We denote this quantity by $\text{egc}(Q)$.

By the excerpt *the number of elementary gates a circuit implements*, we mean the number of gates that are present in the circuit, after all nested quantum circuits are unwrapped until only elementary gates remain. So, for example, the circuit Q defined in Figure 3.4 has an elementary gate complexity of 2, because both of its components, i.e., T and $CNOT$, are elementary gates. Similarly, the elementary gate complexity of the circuit P, defined in Figure 3.9, is 5. We write:

$$\text{egc}(Q) = 2 \qquad \text{and} \qquad \text{egc}(P) = 5$$

At this point, it is instructive to observe that in principle, every unitary matrix of size $2^n \times 2^n$ defines a valid operation on an n-qubit system. We say that a set of quantum gates is *universal* if, for any $2^n \times 2^n$ unitary matrix U, we can build a quantum circuit acting on n qubits whose matrix representation equals U, only using gates from this set. Such sets of gates, we refer to as *universal gate sets*.

The question now naturally arises whether the set of elementary gates defined in Definition 3.3.1 is universal. The answer to this question turns out to be yes, but its proof is not trivial. Moreover, the gate set

$\{H, T, CNOT\}$ is *approximately universal*, meaning that one can build a circuit using these gates whose matrix representation is arbitrarily close to any given unitary $2^n \times 2^n$ matrix U in operator norm. We refer the interested reader to [NC00], where these results are proven in chapter 4. In addition, this proof can also be found in [Cor16], Chapter 4.

The attentive reader may have noted that not all quantum circuits have a well-defined elementary gate complexity. In particular, circuits using non-elementary gates, like the Toffoli gate, indeed do not have a well-defined elementary gate complexity. Usually, however, we can give an implementation of a non-elementary gate by means of a quantum circuit using only elementary gates. Now, we can unwrap these nested quantum circuits, and count the resulting number of elementary gates in order to calculate the elementary gate complexity of the original circuit. Hence, in order for the elementary gate complexity of a circuit to be well-defined, we must provide an implementation of each non-elementary gate that appears in this circuit, consisting only of elementary gates.[5]

Additionally, similarly to the elementary gate complexity, we can also measure how often a specific subcircuit is queried. This we refer to as the *query complexity* of a specific subcircuit in a larger circuit. For instance, the circuit R, defined in Figure 3.5 makes 2 queries to the subcircuit Q, defined in Figure 3.4. We say that the *Q-query complexity of R* is 2. Sometimes, we abbreviate this simply to the *query complexity of R* when confusion about which subcircuit is meant is unlikely.

Finally, observe that we did not introduce a *measurement-gate*, or a similar operation that involves performing a measurement that we can represent within quantum circuits. Some sources, e.g., [NC00], do introduce auxiliary symbols such that they can represent measurement operations in the graphical representation of quantum circuits. We, however, will ensure that quantum circuits do not contain measurement operations, for two reasons.

First of all, we remarked that, similar to quantum gates, we can write down the matrix representation of a quantum circuit. In particular, this implies that every quantum circuit acts as a linear operator on the state space of the physical system it is executed on. If we were to allow measurements in quantum circuits, though, we would break this linearity property, complicating analyses later on in this text considerably.

Secondly, recall from our discussion in Subsection 3.1.1 that measurements can break the superposition. However, if we want to use quantum circuits as a subcircuit in larger circuits, we sometimes don't want to break the superposition of the larger circuit by simply executing the smaller circuit. Not allowing ourselves to perform any measurements in any quantum circuit circumvents having to deal with this problem at all, motivating the exclusion of measurement operations from quantum circuits.

This concludes our discussion on quantum circuits. In the next section, we will investigate what role quantum circuits play in quantum algorithms.

3.4 Quantum algorithms

In this section, we will introduce the notion of *quantum algorithms*. Specifically, we will elaborate on how quantum circuits, introduced in the previous section, fit into quantum algorithms, and how we can use the complexity metrics that we defined for quantum circuits to measure the complexity of these quantum algorithms.

In mathematics, we often deal with *problems*. These can usually be expressed as an imperative sentence, i.e., "Find the prime factorization of 196883," or "Prove Fermat's last theorem or provide a counterexample." Solutions to problems vary enormously in length, as "$47 \cdot 59 \cdot 73$" is a valid solution for the first problem, whereas the first known solution for the second problem was over 100 pages long. Note that solutions to

[5] In Subsection 3.5.2, we will explicitly provide such an implementation for the Toffoli gate.

problems are not necessarily unique, as different proofs of Fermat's last theorem can be equally valid as the one first published by Andrew J. Wiles.

Oftentimes, we encounter many problems with a very similar structure. For example, we can have the problems: "Find the greatest common divisor of 2 and 3," "Find the greatest common divisor of 2 and 4" and "Find the greatest common divisor of 2 and 5." We can bundle all these problems in a particular *class of problems*: "Find the greatest common divisor of a and b, where $a, b > 0$ are integers." For every particular choice for $a > 0$ and $b > 0$, we obtain one specific *instance of the problem* which is contained in the class of problems. We say that the pair (a, b) is an *input* to the class of problems, and hence every input gives rise to one instance of the class of problems. The *output* corresponding to a given input, we define to be the solution to the instance of the class of problems generated by the input. So, for example, let's take $a = 1517$ and $b = 1599$. Then, the instance generated by the input $(1517, 1599)$ is "Find the greatest common divisor of 1517 and 1599." The output corresponding to this input becomes the solution to this instance, which happens to be 41.

An *algorithm* that solves a given class of problems, now, is an unambiguous specification of the steps that we can perform in order to find the output corresponding to any input to this class of problems. For example, Euclid's algorithm is the well-known algorithm that solves the class of problems that we introduced in the previous paragraph: "Find the greatest common divisor of a and b, where $a, b > 0$ are integers." We say that any pair (a, b) where $a, b > 0$ is an input to Euclid's algorithm, and that this algorithm yields the corresponding output $\gcd(a, b)$. So, under the input $(1517, 1599)$, Euclid's algorithm yields the output 41.

Now, suppose that we have an algorithm which solely prescribes the execution of steps that could be executed on a universal Turing machine. Then, we refer to such an algorithm as a *classical algorithm*. All operations that we can perform on any modern-day computer could in principle be performed on a universal Turing machine, and hence every algorithm that we can implement in a classical computer is by definition a classical algorithm.

On the other hand, suppose that we have an algorithm that, aside from operations that could be performed on a universal Turing machine, for at least some inputs also prescribes the execution of one or multiple quantum circuits, and possibly the performance of one or multiple measurements. Then, we refer to such an algorithm as a *quantum algorithm*. The part of the algorithm that could be implemented on a universal Turing machine, we refer to as the *classical part* of the quantum algorithm.

Note that in principle, the quantum circuit that the algorithm prescribes, can depend on the input that is being supplied to the algorithm. Moreover, some quantum algorithms only prescribe the execution of a quantum circuit for some, but not all, inputs. A quantum algorithm that has both these features is Shor's algorithm, which solves the class of problems "For any composite positive integer N, find a non-trivial divisor of N." We refer the interested reader to [NC00], Chapter 5, or [Cor16], Chapter 5.

For every input that is supplied to a quantum algorithm, though, we can calculate the *total elementary gate complexity of the quantum algorithm corresponding to this input*, which is the sum of the elementary gate complexities of the quantum circuits that the quantum algorithm prescribes for the given input. This total elementary gate complexity of the quantum algorithm, we generally simply refer to as the *elementary gate complexity of the quantum algorithm*. Note that this quantity is dependent on the input that is supplied to the quantum algorithm. In other words, the elementary gate complexity of the quantum algorithm is a function that maps any input that can be supplied to the quantum algorithm to a number indicating the total elementary gate complexity corresponding to this input. For example, for Shor's algorithm, the input is a positive integer N, and the total elementary gate complexity is $\mathcal{O}(\log^3(N))$, as is shown in [NC00], Chapter 5, and [Cor16], Chapter 5.

Additionally, we can also have a quantum algorithm whose input consists of a quantum circuit. For example, let's consider the class of problems: "Given any quantum circuit implementing $|j\rangle \otimes |b\rangle \mapsto |j\rangle \otimes |b \oplus x_j\rangle$, where $j \in \{0, 1, \ldots, 2^n - 1\}$ and $x_0, x_1, \ldots, x_{2^n-1}, b \in \{0, 1\}$, find a $j \in \{0, 1, \ldots, 2^n - 1\}$ such that $x_j = 1$ or output that no such j exists." For this class of problems, the input is a quantum circuit, and hence any quantum

algorithm that solves this class of problems will somehow need to prescribe one or more quantum circuits that call this input circuit as a subcircuit at least once. For these circuits prescribed by the algorithm, we can meaningfully calculate the *input circuit query complexity*, which we defined to be the number of times the input circuit is queried by the circuit. The *total query complexity corresponding to the given input*, or the *oracle query complexity corresponding to the given input*, is the number of times the input circuit is being queried in total, throughout all the circuits that are prescribed by the quantum algorithm. This is again a function mapping any input circuit to a number, as the total number of queries that are being performed can depend on the input circuit that is supplied. For the class of problems that we defined earlier on in this paragraph, there exists a quantum algorithm[6] which solves this class of problems. The query complexity of this algorithm is $\mathcal{O}(2^{n/2})$.

Finally, recall from both Section 2.2 and Section 3.1 that the outcome of measurements is determined by probabilistic events. Hence, if we run a quantum algorithm that prescribes at least one quantum measurement, twice with the same input, then we might obtain different results. More precisely, for every input to a quantum algorithm, the output is selected according to a probability distribution. Hence, it can happen that the probability of the quantum algorithm yielding a valid solution to the instance of the problem generated by the input is not 1, but somewhat less than that. We refer to this probability as the *success probability of the quantum algorithm, corresponding to the given input*. This success probability can depend on the input that is supplied to the quantum algorithm, so we again consider this *success probability of the quantum algorithm* as a function mapping the input to a number in the interval $[0, 1]$. For example, if the input integer to Shor's algorithm is even, the success probability corresponding to this input integer is 1. Similarly, if the input integer to Shor's algorithm is a pure power, i.e., it can be written as a^b, with a and b integers and $b > 1$, then the success probability is 1 as well. If, on the other hand, the input integer is neither of the above two, then the success probability is at least $\frac{2}{3}$, as shown in [NC00], chapter 5, or [Cor16], chapter 5. So, we say that the success probability of Shor's algorithm is lower bounded by $\frac{2}{3}$, i.e., the success probability of Shor's algorithm associated to any input is at least $\frac{2}{3}$.

This concludes our discussion on quantum algorithms. In the next section, we will present several examples of quantum circuits and quantum algorithms, so the reader can get a bit more familiar with the notions introduced in the last two sections.

3.5 Examples of quantum circuits and quantum algorithms

In this section, we will cover some examples of quantum circuits and quantum algorithms. Partially, we do this to provide the reader with some examples, such that the reader can get accustomed to analyzing these quantum circuits and quantum algorithms. Simultaneously, though, the techniques that we introduce here often play a central role in the algorithms that we will develop in subsequent chapters, and hence a thorough understanding of the ideas presented here is indispensible to fully grasp the ideas behind the algorithms that we introduce in the remainder of this text.

3.5.1 SWAP

The first quantum circuit that we introduce here, is the SWAP-circuit. It can be used to swap the state of two n-qubit systems. For clarity, we will first introduce the 1-qubit variant, and subsequently, we will make the obvious generalization to n-qubit systems.

So, first of all, we consider the following problem. Suppose that we have a two-qubit system, and that the state of this system is $|\phi\rangle \otimes |\psi\rangle$, where $|\phi\rangle$ and $|\psi\rangle$ are single-qubit states. Next, suppose that we want to

[6] This is a modified version of Grover's algorithm, which we introduce in Subsection 3.5.6.

swap the states of both qubits, i.e., we want to get the two-qubit system into the state $|\psi\rangle \otimes |\phi\rangle$. This is what the SWAP-circuit achieves.

As a first step towards the development of the SWAP-circuit, we note that it is sufficient to determine what we want the action of the SWAP-circuit to be on the computational basis states. As any quantum circuit acts as a linear operator on the state space, we deduce from linearity that the desired behavior extends to all separable input states.

Obviously, we want the SWAP-circuit not to modify the states $|00\rangle$ and $|11\rangle$, as swapping the qubits has no effect when they are in these states. On the other hand, if the initial state, before application of the SWAP-circuit, is $|01\rangle$ or $|10\rangle$, then we want the final state to be $|10\rangle$ or $|01\rangle$, respectively. Hence, we can now write down what we want the matrix representation of the SWAP-circuit to be (with respect to the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, as always):

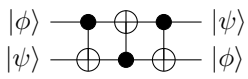
$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, observe that this matrix is very similar to the matrices corresponding to the CNOT- and $\overline{\text{CNOT}}$-gates. In fact, we can obtain the matrix representation of the SWAP-gate by multiplying these matrices corresponding to the CNOT- and $\overline{\text{CNOT}}$-gates together, as such:

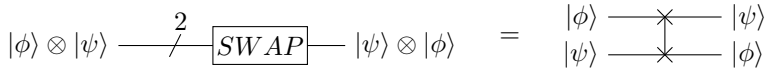
$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \text{CNOT} \cdot \overline{\text{CNOT}} \cdot \text{CNOT}$$

Thus, we have now found a way to construct the SWAP-circuit: we can simply apply a CNOT-gate, a $\overline{\text{CNOT}}$-gate and a CNOT-gate in sequence. This is summarized in Circuit 3.5.1. In addition, we also supply two different shorthand notations for the SWAP-circuit there, that we will use throughout the remainder of this text.

Circuit 3.5.1: SWAP
Description: This quantum circuit swaps the state of two qubits.
Elementary gate complexity: 3.
Circuit: Here, $|\phi\rangle$ and $|\psi\rangle$ are single-qubit states.



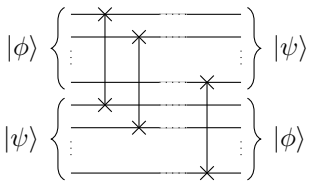
Shorthand notation:



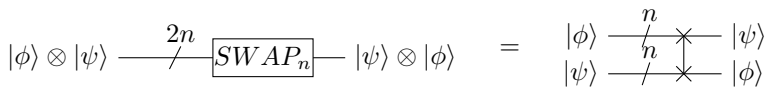
Now that we have formally introduced the SWAP-circuit, we can use it as a subcircuit in larger quantum circuits. We will use this right away in the generalization of the SWAP-circuit that swaps the state of two n-qubit systems, which we refer to as the SWAP_n-circuit.

If we want to devise a circuit SWAP_n that swaps the role of two n-qubit systems, then we must simply devise a circuit that swaps the role of n two-qubit pairs. One can easily observe this by looking at the computational basis states of the $2n$-qubit system, just as we did for the simpler SWAP-circuit. Hence, we obtain the SWAP_n-circuit in a straightforward manner, as shown in Circuit 3.5.2.

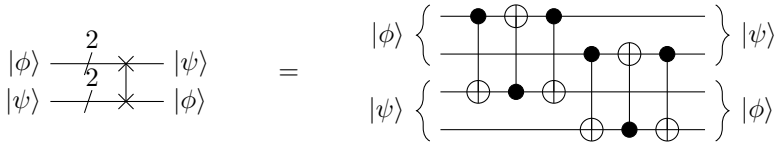
Circuit 3.5.2: SWAP_n
Description: This quantum circuit swaps the states of two n-qubit registers.
Elementary gate complexity: $3n$.
Circuit: Here $|\phi\rangle$ and $|\psi\rangle$ are n-qubit states.



Shorthand notation:



For additional clarity, we give here the SWAP_2 circuit, with both SWAP-subcircuits expanded:

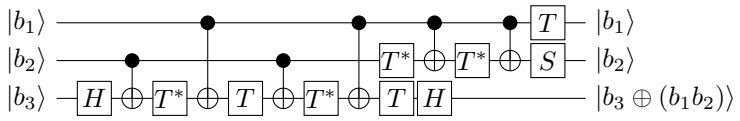


This completes our discussion of the SWAP-circuit. We will encounter it frequently in subsequent sections.

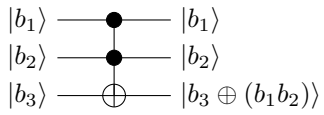
3.5.2 Toffoli gate

Earlier on in this chapter, we introduced the Toffoli-gate. Afterwards, when we introduced the elementary gates, we remarked that the Toffoli-gate is not an elementary gate. We can, however, construct a circuit consisting solely of CNOT-gates and single-qubit gates, that has the same matrix representation as the Toffoli-gate that we introduced in Subsection 3.2.2. We refer to this circuit as the Toffoli-circuit, and it is presented in Circuit 3.5.3.

Circuit 3.5.3: Toffoli
Description: The matrix representation of this circuit is identical to the matrix representation of the Toffoli-gate, and hence this circuit essentially implements this gate solely using elementary gates.
Elementary gate complexity: 16.
Circuit: In the circuit below, we have $b_1, b_2, b_3 \in \{0, 1\}$:



Shorthand notation:



We have taken the construction of Circuit 3.5.3 from [NC00], page 182. Proving that the matrix representation of this circuit indeed coincides with the matrix representation of the Toffoli-gate that we introduced

in Subsection 3.2.2 is straightforward, as we can just multiply together the matrix representations of the elementary gates that appear in the Toffoli-circuit. We leave the details to the reader as they do not provide any additional relevant insight for the remainder of this text.

3.5.3 Quantum Fourier transform

The Fourier transform is used in many different branches of science. Ever since the introduction of the *Fast Fourier Transform* (FFT) in the 60's by Cooley and Tukey [CT65], it has become a basic building block in many applications, e.g., in sound analysis and optical imaging, but also in implementing a fast multiplication of polynomials of high degree.

In this subsection, we elaborate on how we can implement the Fourier transform on a quantum computer. Later on, we will see that the circuit we devise here, is a very important building block in many more quantum algorithms, as it is frequently used as a subcircuit in much larger quantum circuits.

There exist many different definitions of the Fourier transform, though, as every branch of science in which the Fourier transform is used tends to use its own definition. In that respect, quantum computing is no different, as the definition of the Fourier transform used in quantum computing is subtly different from the definitions found in other areas. The version of the Fourier transform used in quantum computing is referred to as the *quantum Fourier transform.*[7] We will spend the next part of this section on introducing this transform and after that, we will elaborate on how this transform can be implemented in a quantum circuit.

For every natural number n, the n-qubit quantum Fourier transform implements the linear extension of the following mapping on the n-qubit computational basis states. Here, j is some integer in $\{0, 1, \ldots, 2^n - 1\}$:

$$|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} |k\rangle \tag{3.5.1}$$

We refer to this mapping by QFT_{2^n}. As the above relation is already expressed in terms of computational basis states, we can easily write down an expression for the entries of the matrix representation of this mapping. For notational convenience, we will assume that all our vectors and matrices are 0-indexed, like we did in Section 3.1. Now, observe that for all $j, k \in \{0, 1, \ldots, 2^n - 1\}$:

$$\left(\mathrm{QFT}_{2^n}\right)_{jk} = \frac{1}{\sqrt{2^n}} \cdot e^{\frac{2\pi i j k}{2^n}} = \frac{1}{\sqrt{2^n}} \cdot \omega_{2^n}^{jk} \qquad \text{with} \qquad \omega_{2^n} = e^{\frac{2\pi i}{2^n}} \tag{3.5.2}$$

The matrix representation of the n-qubit quantum Fourier transform can now concisely be expressed as follows:

$$\mathrm{QFT}_{2^n} = \frac{1}{\sqrt{2^n}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_{2^n} & \omega_{2^n}^2 & \cdots & \omega_{2^n}^{2^n-1} \\ 1 & \omega_{2^n}^2 & \omega_{2^n}^4 & \cdots & \omega_{2^n}^{2(2^n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{2^n}^{2^n-1} & \omega_{2^n}^{2(2^n-1)} & \ldots & \omega_{2^n}^{(2^n-1)(2^n-1)} \end{bmatrix} = \frac{1}{\sqrt{2^n}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_{2^n} & \omega_{2^n}^2 & \cdots & \omega_{2^n}^{-1} \\ 1 & \omega_{2^n}^2 & \omega_{2^n}^4 & \cdots & \omega_{2^n}^{-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{2^n}^{-1} & \omega_{2^n}^{-2} & \cdots & \omega_{2^n} \end{bmatrix}$$

In the last equality, we used that $\omega_{2^n}^{2^n} = 1$. Next, in order to check that the n-qubit quantum Fourier transform is unitary, we can simply check that the above matrix is unitary. To that end, we first of all from

[7] In this text, we will only be using the *Abelian quantum Fourier transform.* Generalizations to the non-Abelian case, albeit having very interesting and powerful properties, are left for another time. The interested reader is referred to [Bea97] or [Har05].

Equation 3.5.2 derive the following relation for the matrix entries of the Hermitian transpose of the matrix we found above, where again $j, k \in \{0, 1, \ldots, 2^n - 1\}$:

$$(\mathrm{QFT}^*_{2^n})_{jk} = \frac{1}{\sqrt{2^n}} \cdot \overline{\omega_{2^n}^{kj}} = \frac{1}{\sqrt{2^n}} \cdot \overline{e^{\frac{2\pi i}{2^n} \cdot kj}} = \frac{1}{\sqrt{2^n}} \cdot e^{\frac{2\pi i}{2^n} \cdot -jk} = \frac{1}{\sqrt{2^n}} \cdot \omega_{2^n}^{-jk}$$

Using this relation, we can now concisely express the matrix $\mathrm{QFT}^*_{2^n}$. Note that it is very similar to the matrix QFT_{2^n}, and that the only difference is the minus signs in the exponents.

$$\mathrm{QFT}^*_{2^n} = \frac{1}{\sqrt{2^n}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_{2^n}^{-1} & \omega_{2^n}^{-2} & \cdots & \omega_{2^n} \\ 1 & \omega_{2^n}^{-2} & \omega_{2^n}^{-4} & \cdots & \omega_{2^n}^{2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{2^n} & \omega_{2^n}^{2} & \cdots & \omega_{2^n}^{-1} \end{bmatrix}$$

Now, in order to check that QFT_{2^n} is unitary, we must check that $\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n} = I_{2^n}$, i.e., the $2^n \times 2^n$ identity matrix. To that end, we can simply calculate the matrix entries of the matrix product $\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n}$. Hence, observe that for all $j, k \in \{0, 1, \ldots, 2^n - 1\}$:

$$(\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n})_{jk} = \sum_{\ell=0}^{2^n-1} (\mathrm{QFT}_{2^n})_{j\ell} (\mathrm{QFT}^*_{2^n})_{\ell k} = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \omega_{2^n}^{\ell(j-k)} \tag{3.5.3}$$

Now if $j = k$, all the terms in the summation on the right-hand side reduce to $\omega_{2^n}^0 = 1$. Hence, then we obtain:

$$(\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n})_{jk} = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \omega_{2^n}^{\ell(j-k)} = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \omega_{2^n}^{0} = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} 1 = \frac{2^n}{2^n} = 1 \tag{3.5.4}$$

On the other hand, if $j \neq k$, then we must do some more work to rewrite the summation on the right-hand side of Equation 3.5.3. To that end, we use the following relation, for all $n \in \mathbb{N}$ and $r \in \mathbb{C} \setminus \{1\}$, which can easily be proven by long division of $1 - r^n$ by $1 - r$:

$$\sum_{k=0}^{n-1} r^k = \frac{1 - r^n}{1 - r}$$

Indeed, if $j \neq k$, then $\omega_{2^n}^{j-k} \neq 1$, and hence the right-hand side of Equation 3.5.3 reduces to:

$$(\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n})_{jk} = \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \left(\omega_{2^n}^{j-k}\right)^{\ell} = \frac{1}{2^n} \cdot \frac{1 - \omega_{2^n}^{2^n(j-k)}}{1 - \omega_{2^n}^{j-k}} = \frac{1}{2^n} \cdot \frac{1 - 1^{j-k}}{1 - \omega_{2^n}^{j-k}} = \frac{0}{2^n(1 - \omega_{2^n}^{j-k})} = 0 \tag{3.5.5}$$

Hence, combining Equation 3.5.4 and Equation 3.5.5, we obtain, for all $j, k \in \{0, 1, \ldots, 2^n - 1\}$:

$$(\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n})_{jk} = \delta_{jk}$$

This indeed implies that $\mathrm{QFT}_{2^n} \mathrm{QFT}^*_{2^n} = I$, indicating that the n-qubit quantum Fourier transform is indeed a unitary operation.

Now, we will have a look at the implementation of the quantum Fourier transform. It so happens that the crucial observation that Cooley and Tukey made in the 1960s which allowed them to devise the FFT algorithm, also plays a central role in the design of a quantum circuit that implements the quantum Fourier transform. This observation, in the context of quantum computing, is that we can rewrite the definition of the n-qubit quantum Fourier transform in terms of the $(n-1)$-qubit quantum Fourier transform. In other

words, the right-hand side of Equation 3.5.1 can be rewritten as follows. Here, we have $j \in \{0, 1, \ldots, 2^n - 1\}$ and we let $j_{n-1}j_{n-2}\ldots j_1 j_0$ be the binary expansion of j. That is, $j = (j_{n-1}j_{n-2}\ldots j_1 j_0)_2$.

$$\begin{aligned}
\mathrm{QFT}_{2^n}|j\rangle &= \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}}|k\rangle = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i j\cdot 2k}{2^n}}|2k\rangle + \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i j\cdot(2k+1)}{2^n}}|2k+1\rangle \\
&= \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i j k}{2^{n-1}}}|k\rangle\otimes|0\rangle + \frac{e^{\frac{2\pi i j}{2^n}}}{\sqrt{2^n}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i j k}{2^{n-1}}}|k\rangle\otimes|1\rangle \\
&= \left(\frac{1}{\sqrt{2^{n-1}}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i j k}{2^{n-1}}}|k\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i j}{2^n}}|1\rangle\right) \\
&= \left(\frac{1}{\sqrt{2^{n-1}}}\sum_{k=0}^{2^{n-1}-1} e^{\frac{2\pi i (j_{n-2}j_{n-3}\cdots j_1 j_0)_2 k}{2^{n-1}}}|k\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i j}{2^n}}|1\rangle\right) \\
&= \mathrm{QFT}_{2^{n-1}}|(j_{n-2}j_{n-3}\ldots j_1 j_0)_2\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i j}{2^n}}|1\rangle\right)
\end{aligned}$$

Thus, we have now expressed the action of the n-qubit quantum Fourier transform on a computational basis state in terms of the $(n-1)$-qubit quantum Fourier transform on a computational basis state. We can now recursively apply this identity, which allows us to obtain the following expression:

$$\mathrm{QFT}_{2^n}|j\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (j_0)_2}{2}}|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (j_1 j_0)_2}{4}}|1\rangle\right)\otimes\cdots\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (j_{n-1}j_{n-2}\cdots j_1 j_0)_2}{2^n}}|1\rangle\right) \tag{3.5.6}$$

From this expression, we readily observe that the quantum Fourier transform of a computational basis state is not an entangled state, as we have expressed it as a pure tensor of single-qubit states. This, in itself, is a remarkable result, and in essence is where the computational speed-up of the Cooley-Tukey classical FFT algorithm originates from.

Now, we are in good position to describe a quantum circuit that implements the n-qubit quantum Fourier transform. To that end, again suppose that our starting state is $|j\rangle$, where $j \in \{0, 1, \ldots, 2^n - 1\}$. We can rewrite this state as follows:

$$|j\rangle = |j_{n-1}\rangle\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle$$

Now, we take a look at the right-most tensor in the right-hand side of Equation 3.5.6. Observe that we can construct this state from $|j\rangle$ in the left-most single-qubit state, as follows. First of all, we apply a Hadamard gate to the first qubit, to obtain the following state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^{j_{n-1}}|1\rangle\right)\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i\cdot 2^{n-1}j_{n-1}}{2^n}}|1\rangle\right)\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle$$

Next, we apply an $R_{\frac{1}{4}}$ gate to the first qubit, controlled on the second qubit. This leaves us with the following state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i\cdot(2^{n-1}j_{n-1}+2^{n-2}j_{n-2})}{2^n}}|1\rangle\right)\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle$$

Next, we apply $R_{\frac{1}{8}}$ to the first qubit, controlled on the third qubit, $R_{\frac{1}{16}}$ controlled on the fourth qubit, etc, up until $R_{\frac{1}{2^n}}$ controlled on the nth qubit. After all these gates, we end up with the following state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i\cdot\sum_{k=0}^{n-1}2^k j_k}{2^n}}|1\rangle\right)\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (j_{n-1}j_{n-2}\cdots j_1 j_0)_2}{2^n}}\right)\otimes|j_{n-2}\rangle\otimes\cdots\otimes|j_1\rangle\otimes|j_0\rangle$$

Hence, we have constructed the right-most single-qubit tensor in Equation 3.5.6, in the left-most qubit of our n-qubit system. Now, we take a look at the second to last single-qubit state in Equation 3.5.6. Here, we have

$(j_{n-2}j_{n-3}\dots j_1j_0)_2$ in the exponent, so we no longer need the value of j_{n-1} to construct this state. Hence, using the last $n-1$ qubits, we can construct this single-qubit state in the second qubit of our system, using the exact same method as for the first qubit. Thus, we first apply a Hadamard gate to the second qubit, and then we apply the $R_{\frac{1}{4}}$-gate to the second qubit, controlled on the third qubit, followed by a $R_{\frac{1}{8}}$-gate controlled on the fourth qubit, etc. Then, we end up with the following state:

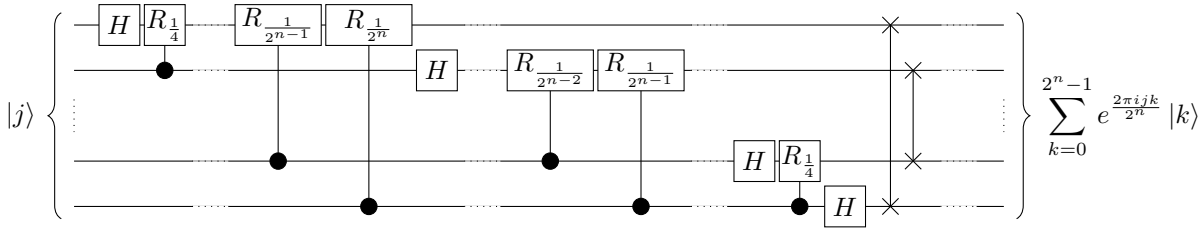
$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(j_{n-1}j_{n-2}\cdots j_1j_0)_2}{2^n}}\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(j_{n-2}j_{n-3}\cdots j_1j_0)_2}{2^{n-1}}}|1\rangle\right) \otimes |j_{n-3}\rangle \otimes \cdots \otimes |j_1\rangle \otimes |j_0\rangle$$

Next, we iterate this procedure to construct similar states in the remaining qubits. Then, we have put our system into the following state:

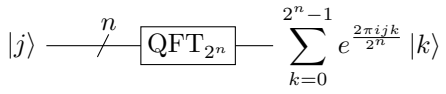
$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(j_{n-1}j_{n-2}\cdots j_1j_0)_2}{2^n}}\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(j_{n-2}j_{n-3}\cdots j_1j_0)_2}{2^{n-1}}}|1\rangle\right) \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(j_0)_2}{2}}|1\rangle\right)$$

Finally, we can swap the nth qubit with the first qubit, the $(n-1)$th qubit with the second qubit, etc., to obtain the state displayed in Equation 3.5.6. This, then, completes our description of the implementation of the quantum Fourier transform as a quantum circuit. The entire process is summarized and depicted in Circuit 3.5.4.

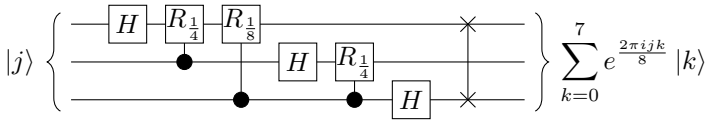
Circuit 3.5.4: n-qubit quantum Fourier transform
Description: This circuit implements the n-qubit quantum Fourier transform.
Elementary gate complexity: $(n+1)n/2 + 3\lfloor n/2 \rfloor = \mathcal{O}(n^2)$.
Circuit: Here, $j \in \{0, 1, \dots, 2^n - 1\}$:



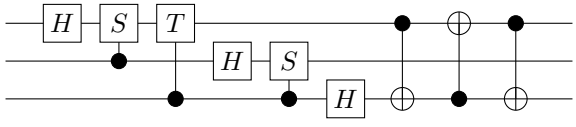
Shorthand notation:



For extra clarification, let us write out the full quantum circuit implementing the three-qubit quantum Fourier transform QFT_{2^3}:



Recall from Subsection 3.2.1 that the $R_{1/4}$ and $R_{1/8}$ gates are equal to the S and T gates. Hence, we can rewrite the circuit as follows:



Here, we have also expanded the SWAP-circuit, to obtain the representation of this circuit in elementary gates. Now, we can also easily observe that the elementary gate complexity is equal to 9, which is equal to

the expected number when $n = 3$:

$$(n+1)n/2 + 3\lfloor n/2 \rfloor = 4 \cdot 3/2 + 3 \cdot 1 = 6 + 3 = 9$$

In this section, we have shown how to implement the mapping QFT_{2^n} exactly, using a number of elementary gates that grows quadratically in n. Note that the number of CNOT-gates and the number of Hadamard gates grows only linearly in n, though, and that only the number of controlled-R-gates grows quadratically in n. More precisely, observe that we use exactly $n - k + 1$ controlled applications of $R_{\frac{1}{2^k}}$. In addition, observe:

$$\left\| C\left(R_{\frac{1}{2^k}}\right) - I \right\| = \left\| \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right\| = \left\| \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^k}} - 1 \end{bmatrix} \right\|$$
$$= \left| e^{\frac{2\pi i}{2^k}} - 1 \right| = 2 \left| \sin\left(\frac{\pi}{2^k}\right) \right| \leq \frac{2\pi}{2^k}$$

In other words, the controlled $R_{\frac{1}{2^k}}$-gates where k is big, are very close to the identity gate in operator norm, indicating that their action is almost negligible. Hence, we might just as well leave out these controlled-R-gates where k is big. This idea allows us to obtain a significant reduction in the elementary gate complexity. We can implement the quantum Fourier transform up to some error ε in operator norm, using only $\mathcal{O}(n \log(n))$ elementary gates. In [BEST96], this construction is made explicit.

This completes our discussion of the quantum Fourier transform. In subsequent subsections, we will see how the quantum Fourier transform is used in a variety of different quantum circuits.

3.5.4 Quantum Fourier adder

In the last subsection, we have introduced the n-qubit quantum Fourier transform, where n could be any natural number. In this section, we will see the first application of the quantum Fourier transform, in a rather unexpected setting. We will show how we can add two n-bit integers a and b modulo 2^n, using two applications of the n-qubit quantum Fourier transform. The construction presented here is due to [Dra00].

So, suppose that we have two n-bit integers a and b, hence $a, b \in \{0, 1, \ldots, 2^n - 1\}$. The goal is to construct a $2n$-qubit quantum circuit, where the first n qubits constitute the first register, and the second n qubits form the second register, implementing the following mapping:

$$|a\rangle \otimes |b\rangle \mapsto |a\rangle \otimes |a + b\rangle$$

The idea is to first apply the quantum Fourier transform on the second register. Then, we obtain the following state, according to Equation 3.5.6, where $a_{n-1}a_{n-2}\ldots a_1a_0$ and $b_{n-1}b_{n-2}\ldots b_1b_0$ are the binary expansions of a and b, respectively:

$$|a_{n-1}\rangle \otimes \cdots \otimes |a_0\rangle \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (b_0)_2}{2}} |1\rangle\right) \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (b_{n-1}b_{n-2}\ldots b_1b_0)_2}{2^n}} |1\rangle\right)$$

It now becomes apparent that the integer b is stored in the phases of the $|1\rangle$ states of the final n qubits. More precisely, we find that the $(n + j)$th qubit is in state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i (b_{j-1}b_{j-2}\cdots b_1b_0)_2}{2^j}} |1\rangle\right)$$

We will now methodically modify this state until the integer $a+b$ is stored in these phases in the exact same way, i.e., until the $(n + j)$th qubit is in the following state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i ((a_{j-1}a_{j-2}\cdots a_1a_0)_2 + (b_{j-1}b_{j-2}\cdots b_1b_0)_2)}{2^j}} |1\rangle\right)$$

This comes down to shifting the phase of the amplitude of the $|1\rangle$ state in the $(n+j)$th qubit by the following amount:

$$\frac{2\pi i(a_{j-1}a_{j-2}\cdots a_1a_0)_2}{2^j}$$

Such a phase shift can be obtained using consecutive controlled rotations. For instance, for the $(n+1)$st qubit, we must shift the phase by $2\pi i a_0/2$. Recall that the state of the nth qubit is $|a_0\rangle$. Hence, if we apply a $R_{1/2}$-gate, controlled on the nth qubit, then nothing happens if $a_0 = 0$, which amounts to a phase shift of $0 = 2\pi i a_0/2$. On the other hand, if $a_0 = 1$, we again obtain a phase shift of $2\pi i/2 = 2\pi i a_0/2$.

Similarly, for the $(n+2)$nd qubit, we must shift the phase by the following amount:

$$\frac{2\pi i(a_1a_0)_2}{4} = \frac{2\pi i a_1}{2} + \frac{2\pi i a_0}{4}$$

Hence, we can use two controlled rotations, a $R_{1/2}$-gate controlled on the $(n-1)$st qubit, and a $R_{1/4}$-gate controlled on the nth qubit. If we recall that the states of these $(n-1)$st and nth qubits are $|a_1\rangle$ and $|a_0\rangle$, respectively, we obtain by the same logic as before that we implement the correct phase shifts.

We can repeat this process for all $(n+j)$th qubits, where j runs from 1 to n, until we have successfully *rotated in* the necessary phases in the final n qubits. The resulting state becomes:

$$|a_{n-1}\rangle\otimes\cdots\otimes|a_0\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i((a_0)_2+(b_0)_2)}{2}}|1\rangle\right)\otimes\cdots\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i((a_{n-1}a_{n-2}\ldots a_1a_0)_2+(b_{n-1}b_{n-2}\ldots b_1b_0)_2)}{2^n}}|1\rangle\right)$$

Equation 3.5.6 tells us that the state of the second register is now the image of $|a+b\rangle$ under the n-qubit quantum Fourier transform. So, we can apply the n-qubit quantum Fourier transform *in reverse*, to obtain the following state:

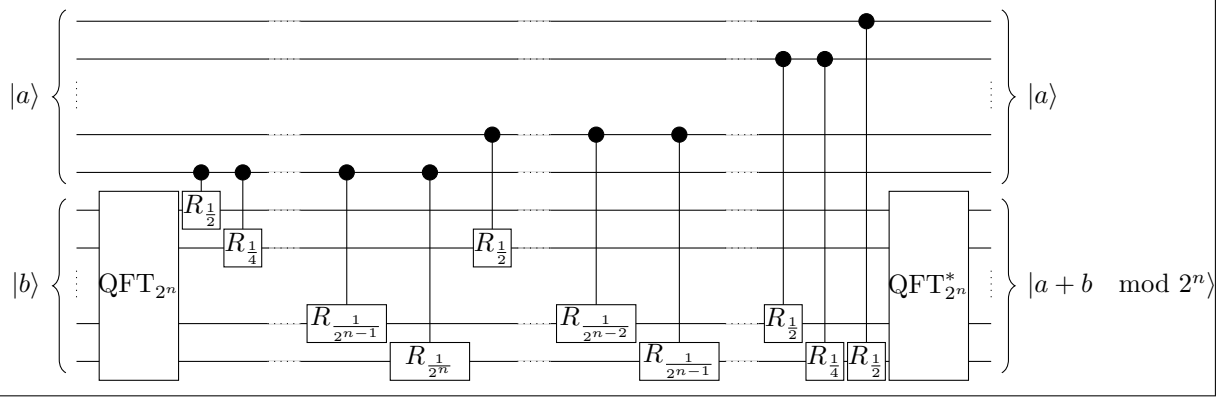
$$|a\rangle\otimes|a+b\rangle$$

This is the state we were after, and hence we have found a way to add two numbers on a quantum computer. We summarize this construction in Circuit 3.5.5, and also provide a graphical depiction of the resulting quantum circuit there.

Circuit 3.5.5: n-qubit quantum Fourier adder
Description: This circuit calculates the sum of two n-bit numbers modulo 2^n.
Elementary gate complexity:

$$3(n+1)n/2 + 3\lfloor n/2\rfloor = \mathcal{O}(n^2)$$

Circuit: Here, $a, b \in \{0, 1, \ldots, 2^n - 1\}$:



Observe that the elementary gate complexity of this circuit scales quadratically with the number of qubits, n. However, again we are using many controlled rotation gates whose action is almost negligible. Hence,

using a similar argument as with the quantum Fourier transform, we can obtain a significant reduction in the elementary gate complexity by leaving out some of the controlled R-gates that have little effect. If we, in addition, also replace the quantum Fourier transform circuits by their approximate versions, then we can reduce the entire elementary gate complexity to $\mathcal{O}(n \log(n))$.

As a final remark, we note that it is also possible to add two numbers modulo 2^n using only $\mathcal{O}(n)$ elementary gates, by translating a reversible classical circuit to the quantum setting using Toffoli gates. The interested reader is referred to [VBE96], or [Cor16], Section 5.3.2. However, in this construction, one requires n auxiliary qubits, which are not needed in the construction presented in Circuit 3.5.5. Thus, if adding a number of qubits is more expensive than adding a number of elementary gates to a quantum circuit, then Circuit 3.5.5 might still have some practical value.

This completes our discussion of the quantum Fourier adder. This was the first application of the quantum Fourier transform that we encountered. In the next subsection, we will see another application of the quantum Fourier transform, in a circuit known as the phase estimation circuit.

3.5.5 Phase estimation

In the previous subsection, we have seen one application of the quantum Fourier transform, namely the quantum Fourier adder. In this subsection, we will see another, arguably more important application of the quantum Fourier transform. We will use the quantum Fourier transform to recover eigenvalues of quantum circuits, with an algorithm known as the phase estimation algorithm.

First of all, we will introduce what we mean by eigenstates and eigenvalues of quantum circuits. After that, we will elaborate on how the quantum Fourier transform can be used to find such eigenvalues of quantum circuits. Finally, we will summarize this method in the phase estimation algorithm, and we will prove a lower bound on the success probability of this algorithm.

Suppose that we have a quantum circuit, U, acting on n qubits. Then, its matrix representation is a $2^n \times 2^n$ unitary matrix, as we argued in Section 3.3, which we also refer to as U. Next, suppose that we have an eigenvector of this matrix U, which we call u. As U is unitary, we have that the eigenvalue corresponding to the eigenvector u has modulus 1, and as such it can be written as $e^{2\pi i\phi}$, for $\phi \in [0, 1)$. Hence, we find $Uu = e^{2\pi i\phi}u$. Moreover, recall that u is identified with a quantum state $|u\rangle$, which is a vector in the state space of an n-qubit system. Now, if we apply the circuit U to an n-qubit system which is prepared in the state vector $|u\rangle$, we obtain:

$$U|u\rangle = e^{2\pi i\phi}|u\rangle$$

Hence, the state of the system is only changed by a global phase factor, which cannot be measured, as we argued in Section 2.2. We refer to $|u\rangle$ as an *eigenstate of the quantum circuit* U. Corresponding to this eigenstate $|u\rangle$, we have the *eigenvalue of the quantum circuit* U, $e^{2\pi i\phi}$.

Next, suppose that we have been given a quantum circuit U, and an eigenstate $|u\rangle$ of this quantum circuit U. Then, we know that corresponding to this eigenstate $|u\rangle$, there exists an eigenvalue of the quantum circuit U, which can be written as $e^{2\pi i\phi}$ with $\phi \in [0, 1)$. Given the quantum circuit U and the eigenstate $|u\rangle$, our goal is now to find an approximation to the value of ϕ. That is, for some $\varepsilon > 0$, we want to find a value $\tilde{\phi}$ such that ϕ and $\tilde{\phi}$ are at most ε apart. Here, we consider the endpoints of $[0, 1)$ to be connected, i.e., we say that 0.99 and 0.01 are only separated by a distance of 0.02.

As a first step towards developing a method to obtain such an approximation to ϕ, we first of all introduce a constant. Let $m = \lceil \log\left(\frac{1}{\varepsilon}\right) \rceil$, where we take the logarithm in base 2. This is the number of bits that we will use to express $\tilde{\phi}$. Next, we make a simplifying assumption, namely, we assume that $2^m\phi$ is an integer. This assumption will make sure that the algorithm succeeds with success probability 1, which simplifies the analysis considerably. Later on, we will relax this assumption, and show that the algorithm still works with some reasonable success probability if $2^m\phi$ is not integer.

As $2^m\phi$ is an integer, and as $\phi \in [0,1)$, we obtain that $2^m\phi \in \{0,1,\ldots,2^m-1\}$. Hence, we find that the m-qubit state $|2^m\phi\rangle$ is a computational basis state. Moreover, its image under the m-qubit quantum Fourier transform is given by:

$$\mathrm{QFT}_{2^m}|2^m\phi\rangle = \frac{1}{\sqrt{2^m}}\sum_{k=0}^{2^m-1} e^{\frac{2\pi i\cdot 2^m\phi k}{2^m}}|k\rangle = \frac{1}{\sqrt{2^m}}\sum_{k=0}^{2^m-1} e^{2\pi i\phi k}|k\rangle \tag{3.5.7}$$

Let $\phi_{m-1}\phi_{m-2}\ldots\phi_1\phi_0$ be the binary expansion of $2^m\phi$, i.e., $2^m\phi = (\phi_{m-1}\phi_{m-2}\ldots\phi_1\phi_0)_2$. Similarly to what we did before in the derivation of Equation 3.5.6, we can write the right-hand side of Equation 3.5.7 as a pure tensor of single-qubit states:

$$\mathrm{QFT}_{2^m}|2^m\phi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(\phi_0)_2}{2}}|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(\phi_0\phi_1)_2}{4}}|1\rangle\right)\otimes\cdots\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi i(\phi_{n-1}\phi_{n-2}\cdots\phi_1\phi_0)_2}{2^m}}|1\rangle\right)$$

This can easily be rewritten into the following form (the terms that are added in the exponents are all multiples of $2\pi i$):

$$\mathrm{QFT}_{2^m}|2^m\phi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-1}\phi}|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-2}\phi}\right)\otimes\cdots\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^0\phi}|1\rangle\right) \tag{3.5.8}$$

But now, we can see how this state can be constructed. We can take an $(n+m)$-qubit system, and prepare the first m qubits in the state $|0\rangle^{\otimes m}$, and the last n qubits in the state $|u\rangle$, i.e., the eigenstate of the quantum circuit U that we have been given. So, the initial state is $|0\rangle^{\otimes m}\otimes|u\rangle$. Next, we apply a Hadamard-gate to the first m qubits. The resulting state is:

$$\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)^{\otimes m}\otimes|u\rangle \tag{3.5.9}$$

Now, we can construct the state displayed in Equation 3.5.8 in the first m qubits of this $(n+m)$-qubit system. To that end, we first of all consider what happens if we do a controlled application of the quantum circuit U. The action of a controlled application of the quantum circuit U can be expressed as follows:

$$|0\rangle\otimes|u\rangle \overset{C(U)}{\mapsto} |0\rangle\otimes|u\rangle \qquad \text{(Nothing happens, as the control qubit is 0)}$$
$$|1\rangle\otimes|u\rangle \overset{C(U)}{\mapsto} |1\rangle\otimes U|u\rangle = |1\rangle\otimes e^{2\pi i\phi}|u\rangle = e^{2\pi i\phi}|1\rangle\otimes|u\rangle \qquad \text{(The quantum circuit } U \text{ is applied)}$$

So, by linearity, we obtain that $C(U)$ has the following action if the control qubit is in a superposition of $|0\rangle$ and $|1\rangle$:

$$\begin{aligned} &\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\otimes|u\rangle = \frac{1}{\sqrt{2}}|0\rangle\otimes|u\rangle + \frac{1}{\sqrt{2}}|1\rangle\otimes|u\rangle \\ \overset{C(U)}{\mapsto}\ &\frac{1}{\sqrt{2}}|0\rangle\otimes|u\rangle + \frac{1}{\sqrt{2}}\cdot e^{2\pi i\phi}|1\rangle\otimes|u\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\phi}|1\rangle\right)\otimes|u\rangle \end{aligned}$$

Hence, if we have our $(n+m)$-qubit system in the state described by Equation 3.5.9, then we can apply the quantum circuit U to the last n qubits, controlled on the mth qubit, in order to obtain the following state:

$$\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)^{\otimes(m-1)}\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\phi}|1\rangle\right)\otimes|u\rangle$$

Next, we can apply the quantum circuit U twice to the last n qubits, controlled on the $(m-1)$st qubit. That is, we apply $C(U^2)$, where the control qubit is the $(m-1)$st qubit, and the target qubits are the last n qubits of the system. Now, the $|1\rangle$-state in the $(m-1)$st qubit picks up a phase factor of $e^{2\pi i\cdot 2\phi} = \left(e^{2\pi i\phi}\right)^2$,

i.e., one phase factor $e^{2\pi i\phi}$ for every time that U is applied. Thus, the $(n+m)$-qubit system ends up in the following state:

$$\left(\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)\right)^{\otimes(m-2)} \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2\phi}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\phi}|1\rangle\right) \otimes |u\rangle$$

Continuing in this manner, i.e., applying a $C(U^4)$-operation with the control qubit being the $(m-2)$nd qubit, etc., all the way to applying a $C(U^{2^{m-1}})$-operation where the first qubit is the control qubit, allows us to construct the following state:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-1}\phi}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-2}\phi}|1\rangle\right) \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{0}\phi}|1\rangle\right) \tag{3.5.10}$$

But this state is equal to $\mathrm{QFT}_{2^m}|2^m\phi\rangle$, as shown in Equation 3.5.8. Hence, if we apply the inverse of the m-qubit quantum Fourier transform to this state, we obtain the state $|2^m\phi\rangle$. We can measure this state in the computational basis to recover $2^m\phi$, from which we can easily calculate ϕ. Thus, under our simplifying assumption, i.e., when $2^m\phi$ is an integer, we have constructed a quantum circuit that allows us to recover ϕ exactly.

Next, we drop the assumption that $2^m\phi$ is an integer and give an indication why the above procedure will still yield a reasonable approximation of ϕ. We can, namely, apply the exact same procedure as we have described above. Just before applying the inverse quantum Fourier transform, it remains valid that the system is in the state described by Equation 3.5.10, even though this time it is not necessarily the image of a computational basis state under the m-qubit quantum Fourier transform. We can consider the function mapping the value of ϕ to this state:

$$\phi \mapsto \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-1}\phi}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{m-2}\phi}|1\rangle\right) \otimes \cdots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\cdot 2^{0}\phi}|1\rangle\right)$$

Note that this mapping is continuous, due to the continuous nature of all its constituents. Hence, intuitively, if $2^m\phi$ is *close to some integer* $2^m\tilde{\phi}$, then the resulting state just before applying the inverse quantum Fourier transform is *close to the state* $\mathrm{QFT}_{2^m}\left|2^m\tilde{\phi}\right\rangle$. Hence, upon measurement in the computational basis after the quantum Fourier transform, we will with reasonable probability obtain the outcome $2^m\tilde{\phi}$, and as $2^m\tilde{\phi}$ was close to $2^m\phi$, this gives rise to a close approximation of ϕ. This argument is made rigorous below.

The quantum algorithm that we have described in this section is summarized in Algorithm 3.5.6. Here, we also present an entirely rigorous analysis of the case when $2^m\phi$ is not an integer.

Algorithm 3.5.6: Phase estimation
Input:

1. A positive real number $\varepsilon > 0$, indicating the precision with which we want to approximate ϕ.
2. A positive real number $\delta > 0$, indicating the maximally tolerated failure probability.
3. A quantum circuit U acting on n qubits, with corresponding n-qubit eigenstate $|u\rangle$.
4. For all $j \in \{0, 1, \ldots, m-1\}$, an oracle circuit $C(U^{2^j})$, i.e., a circuit that applies U^{2^j} conditioned on one qubit (where m is one of the derived constants).

Derived constants:

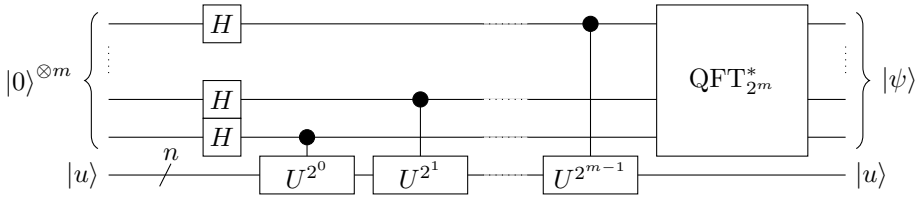
1. $m = \lceil \log(\frac{\frac{1}{\delta}+4}{\varepsilon}) \rceil$.
2. $\phi \in [\varepsilon, 1-\varepsilon)$ such that $e^{2\pi i\phi}$ is the eigenvalue corresponding to $|u\rangle$.

Output: A number $\tilde{\phi} \in [0, 1)$ such that ϕ and $\tilde{\phi}$ are at most ε apart, i.e., $\min\{|\phi - \tilde{\phi}|, 1 - |\phi - \tilde{\phi}|\} < \varepsilon$.
Success probability: Lower bounded by $1 - \delta$.
Number of qubits: $n + m$.
Oracle query complexity: One oracle query to $C(U^{2^j})$, for all $j \in \{0, 1, \ldots, m-1\}$.
Elementary gate complexity:

$$m + \sum_{j=0}^{m-1} \mathrm{egc}\left(C\left(U^{2^j}\right)\right) + (m+1)m/2 + 3\lfloor m/2 \rfloor = \sum_{j=0}^{m-1} \mathrm{egc}\left(C\left(U^{2^j}\right)\right) + \mathcal{O}(m^2)$$

Algorithm:

1. Prepare the $(n+m)$-qubit system in the state $|0\rangle^{\otimes m} \otimes |u\rangle$.
2. Apply the phase estimation circuit:



3. Measure the first register (i.e., the first m qubits) in the computational basis. Interpret the resulting bitstring as an integer, and denote this integer by j.
4. Return $\tilde{\phi} = j/2^m$.

Proof of the lower bound on the success probability of Algorithm 3.5.6. We will track what happens to the state throughout the circuit. The initial state is $|0\rangle^{\otimes m} \otimes |u\rangle$. After the application of the m Hadamard gates, the resulting state is:

$$(H^{\otimes m} \otimes I_{2^n})(|0\rangle^{\otimes m} \otimes |u\rangle) = (H|0\rangle)^{\otimes m} \otimes |u\rangle = \frac{1}{\sqrt{2^m}} (|0\rangle + |1\rangle)^{\otimes m} \otimes |u\rangle = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j\rangle \otimes |u\rangle$$

Now, for the second part, i.e., all the controlled applications of powers of U, suppose that the first register is in state $|j\rangle$. Let $j_{m-1}j_{m-2}\ldots j_1 j_0$ be the binary representation of j. Then, we have:

$$j = (j_{m-1}j_{m-2}\ldots j_1 j_0)_2 = \sum_{k=0}^{m-1} 2^k j_k$$

Also, observe that the following operations are applied to the last n qubits:

$$\left(U^{2^{m-1}}\right)^{j_{m-1}} \cdot \left(U^{2^{m-2}}\right)^{j_{m-2}} \cdot \cdots \cdot \left(U^{2^1}\right)^{j_1} \cdot \left(U^{2^0}\right)^{j_0} = U^{\sum_{k=0}^{m-1} 2^k j_k} = U^j$$

Hence, by linearity, we obtain that the second part performs the following action on the state of the system:

$$\frac{1}{\sqrt{2^m}}\sum_{j=0}^{2^m-1}|j\rangle\otimes|u\rangle \mapsto \frac{1}{\sqrt{2^m}}\sum_{j=0}^{2^m-1}|j\rangle\otimes U^j|u\rangle = \frac{1}{\sqrt{2^m}}\sum_{j=0}^{2^m-1}|j\rangle\otimes e^{2\pi i\phi j}|u\rangle$$

where in the last inequality, we used that $|u\rangle$ is an eigenstate of U, with eigenvalue $e^{2\pi i\phi}$. Next, the inverse quantum Fourier transform is applied to the first m qubits, and hence the resulting output state is:

$$\frac{1}{\sqrt{2^m}}\sum_{j=0}^{2^m-1}\frac{1}{\sqrt{2^m}}\sum_{k=0}^{2^m-1}e^{-\frac{2\pi ijk}{2^m}}|k\rangle\otimes e^{2\pi ij\phi}|u\rangle = \frac{1}{2^m}\sum_{k=0}^{2^m-1}\sum_{j=0}^{2^m-1}e^{\frac{2\pi ij(2^m\phi-k)}{2^m}}|k\rangle\otimes|u\rangle$$

So, we now obtain that:

$$|\psi\rangle = \frac{1}{2^m}\sum_{k=0}^{2^m-1}\sum_{j=0}^{2^m-1}e^{\frac{2\pi ij(2^m\phi-k)}{2^m}}|k\rangle$$

Next, we obtain, for all $\xi\in\{\frac{1}{2^m}\lfloor 2^m\phi\rfloor-\frac{1}{2}+\frac{1}{2^m},\frac{1}{2^m}\lfloor 2^m\phi\rfloor-\frac{1}{2}+\frac{2}{2^m},\ldots,\frac{1}{2^m}\lfloor 2^m\phi\rfloor+\frac{1}{2}\}$:

$$\langle 2^m\xi|\psi\rangle = \frac{1}{2^m}\sum_{j=0}^{2^m-1}e^{\frac{2\pi ij(2^m\phi-2^m\xi)}{2^m}} = \frac{1}{2^m}\sum_{j=0}^{2^m-1}e^{2\pi ij(\phi-\xi)} = \frac{1}{2^m}\sum_{j=0}^{2^m-1}\left(e^{2\pi i(\phi-\xi)}\right)^j$$

If $2^m\phi$ is an integer, then we find that $\langle 2^m\phi|2^m\psi\rangle = 1$, and hence the probability that the algorithm outputs exactly ϕ is 1. Hence, we only have to consider the case when $2^m\phi$ is not integer, in which case we can rewrite the above inner product in terms of a geometric series:

$$\langle 2^m\xi|\psi\rangle = \frac{1}{2^m}\cdot\frac{1-e^{2\pi i\cdot 2^m(\phi-\xi)}}{1-e^{2\pi i(\phi-\xi)}}$$

Taking the modulus on both sides yields, using $|1-e^{2i\phi}| = 2\sin|\phi|$ for all $\phi\in\mathbb{R}$, and $\sin(x)\geq\frac{2}{\pi}x$, for all $x\in(0,\pi/2)$:

$$|\langle 2^m\xi|\psi\rangle| = \frac{1}{2^m}\left|\frac{1-e^{2\pi i\cdot 2^m(\phi-\xi)}}{1-e^{2\pi i(\phi-\xi)}}\right| = \frac{\sin(\pi\cdot 2^m|\phi-\xi|)}{2^m\sin(\pi|\phi-\xi|)} \leq \frac{1}{2^m\cdot\frac{2}{\pi}\cdot\pi|\phi-\xi|} = \frac{1}{2^{m+1}|\phi-\xi|}$$

Next, take $a = \lceil 2^m\varepsilon-2\rceil$. Observe that whenever $k\in\lfloor 2^m\phi\rfloor-a,\ldots,\lfloor 2^m\phi\rfloor+a+1$, we have:

$$-(a+1)\leq 2^m\phi-\lfloor 2^m\phi\rfloor-(a+1)\leq 2^m\phi-k\leq 2^m\phi-\lfloor 2^m\phi\rfloor+a\leq a+1$$

Moreover, we then have:

$$|\phi-2^{-m}k|\leq 2^{-m}(a+1)\leq 2^{-m}\cdot(2^m\varepsilon-1+1)=\varepsilon$$

And so, we find:

$$\begin{aligned}
\mathbb{P}[\tilde{\phi}\notin(\phi-\varepsilon,\phi+\varepsilon)] &\leq \sum_{k=-2^m}^{-(a+1)}|\langle\lfloor 2^m\phi\rfloor+k|\psi\rangle|^2+\sum_{k=a+2}^{2^m-1}|\langle\lfloor 2^m\phi\rfloor+k|\psi\rangle|^2\\
&\leq \sum_{k=-2^m}^{-(a+1)}\frac{1}{4|\phi-\lfloor 2^m\phi\rfloor-k|^2}+\sum_{k=a+2}^{2^m-1}\frac{1}{4|\phi-\lfloor 2^m\phi\rfloor-k|^2}\\
&\leq \sum_{k=-2^m}^{-(a+1)}\frac{1}{4|k|^2}+\sum_{k=a+2}^{2^m-1}\frac{1}{4|k-1|^2}\leq\frac{1}{2}\int_a^\infty\frac{1}{x^2}\,\mathrm{d}x=\frac{1}{2}\left[-\frac{1}{x}\right]_a^\infty=\frac{1}{2a}\\
&\leq \frac{1}{2(2^m\varepsilon-2)}\leq\frac{1}{2(\frac{1}{2}\cdot\frac{\frac{1}{\delta}+4}{\varepsilon}\cdot\varepsilon-2)}=\delta
\end{aligned}$$

Hence the success probability is lower bounded by $1-\delta$. This completes the proof. □

As a final remark, we will take a closer look at the circuit presented in step 2 of Algorithm 3.5.6. Suppose that we prepare an $(n+m)$-qubit system in a state $|0\rangle^{\otimes m} \otimes (\alpha_1 |u_1\rangle + \cdots + \alpha_s |u_s\rangle)$, where $|u_1\rangle$ through $|u_s\rangle$ are different eigenstates of a quantum circuit U, corresponding to not necessarily distinct eigenvalues $e^{2\pi i \phi_1}$ through $e^{2\pi i \phi_s}$. If we now apply the phase estimation circuit to this initial state, then by linearity we obtain the following state:

$$\alpha_1 |\psi_1\rangle |u_1\rangle + \otimes + \alpha_s |\psi_s\rangle |u_s\rangle$$

The analysis of Algorithm 3.5.6 show that measuring $|\psi_r\rangle$, yields a close approximation of $2^m \phi_r$ with probability at least $1 - \delta$. Now, though, as we are dealing with a superposition with amplitudes $\alpha_1, \ldots, \alpha_s$, we obtain a close approximation of $2^m \phi_r$ with probability lower bounded by $|\alpha_r|^2(1 - \delta)$. Summing these probabilities and noting that $|\alpha_1|^2 + \cdots + |\alpha_s|^2 = 1$ yields that we obtain a close approximation to *some* eigenvalue of the quantum circuit U, with probability at least $1 - \delta$. Hence, we do not have to construct a precise eigenstate of U in the second register before the start of the phase estimation algorithm, but instead we can start with a superposition of eigenstates. Then, we don't know which eigenvalue we will be approximating, but we do know that we will obtain an approximation of *some* eigenvalue, which is sometimes all that we're after. In fact, this is the key observation in Shor's algorithm, as can be found in [NC00], Chapter 5, or [Cor16], Chapter 5.

This concludes our discussion on the phase estimation algorithm. We will encounter this algorithm again in Subsection 3.5.7, when we will use some specific circuits $C(U^{2^j})$ to estimate the amplitude of one particular state in a superposition state.

3.5.6 Amplitude amplification

In this subsection, we will consider the amplitude amplification algorithm. Its principal application is to search in large unstructured data sets, with a quadratic reduction in the number of queries to the data over the optimal classical search algorithms, as long as the data can be put into superposition efficiently. This specific database search algorithm is due to Grover and first appeared in [Gro96]. The generalized amplitude amplification algorithm is due to Brassard et al. [BHMT00]

Suppose that we have two orthogonal n-qubit states, $|G\rangle$ and $|B\rangle$, which we will refer to as the *good* and the *bad* state, respectively. Suppose that we want to approximately construct the good state, $|G\rangle$, and that we can use the following two oracle circuits to do so:

1. A circuit Q, acting on n qubits, that has the following action, for some $\theta \in (0, \frac{\pi}{2})$:

$$|0\rangle^{\otimes n} \overset{Q}{\mapsto} |U\rangle = \sin\theta |G\rangle + \cos\theta |B\rangle \tag{3.5.11}$$

2. A circuit R, acting on n qubits, that has the following action for all $\alpha, \beta \in [-1, 1]$ for which $|\alpha|^2 + |\beta|^2 = 1$ holds:

$$\alpha |G\rangle + \beta |B\rangle \overset{R}{\mapsto} -\alpha |G\rangle + \beta |B\rangle$$

The amplitude amplification circuit constructs the state $|G\rangle$ approximately, i.e., the Hilbert space norm of the difference of the state that the circuit constructs and the state $|G\rangle$ is small. How this is done can be very concisely represented in the following visualization.

Consider states of the form $\sin\phi |G\rangle + \cos\phi |B\rangle$, where $\phi \in [0, 2\pi)$. Now, we make the following identification:

$$\sin\phi |G\rangle + \cos\phi |B\rangle \qquad \leftrightarrow \qquad \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}$$

Hence, in other words, we picture all of the states $\sin\phi |G\rangle + \cos\phi |B\rangle$ as lying on the unit circle in $\mathbb{R}^2$. Specifically, we find that $|G\rangle$ points straight up, $|B\rangle$ points to the right, and $|U\rangle = \sin\theta |G\rangle + \cos\theta |B\rangle$ is

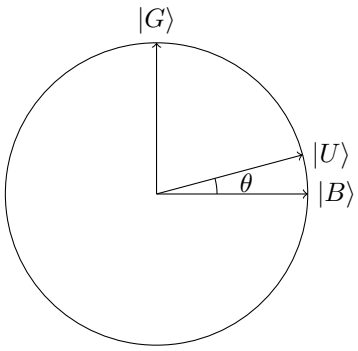


Figure 3.11: Visual interpretation of the three states that are available at the start of the amplitude amplification circuit.

located somewhere in between $|B\rangle$ and $|G\rangle$, as $\theta \in (0, \pi/2)$ by assumption. We have graphically depicted these three states in Figure 3.11.

Now, we start in the state $|U\rangle$, so we define $|\psi_0\rangle = |U\rangle$. Next, we reflect through $|B\rangle$, and the resulting state, we denote by $|\psi_1\rangle$. After that, we reflect through $|U\rangle$, and the state that results from that operation, we denote by $|\psi_2\rangle$. These states are shown in Figure 3.12.

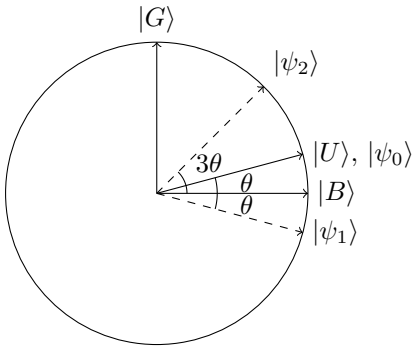


Figure 3.12: Visual interpretation of one Grover's iterate. The system starts out in the state $|\psi_0\rangle$. Then, the state is reflected through $|B\rangle$, and hence the state $|\psi_1\rangle$ is obtained. After that, the state is reflected through $|U\rangle$, which results in the state $|\psi_2\rangle$. One can prove that this combined action comes down to a rotation over an angle of 2θ counterclockwise, i.e., to get from $|\psi_0\rangle$ to $|\psi_2\rangle$, one must rotate along the circle in the counterclockwise direction over an angle of 2θ.

It can be seen that in order to get from $|\psi_0\rangle$ to $|\psi_2\rangle$, one must rotate along the circle by an angle of 2θ. If one were to reflect through $|B\rangle$ and $|U\rangle$ again, then one would rotate another 2θ along the circle. Such a rotation we refer to as a *Grover iterate*. The entire angle between $|U\rangle$ and $|G\rangle$ is $\frac{\pi}{2} - \theta$, and hence a Grover iterate, i.e., a rotation over the angle 2θ counterclockwise, must be applied the following number of times to arrive at $|G\rangle$ exactly:

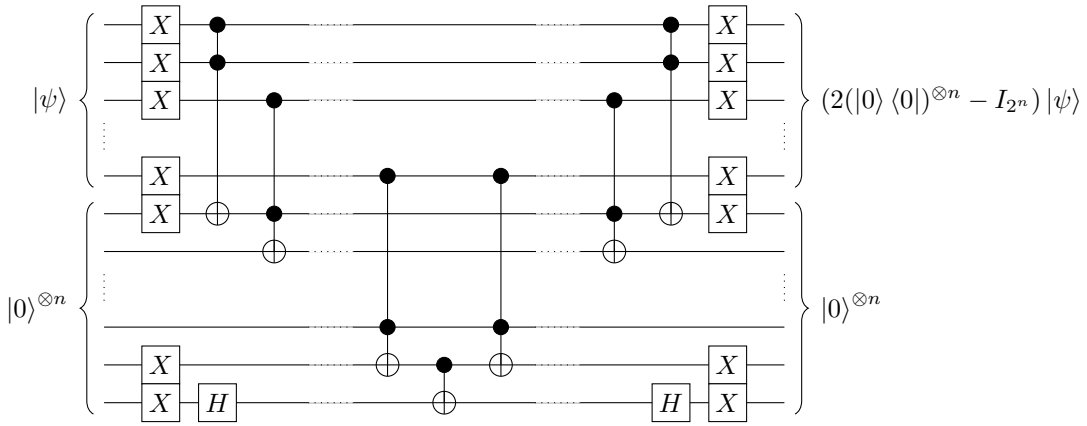
$$\frac{\frac{\pi}{2} - \theta}{2\theta} = \frac{\pi}{4\theta} - \frac{1}{2}$$

However, we can only apply the Grover iterate an integer number of times. Hence, we apply it a total number of $k = [\pi/4\theta - 1/2]$ times, where $[x]$ denotes x rounded to the nearest integer, and arrive at an approximation of $|G\rangle$ instead of precisely at $|G\rangle$. Note:

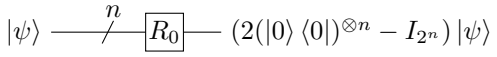
$$k = \left[\frac{\pi}{4\theta} - \frac{1}{2}\right] = \left\lfloor \frac{\pi}{4\theta} \right\rfloor \tag{3.5.12}$$

This is the idea behind the amplitude amplification circuit. How one reflects through $|B\rangle$ is clear, as the circuit R, which we assumed to have access to, does exactly this. How one reflects through $|U\rangle$, though, is not clear at first glance. To that end, we introduce the circuit R_0, which reflects the entire state space of an n-qubit system through the subspace spanned by the state $|0\rangle^{\otimes n}$. It is displayed in Circuit 3.5.7.

Circuit 3.5.7: Reflection through the all-zero state
Description: This circuit reflects any state through the subspace spanned by the all-zero state.
Number of qubits: This circuit acts on n qubits, but it uses n ancilla qubits as well.
Elementary gate complexity:[a] If $n \geq 3$, $2n + 6 + 16 \cdot (2n - 2) + 1 = 34n - 25 = \mathcal{O}(n)$.
Circuit: Here, $|\psi\rangle$ is an n-qubit state.



Shorthand notation:



[a]Note that the elementary gate complexity of a Toffoli gate is 16.

Proof of the reflective action of Circuit 3.5.7. Suppose that $|\psi\rangle$ is a computational basis state. One can easily observe that the middle CNOT-gate performs an X-operation on the bottom qubit if and only if the initial state $|\psi\rangle$ is $|0\rangle^{\otimes n}$. The state of the bottom qubit, prior to this middle Toffoli gate, is:

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

Hence, we now observe that an X-operation on this state introduces an overall sign flip, as:

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

Thus, if $|\psi\rangle$ is a computational basis state, we see that only a sign flip is introduced if $|\psi\rangle$ is $|0\rangle^{\otimes n}$. By linearity, we now deduce that this circuit implements the desired action, completing the proof. □

Now that we have defined the circuit R_0, we can use it to construct the Grover iterate. It turns out, namely, that QR_0Q^* is a reflection through the state $|U\rangle$. Thus, we easily obtain that the circuit QR_0Q^*R is an implementation of the Grover iterate. Circuit 3.5.8 elaborates on the details of this circuit.

Circuit 3.5.8: Grover iterate
Description: This circuit rotates any state in Figure 3.11 counterclockwise over an angle of 2θ.
Requirements:

1. Two orthogonal n-qubit states $|G\rangle$ and $|B\rangle$.
2. A setup circuit Q, which implements the following mapping, for some $\theta \in (0, \frac{\pi}{2})$:

$$|0\rangle^{\otimes n} \overset{Q}{\mapsto} \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$$

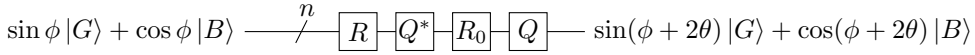
3. A reflection circuit R, which implements a reflection through $|B\rangle$, i.e., implements the following mapping where $\alpha, \beta \in [-1, 1]$ satisfy $|\alpha|^2 + |\beta|^2 = 1$:

$$\alpha\,|G\rangle + \beta\,|B\rangle \overset{R}{\mapsto} -\alpha\,|G\rangle + \beta\,|B\rangle$$

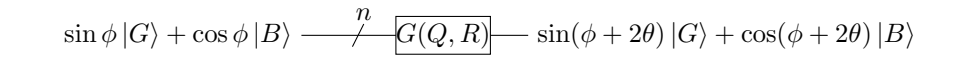
Number of qubits: This circuit acts on n qubits, but at least n ancilla qubits are required, apart from those required by Q and R.
Oracle query complexity: One call to Q, one to its inverse, and one call to R.
Elementary gate complexity:

$$2\text{egc}(Q) + \text{egc}(R) + 34n - 25 \qquad (n \geq 3)$$

Circuit:



Shorthand notation:



Proof of the action of Circuit 3.5.8. Let's trace the state throughout the application of the circuit. Suppose that we start with the initial state $\sin\phi\,|G\rangle + \cos\phi\,|B\rangle$. Then, after applying R, we obtain the state $-\sin\phi\,|G\rangle + \cos\phi\,|B\rangle$. Now, observe that we can rewrite this state, using some trigonometry:

$$\begin{aligned}
-\sin\phi\,|G\rangle + \cos\phi\,|B\rangle &= \sin(-\phi)\,|G\rangle + \cos(-\phi)\,|B\rangle = \sin(-\phi-\theta+\theta)\,|G\rangle + \cos(-\phi-\theta+\theta)\,|B\rangle \\
&= [\sin(-\phi-\theta)\cos\theta + \cos(-\phi-\theta)\sin\theta]\,|G\rangle = [\cos(-\phi-\theta)\cos\theta - \sin(-\phi-\theta)\sin\theta]\,|B\rangle \\
&= \sin(-\phi-\theta)[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle] + \cos(-\phi-\theta)[\sin\theta\,|G\rangle + \cos\theta\,|B\rangle]
\end{aligned}$$

Now, we can easily observe what happens when we apply Q^* to this state. We obtain:

$$\sin(-\phi-\theta)Q^*[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle] + \cos(-\phi-\theta)\,|0\rangle^{\otimes n}$$

We don't know much about the state $Q^*[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle]$, but we do know that $\cos\theta\,|G\rangle - \sin\theta\,|B\rangle$ is orthogonal to $\sin\theta\,|G\rangle + \cos\theta\,|B\rangle$. From this and the unitarity of Q^*, we can deduce that $Q^*[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle]$ must be orthogonal to $Q^*[\sin\theta\,|G\rangle + \cos\theta\,|B\rangle] = |0\rangle^{\otimes n}$. Hence, it picks up a minus sign when R_0 is applied to it. And so, after the application of R_0, we have that the system is in the following state:

$$-\sin(-\phi-\theta)Q^*[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle] + \cos(-\phi-\theta)\,|0\rangle^{\otimes n} = \sin(\phi+\theta)Q^*[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle] + \cos(\phi+\theta)\,|0\rangle^{\otimes n}$$

Now, after applying the final subcircuit, Q, we obtain the final state:

$$\sin(\phi+\theta)[\cos\theta\,|G\rangle - \sin\theta\,|B\rangle] + \cos(\phi+\theta)[\sin\theta\,|G\rangle + \cos\theta\,|B\rangle]$$

This, we can again rewrite, using some trigonometry, into:

$$[\sin(\phi+\theta)\cos\theta + \cos(\phi+\theta)\sin\theta]\,|G\rangle + [-\sin(\phi+\theta)\sin\theta + \cos(\phi+\theta)\cos\theta]\,|B\rangle = \sin(\phi+2\theta)\,|G\rangle + \cos(\phi+2\theta)\,|B\rangle$$

which is indeed the resulting state we claimed in Circuit 3.5.8. □

In the proof above, we have observed that Grover's iterate has the following action on any state which can be written as $\sin\phi\,|G\rangle + \cos\phi\,|B\rangle$, for $\phi \in [0, 2\pi)$:

$$G(Q,R)[\sin\phi\,|G\rangle + \cos\phi\,|B\rangle] = \sin(\phi+2\theta)\,|G\rangle + \cos(\phi+2\theta)\,|B\rangle$$

We can also rewrite this in matrix-vector notation, using the identification that we made earlier in this subsection:

$$G(Q,R)\begin{bmatrix}\cos\phi\\ \sin\phi\end{bmatrix} = \begin{bmatrix}\cos(\phi+2\theta)\\ \sin(\phi+2\theta)\end{bmatrix} = \begin{bmatrix}-\sin\phi\sin(2\theta)+\cos\phi\cos(2\theta)\\ \sin\phi\cos(2\theta)+\cos\phi\sin(2\theta)\end{bmatrix} = \begin{bmatrix}\cos(2\theta) & -\sin(2\theta)\\ \sin(2\theta) & \cos(2\theta)\end{bmatrix}\begin{bmatrix}\cos\phi\\ \sin\phi\end{bmatrix}$$

In the middle equality, we used some trigonometric identities. Hence, we recover the matrix representation of $G(Q,R)$:

$$G(Q,R) = \begin{bmatrix}\cos(2\theta) & -\sin(2\theta)\\ \sin(2\theta) & \cos(2\theta)\end{bmatrix} \tag{3.5.13}$$

Note that indeed, this is a matrix that rotates 2-dimensional vectors over an angle 2θ about the origin in the counterclockwise direction. Thus, the algebraic derivation above indeed agrees with the geometrical interpretation shown in Figure 3.12.

Now that we have an implementation for Grover's iterate, we only have to apply it several times in a row to construct an approximation of $|G\rangle$ from $|U\rangle$, as can be seen in Figure 3.12. So, in order to construct an approximation of $|G\rangle$ from $|0\rangle^{\otimes n}$, we must first of all apply Q once to obtain the state $|U\rangle$, and subsequently apply the Grover iterate a total number of k times, where k was defined in Equation 3.5.12. This circuit is known as the amplitude amplification circuit, and the details are shown in Circuit 3.5.9.

Circuit 3.5.9: Amplitude Amplification
Description: This circuit approximately constructs the state $|G\rangle$ from the state $|0\rangle^{\otimes n}$.
Requirements:

1. Two orthogonal n-qubit states $|G\rangle$ and $|B\rangle$.
2. A setup circuit Q, which implements the following mapping, for some known $\theta \in (0, \frac{\pi}{2})$:

$$|0\rangle^{\otimes n} \overset{Q}{\mapsto} \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$$

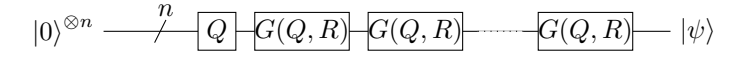
3. A reflection circuit R, which implements a reflection through $|B\rangle$, i.e., implements the following mapping where $\alpha, \beta \in [-1, 1]$ satisfy $|\alpha|^2 + |\beta|^2 = 1$:

$$\alpha\,|G\rangle + \beta\,|B\rangle \overset{R}{\mapsto} -\alpha\,|G\rangle + \beta\,|B\rangle$$

Derived constants: $k = \lfloor \frac{\pi}{4\theta} \rfloor$.
Number of qubits: The circuit acts on n qubits, and excluding the oracle circuits, at least n auxiliary qubits are required, possibly more due to the circuits Q and R.
Oracle query complexity: $2k+1$ queries are performed to Q or its inverse, and k queries are performed to R.
Elementary gate complexity:

$$(2k+1)\,\mathrm{egc}(Q) + k\,\mathrm{egc}(R) + k(34n - 25) \qquad (n \geq 3)$$

Circuit: The Grover iterate is applied k times in the circuit below. The output state satisfies $|\,\langle\psi|G\rangle\,| \geq \frac{1}{2}\sqrt{2}$.



Proof of the property of the output state $|\psi\rangle$ in Circuit 3.5.9. We trace the state of the system throughout application of the circuit. After the application of Q, the system is in state $|U\rangle = \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$. Next,

according to the properties provided by Circuit 3.5.8, the final state of the system, $|\psi\rangle$, after k applications of the Grover iterate, is:

$$|\psi\rangle = \sin((2k+1)\theta)\,|G\rangle + \cos((2k+1)\theta)\,|B\rangle$$

So, we find:

$$\langle\psi|G\rangle = \sin((2k+1)\theta)$$

Now, observe that we defined $k = \lfloor \pi/4\theta \rfloor$. Hence, we obtain:

$$\frac{\pi}{4\theta} - 1 \leq k \leq \frac{\pi}{4\theta}$$

Hence, multiplying by 2 and adding 1, yields:

$$\frac{\pi}{2\theta} - 1 \leq 2k+1 \leq \frac{\pi}{2\theta} + 1$$

And finally, multiplying by θ yields:

$$0 < \frac{\pi}{2} - \theta \leq (2k+1)\theta \leq \frac{\pi}{2} + \theta < \pi$$

Hence, we obtain $\langle\psi|G\rangle = \sin((2k+1)\theta) > 0$. Moreover, if $\theta \in (0, \pi/4]$, we have:

$$\langle\psi|G\rangle = \sin((2k+1)\theta) \geq \sin\left(\frac{\pi}{2} - \theta\right) \geq \sin\left(\frac{\pi}{4}\right) = \frac{1}{2}\sqrt{2}$$

On the other hand, if $\theta \in (\pi/4, \pi/2)$, then we find that $k = \lfloor \pi/4\theta \rfloor = 0$ and hence:

$$\langle\psi|G\rangle = \sin((2k+1)\theta) = \sin(\theta) > \sin\left(\frac{\pi}{4}\right) = \frac{1}{2}\sqrt{2}$$

So, in both cases, $|\,\langle\psi|G\rangle\,| = \langle\psi|G\rangle \geq \frac{1}{2}\sqrt{2}$, proving the claim stated in the box. □

So, now we have fully developed the amplitude amplification circuit. The curious reader might wonder what this circuit is useful for. A prototypical application for this circuit is Grover's algorithm, which aims at searching among an unstructured set of things for an element of that set that satisfies a certain property. For instance, suppose that we have a finite set S, whose elements we identify with distinct n-qubit computational basis states, that is, for all $s \in S$, there is a computational basis state $|s\rangle$ and for all $s, t \in S$, $\langle s|t\rangle = \delta_{st}$. Next, suppose that every element $s \in S$ either satisfies some property p, in which case $p(s) = 1$, or does not satisfy some property p, in which case $p(s) = 0$, and that we know the number of elements $s \in S$ that have property p. Suppose that we are looking for any element $s \in S$ that satisfies property p, and suppose, in addition, that we have a quantum circuit P, which implements the following action, for all $s \in S$ and $b \in \{0, 1\}$:

$$|s\rangle \otimes |b\rangle \mapsto |s\rangle \otimes |b \oplus p(s)\rangle$$

Now, we can define $S_G \subseteq S$ to be the set containing all elements that have property p, i.e., $s \in S_G \Leftrightarrow p(s) = 1$, and similarly we define $S_B = S \setminus S_G$. Observe that we know the value of $|S_G|$. We can now define the good and bad state as follows:

$$|G\rangle = \frac{1}{\sqrt{|S_G|}} \sum_{s \in S_G} |s\rangle \qquad \text{and} \qquad |B\rangle = \frac{1}{\sqrt{|S_B|}} \sum_{s \in S_B} |s\rangle$$

If we can now construct the good state, $|G\rangle$, we can measure it in the computational basis and obtain an element s which has property p, i.e., for which $p(s) = 1$. So, we aim to use the amplitude amplification circuit to construct $|G\rangle$. To that end, we need a setup circuit Q, and a reflection circuit R. We can construct the setup circuit easily, because we know what elements are in the set S:

$$Q : |0\rangle^{\otimes n} \mapsto |U\rangle = \frac{1}{\sqrt{|S|}} \sum_{s \in S} |s\rangle = \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$$

where $\theta = \arcsin(\sqrt{|S_G|/|S|})$. We can also easily construct the reflection circuit, acting on n qubits and using one auxiliary qubit, as follows, where $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$:

$$R = (I_{2^n} \otimes (XH))P(I_{2^n} \otimes (HX)) : \alpha |G\rangle + \beta |B\rangle \mapsto -\alpha |G\rangle + \beta |B\rangle$$

Now, we have satisfied all conditions that are necessary for implementing the amplitude amplification circuit, and hence we can (approximately) construct $|G\rangle$. The number of times we execute P, now, is given by Circuit 3.5.9 as the oracle complexity of R, as we execute P once every time we execute R:

$$k = \left\lfloor \frac{\pi}{4\theta} \right\rfloor \approx \frac{\pi}{4} \cdot \frac{1}{\arcsin(\sqrt{|S_G|/|S|})} \approx \frac{\pi}{4} \cdot \sqrt{\frac{|S|}{|S_G|}} \qquad (\text{as } |S_G| \ll |S|)$$

Finally, once we have constructed the n-qubit state $|G\rangle$, we can simply measure these n qubits in the computational basis, and then we will obtain an element $s \in S_G$, i.e., an element that has property p. In doing so, we only queried P a number of times proportional to the square root of the inverse fraction of elements that have property p, namely approximately $\sqrt{|S|/|S_G|}$ times. In contrast, in a classical implementation, we would on average need to check $|S|/|S_G|$ elements in S before finding one that has property p. Hence, Grover's algorithm quadratically reduces the number of queries to the routine that checks whether an element in S has a certain property, and so if this routine computationally dominates all the other operations, this implies a quadratic speed-up of the algorithm.

There is of course an obvious catch to this method, though. We require that we know $|S_G|$, i.e., the number of elements that satisfy the property p, beforehand, and it is not clear how we would proceed if we do not know $|S_G|$ before the algorithm starts. This is where the algorithm in the next subsection comes in, known as amplitude estimation.

3.5.7 Amplitude estimation

In the previous section, we have seen how the amplitude amplification circuit is implemented. Moreover, we mentioned an application of this amplitude amplification circuit, known as Grover's algorithm, which allows us to find an element in an unstructured set satisfying some property, if we know how many elements satisfy this property beforehand.

The quantum algorithm that we introduce in this section, though, deals with this *if* statement, i.e., it provides a way to figure out the number of elements that satisfy the given property. The algorithm is known as the *amplitude estimation algorithm*, and it combines the Grover iterate with the phase estimation algorithm. Its goal is to estimate θ, as defined in Equation 3.5.11, and with it the amplitude of the good state $|G\rangle$ in the state $|U\rangle$, in the framework of the amplitude amplification circuit. The algorithm first appeared in [BHMT00].

Recall from Equation 3.5.13 that we can write Grover's iterate as a matrix, acting on the subspace of the state space of the n-qubit system, spanned by $|G\rangle$ and $|B\rangle$. Using this matrix representation, it is not difficult to find two eigenvalue-eigenvector pairs of the Grover iterate $G(Q, R)$. One can easily check using ordinary matrix-vector multiplication that these are:

$$|S_+\rangle = \frac{1}{\sqrt{2}}(|G\rangle + i |B\rangle) \qquad \text{with} \qquad G(Q,R) |S_+\rangle = e^{2i\theta} |S_+\rangle$$

$$|S_-\rangle = \frac{1}{\sqrt{2}}(|G\rangle - i |B\rangle) \qquad \text{with} \qquad G(Q,R) |S_-\rangle = e^{-2i\theta} |S_-\rangle$$

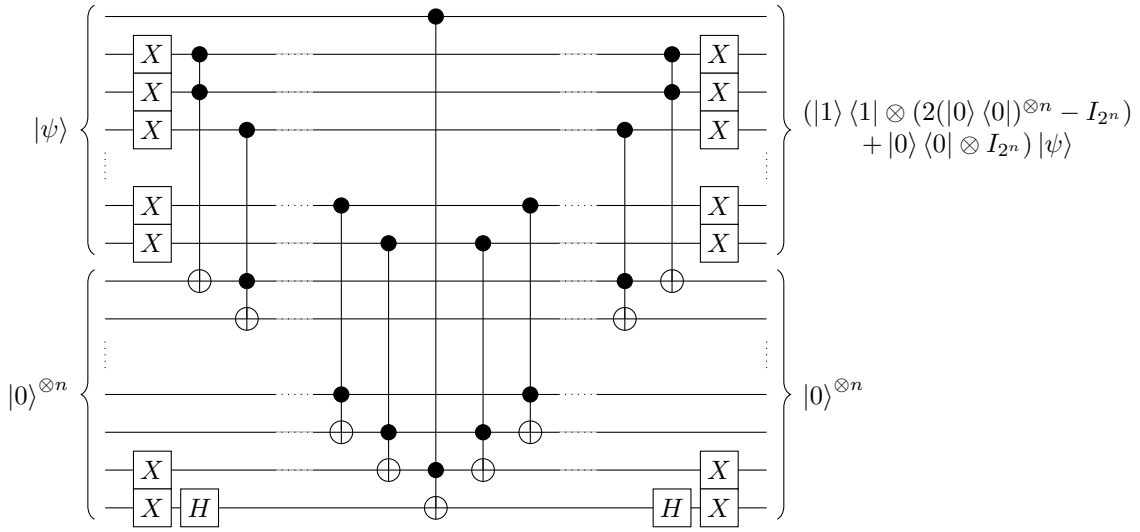
Moreover, observe that we have:

$$\begin{aligned}\frac{i}{\sqrt{2}}\left(-e^{i\theta}\left|S_+\right\rangle + e^{-i\theta}\left|S_-\right\rangle\right) &= \frac{i}{2}\left(-e^{i\theta}(\left|G\right\rangle + i\left|B\right\rangle) + e^{-i\theta}(\left|G\right\rangle - i\left|B\right\rangle)\right)\\ &= \frac{i}{2}\left(\left[-e^{i\theta}+e^{-i\theta}\right]\left|G\right\rangle + i\left[-e^{i\theta}-e^{i\theta}\right]\left|B\right\rangle\right)\\ &= \frac{e^{i\theta}-e^{-i\theta}}{2i}\left|G\right\rangle + \frac{e^{i\theta}+e^{-i\theta}}{2}\left|B\right\rangle = \sin\theta\left|G\right\rangle + \cos\theta\left|B\right\rangle = \left|U\right\rangle\end{aligned}$$

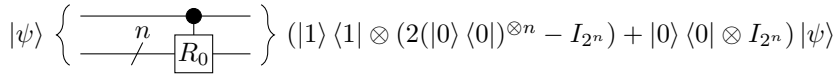
So, we have that $|U\rangle$ is a superposition of $|S_+\rangle$ and $|S_-\rangle$ and the moduli of the amplitudes of both $|S_+\rangle$ and $|S_-\rangle$ are equal. Moreover, both $|S_+\rangle$ and $|S_-\rangle$ are eigenstates of the Grover iterate, $G(Q,R)$, with eigenvalues $e^{2i\theta}$ and $e^{-2i\theta}$, respectively. Hence, the idea is to run the phase estimation algorithm, Algorithm 3.5.6, substituting Grover's iterate for the unknown quantum circuit U, and using $|U\rangle$ as a superposition of two eigenstates of the oracle quantum circuit $U = G(Q,R)$. The result will be an approximation of either θ/π or $-\theta/\pi$, with success probability at least $1-\delta$. But we ensured in Circuit 3.5.8 that $\theta \in (0, \pi/2 - \varepsilon)$, and hence we can simply discard the sign. After that, we can simply multiply the result by π to obtain an approximation of θ.

In order to be able to apply the phase estimation circuit, though, we must be able to implement controlled versions of powers of Grover's iterate. To that end, we first of all introduce the controlled version of the R_0-circuit, Circuit 3.5.7, in Circuit 3.5.10.

Circuit 3.5.10: Controlled reflection through the all-zero state.
Description: This circuit reflects any state through the subspace spanned by the all-zero state, controlled on a single qubit. This is the controlled version of Circuit 3.5.7.
Number of qubits: This circuit acts on $n+1$ qubits, but uses n ancilla qubits.
Elementary gate complexity: If $n \geq 3$, $2n + 6 + 16 \cdot (2n-1) = 34n - 10 = \mathcal{O}(n)$.
Circuit:



Shorthand notation:



Proof of the reflective action of Circuit 3.5.10. If the control qubit is in state $|0\rangle$, then the middle Toffoli circuit will act as the identity operation. But as the circuit on the right side of the middle Toffoli gate is

the reverse of the circuit that is on shown on the left side of the middle Toffoli gate, the entire circuit acts as the identity gate if the control qubit is in state $|0\rangle$, as expected.

On the other hand, if the control qubit is in state $|1\rangle$, then the action of this circuit on all the qubits except the first one is the same as the action of the circuit shown in Circuit 3.5.7. So, we find that the circuit shown here indeed implements a controlled version of Circuit 3.5.7. This completes the proof. □

Now, recall that Grover's iterate can be concisely written as $G(Q, R) = RQ^*R_0Q$. Hence, in order to construct a controlled version of Grover's iterate, we only have to implement the R- and R_0-circuits in a controlled manner, because the Q^*- and Q-circuits undo each other's action. Circuit 3.5.11 exploits this observation, and also shows some additional properties of this controlled implementation of Grover's iterate.

Circuit 3.5.11: Controlled Grover iterate
Description: This circuit is a controlled version of Circuit 3.5.8. Controlled on a single qubit, it rotates any state on the circle shown in Figure 3.11 counterclockwise over an angle of 2θ.
Requirements:

1. Two orthogonal n-qubit states $|G\rangle$ and $|B\rangle$.
2. A setup circuit Q, which implements the following mapping, for some $\theta \in (0, \frac{\pi}{2})$:

$$|0\rangle^{\otimes n} \overset{Q}{\mapsto} \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$$

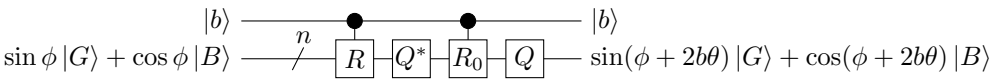
3. A controlled reflection circuit $C(R)$, which implements a reflection through $|B\rangle$ controlled on a single qubit, i.e., implements the following mapping where $\alpha, \beta \in [-1, 1]$ satisfy $|\alpha|^2 + |\beta|^2 = 1$ and $b \in \{0, 1\}$:

$$|b\rangle \otimes (\alpha\,|G\rangle + \beta\,|B\rangle) \overset{C(R)}{\mapsto} |b\rangle \otimes ((-1)^b\alpha\,|G\rangle + \beta\,|B\rangle)$$

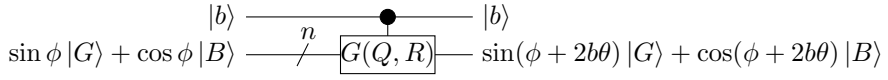
Number of qubits: This circuit acts on $n + 1$ qubits, but it requires n ancilla qubits, not taking into account those required by Q and $C(R)$.
Oracle query complexity: One call to Q, one to its inverse and one to $C(R)$.
Elementary gate complexity:

$$2\,\mathrm{egc}(Q) + \mathrm{egc}(C(R)) + 34n - 10 \qquad (n \geq 3)$$

Circuit: Here, $b \in \{0, 1\}$ and $\phi \in [0, 2\pi)$.



Shorthand notation:



Finally, now that we have introduced the controlled version of the Grover iterate, we can describe how the amplitude estimation algorithm works. This is done in Algorithm 3.5.12.

Algorithm 3.5.12: Amplitude estimation
Description: This algorithm approximates θ in the context of Grover's iterate.
Input:

1. A real number $\varepsilon > 0$, indicating the accuracy of the approximation θ.
2. A real number $\delta > 0$, indicating the maximal tolerable probability of failure.
3. Two orthogonal n-qubit states $|G\rangle$ and $|B\rangle$.
4. An n-qubit circuit Q that has the following action, for some $\theta \in [0, \pi/2 - \varepsilon)$:

$$|0\rangle^{\otimes n} \overset{Q}{\mapsto} \sin\theta\,|G\rangle + \cos\theta\,|B\rangle$$

5. An controlled n-qubit reflection circuit $C(R)$ that performs the following action, for all $\alpha, \beta \in \mathbb{C}$ satisfying $|\alpha|^2 + |\beta|^2 = 1$ and $b \in \{0, 1\}$:

$$|b\rangle \otimes (\alpha\,|G\rangle + \beta\,|B\rangle) \overset{C(R)}{\mapsto} |b\rangle \otimes ((-1)^b \alpha\,|G\rangle + \beta\,|B\rangle)$$

Derived constants:

1. $m = \lceil \log(\pi(1/\delta + 4)/\varepsilon) \rceil$, indicating the number of qubits required to obtain a sufficiently close approximation to θ.

Output: A number $\tilde{\theta} \in [0, \pi/2]$ such that $|\theta - \tilde{\theta}| \leq \varepsilon$.
Number of qubits: $m + n$ qubits are acted upon, and at least n auxiliary qubits are required, not taking into account those that are required by Q and $C(R)$.
Success probability: Lower bounded by $1 - \delta$.
Query complexity: 2^m queries to Q are performed, and $2^m - 1$ queries to its inverse. Furthermore, $2^m - 1$ queries are performed to $C(R)$.
Elementary gate complexity:

$$(2^{2m} - 1)\,\mathrm{egc}(Q) + (2^m - 1)\,\mathrm{egc}(C(R)) + (2^m - 1)(34n - 10) + (m + 3)m/2 + 3\lfloor m/2 \rfloor \qquad (n \geq 3)$$

Algorithm:

1. Prepare the $m + 2n$ qubit system in the state $|0\rangle^{\otimes(m+2n)}$.
2. Apply the Q-circuit to qubits $m + 1$ through $m + n$.
3. Run the phase estimation algorithm on qubits 1 through $m + n$. The accuracy parameter is ε/π. The oracle circuit is $G(Q, R)$. The controlled applications of the oracle circuit are implemented by $C(G(Q, R)^{2^j}) = C(G(Q, r))^{2^j}$, for all $j \in \{0, 1, \ldots, m - 1\}$. Denote the output by ξ, and if $\xi > \frac{1}{2}$, subtract one from ξ.
4. Return $\tilde{\theta} = |\xi| \cdot \pi$.

As a final remark, suppose that we have access to a controlled version of P, where P is the circuit that checks whether a given element of $s \in S$ has some property p. Then, we can build the circuits Q and $C(R)$, just like we did in the previous subsection, and we can supply these circuits to the amplitude estimation algorithm. This allows us to obtain an approximation of θ, which we can subsequently use in the amplitude amplification circuit to approximately construct $|G\rangle$. This solves the problem that we pinpointed at the end of the previous subsection.

This concludes our discussion of the amplitude estimation algorithm, and with it, we also conclude the discussion of the most important techniques in quantum computing. In subsequent chapters, we will apply these techniques to the gradient estimation problem, and afterwards we will use the gradient estimation algorithm to solve the quantum reinforcement learning problem.

4 Quantum gradient estimation

In the previous chapters, we have introduced quantum computing in a very general setting. Chapter 2 elaborated on the fundamentals of quantum mechanics, and Chapter 3 introduced the field of quantum computing. In this chapter, we will specialize towards one specific problem that can be solved with quantum computing, which is estimating gradients of high-dimensional functions using function evaluations. In particular, we will develop the necessary tools in order to implement such a quantum gradient estimation algorithm. In Chapter 5, we will prove that the algorithm presented in this chapter is essentially optimal in the number of function evaluations. Afterwards, in the last chapter, Chapter 6, we will see how this can be applied to construct quantum gradient descent algorithms, and then we will apply this technique to develop the basics of quantum reinforcement learning.

The first quantum gradient estimation algorithm was presented by Jordan in [Jor05]. This algorithm relied solely on the quantum Fourier transform. Subsequently it was improved by Gilyén et al. in [GAW17], who added a numerical method to improve the accuracy. Here, we present a slightly more generalized version of Gilyén et al.'s algorithm, which also allows for estimating gradients with respect to general ℓ^p norms.

This chapter will be subdivided into several parts. First of all, we will fix some notation, in Section 4.1. Afterwards, in Section 4.2, we develop the theory necessary to implement fractional phase queries, based on [GSLW18]. These fractional phase queries are used in the quantum gradient estimation algorithm that was developed by Gilyén et al. in [GAW17]. The complete algorithm we present in Section 4.3.

4.1 Nomenclature

In this section, we first of all fix some terminology, mainly to avoid confusion in the remainder in this chapter. As we are discussing gradients of multidimensional functions, we start by fixing the notation for differentiating real-valued functions that are defined on a multidimensional domain, in Subsection 4.1.1. Then, in Subsection 4.1.2, we define some function classes, which we can use to denote spaces of input functions to quantum gradient estimation algorithms. Subsequently, in Subsection 4.1.3, we discuss how we can generalize the classical notion of "performing a function evaluation" to the quantum computing setting. Finally, in Subsection 4.1.4, we formally define what we mean by a quantum gradient estimation algorithm.

4.1.1 Derivatives

Throughout this chapter, we will frequently be dealing with derivatives of multidimensional functions. There exist many different ways to express these derivatives concisely, so to avoid confusion, we will precisely define which notations we will be using here. These notations might take some getting used to at first, but they greatly reduce the cumbersomeness of the expressions in the remainder of this chapter.

First of all, for all $n \in \mathbb{N}$, we let $[n]$ denote $\{1, 2, \ldots, n\}$. Hence, for all $k, n \in \mathbb{N}$, the set $[n]^k$ is the set that contains all finite sequences of length k whose entries are elements from $[n] = \{1, 2, \ldots, n\}$.

Next, suppose that $d \in \mathbb{N}$, and that f is a function mapping the open set $\Omega \subseteq \mathbb{R}^d$ into $\mathbb{R}$, i.e., $f : \Omega \to \mathbb{R}$. We let $j \in [d]$. Then, we define $\partial_j f$ to be the function f, differentiated with respect to the jth variable. Note that again $\partial_j f : \Omega \to \mathbb{R}$. Furthermore, we define $\partial_j^k f$ to be the function f differentiated to the jth variable k times.

Now, let $k \in \mathbb{N}$ and $\alpha \in [d]^k$. For all $j \in \{1, 2, \ldots, k\}$, we have that $\alpha_j \in \{1, 2, \ldots, d\}$. We let $\partial_\alpha f = \partial_{\alpha_1} \partial_{\alpha_2} \cdots \partial_{\alpha_k} f$. Hence, $\partial_\alpha f$ is a kth order derivative of f, and $\delta_\alpha f : \Omega \to \mathbb{R}$. Hence, every entry of α corresponds to a differentiation operation of f to the coordinate specified by this entry. Furthermore, for some vector $\mathbf{x} \in \mathbb{R}^d$, we define $\mathbf{x}^\alpha$ to be $x_{\alpha_1} x_{\alpha_2} \cdots x_{\alpha_d} \in \mathbb{R}$.

We also introduce another notation for the derivative of f. Let $\beta \in \mathbb{N}_0^d$. Then, we say that $D^\beta f = \partial_1^{\beta_1} \partial_2^{\beta_2} \cdots \partial_d^{\beta_d} f$.[1] Both notations we will use throughout the remainder of this chapter, as one is sometimes more convenient than the other and vice versa. Furthermore, we define $|\beta| = \beta_1 + \beta_2 + \cdots + \beta_d$. Finally, for all $\mathbf{x} \in \mathbb{R}^d$, we also define $\mathbf{x}^{(\beta)} = x_1^{\beta_1} x_2^{\beta_2} \cdots x_d^{\beta_d} \in \mathbb{R}$, where the parentheses distinguish the notation from the one introduced in the previous paragraph.

Let's illustrate these notations by means of an example. Suppose that $f : \mathbb{R}^3 \to \mathbb{R}$, with $f(x, y, z) = x^2yz$. Then:

$$\begin{array}{ccc} \partial_1 f(x,y,z) = 2xyz & \partial_2 f(x,y,z) = x^2 z & \partial_3 f(x,y,z) = x^2 y \\ \partial_{(1,1)} f(x,y,z) = 2yz & \partial_{(1,2)} f(x,y,z) = 2xz & \partial_{(2,1,1)} f(x,y,z) = 2z \\ D^{(1,0,0)} f(x,y,z) = 2xyz & D^{(2,1,0)} f(x,y,z) = 2z & D^{(2,1,1)} f(x,y,z) = 2 \end{array}$$

In the next subsection, we will use the notation introduced in this subsection to introduce classes of functions whose elements' derivatives satisfy certain smoothness conditions.

4.1.2 Gevrey function classes

Consider the set of all differentiable functions $f : \mathbb{R}^d \to \mathbb{R}$, for some $d \in \mathbb{N}$. This set contains a wide range of functions. Some are very well-behaved, for example the linear functions $f_{\mathbf{v}}$ for some $\mathbf{v} \in \mathbb{R}^d$, defined by $\mathbf{x} \mapsto \mathbf{v} \cdot \mathbf{x}$. However, others behave in a very wild manner, for instance the function $\mathbf{x} \mapsto (x_j^2 \cdot \sin(1/x_j))_{j=1}^d$. For this huge set of functions, one cannot hope to develop an algorithm that accurately estimates the gradient, simply because the behavior of some of these functions is too wild to cope with. Hence, when one develops a gradient estimation algorithm, it is indispensable to impose some restrictions on the set of functions for which the algorithm is supposed to yield accurate results.

In this subsection, we introduce one such set of restrictions on the class of functions for which we want the gradient estimation algorithm to yield accurate results. This we do in the following definition. The resulting class of functions is in the literature referred to as a *Gevrey function class*.

Definition 4.1.1: Gevrey function classes
Let $d \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$ open, $M > 0$, $c > 0$ and $\sigma \in \mathbb{R}$. Now, we define the *Gevrey function class* $\mathcal{G}^\sigma_{d,\Omega,M,c}$ to be the set containing all functions $f : \Omega \to \mathbb{R}$ satisfying the following conditions[a]:

1. f must be smooth (i.e., all derivatives must exist, $f \in C^\infty(\Omega)$).
2. The power series of f around any point $\mathbf{x} \in \Omega$ must converge on all of Ω.
3. For all multi-indices $\alpha \in \mathbb{N}_0^d$, we have:

$$\|D^\alpha f\|_\infty \leq M c^{|\alpha|} (|\alpha|!)^\sigma$$

[a] Condition 2 actually implies condition 1, but we included condition 1 for extra clarity. Furthermore, if $\sigma < 1$, then condition 3 implies condition 2.

It is not difficult to see that when one increases M, c or σ, the restrictions defined in this definition become weaker. Moreover, if one shrinks the set Ω, the condition becomes weaker as well. These statements are made precise in the following theorem.

[1] Note that if f is sufficiently smooth, the order of differentiation does not matter. These smoothness conditions are automatically satisfied by the assumptions made in Subsection 4.1.2, and hence we will not worry about the order of differentiation in the remainder of this text.

Theorem 4.1.2: Properties of Gevrey function classes
Let $d \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$ open, $M > 0$, $c > 0$ and $\sigma \in \mathbb{R}$. The Gevrey function classes satisfy the following properties:

1. For all open sets $\Omega_1 \subseteq \Omega_2 \subseteq \mathbb{R}^d$, we have $f \in \mathcal{G}^\sigma_{d,\Omega_2,M,c} \Rightarrow f|_{\Omega_1} \in \mathcal{G}^\sigma_{d,\Omega_1,M,c}$.
2. For all $M_1 \geq M_2 > 0$, we have $\mathcal{G}^\sigma_{d,\Omega,M_2,c} \subseteq \mathcal{G}^\sigma_{d,\Omega,M_1,c}$.
3. For all $c_1 \geq c_2 > 0$, we have $\mathcal{G}^\sigma_{d,\Omega,M,c_2} \subseteq \mathcal{G}^\sigma_{d,\Omega,M,c_1}$.
4. For all $\sigma_1 \geq \sigma_2$, we have $\mathcal{G}^{\sigma_2}_{d,\Omega,M,c} \subseteq \mathcal{G}^{\sigma_1}_{d,\Omega,M,c}$.

Proof. The statements in this theorem follow immediately from the restrictions in Definition 4.1.1. □

According to the previous theorem, the class of functions $\mathcal{G}^\sigma_{d,\Omega,M,c}$ becomes bigger when we increase M, c or σ. Hence, for some function $f : \Omega \to \mathbb{R}$, it is sensible to wonder what is the minimal σ such that $f \in \mathcal{G}^\sigma_{d,\Omega,M,c}$ for some values of M and c. This value of σ, we refer to as the Gevrey-type of f. We formalize this idea in the definition below.

Definition 4.1.3: Gevrey-type of functions
Let $d \in \mathbb{N}$ and $\Omega \subseteq \mathbb{R}^d$ open. Let $f \in \mathcal{G}^\sigma_{d,\Omega,M,c}$ for some $M > 0$, $c > 0$ and $\sigma \in \mathbb{R}$. Next, we define:

$$\mathrm{GT}(f) = \inf\{\sigma \in \mathbb{R} : \exists M, c > 0 : f \in \mathcal{G}^\sigma_{d,\Omega,M,c}\}$$

We refer to this quantity as the *Gevrey-type* of the function f. Additionally, we say that f is *of Gevrey-type* $\mathrm{GT}(f)$.

Finally, it is interesting to figure out how the Gevrey function classes are related under elementary modifications of their elements. The following theorem provides these relations.

Theorem 4.1.4: Modifications of Gevrey functions
Let $d \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$ open, $M > 0$, $c > 0$, $\sigma \in \mathbb{R}$ and $f \in \mathcal{G}^\sigma_{d,\Omega,M,c}$. Then, the following statements hold.

1. We define the following function, with $0 \neq M' \in \mathbb{R}$:
$$g : \Omega \to \mathbb{R}, \qquad g(\mathbf{x}) = \frac{f(\mathbf{x})}{M'}$$
Then $g \in \mathcal{G}^\sigma_{d,\Omega,M/M',c}$.
2. We define the set Ω' and function h as follows, where $0 \neq c' \in \mathbb{R}$:
$$\Omega' = \{c'\mathbf{x} : \mathbf{x} \in \Omega\}, \qquad \text{and} \qquad h : \Omega' \to \mathbb{R}, \qquad h(\mathbf{x}) = f(c'\mathbf{x})$$
Then $h \in \mathcal{G}^\sigma_{d,\Omega',M,cc'}$.

Proof. The proofs are immediate from the restrictions provided in Definition 4.1.1. □

As a final note on Gevrey function classes, we will determine the Gevrey-type of some typical functions. First of all, we will consider polynomials defined on a bounded set.

Example 4.1.5: Let $\Omega \subseteq \mathbb{R}^d$ be bounded, $p : \Omega \to \mathbb{R}$ be a multivariate polynomial, and let $\deg(p)$ be its total degree. Then, we find that all kth order derivatives of p vanish whenever $k > \deg(p)$. Hence, we can simply take, for all $\sigma \in \mathbb{R}$:

$$M_\sigma = \sup_{k \in \{0,1,\ldots,\deg(p)\}} \sup_{\alpha \in [d]^k} \sup_{\mathbf{x} \in \Omega} |\partial_\alpha p(\mathbf{x})| \cdot (\deg(p)!)^{-\sigma}$$

Then, we find that $M_\sigma < \infty$ and $p \in \mathcal{G}^\sigma_{d,\Omega,M_\sigma,c}$. Thus, we find that $\mathrm{GT}(p) = -\infty$. This does not hold for polynomials that are defined on unbounded regions Ω, however, as the partial derivatives blow up at infinity.

In the example we just considered, we remarked that having a bounded region on which the polynomial is defined is of crucial importance for its Gevrey type. It is natural to wonder whether there exist functions whose Gevrey types do not exhibit this dependence on their domain. Below, we will give an example of such a function. We will again encounter this function in Section 5.1.

Example 4.1.6: Let $c > 0$ and consider the function $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \sin(cx)$. The derivatives of this function are given by, with $n \in \mathbb{N}_0$:

$$f^{(n)}(x) = c^n \cdot \begin{cases} (-1)^{(n-1)/2} \cos(cx), & \text{if } n \text{ is odd} \\ (-1)^{n/2} \sin(cx), & \text{if } n \text{ is even} \end{cases}$$

Hence, we find, for all $n \in \mathbb{N}_0$:

$$|f^{(n)}(x)| \leq c^n = c^n (n!)^0$$

So, $f \in \mathcal{G}^0_{1,\mathbb{R},1,c}$. Moreover, for every $n \in \mathbb{N}_0$, equality is reached in the above inequality for some values of x, and hence there does not exist a $\sigma < 0$ such that f is of Gevrey-type σ. Thus, we find $\mathrm{GT}(f) = 0$.

This concludes the introduction of the Gevrey function classes. In the next subsection, we will have a look at how we can evaluate functions from these function classes in the quantum computing setting.

4.1.3 Phase oracle queries

In classical computing, *performing a function evaluation of f* comes down to executing a procedure that yields the value $f(\mathbf{x})$ for some input value $\mathbf{x}$. Generally, the input value of $\mathbf{x}$ has to satisfy some condition such that it lies in the domain of f.

In order to develop the quantum gradient estimation algorithm and analyze the number of function evaluations it performs, we have to precisely define what we mean by *performing a function evaluation of f* within the framework of quantum computing. To that end, we introduce a *phase oracle*, which calculates the function value at some predefined point in the domain and outputs the result by means of a phase shift. We say that we perform one function evaluation of f if we query the oracle quantum circuit presented in the next definition once.

Definition 4.1.7: Phase oracle
Let $f : \Omega \to \mathbb{R}$, $G \subseteq \Omega$ and $n \in \mathbb{N}$. Let $\{|\mathbf{x}\rangle : \mathbf{x} \in G\}$ be an orthogonal set of n-qubit states. Then $O_{f,G}$ is a *phase oracle of f on G* if it is a quantum circuit acting on n qubits that implements the following mapping:

$$O_{f,G} : |\mathbf{x}\rangle \mapsto e^{if(\mathbf{x})} |\mathbf{x}\rangle$$

Moreover, we require $O_{f,G}$ to act as the identity on all the states that are in the orthogonal complement of $\{|\mathbf{x}\rangle : \mathbf{x} \in G\}$. We refer to G as the *grid* corresponding to the phase oracle $O_{f,G}$, even though the layout of G in $\mathbb{R}^d$ does not necessarily resemble a grid at all.

Note that in the above definition, we must have that $|G| \leq 2^n$, because the n-qubit state space is 2^n-dimensional.

As a final note, observe that if we want to obtain the value of $f(\mathbf{x})$ for some $\mathbf{x} \in G$, then in general we cannot find this value using one call to $O_{f,G}$. This is because the function value is only stored in the phase of the amplitude of the state $|\mathbf{x}\rangle$. In order to extract this phase information, we can execute the phase estimation algorithm of Subsection 3.5.5, but this involves multiple calls to $O_{f,G}$. In this sense, the function evaluation model in quantum computing is weaker than the one in classical computing, where we can obtain the value of $f(\mathbf{x})$ using just one function evaluation.

This concludes the introduction of the phase oracle. In the next subsection, we will use these phase oracles to define what exactly we mean by a quantum gradient estimation algorithm.

4.1.4 Quantum gradient estimation algorithms

The goal of this chapter is to develop a quantum algorithm that, given a phase oracle of some function, calculates an approximation of its gradient at some point. We should, however, specify what exactly we mean by "an approximation", and when we deem this approximation to be close enough to the true value for the algorithm to have terminated successfully. The following definition formalizes all such details.

Definition 4.1.8: Quantum gradient estimation algorithms
Let $d \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$ open, $\mathbf{a} \in \Omega$, $G \subseteq \Omega$, $\varepsilon > 0$, $p \in [1, \infty]$ and $P \in [0, 1]$. Let $\mathcal{F}$ be a class of functions, whose elements map Ω to $\mathbb{R}$ in an at least once differentiable manner. Then *an ε-precise ℓ^p-approximate quantum gradient estimation algorithm for $\mathcal{F}$ in $\mathbf{a}$ on G with success probability P* is a quantum algorithm that satisfies the following properties:

1. For every $f \in \mathcal{F}$, the input is a phase oracle of f on G, controlled on one qubit, denoted by $C(O_{f,G})$.
2. For every input $C(O_{f,G})$ where $f \in \mathcal{F}$, with probability at least P, the algorithm calculates a vector $\mathbf{v} \in \mathbb{R}^d$ such that:
$$\|\mathbf{v} - \nabla f(\mathbf{a})\|_p \leq \varepsilon$$

Loosely speaking, a quantum gradient estimation algorithm solves problems of the following form with constant probability: "Given a controlled phase oracle $C(O_{f,G})$, where $f \in \mathcal{F}$, find an approximation of $\nabla f(\mathbf{a})$ which differs from the true value by at most ε in ℓ^p-norm."

Note that the above definition closes some obvious loopholes. For instance, if we want to calculate the gradient of f, we must use the controlled phase oracle $C(O_{f,G})$, and we cannot use the circuit $O_{g,G}$ for some other function $g \in \mathcal{F}$, simply because it is not part of the input.

This completes the formal definition of a quantum gradient estimation algorithm. In the next section, we will set the first step towards developing such an algorithm.

4.2 Fractional phase queries

In this section, we will develop one of the building blocks of Gilyén et al.'s quantum gradient estimation algorithm that we will present in the next section. More precisely, we explain how we can build a quantum circuit that implements a *fractional phase query*. This is a quantum circuit that approximately implements the following operation, for some $\xi \in \mathbb{R}$:

$$O_{f,G}^{\xi} : |\mathbf{x}\rangle \mapsto e^{i\xi f(\mathbf{x})} |\mathbf{x}\rangle$$

For the most part, we follow the construction provided by [GSLW18], and provide a construction of a quantum circuit that implements this operation approximately using a number of calls to the controlled phase oracle $C(O_{f,G})$ only logarithmic in the precision.

This section is subdivided in four subsections. First of all, we introduce the concept of block-encodings, in Subsection 4.2.1. Next, we build a circuit that implements a block-encoding of the sine of a function, in Subsection 4.2.2. After that, we construct polynomial approximations to the functions $x \mapsto \sin(t \arcsin(x))$ and $x \mapsto \cos(t \arcsin(x))$ in Subsection 4.2.3. Next, we explain how we can implement block-encoded polynomials of arbitrary operators in Subsection 4.2.4. Subsequently, we describe how we can add two polynomials in Subsection 4.2.5. Finally, we merge all ideas together in a complete quantum circuit, which we provide in Subsection 4.2.6.

4.2.1 Block-encodings

First of all, we introduce the concept of block-encodings. The approach presented here is taken from [GSLW18].

Definition 4.2.1: Block-encodings
Let U be a unitary operator acting on the n-qubit state space. Suppose that A is another unitary operator, acting on $(n+a)$-qubits, where $a \geq 0$. Furthermore, let $\delta \geq 0$ and $\alpha \in \mathbb{C}$. We say that A is an (α, a, δ)-block-encoding of U if:

$$\left\| A - \alpha(\langle 0|^{\otimes a} \otimes I_{2^n}) U (|0\rangle^{\otimes a} \otimes I_{2^n}) \right\| \leq \delta$$

By $\{U\}_{(\alpha,a,\delta)}$, we denote the set of all (α, a, δ)-block-encodings of U.

We can easily see that A and U as in the definition above are very concretely related. Consider, namely, the matrix representation of A, which is a $2^{n+a} \times 2^{n+a}$ unitary matrix. As such, its layout can be viewed as $2^a \times 2^a$ *blocks*, all of which have dimension $2^n \times 2^n$. If A is an (α, a, ε)-block-encoding of U, then, according to Definition 4.2.1, the top-left block of A almost equals αU. More precisely, the matrix αU and the top-left $2^n \times 2^n$ block of A differ only by δ in operator norm. Thus, we can think of the matrix representation of A as being approximately equal to:

$$A \approx \begin{bmatrix} \alpha U & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \\ \cdot & \cdot & \cdots & \cdot \end{bmatrix}$$

where all the single dots (the $\cdot$'s) represent other, unimportant, $2^n \times 2^n$-blocks. Thus, we find that the matrix representation of U is somehow *encoded* in the upper-left *block* of the matrix representation of A, hence the name. Rather trivially, we find that every unitary operator is a $(1, 0, 0)$-block-encoding of itself.

At this point, it is not obvious why block-encodings are useful. The following definition, though, will provide us with a very natural application of block-encodings.

Definition 4.2.2: Implementation of function compositions
Let $f : \Omega \to \mathbb{R}$, $G \subseteq \Omega$ and $\{|\mathbf{x}\rangle : \mathbf{x} \in G\}$ be an orthonormal set of n-qubit states. Furthermore, let U be a (not necessarily unitary) operator acting on the n-qubit state space, and let $h : \mathrm{img}(f) \to \mathbb{C}$. We say that U *implements* $h \circ f$ *on* G if:

$$U = \sum_{\mathbf{x} \in G} h(f(\mathbf{x})) \, |\mathbf{x}\rangle\langle\mathbf{x}|$$

We write $U = h(f)_G$.

Specifically, observe that we can write the phase oracle $O_{f,G}$ as follows:

$$O_{f,G} = \sum_{\mathbf{x} \in G} e^{if(\mathbf{x})} \, |\mathbf{x}\rangle\langle\mathbf{x}|$$

and hence we can write $O_{f,G} = \left(e^{if}\right)_G$. Similarly, suppose that we had an operator U that implements the following mapping:

$$U = \sum_{\mathbf{x} \in G} f(\mathbf{x}) \, |\mathbf{x}\rangle\langle\mathbf{x}|$$

Then we would write $U = f_G$, because the function h in this case would simply be the identity function.

Note that if not all function values $f(\mathbf{x})$ are located on the unit disc in the complex plane, then the operator $U = f_G$ defined above is not unitary. Hence, we cannot hope to implement this operator U in a quantum

circuit directly. However, we can hope to implement a block-encoding of U, because the upper-left block of a unitary matrix is not necessarily unitary itself. This is what we will primarily be using block-encodings for, namely implementing non-unitary operators as part of bigger unitary operators.[2]

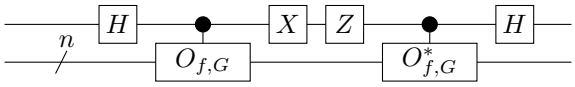
This concludes the introduction of the concept of block-encodings. In the next subsection, we will construct the first quantum circuit that is a block-encoding of a non-unitary operator.

4.2.2 Implementation of a $(1,1,0)$-block-encoding of $\sin(f)_G$

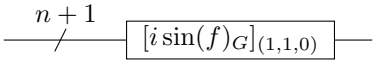
In this subsection we will construct a quantum circuit that is a block-encoding of the unitary operator that implements $\sin(f)$ on G. Moreover, we will construct this circuit by only making calls to the controlled phase oracle $C(O_{f,G})$.

We start by giving the circuit, and subsequently we will prove that the action is as claimed.

Circuit 4.2.3: Block-encoding of the operator that implements $\sin\circ f$ on G
Description: This quantum circuit is a $(1,1,0)$-block-encoding of $i\sin(f)_G$.
Requirements: A controlled version of a phase oracle $O_{f,G}$, denoted by $C(O_{f,G})$, acting on n qubits.
Number of qubits: This circuit acts on $n+1$ qubits, but possibly $O_{f,G}$ requires some extra auxilliary qubits.
Oracle query complexity: One call to $C(O_{f,G})$ and one to its inverse.
Elementary gate complexity: $2\,\mathrm{egc}(C(O_{f,G}))+4$.
Circuit:



Shorthand notation:



Proof that the circuit in Circuit 4.2.3 is a $(1,1,0)$-block-encoding of $i\sin(f)_G$. Let's consider the matrix representation of Circuit 4.2.3. By simply writing out the matrix representation of the elements in the quantum circuit, we find:

$$
\begin{aligned}
[i\sin(f)_G]_{(1,1,0)} &= (H\otimes I_{2^n})C(O^*_{f,G})(Z\otimes I_{2^n})(X\otimes I_{2^n})C(O_{f,G})(H\otimes I_{2^n}) \\
&= \frac{1}{2}\begin{bmatrix} I_{2^n} & I_{2^n} \\ I_{2^n} & -I_{2^n}\end{bmatrix}\begin{bmatrix} I_{2^n} & 0 \\ 0 & O^*_{f,G}\end{bmatrix}\begin{bmatrix} I_{2^n} & 0 \\ 0 & -I_{2^n}\end{bmatrix}\begin{bmatrix} 0 & I_{2^n} \\ I_{2^n} & 0\end{bmatrix}\begin{bmatrix} I_{2^n} & 0 \\ 0 & O_{f,G}\end{bmatrix}\begin{bmatrix} I_{2^n} & I_{2^n} \\ I_{2^n} & -I_{2^n}\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix} I_{2^n} & O^*_{f,G} \\ I_{2^n} & -O^*_{f,G}\end{bmatrix}\begin{bmatrix} 0 & I_{2^n} \\ -I_{2^n} & 0\end{bmatrix}\begin{bmatrix} I_{2^n} & I_{2^n} \\ O_{f,G} & -O_{f,G}\end{bmatrix} = \frac{1}{2}\begin{bmatrix} I_{2^n} & O^*_{f,G} \\ I_{2^n} & -O^*_{f,G}\end{bmatrix}\begin{bmatrix} O_{f,G} & -O_{f,G} \\ -I_{2^n} & -I_{2^n}\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix} O_{f,G}-O^*_{f,G} & -O_{f,G}-O^*_{f,G} \\ O_{f,G}+O^*_{f,G} & -O_{f,G}+O^*_{f,G}\end{bmatrix} = \frac{1}{2}\begin{bmatrix} \left(e^{if}\right)_G-\left(e^{-if}\right)_G & -\left(e^{if}\right)_G-\left(e^{-if}\right)_G \\ \left(e^{if}\right)_G+\left(e^{-if}\right)_G & -\left(e^{if}\right)_G+\left(e^{-if}\right)_G\end{bmatrix} \\
&= \begin{bmatrix} i\sin(f)_G & -\cos(f)_G \\ \cos(f)_G & -i\sin(f)_G\end{bmatrix}
\end{aligned}
$$

Hence, indeed, we note that $[i\sin(f)_G]_{(1,1,0)}$ is a $(1,1,0)$-block-encoding of $i\sin(f)_G$, and so we write $[i\sin(f)_G]_{(1,1,0)}\in\{i\sin(f)_G\}_{(1,1,0)}$. This completes the proof. □

[2] In the area of functional analysis, the method of embedding operators that are not necessarily unitary in unitary operators acting on larger spaces, is referred to as the *transference principle*. The space of operators acting on this larger space is sometimes referred to as *operator heaven*. [Haa18]

Note that we can quite easily modify this circuit such that it is controlled on a single extra qubit. We can, namely, simply control the application of the X and Z gates in the middle on the extra qubit. This does not have any influence on the elementary gate complexity, or the oracle query complexity.

Using Circuit 4.2.3, we can now implement the function $\mathbf{x} \mapsto i\sin(f(\mathbf{x}))$, using only calls to the controlled phase oracle $C(O_{f,G})$. In the next subsections, we will use this circuit as a subcircuit in constructing the fractional phase query.

4.2.3 Approximation of the function $\exp(it\arcsin(x))$

In the previous subsection, we have constructed a quantum circuit which implements the function $\mathbf{x} \mapsto \sin(f(\mathbf{x}))$. The next step will be to choose some $t \in [0, 1/48]$, and construct quantum circuits that implement the functions $\mathbf{x} \mapsto \cos(tf(\mathbf{x}))$ and $\mathbf{x} \mapsto \sin(tf(\mathbf{x}))$. To that end, we observe, for all $\mathbf{x} \in G$:

$$\cos(tf(\mathbf{x})) = \cos(t\arcsin(\sin(f(\mathbf{x})))) \qquad \text{and} \qquad \sin(tf(\mathbf{x})) = \sin(t\arcsin(\sin(f(\mathbf{x}))))$$

We already know how to implement $\mathbf{x} \mapsto \sin(f(\mathbf{x}))$, so if we find a way to compose this function with the functions $x \mapsto \cos(t\arcsin(x))$ and $x \mapsto \sin(t\arcsin(x))$, we are done. This, we will achieve by finding polynomial approximations to $x \mapsto \cos(t\arcsin(x))$ and $x \mapsto \sin(t\arcsin(x))$, and subsequently finding circuits that implement these polynomials. This section focuses on finding the polynomial approximations, and the next subsection is devoted to developing the quantum circuit that implements these polynomials.

The main result that we reach in this section is the following theorem:

Theorem 4.2.4: Polynomial approximations of $x \mapsto \sin(t\arcsin(x))$ and $x \mapsto \cos(t\arcsin(x))$
Let $t \in [0, 1/48]$ and $\delta > 0$. Define $N = \max\{\lceil \log(1/\delta) \rceil, 1\} + 1$. Let, for all integer n satisfying $0 \leq n \leq N-1$:

$$b_n = \begin{cases} \frac{(2k)!}{(2^k k!)^2}, & \text{if } n = 2k \text{ with } k \in \mathbb{N}_0 \\ 0, & \text{otherwise} \end{cases} \qquad \text{and} \qquad a_n^{(t)} = \begin{cases} 1, & \text{if } n = 0 \\ \frac{it}{n} \cdot \sum_{k=0}^{n-1} a_k^{(t)} b_{n-k-1}, & \text{otherwise} \end{cases}$$

Next, we define:

$$C(x) = \sum_{n=0}^{N-1} \mathrm{Re}\left(a_n^{(t)}\right) x^n \qquad \text{and} \qquad S(x) = \sum_{n=0}^{N-1} \mathrm{Im}\left(a_n^{(t)}\right) x^n$$

Then, we find for all $x \in [-1/2, 1/2]$:

$$|\cos(t\arcsin(x)) - C(x)| \leq \delta \qquad \text{and} \qquad |\sin(t\arcsin(x)) - S(x)| \leq \delta$$

Moreover, we have that C and S are even and odd, respectively, and for all $x \in [-1, 1]$: $C(x), S(x) \in [-1, 1]$.

Before we give the proof of Theorem 4.2.4, we will first of all provide some lemmas. We start with a bound on the sequence $(b_n)_{n=0}^{\infty}$ that we defined in the theorem above.

Lemma 4.2.5: Bound on the coefficients b_n
Let $n \in \mathbb{N}$. Then, we find:

$$0 \leq b_n \leq \frac{1}{\sqrt{n}}$$

Proof. We trivially find that $b_n \geq 0$, which proves the lower bound on the b_n's. For the upper bound, we

employ Stirling's approximation and obtain, for all $n \in \mathbb{N}$:

$$b_{2n} \leq \frac{(2n)!}{(2^n n!)^2} \leq \frac{(2n)^{2n} \cdot e^{-2n} \cdot \sqrt{2n} \cdot e}{(2^n \cdot n^n \cdot e^{-n} \cdot \sqrt{n} \cdot \sqrt{2\pi})^2} \leq \frac{\sqrt{2ne}}{2\pi n} = \frac{e}{\pi} \cdot \frac{1}{\sqrt{2n}} \leq \frac{1}{\sqrt{2n}}$$

Hence, for all even $n \in \mathbb{N}$, the result holds. For odd $n \in \mathbb{N}$, the result is trivial, and hence this completes the proof. □

In the next lemma, we show how the sequence $(b_n)_{n=0}^{\infty}$ is related to the function $1/\sqrt{1-x^2}$.

Lemma 4.2.6: Taylor series of $x \mapsto 1/\sqrt{1-x^2}$
Let $(b_n)_{n=0}^{\infty}$ be defined as in Theorem 4.2.4. Then, for all $x \in (-1,1)$, we have:

$$\frac{1}{\sqrt{1-x^2}} = \sum_{n=0}^{\infty} b_n x^n$$

Proof. Let's define $g(t) = 1/\sqrt{1+t} = (1+t)^{-1/2}$. We easily obtain, for all $n \in \mathbb{N}_0$:[3]

$$\frac{\mathrm{d}^n g}{\mathrm{d}t^n}(t) = \left(-\frac{1}{2} - n + 1\right)_n (1+t)^{-\frac{1}{2}-n} = \left(\frac{1}{2} - n\right)_n (1+t)^{-\frac{1}{2}-n}$$

So, filling in $t = 0$ yields:

$$\frac{\mathrm{d}^n g}{\mathrm{d}t^n}(0) = \left(\frac{1}{2} - n\right)_n = (-1)^n \left(\frac{1}{2}\right)_n = (-1)^n \cdot \frac{1 \cdot 3 \cdot \cdots \cdot (2n-1)}{2^n} = (-1)^n \cdot \frac{(2n)!}{2^{2n} n!}$$

Hence, the Taylor series formally becomes the following expression:

$$\frac{1}{\sqrt{1+t}} = \sum_{n=0}^{\infty} \frac{\frac{\mathrm{d}^n g}{\mathrm{d}t^n}(0)}{n!} t^n = \sum_{n=0}^{\infty} (-1)^n \frac{(2n)!}{(2^n n!)^2} t^n \tag{4.2.1}$$

Substituting t with $-x^2$ leaves us with:

$$\frac{1}{\sqrt{1-x^2}} = \sum_{n=0}^{\infty} \frac{(2n)!}{(2^n n!)^2} x^{2n} = \sum_{n=0}^{\infty} b_n x^n$$

Now, it remains to check the radius of convergence of this Taylor series. We obtain, for all $x \in (-1,1)$, using Lemma 4.2.5:

$$\sum_{n=0}^{\infty} |b_n x^n| \leq b_0 + \sum_{n=1}^{\infty} \frac{|x|^n}{\sqrt{n}} \leq 1 + \sum_{n=1}^{\infty} |x|^n < \infty$$

Hence the Taylor series converges for all $x \in (-1,1)$, and so by Taylor's theorem is also converges to $f(x)$.[4] □

Similarly to the b_n's in Lemma 4.2.5, we can also obtain a bound for the $a_n^{(t)}$'s. This, we achieve in the lemma below.

Lemma 4.2.7: Bound on the coefficients $a_n^{(t)}$
Let $t \in [0, 1/12]$, and let $(a_n^{(t)})_{n=0}^{\infty}$ be defined as in Theorem 4.2.4. Then, we find, for all $n \in \mathbb{N}$:

$$0 \leq \left|a_n^{(t)}\right| \leq \frac{12t}{n^{\frac{3}{2}}}$$

[3] Here, $(a)_n$ denotes the Pochhammer symbol, which for $a \in \mathbb{C}$ and $n \in \mathbb{N}$ is defined as $(a)_n = a \cdot (a+1) \cdot (a+2) \cdot \cdots \cdot (a+n-1)$.
[4] One can also employ Abel's lemma to the holomorphic function $z \mapsto 1/\sqrt{1-z^2}$.

Proof. The lower bound is clear. We prove the upper bound using the principle of mathematical induction. For the induction basis, we must prove that the statement holds for $n = 1$ and $n = 2$. This follows easily, as:

$$\left|a_1^{(t)}\right| = \frac{t}{1} \cdot \left|\sum_{k=0}^{0} a_k^{(t)} b_{1-k-1}\right| = t \cdot 1 \cdot |b_0| = t \leq 1 = \frac{1}{1^{\frac{3}{2}}}$$

Similarly, we find:

$$\left|a_2^{(t)}\right| \leq \frac{t}{2} \cdot \left(\left|a_0^{(t)} b_1\right| + \left|a_1^{(t)} b_0\right|\right) = \frac{t}{2} \cdot (0+1) \leq \frac{1}{2} \cdot \frac{1}{12} \leq \frac{1}{2^{\frac{3}{2}}}$$

Hence, we have established the basis for induction. Now, for the induction step, we assume that the statement holds for all integer k satisfying $1 \leq k \leq n$, where $n \geq 2$. We find:

$$\left|a_{n+1}^{(t)}\right| = \frac{t}{n+1} \cdot \left|\sum_{k=0}^{n} a_k^{(t)} b_{n-k}\right| \leq \frac{t}{n+1} \cdot \left(\left|a_0^{(t)} b_n\right| + \left|a_1^{(t)} b_{n-1}\right| + \left|a_{n-1}^{(t)} b_1\right| + \left|a_n^{(t)} b_0\right| + \sum_{k=2}^{n-2} a_k^{(t)} b_{n-k}\right)$$

Filling in the bounds that we know from the induction hypothesis and from Lemma 4.2.5, leaves us with:

$$\begin{aligned}\left|a_{n+1}^{(t)}\right| &\leq \frac{t}{n+1} \cdot \left(\frac{1}{\sqrt{n}} + \frac{t}{\sqrt{n-1}} + 0 + \frac{1}{n\sqrt{n}} + \sum_{k=2}^{n-2} \frac{1}{k\sqrt{k}} \cdot \frac{1}{\sqrt{n-k}}\right) \\ &\leq \frac{t}{n+1} \cdot \left(\frac{1+\sqrt{2}t+\frac{1}{2}}{\sqrt{n}} + 2\sum_{k=2}^{\lfloor n/2 \rfloor} \frac{1}{k\sqrt{k(n-k)}}\right) \leq \frac{t}{n+1} \cdot \left(\frac{2}{\sqrt{n}} + 2\sum_{k=2}^{\lfloor n/2 \rfloor} \frac{1}{k\sqrt{k(n-k)}}\right)\end{aligned}$$

Observe that for all $y \in [0, 1/2]$:

$$y(1-y) = y - y^2 \geq y - \frac{1}{2}y = \frac{1}{2}y$$

Hence, we find for all $k \in [0, n/2]$:

$$k(n-k) = n^2 \cdot \frac{k}{n}\left(1 - \frac{k}{n}\right) \geq n^2 \cdot \frac{k}{2n} = \frac{kn}{2}$$

Thus, we obtain:

$$\begin{aligned}\left|a_{n+1}^{(t)}\right| &\leq \frac{t}{n+1} \cdot \left(\frac{2}{\sqrt{n}} + \sum_{k=2}^{\lfloor n/2 \rfloor} \frac{1}{k\sqrt{k(n-k)}}\right) \leq \frac{t}{n+1} \cdot \left(\frac{2}{\sqrt{n}} + \frac{2\sqrt{2}}{\sqrt{n}} \sum_{k=2}^{\lfloor n/2 \rfloor} \frac{1}{k\sqrt{k}}\right) \\ &\leq \frac{2t\sqrt{2}}{(n+1)\sqrt{n}} \cdot \left(1 + \sum_{k=2}^{\infty} \frac{1}{k\sqrt{k}}\right) \leq \frac{2t\sqrt{2}}{(n+1)\sqrt{n}} \cdot \left(1 + \int_1^{\infty} \frac{1}{x^{\frac{3}{2}}}\,\mathrm{d}x\right) = \frac{2t\sqrt{2}}{(n+1)\sqrt{n}} \cdot \left(1 + \left[-\frac{2}{\sqrt{x}}\right]_1^{\infty}\right) \\ &= \frac{2t\sqrt{2}}{(n+1)\sqrt{n}} \cdot (1+2) = \frac{6t\sqrt{2}}{(n+1)\sqrt{n}} \leq \frac{12t}{(n+1)\sqrt{n+1}}\end{aligned}$$

This completes the induction step, and hence, by the principle of mathematical induction, we obtain the result that we were after. This completes the proof. □

Observe that the derivative of the function $\arcsin(x)$ is $1/\sqrt{1-x^2}$. Hence, we can use the Taylor series of $1/\sqrt{1-x^2}$ to derive a Taylor series of $\exp(it \arcsin(x))$, for $t \in [0, 1/12]$. This, we achieve in the following lemma.

Lemma 4.2.8: Taylor series of $x \mapsto \exp(it \arcsin(x))$
Let $t \in [0, 1/12]$, and let $(a_n^{(t)})_{n=0}^{\infty}$ be as in Theorem 4.2.4. Then, we find, for all $x \in (-1, 1)$:

$$\exp(it \arcsin(x)) = \sum_{n=0}^{\infty} a_n^{(t)} x^n$$

Here, $\arcsin : [-1, 1] \to [-\pi/2, \pi/2]$.[a]

[a] Another way to approximate $x \mapsto \exp(it \arcsin(x))$ is by approximating $x \mapsto \arcsin(x)$ and $x \mapsto \exp(itx)$ separately, and then composing the polynomials. This does add another factor that is logarithmic in the precision to the query complexity of Circuit 4.2.18, though.

Proof. First of all, observe that for all $x \in [-\pi/2, \pi/2]$, we have that $\cos(x) \geq 0$. Hence, we find:

$$\frac{\mathrm{d}}{\mathrm{d}x} \sin(x) = \cos(x) = \sqrt{1 - \sin^2(x)}$$

Now we can apply the inverse function theorem, which yields, for all $x \in (-1, 1)$:

$$\frac{\mathrm{d}}{\mathrm{d}x} \arcsin(x) = \frac{1}{\frac{\mathrm{d}}{\mathrm{d}x} \sin(\arcsin(x))} = \frac{1}{\sqrt{1 - \sin^2(\arcsin(x))}} = \frac{1}{\sqrt{1 - x^2}}$$

Next, we formally write the Taylor series of $\exp(it \arcsin(x))$ as follows, which is allowed as the function $x \mapsto \exp(it \arcsin(x))$ is smooth around $x = 0$:

$$\exp(it \arcsin(x)) = \sum_{n=0}^{\infty} c_n^{(t)} x^n$$

We now check that the coefficients $c_n^{(t)}$ defined through this Taylor series agree with the coefficients $a_n^{(t)}$ defined in Theorem 4.2.4. To that end, we differentiate both sides and observe:

$$\begin{aligned}\sum_{n=0}^{\infty} (n+1) c_{n+1}^{(t)} x^n &= \frac{\mathrm{d}}{\mathrm{d}x} \exp(it \arcsin(x)) = \frac{it}{\sqrt{1 - x^2}} \exp(it \arcsin(x)) = it \left(\sum_{n=0}^{\infty} b_n x^n \right) \left(\sum_{n=0}^{\infty} c_n^{(t)} x^n \right) \\ &= it \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} c_k^{(t)} b_{n-k} \right) x^n\end{aligned}$$

As the above relation holds for all $x \in (-1, 1)$, we obtain the following recurrence relation:

$$c_0 = 1 \qquad \text{and} \qquad c_{n+1}^{(t)} = \frac{it}{n+1} \sum_{k=0}^{n} c_k^{(t)} b_{n-k}$$

Thus, we readily check that the coefficients $c_n^{(t)}$ indeed are in agreement with the coefficients $a_n^{(t)}$ defined on Theorem 4.2.4.

Now, it remains to check that the radius of convergence of the Taylor series is indeed at least 1. To that end, observe that for all $x \in [-1, 1]$, we have, using Lemma 4.2.7:

$$\sum_{n=0}^{\infty} \left| a_n^{(t)} x^n \right| \leq 1 + \sum_{n=1}^{\infty} \frac{|x|^n}{n\sqrt{n}} \leq 1 + \sum_{n=1}^{\infty} \frac{1}{n\sqrt{n}} = 1 + \zeta\left(\frac{3}{2}\right) < \infty$$

Hence, the Taylor series converges for all $x \in [-1, 1]$, and so it converges to $\exp(it \arcsin(x))$ by Taylor's theorem. Thus, we find that the radius of convergence is at least 1, completing the proof. □

Now, we are finally in good position to give the proof of Theorem 4.2.4.

Proof of Theorem 4.2.4. From Lemma 4.2.8, we find, for all $t \in [0, 1/12]$:

$$\exp(it \arcsin(x)) = \sum_{n=0}^{\infty} a_n^{(t)} x^n$$

Hence, taking the real and imaginary parts yields:

$$\cos(t \arcsin(x)) = \sum_{n=0}^{\infty} \mathrm{Re}\left(a_n^{(t)}\right) x^n = C(x) \qquad \text{and} \qquad \sin(t \arcsin(x)) = \sum_{n=0}^{\infty} \mathrm{Im}\left(a_n^{(t)}\right) x^n = S(x)$$

So, for all $\delta > 0$ and $x \in [-1/2, 1/2]$, using the bound from Lemma 4.2.7:

$$\begin{aligned}\left|\cos(t \arcsin(x)) - \sum_{n=0}^{N-1} \mathrm{Re}\left(a_n^{(t)}\right) x^n\right| &\leq \sum_{n=N}^{\infty} \left|a_n^{(t)} x^n\right| \leq \sum_{n=N}^{\infty} \frac{|x|^n}{n\sqrt{n}} \leq \left(\frac{1}{2}\right)^N \cdot \sum_{n=2}^{\infty} \frac{1}{n\sqrt{n}} \leq 2^{-N} \int_1^{\infty} \frac{1}{x^{\frac{3}{2}}}\, \mathrm{d}x \\ &\leq 2^{-N} \left[-\frac{2}{\sqrt{x}}\right]_1^{\infty} = 2^{-N+1} \leq 2^{-\log(1/\delta)} = \delta\end{aligned}$$

The claim for $\sin(t \arcsin(x))$ is proven in the exact same way. Moreover, we easily find using mathematical induction that for all even n, $a_n^{(t)}$ is real, and for all odd n, $a_n^{(t)}$ is purely imaginary. Hence, we readily find that C is even and S is odd.

Finally, we must prove that for all $x \in [-1, 1]$, we have $C(x), S(x) \in [-1, 1]$. For all $x \in [-1, 1]$, we readily find, using $\mathrm{Im}(a_0^{(t)}) = 0$ and Lemma 4.2.7:

$$|S(x)| \leq \sum_{n=1}^{\infty} \left|\mathrm{Im}(a_n^{(t)})\right| |x|^n \leq \frac{1}{4} \sum_{n=1}^{\infty} \frac{1}{n\sqrt{n}} \leq \frac{1}{4} + \frac{1}{4} \int_1^{\infty} \frac{1}{x^{\frac{3}{2}}}\, \mathrm{d}x = \frac{1}{4} + \frac{1}{4} \left[-\frac{2}{\sqrt{x}}\right]_1^{\infty} = \frac{1}{4} + \frac{1}{2} < 1$$

So, it remains to show that $C(x) \in [-1, 1]$, for all $x \in [-1, 1]$. To that end, we prove that for all $n \in \mathbb{N}$, we have $a_{2n}^{(t)} \leq 0$ and $-ia_{2n+1}^{(t)} \geq 0$. This is clear for $n = 1$, and moreover, we have, for all $n \in \mathbb{N}$, by the induction hypothesis:

$$a_{2n}^{(t)} = \frac{it}{2n-1} \sum_{k=0}^{2n-1} a_k^{(t)} b_{2n-k-1} = -\frac{t}{2n+1} \sum_{k=0}^{n-1} (-ia_{2k+1}^{(t)}) b_{2(n-1)-2k} \leq 0$$

And on the other hand, as $t \leq 1/48$:

$$\begin{aligned}-ia_{2n+1}^{(t)} &= \frac{t}{2n+1} \sum_{k=0}^{n} a_{2k}^{(t)} b_{2n-2k} = \frac{t}{2n+1} \left(a_0^{(t)} b_{2n} + \sum_{k=1}^{n} a_{2k}^{(t)} b_{2n-2k}\right) \\ &\geq \frac{t}{2n+1} \left(b_{2n} - \sum_{k=1}^{n} \frac{12t}{2k\sqrt{2k(2n-2k)}}\right) \geq \frac{t}{(2n+1)\sqrt{n}} \left(\frac{1}{2\sqrt{2}} - 12t\sqrt{2}\right) \geq 0\end{aligned}$$

So, we find that $C(x) \leq 1$ for all $x \in [-1, 1]$. That it is lower bounded by -1 can be seen from the same argument as the one used for $S(x)$. This completes the proof. □

In [GSLW18], a similar result is reached, but for $t \in [0, 2/\pi]$. The proof is much more involved, though, and hence we will refer the interested reader there. Their result allows for a speed-up of a constant factor of the query complexity of $C(O_{f,G})$ in the construction of the fractional phase query circuit, i.e., Circuit 4.2.18.

As a final note, $C(x)$ and $S(x)$ are depicted in Figure 4.1.

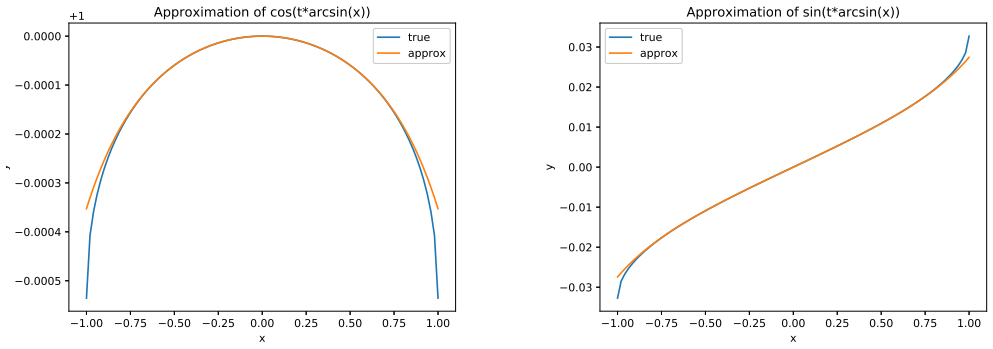


Figure 4.1: The approximations of $x \mapsto \sin(t \arcsin(x))$ and $x \mapsto \cos(t \arcsin(x))$ with $t = 1/48$ and $N = 10$.

4.2.4 Implementation of block-encodings of polynomials of arbitrary operators

In this subsection, we will attempt to solve the following problem. Suppose that we have an operator U, acting on the n-qubit state space, and suppose that we have a quantum circuit which implements a block-encoding of this operator. The goal is to construct a quantum circuit that implements a block-encoding of $P(U)$, where P is a polynomial, and $P(U)$ is defined in the obvious way. It turns out that this is possible if the block-encoding of U and P satisfy certain properties.

As a first step towards achieving this goal, we define a *helper polynomial*, which will help us during the construction of the quantum circuit.

Definition 4.2.9: Helper polynomial
Let $P \in \mathbb{R}[x]$ be a real polynomial. Then, we define the *helper polynomial of* P to be the following polynomial:

$$1 - P^2 \in \mathbb{R}[x]$$

In particular, observe that the helper polynomial of an even or odd polynomial is always even. Next, we define the notion of an *implementation pair*:

Definition 4.2.10: Implementation pair
Let $P \in \mathbb{R}[x]$ be an odd or even polynomial such that for all $x \in [-1, 1]$, we have $|P(x)| \leq 1$. We say that $\tilde{P}, \tilde{Q} \in \mathbb{C}[x]$ is an *implementation pair of* P if:

1. $\mathrm{Re}(\tilde{P}) = P$.
2. $\deg(\tilde{Q}) < \deg(\tilde{P})$.
3. $\forall x \in [-1, 1]$, $|\tilde{P}(x)|^2 + (1 - x^2)|\tilde{Q}(x)|^2 = 1$.
4. $\tilde{P}$ and $\tilde{Q}$ have opposite parity, i.e., either $\tilde{P}$ is even and $\tilde{Q}$ is odd, or vice versa.

We now prove that for real any polynomial that is either even or odd, and that on the interval $[-1, 1]$ is bounded in absolute value by 1, we can always construct such an implementation pair.

Theorem 4.2.11: Construction of an implementation pair
Let $P \in \mathbb{R}[x]$ be either even or odd, such that for all $x \in [-1,1]$, we have $|P(x)| \leq 1$. Let R be the set of roots of the helper polynomial of P, i.e., $1 - P(r)^2 = 0$ if and only if $r \in R$. Moreover, for all $r \in R$, let m_r be the multiplicity of the root r. Then, we can write, for some $\tilde{K} \in \mathbb{R}$:

$$1 - P(x)^2 = \tilde{K} \prod_{r \in R} (x - r)^{m_r}$$

Next, we define:

1. $R_0 = \{r \in R : r = 0\}$ (and $m_0 = 0$ if $R_0 = \varnothing$).
2. $R_{(0,1)} = \{r \in R : r \in (0,1)\}$.
3. $R_{[1,\infty)} = \{r \in R : r \in [1,\infty)\}$.
4. $R_I = \{r \in R : \mathrm{Re}(r) = 0, \mathrm{Im}(r) > 0\}$.
5. $R_C = \{r \in R : \mathrm{Re}(r) > 0, \mathrm{Im}(r) > 0\}$.

Furthermore, we formally define:

1. $A_r(x) = \sqrt{r^2 - 1}x + ir\sqrt{1 - x^2}$.
2. $B_r(x) = \sqrt{|r|^2 + 1}x + i|r|\sqrt{1 - x^2}$.
3. $C_{a,b}(x) = cx^2 - (a^2 + b^2) + i\sqrt{c^2 - 1}x\sqrt{1 - x^2}$ where $c = a^2 + b^2 + \sqrt{2(a^2+1)b^2 + (a^2-1)^2 + b^4}$.

Moreover, we formally define:

$$W(x) = \sqrt{K}x^{\frac{m_0}{2}} \prod_{r \in R_{(0,1)}} \left(x^2 - r^2\right)^{\frac{m_r}{2}} \prod_{r \in R_{[1,\infty)}} A_r(x)^{m_r} \prod_{r \in R_I} B_r(x)^{m_r} \prod_{a+bi \in R_C} C_{a,b}(x)^{m_r}$$

Factoring out the above expression yields unique $B, C \in \mathbb{R}[x]$ such that $W(x) = B(x) + i\sqrt{1 - x^2}C(x)$. Next we define $\tilde{P} = P + iB$ and $\tilde{Q} = iC$. Then $\tilde{P}, \tilde{Q}$ is an implementation pair of P.

Proof. First of all, observe that the helper polynomial is even and real. Hence if r is a root of $1 - P^2$, then so are $-r$, r^* and $-r^*$, and the multiplicities of these roots are equal. Thus, we find that we can write, for some $K \in \mathbb{R}$:

$$\begin{aligned}
&1 - P(x)^2 = \tilde{K} \prod_{r \in R} (x - r)^{m_r} \\
=&Kx^{m_0} \prod_{r \in R_{(0,1)}} (x^2 - r^2)^{m_r} \prod_{r \in R_{[1,\infty)}} (r^2 - x^2)^{m_r} \prod_{r \in R_I} (x^2 + |r|^2)^{m_r} \prod_{a+bi \in R_C} (x^4 + 2x^2(b^2 - a^2) + (a^2 + b^2)^2)^{m_r}
\end{aligned}$$

Moreover, as $|P(x)| \leq 1$ for all $x \in [-1,1]$, we find that $1 - P(x)^2 \geq 0$ for all $x \in [-1,1]$. In particular, $1 - P(0)^2 \geq 0$. Hence, plugging in $x = 0$ into the above equation allows us to conclude that m_0 is even and $K \geq 0$. Moreover, plugging in $r \in R_{(0,1)}$ shows that m_r is even for all $r \in R_{(0,1)}$ as well.

Now, observe that all factors of W are polynomials of the form[5] $B'(x) + i\sqrt{1 - x^2}C'(x)$, with $B', C' \in \mathbb{R}[x]$ having opposite parity and $\deg(C') < \deg(B')$. The product of two such expressions is again of that form:

$$\begin{aligned}
&\left(B_1(x) + i\sqrt{1 - x^2}C_1(x)\right)\left(B_2(x) + i\sqrt{1 - x^2}C_2(x)\right) \\
=\ &B_1(x)B_2(x) - (1 - x^2)C_1(x)C_2(x) + i\sqrt{1 - x^2}(B_1(x)C_2(x) + B_2(x)C_1(x))
\end{aligned}$$

where $B_1(x)B_2(x) - (1 - x^2)C_1(x)C_2(x)$ is again of opposite parity from $B_1(x)C_2(x) + B_2(x)C_1(x)$, and:

$$\begin{aligned}
\deg(B_1(x)B_2(x) + (1 - x^2)C_1(x)C_2(x)) &= \max(\deg(B_1) + \deg(B_2), \deg(C_1) + \deg(C_2) + 2) \\
&= \deg(B_1) + \deg(B_2) \\
&> \max(\deg(B_1) + \deg(C_2), \deg(B_2) + \deg(C_1)) \\
&= \deg(B_1(x)C_2(x) + B_2(x)C_1(x))
\end{aligned}$$

[5] B' and C' are not the derivatives of B and C in this context.

So we can write $W(x) = B(x)+i\sqrt{1-x^2}C(x)$ with $B, C \in \mathbb{R}[x]$ having opposite parity and $\deg(B) > \deg(C)$. Moreover, we have, for real x:

$$\begin{aligned} & B(x)^2 + (1-x^2)C(x)^2 = |W(x)|^2 = W(x)\overline{W(x)} \\ = \; & K^2 x^{m_0} \prod_{r \in R_{(0,1)}} (x^2 - r^2)^{m_r} \prod_{r \in R_{[1,\infty)}} (|A_r(x)|^2)^{m_r} \prod_{r \in R_I} (|B_r(x)|^2)^{m_r} \prod_{a+bi \in R_C} (|C_{a,b}(x)|^2)^{m_r} = 1 - P(x)^2 \end{aligned}$$

because we have:

$$\begin{aligned} |A_r(x)|^2 &= (r^2-1)x^2 + r^2(1-x^2) = r^2 - x^2 \\ |B_r(x)|^2 &= (|r|^2+1)x^2 + |r|^2(1-x^2) = x^2 + |r|^2 \\ |C_{a,b}(x)|^2 &= (cx^2 - (a^2+b^2))^2 + (c^2-1)x^2(1-x^2) \\ &= c^2x^4 - 2(a^2+b^2)c^2x^2 + (a^2+b^2)^2 + (1-c^2)x^4 + (c^2-1)x^2 \\ &= x^4 + (c^2(1-2(a^2+b^2)) - 1)x^2 + (a^2+b^2)^2 \\ &= x^4 + 2(b^2-a^2)x^2 + (a^2+b^2)^2 \end{aligned}$$

Now, we check that $\tilde{P}$ and $\tilde{Q}$ satisfy the properties of an implementation pair of P. First of all, we observe that $\mathrm{Re}(\tilde{P}) = \mathrm{Re}(P + iB) = P$, and hence the first property is satisfied. Furthermore, we have:

$$\deg(\tilde{P}) = \max(\deg(P), \deg(B)) \geq \deg(B) > \deg(C) = \deg(\tilde{Q})$$

and so the second property is satisfied as well. The third property is satisfied because:

$$|\tilde{P}(x)|^2 + (1-x^2)|\tilde{Q}(x)|^2 = P(x)^2 + B(x)^2 + (1-x^2)C^2(x) = P(x)^2 + 1 - P(x)^2 = 1$$

which implies that the third property is also satisfied. Finally, observe that $B(x)^2+(1-x^2)C(x)^2 = 1-P(x)^2$, and $\deg(B) > \deg(C)$, so $2\deg(B) = 2\deg(P) \Leftrightarrow \deg(B) = \deg(P)$. So B is of the same parity as P, and of opposite parity from C, implying that $\tilde{P}$ and $\tilde{Q}$ are indeed of opposite parity. This completes the proof. □

Next, we introduce the concept of an implementing phase vector:

Definition 4.2.12: Phase vector

Let $P, Q \in \mathbb{C}[x]$ be polynomials that satisfy the following requirements:

1. P and Q are both either even or odd, and they are of opposite parity.
2. $\deg(Q) < \deg(P)$.
3. For all $x \in [-1, 1]$, we have $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1$.

Then, a vector $\Phi \in \mathbb{R}^{\deg(P)+1}$ is an *implementing phase vector of the pair P, Q with global constant* $\alpha \in \mathbb{C}$, if:

$$\left[\prod_{k=1}^{\deg(P)} e^{i\phi_k Z} R(x)\right] e^{i\phi_{\deg(P)+1} Z} = \alpha \begin{bmatrix} P(x) & iQ(x)\sqrt{1-x^2} \\ iQ^*(x)\sqrt{1-x^2} & P^*(x) \end{bmatrix}$$

where:

$$e^{i\phi Z} = \begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix} \qquad \text{and} \qquad R(x) = \begin{bmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{bmatrix}$$

We show that using an implementation pair of P, we can always find an implementing phase vector of P. We will do this using the principle of mathematical induction in Theorem 4.2.14, but we first of all provide the induction step in Lemma 4.2.13.

Lemma 4.2.13: Induction step in the construction of an implementing phase vector
Let $P, Q \in \mathbb{C}[x]$ be a pair of polynomials satisfying the requirements of Definition 4.2.12. Let $\deg(P) = k \geq 1$. Then we can find a $\phi \in \mathbb{R}$ such that $e^{2i\phi} = P_k/Q_{k-1}$. We define the following polynomials:

$$\overline{P}(x) = e^{-i\phi}\left(xP(x) + \frac{P_k}{Q_{k-1}}(1-x^2)Q(x)\right) \qquad \text{and} \qquad \overline{Q}(x) = e^{-i\phi}\left(\frac{P_k}{Q_{k-1}}xQ(x) - P(x)\right)$$

Then the pair $\overline{P}, \overline{Q}$ satisfies the requirements of Definition 4.2.12 as well, and we have $k-1 = \deg(\overline{P}) < \deg(P) = k$. Moreover, suppose that $\Psi \in \mathbb{R}^k$ is an implementing phase vector of $\overline{P}, \overline{Q}$. Define $\Phi = (\psi_1, \ldots, \psi_{k-1}, \psi_k - \frac{\pi}{4}, \phi - \frac{\pi}{4})$. Then Φ is an implementing phase vector for P, Q with global constant $-i\alpha$.

Proof. First of all, we have that $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1$, and hence:

$$2\deg(P) = \deg(|P(x)|^2) = \deg(1 - (1-x^2)|Q(x)|^2) = 2 + 2\deg(Q)$$

So, we have $\deg(Q) = \deg(P) - 1$, and hence $Q_{k-1} \neq 0$. Moreover, the highest-degree term of $|P(x)|^2 + (1-x^2)|Q(x)|^2$ must vanish, hence $|P_k|^2 = |Q_{k-1}|^2$. So, indeed, we can find a $\phi \in \mathbb{R}$ such that $e^{2i\phi} = P_k/Q_{k-1}$. We can also easily observe that the highest-degree term of $\overline{P}$ vanishes as well, which implies that $\deg(\overline{P}) < \deg(P)$. The parities of $\overline{P}$ and $\overline{Q}$ are the opposite of those of P and Q, respectively. Furthermore, observe:

$$\begin{aligned} e^{-i\frac{\pi}{4}Z}R(x)e^{-i\frac{\pi}{4}Z} &= \begin{bmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} \begin{bmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{bmatrix} \begin{bmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} = \begin{bmatrix} -ix & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -ix \end{bmatrix} \\ &= -i\begin{bmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{bmatrix} \end{aligned}$$

Then, we find:

$$\begin{aligned} \left[\prod_{j=1}^{k} e^{i\phi_j Z}R(x)\right] e^{i\phi_{k+1}Z} &= \left[\prod_{j=1}^{k-1} e^{i\psi_j Z}R(x)\right] e^{i(\psi_k - \frac{\pi}{4})Z}R(x)e^{i(\phi-\frac{\pi}{4})Z} \\ &= -i\left[\prod_{j=1}^{k-1} e^{i\psi_j Z}R(x)\right] e^{i\psi_k Z}\begin{bmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{bmatrix} e^{i\phi Z} \\ &= -i\alpha\begin{bmatrix} \overline{P}(x) & i\overline{Q}(x)\sqrt{1-x^2} \\ i\overline{Q}^*(x)\sqrt{1-x^2} & \overline{P}^*(x) \end{bmatrix}\begin{bmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{bmatrix}\begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix} \\ &= -i\alpha\begin{bmatrix} \overline{P}(x) & i\overline{Q}(x)\sqrt{1-x^2} \\ i\overline{Q}^*(x)\sqrt{1-x^2} & \overline{P}^*(x) \end{bmatrix}\begin{bmatrix} e^{i\phi}x & ie^{-i\phi}\sqrt{1-x^2} \\ ie^{i\phi}\sqrt{1-x^2} & e^{-i\phi}x \end{bmatrix} \\ &= -i\alpha\begin{bmatrix} e^{i\phi}x\overline{P}(x) - e^{i\phi}\overline{Q}(x)(1-x^2) & i(e^{-i\phi}\overline{P}(x) + e^{-i\phi}x\overline{Q}(x))\sqrt{1-x^2} \\ i(e^{i\phi}x\overline{Q}^*(x) + e^{i\phi}\overline{P}^*(x))\sqrt{1-x^2} & -e^{-i\phi}\overline{Q}^*(x)(1-x^2) + e^{-i\phi}x\overline{P}^*(x) \end{bmatrix} \\ &= -i\alpha\begin{bmatrix} P(x) & iQ(x)\sqrt{1-x^2} \\ iQ^*(x)\sqrt{1-x^2} & P^*(x) \end{bmatrix} \end{aligned}$$

The last equality follows from simply writing out the top-left and top-right entry, and next observing that the lower-left and lower-right simply follow from taking conjugates. Finally, we find that $|\overline{P}(x)|^2 + (1-x^2)|\overline{Q}(x)|^2 = 1$ from the unitarity of the above matrices. The matrix equation also implies that $\deg(\overline{P}) \geq k-1$, completing the proof. □

Theorem 4.2.14: Construction of the implementing phase vector
Let $P \in \mathbb{R}[x]$ be an even or odd real polynomial such that for all $x \in [-1,1]$, we have $|P(x)| \leq 1$. Let $\tilde{P}, \tilde{Q} \in \mathbb{C}[x]$ be an implementation pair of P. Then $\tilde{P}, \tilde{Q}$ satisfies the requirements of Definition 4.2.12. If $\deg(\tilde{P}) = 0$, then $P = e^{i\phi}$ for some $\phi \in \mathbb{R}$ and $\tilde{Q} = 0$ and hence an implementing vector for $\tilde{P}, \tilde{Q}$ with global constant 1 is $\Phi = (\phi)$. If $\deg(\tilde{P}) \geq 1$, then we can use Lemma 4.2.13 inductively to obtain an implementing phase vector $\Phi \in \mathbb{R}^{\deg(P)+1}$ with global constant $(-i)^{\deg(P)}$.

Proof. The only real thing that is left to prove is the induction basis, but this is clear. □

Now, all that is left to show is how we can use these implementing phase vectors to implement block-encodings of real polynomials applied to operators. To that end, we denote, for all normal operators U with $\|U\| \leq 1$:

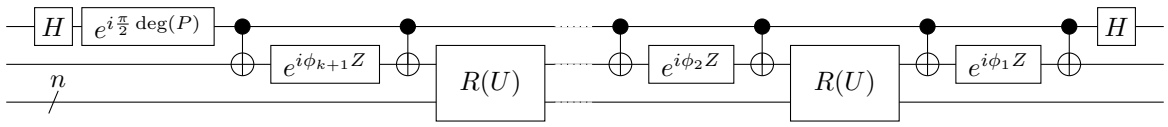
$$R(U) = \begin{bmatrix} U & (I-U^2)^{\frac{1}{2}} \\ (I-U^2)^{\frac{1}{2}} & -U \end{bmatrix}$$

Note that $R(U)$ is unitary if U is self-adjoint, as we then have:

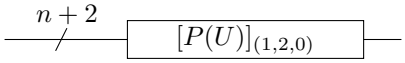
$$\begin{aligned} R(U)R(U)^* = R(U)^*R(U) = R(U)^2 &= \begin{bmatrix} U & (I-U^2)^{\frac{1}{2}} \\ (I-U^2)^{\frac{1}{2}} & -U \end{bmatrix}^2 \\ &= \begin{bmatrix} U^2 + (I-U^2) & U(I-U^2)^{\frac{1}{2}} - U(I-U^2)^{\frac{1}{2}} \\ U(I-U^2)^{\frac{1}{2}} - U(I-U^2)^{\frac{1}{2}} & I - U^2 + U^2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \end{aligned}$$

It now feels logical to replace all the $R(x)$'s with $R(U)$'s in Definition 4.2.12. This is exactly what the next circuit is all about.

Circuit 4.2.15: Real polynomial implementation
Description: This circuit implements a real polynomial $P \in \mathbb{R}[x]$ of degree k, satisfying $|P(x)| \leq 1$ for all $x \in [-1,1]$, using an implementing phase vector $\Phi \in \mathbb{R}^{k+1}$ of an implementation pair of P with global constant $(-i)^{\deg(P)}$.
Oracle circuit: A circuit implementing $R(U)$, which acts on $n+1$ qubits.
Query complexity: The query complexity to $R(U)$ is $k = \deg(P)$.
Number of qubits: $n+2$.
Circuit:



Shorthand notation:



Proof. Let Φ be the implementing phase vector of the pair $\tilde{P}, \tilde{Q}$, with global constant $(-i)^{\deg(P)}$. Then, we find, by Definition 4.2.12:

$$\left[\prod_{j=1}^{k} e^{i\phi_j Z} R(x)\right] e^{i\phi_{k+1}Z} = (-i)^{\deg(P)} \begin{bmatrix} \tilde{P}(x) & i\tilde{Q}(x)\sqrt{1-x^2} \\ i\tilde{Q}^*(x)\sqrt{1-x^2} & \tilde{P}^*(x) \end{bmatrix}$$

Taking conjugations everywhere yields:

$$\left[\prod_{j=1}^{k} e^{-i\phi_j Z} R(x)\right] e^{-i\phi_{\deg(P)+1}Z} = i^{\deg(P)} \begin{bmatrix} \tilde{P}^*(x) & -i\tilde{Q}(x)\sqrt{1-x^2} \\ -i\tilde{Q}^*(x)\sqrt{1-x^2} & \tilde{P}(x) \end{bmatrix}$$

and hence we find that $-\Phi$ is an implementing phase vector of the pair $\tilde{P}^*, -\tilde{Q}$, with global constant $i^{\deg(P)}$.

Now, observe that on the first two qubits, we iteratively apply an operation with the following matrix representation:

$$\begin{aligned}\mathrm{CNOT}\left(I_2 \otimes e^{i\phi Z}\right)\mathrm{CNOT} &= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}e^{i\phi}&0&0&0\\0&e^{-i\phi}&0&0\\0&0&e^{i\phi}&0\\0&0&0&e^{-i\phi}\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\\ &= \begin{bmatrix}e^{i\phi}&0&0&0\\0&e^{-i\phi}&0&0\\0&0&e^{-i\phi}&0\\0&0&0&e^{i\phi}\end{bmatrix} = \begin{bmatrix}e^{i\phi Z}&0\\0&e^{-i\phi Z}\end{bmatrix}\end{aligned}$$

Between these operations, we interlace the application of $R(U)$ on the last $n+1$ qubits. The matrix representation of $R(U)$ applied to full $n+2$-qubit system is given by:

$$I_2 \otimes R(U) = \begin{bmatrix}R(U)&0\\0&R(U)\end{bmatrix}$$

Both operations are diagonal, which simplifies taking their matrix products. Hence, we find that the total action of the circuit, without both of the Hadamard gates, can be expressed with the following matrix representation:

$$\begin{aligned}&\begin{bmatrix}\prod_{j=1}^{k}\left[e^{i\phi_j Z}R(U)\right]e^{i\phi_{k+1}Z} & 0\\ 0 & \prod_{j=1}^{k}\left[e^{-i\phi_j Z}R(U)\right]e^{-i\phi_{k+1}Z}\end{bmatrix}\begin{bmatrix}e^{i\frac{\pi}{2}\deg(P)}&0\\0&e^{-i\frac{\pi}{2}\deg(P)}\end{bmatrix}\\ =&\begin{bmatrix}i^{\deg(P)}\prod_{j=1}^{k}\left[e^{i\phi_j Z}R(U)\right]e^{i\phi_{k+1}Z} & 0\\ 0 & (-i)^{\deg(P)}\prod_{j=1}^{k}\left[e^{-i\phi_j Z}R(U)\right]e^{-i\phi_{k+1}Z}\end{bmatrix}\\ =&\begin{bmatrix}\tilde{P}(U) & i\tilde{Q}(U)(I-U^2)^{\frac{1}{2}} & 0 & 0\\ i\tilde{Q}^*(U)(I-U^2)^{\frac{1}{2}} & \tilde{P}^*(U) & 0 & 0\\ 0 & 0 & \tilde{P}^*(U) & -i\tilde{Q}(U)(I-U^2)^{\frac{1}{2}}\\ 0 & 0 & -i\tilde{Q}^*(U) & \tilde{P}(U)\end{bmatrix}\end{aligned}$$

Finally, we take into account the Hadamard gates. Observe, with A, B matrices of appropriate sizes:

$$H\begin{bmatrix}A&0\\0&B\end{bmatrix}H = \frac{1}{2}\begin{bmatrix}I&I\\I&-I\end{bmatrix}\begin{bmatrix}A&0\\0&B\end{bmatrix}\begin{bmatrix}I&I\\I&-I\end{bmatrix} = \frac{1}{2}\begin{bmatrix}I&I\\I&-I\end{bmatrix}\begin{bmatrix}A&A\\B&-B\end{bmatrix} = \frac{1}{2}\begin{bmatrix}A+B&A-B\\A-B&A+B\end{bmatrix}$$

And hence in the top-left $2^n \times 2^n$-block of the matrix representation of the circuit, we find $\frac{1}{2}(\tilde{P}(U)+\tilde{P}^*(U)) = \mathrm{Re}(\tilde{P})(U) = P(U)$. This completes the proof. □

As a final note, observe that we can implement this circuit in a controlled manner pretty easily as well, provided that we have access to a controlled version of $R(U)$. We can simply use controlled versions of all of the $e^{i\phi Z}$ gates and the $R(U)$-gates, and what is left will cancel out if the control qubit is in state $|0\rangle$.

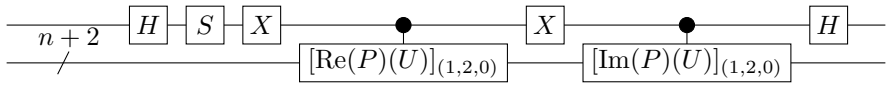
4.2.5 Addition of real and complex parts

In the previous section, we showed how we can implement a block-encoding of $P(U)$, where U is an operator on the n-qubit state space, and P is a real polynomial that is either even or odd such that for all $x \in [-1, 1]$, we have $|P(x)| \leq 1$. If we, on top of that, know how we can add two polynomials, then this is sufficient to implement block-encodings of $P(U)$ for all polynomials $P \in \mathbb{C}[x]$ satisfying $|P(x)| \leq 1$ for all $x \in [-1, 1]$, simply because we can decompose P into its even real, odd real, even imaginary and odd imaginary parts.

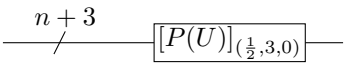
The function that we are trying to approximate, though, which is the function $x \mapsto \exp(it \arcsin(x))$, is special in the sense that its odd real part and its even imaginary part vanish. This was reflected in the approximating polynomials that we constructed in Theorem 4.2.4. Hence, in this section, we only need to add two polynomials together, whereas in the most general case, one would have to add four polynomials together. In an attempt not to digress too much from the roadmap to implementing fractional phase oracles, we will present the special case of vanishing odd real and even complex polynomials here, and refer the interested reader to [GSLW18] for the general case.

The next circuit shows us how we can obtain a block-encoding of $P(U) + iQ(U)$, if we have circuits implementing block-encodings of $P(U)$ and $Q(U)$.

Circuit 4.2.16: Addition of real and imaginary parts
Description: Let $P \in \mathbb{C}[x]$ be a polynomial, such that $\mathrm{Re}(P)$ is either odd or even, and so is $\mathrm{Im}(P)$, and such that for all $x \in [-1, 1]$, we have $|\mathrm{Re}(P)(x)| \leq 1$ and $|\mathrm{Im}(P)(x)| \leq 1$. Then, this circuit implements $P(U)$, given a controlled version of $R(U)$, for some self-adjoint operator U, acting on n qubits.
Query complexity: The query complexity to $C(R(U))$ is $2 \deg(P)$.
Circuit:



Shorthand notation:



Proof that the circuit implements a $(\frac{1}{2}, 3, 0)$-block-encoding of $P(U)$. Observe that if the first qubit is in state $|1\rangle$ after the S-gate, then the action of the circuit up to but excluding the final Hadamard gate reduces to a $(1, 3, 0)$-block-encoding of $\mathrm{Im}(P)(U)$. Similarly, if the first qubit is in state $|0\rangle$ after the S-gate, then the action of this part of the circuit reduces to a $(1, 3, 0)$-block-encoding of $\mathrm{Re}(P)(U)$. The S introduces an i for the imaginary part, and the Hadamards add the two block-encodings, at the cost of a constant factor of $\frac{1}{2}$. This completes the proof. □

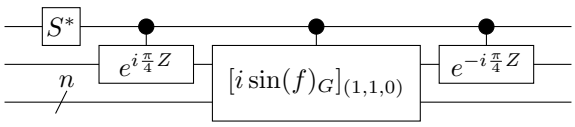
As mentioned before, Circuit 4.2.16 is not as generic as possible, but general enough to suit our needs. In the following subsection, we will show how we can use this circuit to implement fractional phase queries.

4.2.6 Quantum circuit of the fractional phase query

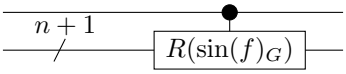
Using the polynomial approximations provided in Theorem 4.2.4, and the circuit provided in Circuit 4.2.16, we are almost ready to present the circuit that implements a block-encoding of the fractional phase query. The final ingredient that we need is a circuit that implements $C(R(\sin(f)_G))$, because such a circuit is required as a subcircuit in Circuit 4.2.16.

Luckily, we can construct a circuit $C(R(\sin(f)_G))$ easily from the controlled version of Circuit 4.2.3, which we introduced all the way back in Subsection 4.2.2. The details are shown in the circuit below.

Circuit 4.2.17: Implementation of a controlled $R(\sin(f)_G)$
Description: This circuit implements $C(\sin(f)_G)$. We require that $\|f\|_\infty \leq \pi/2$ on G, such that $\cos(f)_G \geq 0$.
Query complexity: The query complexity to $C(O_{f,G})$ is 2.
Circuit:



Shorthand notation:



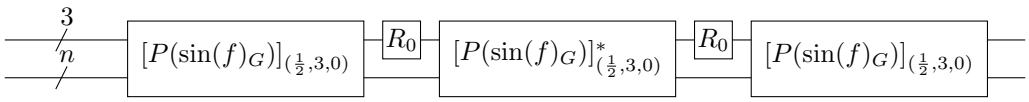
Proof. Recall the matrix representation of $[i\sin(f)_G]_{(1,1,0)}$ from the proof of Circuit 4.2.3. We find that the matrix representation of Circuit 4.2.17, for the moment disregarding the control qubit altogether, can be calculated as follows:

$$\begin{bmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} \begin{bmatrix} i\sin(f)_G & -\cos(f)_G \\ \cos(f)_G & -i\sin(f)_G \end{bmatrix} \begin{bmatrix} e^{i\frac{\pi}{4}} & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{bmatrix} = i \begin{bmatrix} \sin(f)_G & \cos(f)_G \\ \cos(f)_G & -\sin(f)_G \end{bmatrix}$$

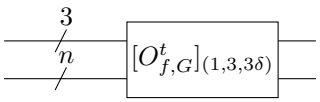
Now, if the control qubit is in the state $|0\rangle$, then nothing happens. However, if it is in the state $|1\rangle$, then first of all a global factor of $-i$ is induced by S^*, and subsequently the operation given in the equation above is applied to the last $n+1$ qubits. Hence, the $-i$ and the i cancel, and we also have that $\cos(f)_G = \left(\sqrt{1-\sin^2(f)}\right)_G$, because $\|f\|_\infty \leq \pi/2$. Thus, indeed this circuit implements a controlled version of $R(\sin(f)_G)$. □

Now, finally, we are able to merge the approximation in Theorem 4.2.4, and both circuits, Circuit 4.2.16 and Circuit 4.2.17, to obtain a fractional phase query. The details are provided in the circuit displayed below.

Circuit 4.2.18: Fractional phase query for $t \in [0, 1/48]$
Description: Let $\|f\|_\infty \leq \frac{1}{2}$ on G. Let $t \in [0, 1/48]$ and $\delta > 0$. Let C, S as in Theorem 4.2.4, and let $P = C + iS$. Then this circuit is $(1, 3, 6\delta)$-encoding of $O_{f,G}^t$.
Parameters: A precision parameter, $\delta > 0$.
Derived constants: $N = 1 + \max\{\lceil\log(1/\delta)\rceil, 1\}$.
Oracle circuit: A controlled phase oracle $C(O_{f,G})$, acting on n qubits.
Number of qubits: $n + 3$, plus the auxiliary qubits used by $O_{f,G}$.
Query complexity: The query complexity to $C(O_{f,G})$ is $3 \cdot 2 \cdot 2N = 12 + 12\max\{\lceil\log(1/\delta)\rceil, 1\}$.
Circuit:



Shorthand notation:



Proof that this circuit is a $(1, 3, 6\delta)$-block-encoding of $O_{f,G}^t$. First of all, observe that $\sin(f)_G$ only has real eigenvalues in the interval $[-1/2, 1/2]$. Moreover, observe that $P(x)$ approximates $\exp(it\arcsin(x))$ on $[-1/2, 1/2]$ with supremum distance bounded above by 2δ. Hence, we find that the operator norm between $P(\sin(f)_G)$ and $\exp(it\arcsin(\sin(f)_G))$, which is the maximum of the difference in eigenvalues, is bounded above by 2δ. Hence, $[P\sin(f)_G]_{(\frac{1}{2},3,0)}$ is a $(\frac{1}{2}, 3, 2\delta)$-block-encoding of $O_{f,G}^t$.

Neglecting these 2δ-errors for the moment, we find that if we apply $[P\sin(f)_G]_{(\frac{1}{2},3,0)}$ to $|0\rangle^{\otimes 3} \otimes |\psi\rangle$, where $|\psi\rangle$ is an n-qubit state, then we obtain a state:

$$\frac{1}{2}|0\rangle^{\otimes 3} \otimes O_{f,G}^t |\psi\rangle + \frac{\sqrt{3}}{2}\left|\psi^\perp\right\rangle$$

where for any n-qubit state $|\phi\rangle$, we have $\left(\langle 0|^{\otimes 3} \otimes \langle\phi|\right)\left|\psi^\perp\right\rangle = 0$. We define:

$$|G\rangle = |0\rangle^{\otimes 3} \otimes O_{f,G}^t |\psi\rangle \qquad \text{and} \qquad |B\rangle = \left|\psi^\perp\right\rangle$$

And so we have the state:

$$\sin\left(\frac{\pi}{6}\right)|G\rangle + \cos\left(\frac{\pi}{6}\right)|B\rangle$$

Now, in a similar manner as we did with the amplitude estimation circuit, Circuit 3.5.9, i.e., by splitting the sine and cosine using trigonometric formulas, we can deduce that the final state is equal to:

$$\sin\left(\frac{\pi}{2}\right)|G\rangle + \cos\left(\frac{\pi}{2}\right)|B\rangle = |0\rangle^{\otimes 3} \otimes O_{f,G}^t |\psi\rangle$$

There are 3 places where an error of norm at most 2δ is introduced, and hence the resulting error is bounded by 6δ. So, this circuit is a $(1, 3, 6\delta)$-block-encoding of $O_{f,G}^t$. □

Circuit 4.2.18 is the main result of this section. We can apply it repeatedly to implement *any* real power of $O_{f,G}$.[6] This will be particularly useful in constructing quantum gradient estimation algorithms, as we will show in the subsequent section.

4.3 Gilyén et al.'s quantum gradient estimation algorithm

In the previous section, we have seen how we can implement a quantum circuit that performs a fractional phase query. In this section, we will elaborate on how we can use this quantum circuit to estimate the gradient of a high-dimensional function. The approach taken in this section is a generalization of the one presented in [GAW17] to different ℓ^p-norms and larger Gevrey-classes of functions.

On high level, the idea of Gilyén et al.'s quantum gradient estimation algorithm, which estimates the gradient of some function f at some points $\mathbf{a}$, is to use a numerical method to approximately calculate the values of $\nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})$, where the vectors $\mathbf{x}$ are taken from some grid that is centered around $\mathbf{a}$, and store these approximately calculated values in the phase of the amplitudes corresponding to the state $|\mathbf{x}\rangle$. Then, the inverse quantum Fourier transform can be employed to obtain an approximation of the gradient of f at $\mathbf{a}$, which was already observed by Jordan in 2005 [Jor05].

In Subsection 4.3.1, we will first of all introduce the grid of points on which Gilyén et al.'s gradient estimation algorithm performs function evaluations. Secondly, we will elaborate on which numerical method is used to approximate the values of $\nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})$, in Subsection 4.3.2. Finally, in Subsection 4.3.3, we will show how we can use fractional phase queries and the inverse quantum Fourier transform to arrive at an approximation of $\nabla f(\mathbf{a})$.

[6] This construction is reminiscent of the *method of continuity* that one encounters when studying elliptic partial differential operators. [Kry08], Theorem 1.4.4.

4.3.1 Grid

Intuitively, it is clear that every algorithm that estimates the gradient based on function evaluations must somehow describe a set of points on which the function is to be evaluated. Subsequently, from the function values on this set of points, the gradient is to be estimated. Gilyén et al.'s quantum gradient estimation algorithm is no different, and in this subsection, we describe this set of points, which we will refer to as the *grid*.

Definition 4.3.1: Grid used in Gilyén et al.'s quantum gradient estimation algorithm
Let $d, n \in \mathbb{N}$, $\mathbf{a} \in \mathbb{R}^d$ and $r > 0$. Then, we define:

$$G_{n,r,\mathbf{a}} = \bigtimes_{j=1}^{d} \left\{ a_j + \frac{r}{2^n} \cdot \left(k + \frac{1}{2} \right) : k \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\} \right\} \subseteq \mathbb{R}^d$$

We refer to $G_{n,r,\mathbf{a}}$ as *the grid around* $\mathbf{a}$ *with side length* r. Moreover, whenever it is clear that we are talking about vectors that are elements of $G_{n,r,\mathbf{a}}$, then we use the following shorthand notation. For all $\mathbf{j} \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\}^d$, we define:

$$\mathbf{x_j} = \mathbf{a} + \frac{r}{2^n} \left(\mathbf{j} + \frac{1}{2} \right) \in G_{n,r,\mathbf{a}}$$

Note that there are 2^{nd} points in this grid. They are arranged in a square lattice with side length r, centered around the point $\mathbf{a} \in \mathbb{R}^d$. Below, in Figure 4.2, we show an example of such a grid.

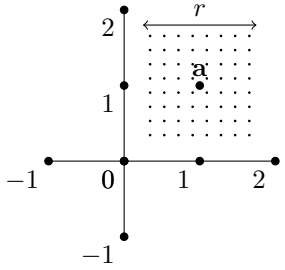


Figure 4.2: This figure portrays an example grid, namely $G_{n,r,\mathbf{a}} \subseteq \mathbb{R}^2$, where $\mathbf{a} = (1, 1)$, $r = 3/2$ and $n = 3$. We can see that the side length of the lattice is approximately r, and that the lattice has $2^n = 8$ points in each direction.

If we just calculated the function values of f at all these points of $G_{n,r,\mathbf{a}}$ in superposition, and subsequently applied the inverse Fourier transform, then we would recover Jordan's algorithm, described in [Jor05]. However, we proceed using Gilyén et al.'s approach, portrayed in [GAW17], which entails using a numerical method to *extend the region of approximate linearity.*[7] In the next subsection, we elaborate on this numerical method.

4.3.2 Numerical method

As already described at the end of the previous subsection, we will use a numerical method to extend the region of approximate linearity of f. Another way to think about it is to smooth out the highly oscillatory ripples of f by canceling the higher-order derivative terms in its Taylor series. This is achieved by employing the numerical method described in this section.

[7] Quoted literally from [GAW17].

The numerical method that we describe here is a central difference scheme of some fixed order. The coefficients of this scheme are given in the definition below.

Definition 4.3.2: Coefficients of the $2m$th order central difference scheme
Let $m \in \mathbb{N}$. For all $\ell \in \{-m, -m+1, \ldots, m-1, m\}$, we define:

$$a_\ell^{(2m)} = \begin{cases} \frac{(-1)^{\ell-1}}{\ell} \cdot \frac{\binom{m}{|\ell|}}{\binom{m+|\ell|}{|\ell|}}, & \text{if } \ell \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

We can prove that these are indeed the coefficients of the $2m$th order central difference scheme. This is achieved in the following lemma.

Lemma 4.3.3: Sums of the coefficients of the $2m$th order central difference scheme
Let $m \in \mathbb{N}$ and $k \in \{0, 1, \ldots, 2m\}$. We find:

$$\sum_{\ell=-m}^{m} a_\ell^{(2m)} \ell^k = \begin{cases} 1, & \text{if } k = 1 \\ 0, & \text{otherwise} \end{cases}$$

Proof (based on lemma 21 of [GAW17]). Suppose that P is a polynomial of degree at most $2m$. Then, we find, by Lagrange's interpolation formula:

$$P(x) = \sum_{\ell=-m}^{m} P(\ell) \prod_{\substack{j=-m \\ j \neq \ell}}^{m} \frac{x-j}{\ell-j}$$

Taking the derivative on both sides yields:

$$P'(x) = \sum_{\ell=-m}^{m} P(\ell) \sum_{\substack{j=-m \\ j \neq \ell}}^{m} \frac{1}{\ell-j} \prod_{\substack{k=-m \\ k \neq j, k \neq \ell}}^{m} \frac{x-k}{\ell-k}$$

Plugging in $x = 0$ leaves us with:

$$P'(0) = \sum_{\ell=-m}^{m} P(\ell) \sum_{\substack{j=-m \\ j \neq \ell}}^{m} \frac{1}{\ell-j} \prod_{\substack{k=-m \\ k \neq j, k \neq \ell}}^{m} \frac{-k}{\ell-k}$$

Observe that if $\ell \neq 0$ and $j \neq 0$, then one of the factors in the product on the right-hand side becomes 0, and hence the term vanishes. Moreover, if $\ell = 0$, then $j \neq 0$ as well. So, the contribution of $\ell = 0$ is 0, and if $\ell \neq 0$, then we must have that $j = 0$ in order to obtain non-zero terms. Thus, we obtain:

$$\mathbb{P}'(0) = \sum_{\substack{\ell=-m \\ \ell \neq 0}}^{m} P(\ell) \cdot \frac{1}{\ell} \cdot \prod_{\substack{k=-m \\ k \neq 0, k \neq \ell}}^{m} \frac{-k}{\ell-k}$$

For all $\ell \neq 0$, we obtain:

$$\left| \prod_{\substack{k=-m \\ k \neq 0, k \neq \ell}}^{m} \frac{-k}{\ell-k} \right| = \frac{(m!)^2}{(m-\ell)!(m+\ell)!}$$

Moreover, the product shown in the above equation picks up a sign for every k that is in between 0 and ℓ, which is satisfied for exactly $|\ell| - 1$ choices of k. Hence, we obtain:

$$P'(0) = \sum_{\substack{\ell=-m \\ \ell \neq 0}}^{m} \frac{P(\ell)}{\ell} \cdot (-1)^{|\ell|-1} \cdot \frac{(m!)^2}{(m-\ell)!(m+\ell)!}$$

From elementary rewriting, we obtain:

$$P'(0) = \sum_{\substack{\ell=-m \\ \ell\neq 0}}^{m} P(\ell) \cdot \frac{(-1)^{\ell-1}}{\ell} \cdot \frac{m!}{(m-|\ell|)!} \cdot \frac{m!}{(m+|\ell|)!} = \sum_{\substack{\ell=-m \\ \ell\neq 0}}^{m} P(\ell) \cdot \frac{(-1)^{\ell-1}}{\ell} \cdot \frac{\binom{m}{|\ell|}}{\binom{m+|\ell|}{\ell}} = \sum_{\substack{\ell=-m \\ \ell\neq 0}}^{m} P(\ell) \cdot a_\ell^{(2m)}$$

Finally, choosing $P(x) = x^k$, where k is an integer between 0 and $2m$, yields:

$$\sum_{\ell=-m}^{m} a_\ell^{(2m)} \ell^k = \sum_{\substack{\ell=-m \\ \ell\neq 0}}^{m} a_\ell^{(2m)} \ell^k = \begin{cases} 1, & \text{if } k = 1 \\ 0, & \text{otherwise} \end{cases}$$

This completes the proof. □

Next, we prove some bounds on these coefficients.

Lemma 4.3.4: Bounds on the coefficients of the $2m$th order central difference scheme
Let $m \in \mathbb{N}$. Then for all $\ell \in \{-m, \dots, m\} \setminus \{0\}$:

$$\left|a_\ell^{(2m)}\right| \leq \frac{1}{|\ell|}$$

We also have:

$$\sum_{\ell=-m}^{m} \left|a_\ell^{(2m)}\right| \leq 2\ln(m) + 2$$

And for all $k \geq 2m+1$, we have:

$$\sum_{\ell=-m}^{m} \left|a_\ell^{(2m)} \ell^k\right| \leq 6e^{-\frac{7m}{6}} m^{k+\frac{1}{2}}$$

Proof. First of all, observe, for all $\ell \in \{-m, \dots, m\} \setminus \{0\}$:

$$\left|a_\ell^{(2m)}\right| = \frac{1}{|\ell|} \cdot \frac{\binom{m}{|\ell|}}{\binom{m+|\ell|}{|\ell|}} = \frac{1}{|\ell|} \cdot \frac{m \cdot \dots \cdot (m-|\ell|+1)}{|\ell| \cdot \dots \cdot 1} \cdot \frac{|\ell| \cdot \dots \cdot 1}{(m+|\ell|) \cdot \dots \cdot (m+1)} = \frac{1}{|\ell|} \prod_{j=1}^{|\ell|} \frac{m-|\ell|+j}{m+j} \leq \frac{1}{|\ell|}$$

This implies, using the fact that $x \mapsto 1/x$ is decreasing on $(0, \infty)$:

$$\sum_{\ell=-m}^{m} \left|a_\ell^{(2m)}\right| \leq 2\sum_{\ell=1}^{m} \frac{1}{\ell} \leq 2 + 2\int_1^m \frac{1}{x}\,\mathrm{d}x = 2\ln(m) + 2$$

The last statement is Lemma 32 of [GAW17]. □

Next, we provide a lemma that will help us determine a good bound on the error that we make when using this numerical method to estimate the gradient of a high-dimensional function.

Lemma 4.3.5: Error induced by calculating the function values symmetrically on the grid
Let $d, n, k \in \mathbb{N}$, $q \in [0, 1]$, and $H : [d]^k \to \mathbb{R}$, such that for all $\alpha \in [d]^k$, $|H(\alpha)| \leq 1$. For at least a $(1 - 1/16^k)$-fraction of points $\mathbf{x} \in G_{n,1,\mathbf{0}}$, where $\mathbf{0} \in \mathbb{R}^d$, we have:

$$\left|\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{x}^\alpha\right| \leq 4^k \sqrt{2^q k^{qk} \left(\frac{d}{2}\right)^{2k-qk}}$$

Proof (based on Lemma 34 in [GAW17]). Suppose that $X_1, \dots, X_d$ are independent identically distributed random variables, taking values in $[-1/2, 1/2]$, and whose distribution is symmetric around 0. Let $\mathbf{X} = (X_1, \dots, X_d)$ be the vector of all these random variables. Then, we obtain, using the linearity of the expectation and some elementary rewriting:

$$\begin{aligned}\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] &= \mathbb{E}\left[\sum_{\alpha,\beta\in[d]^k} H(\alpha)H(\beta)\mathbf{X}^\alpha\mathbf{X}^\beta\right] = \sum_{\alpha,\beta\in[d]^k} H(\alpha)H(\beta)\mathbb{E}\left[\mathbf{X}^\alpha\mathbf{X}^\beta\right] \\ &\leq \sum_{\alpha,\beta\in[d]^k} \mathbb{E}\left[\mathbf{X}^\alpha\mathbf{X}^\beta\right] = \mathbb{E}\left[\sum_{\alpha,\beta\in[d]^k} \mathbf{X}^\alpha\mathbf{X}^\beta\right] = \mathbb{E}\left[\sum_{\alpha\in[d]^{2k}} \mathbf{X}^\alpha\right] \\ &= \mathbb{E}\left[(X_1 + \cdots + X_d)^{2k}\right]\end{aligned}$$

This, now, we can rewrite as follows, which can be proven writing out the definitions and using Fubini's theorem:

$$\mathbb{E}\left[(X_1 + \cdots + X_d)^{2k}\right] = \int_0^\infty \mathbb{P}\left[(X_1 + \cdots + X_d)^{2k} \geq t\right]\,\mathrm{d}t$$

Plugging this into the relation above, we find:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq \int_0^\infty \mathbb{P}\left[(X_1 + \cdots + X_d)^{2k} \geq t\right]\,\mathrm{d}t = \int_0^\infty \mathbb{P}\left[|X_1 + \cdots + X_d| \geq t^{\frac{1}{2k}}\right]\,\mathrm{d}t$$

Now, we can invoke the Hoeffding bound. [Hoe63] As all the X_j's are bounded in the interval $[-1/2, 1/2]$ and $\mathbb{E}(X_j) = 0$ for all $j \in [d]$, we obtain, for all $s > 0$:

$$\mathbb{P}\left[|X_1 + \cdots + X_d| \geq s\right] \leq 2e^{-\frac{2s^2}{d}}$$

Plugging in $s = t^{\frac{1}{2k}}$ yields:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq 2\int_0^\infty e^{-2\frac{t^{\frac{1}{k}}}{d}}\,\mathrm{d}t$$

Next, we perform the substitution $y = 2t^{\frac{1}{k}}/d$. We find $t = (yd/2)^k$ and hence $\mathrm{d}t = (d/2)^k k y^{k-1}\mathrm{d}y$, so:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq 2k\left(\frac{d}{2}\right)^k \int_0^\infty e^{-y}\cdot y^{k-1}\,\mathrm{d}y = 2\left(\frac{d}{2}\right)^k k\Gamma(k) = 2\left(\frac{d}{2}\right)^k k!$$

Finally, we can rewrite this using Stirling's approximation:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq 2\left(\frac{d}{2}\right)^k k! \leq 2\left(\frac{d}{2}\right)^k \cdot e^{1-k}\sqrt{k}k^k \leq 2\left(\frac{dk}{2}\right)^k$$

But, we also trivially have the following estimate, because all the X_j's are bounded by the interval $[-1/2, 1/2]$:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq \mathbb{E}\left[(X_1 + \cdots + X_d)^{2k}\right] \leq \left(\frac{d}{2}\right)^{2k}$$

So, we can geometrically average the last two equations as follows, where $q \in [0,1]$:

$$\mathbb{E}\left[\left(\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right)^2\right] \leq \left[2\left(\frac{dk}{2}\right)^k\right]^q \cdot \left(\frac{d}{2}\right)^{2k(1-q)} = 2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}$$

Hence, by Markov's inequality, we obtain:

$$\begin{aligned}\mathbb{P}\left[\left|\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right| \geq 4^k \cdot \sqrt{2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}}\right] &= \mathbb{P}\left[\left|\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right|^2 \geq 16^k 2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}\right] \\ &\leq \frac{\mathbb{E}\left[\left|\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{X}^\alpha\right|^2\right]}{16^k \cdot 2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}} \leq \frac{2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}}{16^k \cdot 2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}} = \frac{1}{16^k}\end{aligned}$$

Finally, let $\mathbf{X}$ be the random variable uniformly distributed on $G_{n,1,\mathbf{0}}$. Then, we find that the components of $\mathbf{X}$ indeed are independently identically distributed random variables bounded in the interval $[-1/2, 1/2]$. So, the final displayed equation tells us that for at most a $1/16^k$-fraction of the points in $\mathbf{x} \in G_{n,1,\mathbf{0}}$, we have:

$$\left|\sum_{\alpha\in[d]^k} H(\alpha)\mathbf{x}^\alpha\right| \geq 4^k\sqrt{2^q k^{qk}\left(\frac{d}{2}\right)^{2k-qk}}$$

Hence the reverse inequality must hold for at least a $(1-1/16^k)$-fraction of points $\mathbf{x} \in G_{n,1,\mathbf{0}}$. This completes the proof. □

Using Lemma 4.3.3, Lemma 4.3.4 and Lemma 4.3.5, we can prove a bound on the error that arises from using this numerical method.

Theorem 4.3.6: Error induced by the numerical method
Let $d, m, n \in \mathbb{N}$, $f \in \mathcal{G}^\sigma_{d,\mathbb{R}^d,M,c}$, $\mathbf{a} \in \mathbb{R}^d$ and $r > 0$. Then, for at least a 999/1000-fraction of the points $\mathbf{x} \in G_{n,r,\mathbf{a}}$, we have:

$$\left|\nabla f(\mathbf{a}) \cdot (\mathbf{x}-\mathbf{a}) - \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a}+\ell(\mathbf{x}-\mathbf{a}))\right| \leq \sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k$$

Proof (based on theorem 23 in [GAW17]). Observe that f is an element from the Gevrey-class of functions. Recall from Definition 4.1.1 that this implies that we must have that the Taylor series of f around $\mathbf{a}$ converges on all of $\mathbb{R}^d$. Thus, for all $\mathbf{x} \in \mathbb{R}^d$, we can write:

$$f(\mathbf{a}+\mathbf{x}) = \sum_{k=0}^{\infty}\sum_{\alpha\in[d]^k} \frac{\partial_\alpha f(\mathbf{a}) \cdot \mathbf{x}^\alpha}{k!}$$

Hence, by simply plugging in the above series and rearranging, we obtain:

$$\begin{aligned}\sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a}+\ell(\mathbf{x}-\mathbf{a})) &= \sum_{\ell=-m}^{m} a_\ell^{(2m)} \sum_{k=0}^{\infty}\frac{1}{k!}\sum_{\alpha\in[d]^k} \partial_\alpha f(\mathbf{a}) \cdot \ell^k (\mathbf{x}-\mathbf{a})^\alpha \\ &= \sum_{k=0}^{\infty}\frac{1}{k!}\left[\sum_{\alpha\in[d]^k} \partial_\alpha f(\mathbf{a})(\mathbf{x}-\mathbf{a})^\alpha\right] \cdot \left[\sum_{\ell=-m}^{m} a_\ell^{(2m)} \ell^k\right]\end{aligned}$$

Now, we can use Lemma 4.3.3, we obtain that the right-most factor in the sum vanishes for $k = 0$ and all $k \in \{2, 3, \ldots, 2m\}$. Moreover, the term where $k = 1$ exactly equals $\nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})$. Hence, we are only left with the terms where $k \geq 2m + 1$ in the following relation:

$$\left| \nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) - \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a} + \ell(\mathbf{x} - \mathbf{a})) \right| \leq \sum_{k=2m+1}^{\infty} \frac{1}{k!} \left| \sum_{\alpha \in [d]^k} \partial_\alpha f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})^\alpha \right| \cdot \left| \sum_{\ell=-m}^{m} a_\ell^{(2m)} \ell^k \right|$$

We can bound the right-most factor by Lemma 4.3.4. Moreover, observe that for all $k \geq 2m+1$, Lemma 4.3.5 implies that for at least a $(1 - 1/16^k)$-fraction of the vectors $\mathbf{x} \in G_{n,r,\mathbf{a}}$, we have the following bound, where we have picked $q = 2(1 - \sigma)$:

$$\left| \sum_{\alpha \in [d]^k} \frac{\partial_\alpha f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})^\alpha}{r^k c^k k^{\sigma k}} \right| \leq 4^k \sqrt{2^{2(1-\sigma)} k^{2k(1-\sigma)} \left(\frac{d}{2}\right)^{2k-2k(1-\sigma)}} = 4^k 2^{1-\sigma} k^k k^{-\sigma k} \left(\frac{d}{2}\right)^{\sigma k}$$

Here, we used that for all $k \in \mathbb{N}$, $\partial_\alpha f(\mathbf{a}) \leq c^k (k!)^\sigma \leq c^k k^{\sigma k}$. We can use this bound for *all* terms where $k \geq 2m + 1$, for at least the following fraction of points:

$$1 - \sum_{k=2m+1}^{\infty} \frac{1}{16^k} \geq 1 - \sum_{k=3}^{\infty} \left(\frac{1}{16}\right)^k = 1 - \frac{1}{16^3} \cdot \frac{1}{1 - \frac{1}{16}} = 1 - \frac{1}{16^3 - 16^2} \geq \frac{999}{1000}$$

So, we obtain, for at least a 999/1000-fraction of points $\mathbf{x} \in G_{n,r,\mathbf{a}}$:

$$\begin{aligned}
&\left| \nabla f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) - \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a} + \ell(\mathbf{x} - \mathbf{a})) \right| \\
\leq &\sum_{k=2m+1}^{\infty} \frac{1}{k!} \left| \sum_{\alpha \in [d]^k} \frac{\partial_\alpha f(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})^\alpha}{r^k c^k k^{\sigma k}} \right| \cdot r^k c^k k^{\sigma k} \cdot \left| \sum_{\ell=-m}^{m} a_\ell^{(2m)} \ell^k \right| \\
\leq &\sum_{k=2m+1}^{\infty} \frac{4^k 2^{1-\sigma} k^k k^{-\sigma k} \left(\frac{d}{2}\right)^{\sigma k} r^k c^k k^{\sigma k} \cdot 6e^{-\frac{7m}{6}} m^{k+\frac{1}{2}}}{\sqrt{2\pi} e^{-k} k^k \sqrt{k}} \\
= &\sum_{k=2m+1}^{\infty} \frac{e^k 4^k 2^{1-\sigma} \left(\frac{d}{2}\right)^{\sigma k} r^k c^k \cdot 6e^{-\frac{7m}{6}} m^k \sqrt{m}}{\sqrt{2\pi}\sqrt{k}} \\
= &\sum_{k=2m+1}^{\infty} \frac{6 \cdot 2^{1-\sigma} e^{-\frac{7m}{6}} \sqrt{m}}{\sqrt{2\pi k}} \left(\frac{4erc m d^\sigma}{2^\sigma}\right)^k \\
\leq &\sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k
\end{aligned}$$

In the last step, we used that $4e/\sqrt{2} \leq 8$ and:

$$\frac{6 \cdot 2^{1-\sigma} \cdot e^{-\frac{-7m}{6}} \cdot \sqrt{m}}{\sqrt{2\pi(2m+1)}} \leq \frac{6 \cdot \sqrt{2} \cdot e^{-\frac{7m}{6}}}{\sqrt{4\pi}} \leq \frac{6\sqrt{2}}{\sqrt{4\pi} \cdot e} \leq 1$$

This completes the proof. □

Now that we have developed a numerical method, and proved a bound on the error that arises from using this numerical method, we can proceed to describe how Gilyén et al.'s algorithm implements this numerical method. This is the main objective of the next subsection.

4.3.3 Algorithm

Now, we are finally in good position to state Gilyén et al.'s quantum gradient estimation algorithm, and prove that it indeed correctly estimates the gradient of a multidimensional function with the indicated success probability. The algorithm presented here is essentially the same algorithm as described in [GAW17]. However, some generalizations are made. Most notably, one can now choose to approximate the gradient with respect to an arbitrary ℓ^p-norm, and one can estimate the gradient of functions whose Gevrey-type is 1 or smaller, instead of $\frac{1}{2}$ or smaller. Moreover, the extra (necessary) requirement that $\|f\|_\infty \leq \frac{1}{2}$ has been explicitly stated here.

Gilyén et al.'s gradient estimation algorithm makes use of a phase oracle via which we can query f on a grid $G \subseteq \mathbb{R}^d$. We require the states $\{|\mathbf{x}\rangle : \mathbf{x} \in G\} \subseteq \mathbb{C}^{2^q}$ to form an orthonormal set of q-qubit states, however we do not specify exactly what these states should be. In general, we can think of these states to somehow encode vectors in $\mathbb{R}^d$ in an approximate manner, just like tuples of floating point numbers can represent vectors in $\mathbb{R}^d$ approximately in a classical computer.

In contrast, what we do require is to have access to a circuit that generate these states. In particular, given a grid $G = G_{n,r,\mathbf{a}} \subseteq \mathbb{R}^d$, defined in Definition 4.3.1, we require that we have a circuit C_G, acting on nd qubits, which implements the following action for all $\mathbf{j} \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\}^d$:

$$C_G : |\mathbf{j}\rangle |0\rangle^{\otimes q} \mapsto |\mathbf{j}\rangle |\mathbf{x}_\mathbf{j}\rangle \qquad \text{where} \qquad |\mathbf{j}\rangle = |j_1\rangle |j_2\rangle \cdots |j_d\rangle$$

Hence, we can think of the nd-qubit states $|\mathbf{j}\rangle$ to be the indices, or labels, of the points $\mathbf{x}_\mathbf{j} \in G$.

In the algorithm, it will be required to perform some basic arithmetic operations on these labels. In particular, we have to multiply a vector $\mathbf{j}$ by some predefined integer ℓ. Hence, we must construct a *multiplication by ℓ* quantum circuit, which has the following action for all $\mathbf{j} \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\}^d$:

$$M_\ell : |\mathbf{j}\rangle |0\rangle^{\otimes (n + \lceil \log(\ell) \rceil) d} \mapsto |\mathbf{j}\rangle |\ell \mathbf{j}\rangle$$

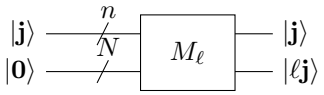
Here, the vectors $|\mathbf{j}\rangle$ are encoded as before, in nd qubits, but in order to make sure that the result does not overflow, we must reserve $(n + \lceil \log(m) \rceil)d$ qubits to store the resulting vector, where $m \geq \ell$.

Luckily, the construction of such a circuit is not all that difficult. We present its construction in the box below.

Circuit 4.3.7: Mutliplication by ℓ
Description: This circuit implements multiplication by a constant $0 \leq \ell \leq m$ for the labels that we will encounter in Gilyén et al.'s gradient estimation algorithm.
Qubit layout: Three registers. The first one is nd qubits in size, and the last two are $(n + \lceil \log(m) \rceil)d$ qubits in size. The middle register is to be considered auxiliary.
Elementary gate complexity: $\mathcal{O}(((n + \lceil \log(m) \rceil)d)^2)$
Circuit: The idea is to write the constant $|\ell|$ in its binary form: $|\ell| = (\ell_{\lceil \log(m) \rceil - 1} \cdots \ell_0)_2$. Then, for every k for which ℓ_k is non-zero, we perform the following three actions:

1. First of all, we perform the following action on the first two registers:
$$|\mathbf{j}\rangle |\mathbf{0}\rangle \mapsto |\mathbf{j}\rangle \left|2^k \mathbf{j}\right\rangle$$
This can be implemented using at most nd CNOT-gates.
2. Next, we call the quantum Fourier adder to add the second register to the third one. This takes $\mathcal{O}(((n + \lceil \log(m) \rceil)d)^2)$ elementary gates.
3. Finally, we uncompute the result that we temporarily stored in the second register, which again takes at most nd CNOT-gates.

In the third register, the resulting value $|\ell|\mathbf{j} = ((\ell_0 + 2\ell_1 + \cdots + 2^{\lceil \log(m) \rceil - 1}\ell_{\lceil \log(m) \rceil - 1})\mathbf{j}$ accumulates. If ℓ is negative, then the final part of the circuit is to multiply the last circuit with -1. This can be done by exploiting the two's complement notation of the integer states, hence by applying an X to all qubits of the last register, preparing a vector $(1)^d$ in the second register, again applying the quantum Fourier adder, and finally reverting the second register back to $(0)^d$.
Shorthand notation: Here, $N = n + \lceil \log(m) \rceil$:



Now, we have developed all the ingredients necessary to state the full circuit that we will be using in Gilyén et al.'s algorithm. The circuit will construct a state that upon measurement yields an approximation to the gradient of the function that is being called by the phase oracle.

The idea is that the fractional phase queries can be used to obtain linear combinations of function values in the exponents. In particular, the circuit uses fractional phase queries to construct the following state:

$$\frac{1}{\sqrt{2^{nd}}} \sum_{\mathbf{j} \in \{-2^{n-1}, \ldots, 2^{n-1}-1\}^d} e^{iS \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{x}_{\ell \mathbf{j}})} |\mathbf{j}\rangle$$

The details of the circuit are shown in the box below.

Circuit 4.3.8: The main routine in Gilyén et al.'s quantum gradient estimation algorithm
Description: This is the central quantum circuit in the Gilyén et al.'s quantum gradient algorithm.
Input: The input consists of a controlled phase oracle $C(O_{f,G})$, where $G = G_{n+\lceil \log(m) \rceil, 2^{\lceil \log(m) \rceil} r, \mathbf{a}}$.
Parameters: The following parameters can be adjusted:

1. $S > 0$.
2. $\delta > 0$.
3. $m > 0$.

Qubit layout: Three registers. The first register consists of nd qubits and is able to contain labels of points in G. The second one is $(n + \lceil \log(m) \rceil)d$ qubits in size, and can hold constant multiples of labels of points in G. The third one consists of q qubits, and can hold the actual states that correspond to the vectors in G. On top of that, at least $(n + \lceil \log(m) \rceil)d + 3$ auxiliary qubits are required, plus some extra if they are required by $C(O_{f,G})$.
Circuit: The circuit consists of the following parts.

1. First of all, nd Hadamard gates are applied to the first register to obtain a uniform superposition of the labels of the points in $G_{n,r,\mathbf{a}}$.
2. Next, for all $\ell \in \{-m, \ldots, m\} \setminus \{0\}$, the following steps are performed:
   (a) First of all, the multiplication circuit M_ℓ is applied to the first two registers.
   (b) Then the state construction circuit C_G is applied to the second and third registers.
   (c) Next, a fractional phase query is applied, where the third register indicates the point at which the function f is to be evaluated. The power of the fractional phase query is given by $a_\ell^{(2m)} S$, and the precision parameter is δ.
   (d) Finally, steps (b) and (a) are reversed.
3. Finally, the d-dimensional inverse quantum Fourier transform is applied to the first nd qubits, i.e., the n-qubit inverse quantum Fourier transform is applied d times, to the d sets of n qubits that comprise the first register.

Query complexity: The number of queries to $C(O_{f,G})$ is upper bounded by:

$$48S \sum_{\ell=-m}^{m} \left(\left| a_\ell^{(2m)} \right| + \frac{1}{48} \right) \cdot 12(\max\{\lceil \log(1/\delta) \rceil, 1\} + 1)$$

Next, we show how the quantum circuit introduced in Circuit 4.3.8 is used in Gilyén et al.'s quantum gradient estimation algorithm.

Algorithm 4.3.9: Gilyén et al.'s quantum gradient estimation algorithm
Description: This algorithm is an ε-precise ℓ^p-approximate quantum gradient estimation algorithm for $\mathcal{G}^{\sigma}_{d,\mathbb{R}^d,\frac{1}{2},c}$ in $\mathbf{a}$ on $G = G_{n+\lceil \log(m), 2^{\lceil \log(m) \rceil} r, \mathbf{a}}$, with success probability 2/3.
Parameters:
1. $\varepsilon > 0$. The accuracy with which we want to find an approximation of the gradient.
2. $c > 0$. One of the parameters that determines the class of functions for which we want the algorithm to work.
3. $d \in \mathbb{N}$. The dimension of the domain of the function.
4. $\sigma \in [1/2, 1]$. An upper bound on the Gevrey-type of the function.
5. $\mathbf{a} \in \mathbb{R}^d$. The point where we want to estimate the gradient.
6. $p \in [1, \infty]$. The norm with which we want to measure the accuracy.

Derived constants:
1. $\varepsilon' = \frac{\varepsilon}{d^{\frac{1}{p}}}$.
2. $m = \lceil \log\left(\frac{cd^{\sigma}}{\varepsilon'}\right) \rceil$.
3. $r = \frac{(81 \cdot 8 \cdot 42\pi cmd^{\sigma}/\varepsilon')^{-\frac{1}{2m}}}{9cmd^{\sigma}}$.
4. $S = 2\pi \cdot 2^{\lceil \log(\frac{4}{r\varepsilon'}) \rceil}$.
5. $n = \lceil \log(\frac{4}{r\varepsilon'}) \rceil + \lceil \log(3rc) \rceil$.
6. $\delta = \frac{1}{2000 \cdot (12mS + 288S(2\ln(m)+2))}$.
7. $N = \lceil \frac{2\log(3d)}{\log(3)} \rceil$.

Input: A $(\lceil \log(m) \rceil + n)$-qubit phase oracle of f on G, controlled on one qubit, denoted by $C(O_{f,G})$.
Output: A vector $\mathbf{v} \in \mathbb{R}^d$ such that $\|\mathbf{v} - \nabla f(\mathbf{a})\|_p \leq \varepsilon$.
Success probability: Lower bounded by 2/3.
Number of qubits: $nd + (\lceil \log(m) \rceil + n)d$, plus the auxilliary qubits that are required by $C(O_{f,G})$.
Query complexity: The query complexity to $C(O_{f,G})$ is upper bounded by:

$$SN\left[2m + 48 \cdot (2\ln(m) + 2)\right] \cdot \left[12 + 12\max\{\lceil \log(1/\delta) \rceil, 1\}\right] = \tilde{O}(S) = \tilde{O}\left(\frac{1}{\varepsilon' r}\right) = \tilde{O}\left(\frac{cd^{\sigma+\frac{1}{p}}}{\varepsilon}\right)$$

Algorithm:
1. Repeat N times:
   (a) Run Circuit 4.3.8.
   (b) Measure the first nd qubits, and interpret the result as $\mathbf{h} \in \{-2^{n-1}, \ldots, 2^{n-1} - 1\}^d$.
   (c) Denote $\mathbf{g} = \frac{2\pi}{Sr}\mathbf{h}$.
2. Let $\mathbf{v}$ be the coordinate-wise mean of the N vectors $\mathbf{g}$ obtained in the previous step.
3. Return $\mathbf{v}$.

Proof of the lower bound on the success probability in Algorithm 4.3.9. Note that in total, we have to implement at most the following number of calls to a fractional phase query circuit shown in Circuit 4.2.18:

$$48 \cdot S \cdot \sum_{\substack{\ell=-m \\ \ell \neq 0}}^{m} \left(\left| a_{\ell}^{(2m)} \right| + \frac{1}{48} \right) \leq 2mS + 48S \cdot (2\ln(m) + 2)$$

Every fractional phase query introduces a norm error of 6δ. Hence, from Lemma B.2, we can deduce that the total error introduced by all the imperfect implementations of the fractional phase queries, is upper bounded by:

$$\left[2mS + 48S \cdot (2\ln(m) + 2)\right] \cdot 6\delta \leq \frac{12mS + 288S(2\ln(m) + 2)}{2000 \cdot (12mS + 288S(2\ln(m) + 2))} \leq \frac{1}{2000}$$

If we ignore this error introduced by the fractional phase queries, observe that the state of the first nd qubits in the quantum system, after application of the circuit, but before application of the Quantum Fourier

Transform, is given by:

$$|\psi\rangle = \frac{1}{\sqrt{2^{nd}}} \sum_{\mathbf{j}\in\{2^{n-1},\ldots,2^{n-1}-1\}^d} e^{iS\sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a}+\ell(\mathbf{x_j}-\mathbf{a}))} |\mathbf{j}\rangle$$

We now calculate the difference in norm to the state:

$$\left|\tilde{\psi}\right\rangle = \frac{1}{\sqrt{2^{nd}}} \sum_{\mathbf{j}\in\{-2^{n-1},\ldots,2^{n-1}-1\}^d} e^{iS\nabla f(\mathbf{a})\cdot(\mathbf{x_j}-\mathbf{a})} |\mathbf{j}\rangle$$

Next, let $R \subseteq G_{n,r,\mathbf{a}}$ be the set of points for which the bound of Theorem 4.3.6 holds. Then, we find $|R| \geq 999/1000 \cdot |G_{n,r,\mathbf{a}}| = 999/1000 \cdot 2^{nd}$. Hence, the number of points for which the bound of Theorem 4.3.6 does not hold is at most $2^{nd}/1000$. We obtain:

$$\begin{aligned}
\left\||\psi\rangle - \left|\tilde{\psi}\right\rangle\right\|^2 &\leq \frac{1}{2^{nd}} \sum_{\mathbf{x}\in G_{n,r,\mathbf{a}}} \left| e^{iS\left(\nabla f(\mathbf{a})\cdot(\mathbf{x}-\mathbf{a}) - \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a}+\ell(\mathbf{x}-\mathbf{a}))\right)} - 1 \right|^2 \\
&\leq \frac{1}{2^{nd}} \sum_{\mathbf{x}\in R} S^2 \left| \nabla f(\mathbf{a})\cdot(\mathbf{x}-\mathbf{a}) - \sum_{\ell=-m}^{m} a_\ell^{(2m)} f(\mathbf{a}+\ell(\mathbf{x}-\mathbf{a})) \right|^2 + \frac{4}{1000} \\
&\leq S^2 \cdot \left(\sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k \right)^2 + \frac{1}{250} = \left(S \sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k \right)^2 + \frac{1}{250}
\end{aligned}$$

Now, we bound the expression within parentheses. Observe that we have:

$$8rcmd^\sigma = \frac{8}{9} \cdot \left(81 \cdot 8 \cdot 42\pi cmd^\sigma/\varepsilon'\right)^{-1/2m}$$

And as $\varepsilon' < c$, we find $8rcmd^\sigma \leq \frac{8}{9}$. Hence:

$$\begin{aligned}
\sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k &= (8rcmd^\sigma)^{2m+1} \sum_{k=0}^{\infty} (8rcmd^\sigma)^k \\
&\leq \left(\frac{8}{9} \cdot \left(81 \cdot 8 \cdot 42\pi cmd^\sigma/\varepsilon'\right)^{-1/2m} \right)^{2m+1} \cdot \sum_{k=0}^{\infty} \left(\frac{8}{9}\right)^k \\
&= \left(\frac{8}{9}\right)^{2m+1} \cdot \frac{\varepsilon'}{81 \cdot 8 \cdot 42\pi cmd^\sigma} \cdot \left(81 \cdot 8 \cdot 42\pi cmd^\sigma/\varepsilon'\right)^{-1/2m} \cdot 9 \\
&\leq 1 \cdot \frac{\varepsilon'}{8 \cdot 42\pi} \cdot \frac{\left(81 \cdot 8 \cdot 42\pi cmd^\sigma/\varepsilon'\right)^{-1/2m}}{9cmd^\sigma} = \frac{\varepsilon' r}{8 \cdot 42\pi}
\end{aligned}$$

So, we obtain:

$$S \sum_{k=2m+1}^{\infty} (8rcmd^\sigma)^k \leq 2\pi \cdot 2^{\lceil \log(\frac{4}{r\varepsilon'}) \rceil} \cdot \frac{\varepsilon' r}{8 \cdot 42\pi} \leq \frac{2\pi \cdot 8}{\varepsilon' r} \cdot \frac{\varepsilon' r}{8 \cdot 42\pi} = \frac{1}{21}$$

Thus:

$$\left\||\psi\rangle - \left|\tilde{\psi}\right\rangle\right\|^2 \leq \left(\frac{1}{21}\right)^2 + \frac{1}{250}$$

Furthermore, we have:

$$\left|\tilde{\psi}\right\rangle = \frac{1}{\sqrt{2^{nd}}} \sum_{j\in\{0,1,\ldots,2^{n-1}\}^d} e^{iS\nabla f(\mathbf{a})\cdot\frac{jr}{2^n}} |j\rangle = \frac{1}{\sqrt{2^{nd}}} \sum_{j\in\{0,1,\ldots,2^{n-1}\}^d} e^{\frac{2\pi i}{2^n}\cdot\frac{Sr}{2\pi}\cdot\nabla f(\mathbf{a})\cdot j} |j\rangle$$

Observe, for all $j \in [d]$:

$$\frac{Sr}{2\pi \cdot 2^n} \cdot |\nabla f(\mathbf{a})_j| = \frac{2\pi \cdot 2^{\lceil \log(\frac{4}{r\varepsilon'}) \rceil} r}{2\pi \cdot 2^{\lceil \log(\frac{4}{r\varepsilon'}) \rceil + \lceil \log(3rc) \rceil}} \cdot |\nabla f(\mathbf{a})_j| \leq \frac{r}{3rc} \cdot \frac{1}{2} c = \frac{1}{6}$$

Hence, we find that the value that is to be approximated is bounded away from the edge of the spectrum that can be reached with the Fourier transform, which would have been $\frac{1}{2}$ in the above equation. In particular, it follows from well-known theory, which can for example be found in [NC00], Equation 5.34, that (where $\left|\tilde{\psi}_j\right\rangle$ denotes the state of the jth set of n qubits in $|\psi\rangle$, which is well-defined because these sets of n qubits are not entangled):

$$\sum_{\substack{k \in \{-2^{n-1}, \ldots, 2^{n-1}-1\} \\ |k - Sr\nabla f(\mathbf{a})_j / 2\pi| > 4}} \left| \langle k | \operatorname{QFT}^*_{2^n} \left|\tilde{\psi}_j\right\rangle \right|^2 \leq \frac{1}{6}$$

But the real state of the quantum system just before the application of the inverse quantum Fourier transform is not exactly $\left|\tilde{\psi}\right\rangle$, but another state $|\phi\rangle$ instead. We have, according to the triangle inequality:

$$\left\| |\phi\rangle - \left|\tilde{\psi}\right\rangle \right\| \leq \| |\phi\rangle - |\psi\rangle \| + \left\| |\psi\rangle - \left|\tilde{\psi}\right\rangle \right\| \leq \frac{1}{2000} + \sqrt{\left(\frac{1}{21}\right)^2 + \frac{1}{250}}$$

And hence, using Lemma B.1, we find that the probability that $|h_j - Sr\nabla f(\mathbf{a})/2\pi| \geq 4$ is bounded above by:

$$\frac{1}{6} + \left\| |\phi\rangle - \left|\tilde{\psi}\right\rangle \right\| \leq \frac{1}{6} + \sqrt{\left(\frac{1}{21}\right)^2 + \frac{1}{250}} + \frac{1}{2000} \leq \frac{1}{3}$$

Now, we obtain that g_j is likely to approximate $\nabla f(\mathbf{a})_j$ well, as:

$$\begin{aligned}
\mathbb{P}\left[|g_j - \nabla f(\mathbf{a})_j| > \varepsilon'\right] &= \mathbb{P}\left[\left|\frac{Sr}{2\pi} g_j - \frac{Sr}{2\pi}\nabla f(\mathbf{a})_j\right| > \frac{Sr\varepsilon'}{2\pi}\right] = \mathbb{P}\left[\left|h_j - \frac{Sr}{2\pi}\nabla f(\mathbf{a})_j\right| > \frac{2\pi \cdot 2^{\lceil \log\left(\frac{4}{r\varepsilon'}\right) \rceil} r\varepsilon'}{2\pi}\right] \\
&\leq \mathbb{P}\left[\left|h_j - \frac{Sr}{2\pi}\nabla f(\mathbf{a})_j\right| > \frac{4r\varepsilon'}{r\varepsilon'}\right] = \mathbb{P}\left[\left|h_j - \frac{Sr}{2\pi}\nabla f(\mathbf{a})_j\right| > 4\right] \leq \frac{1}{3}
\end{aligned}$$

But our final result can only differ by more than ε' from $\nabla f(\mathbf{a})$ in the jth coordinate if more than half of the g_j's are located outside of the interval $[\nabla f(\mathbf{a})_j - \varepsilon', \nabla f(\mathbf{a})_j + \varepsilon']$. This probability we denote by P, and we can bound it as follows:

$$P \leq \binom{N}{N/2} \left(\frac{1}{3}\right)^{N/2} \leq \frac{N^{N+\frac{1}{2}} e^{-N} e}{\left[(N/2)^{N/2+\frac{1}{2}}\, e^{-N/2}\sqrt{2\pi}\right]^2} \cdot \left(\frac{1}{3}\right)^{N/2} \leq \frac{e}{2\pi\sqrt{N}} \cdot \left(\frac{1}{3}\right)^{N/2} \leq 3^{-N/2}$$

Hence, we obtain that the probability that all coordinates of the gradient are well-estimated (i.e., with error at most ε') is lower bounded by:

$$(1-P)^d \geq 1 - dP \geq 1 - d3^{-N/2} \geq 1 - d \cdot 3^{-\frac{2\log(3d)}{2\log(3)}} = 1 - d \cdot 3^{-\log_3(3d)} = 1 - \frac{d}{3d} = \frac{2}{3}$$

Thus, with probability at least 2/3, we find that the ℓ^∞-norm error is at most $\varepsilon' = \varepsilon / d^{1/p}$. But we also have for all $\mathbf{x} \in \mathbb{R}^d$ and $p \in [1, \infty)$, by Hölder's inequality:

$$\|\mathbf{x}\|_p \leq d^{\frac{1}{p}} \|\mathbf{x}\|_\infty$$

And hence with probability at least 2/3, the result differs at most ε from the true gradient, measured in ℓ^p-norm. □

As a final note, let's take a step back and ponder over the properties of the algorithm that we presented in this section. We developed a quantum gradient estimation algorithm which perform the following number of function evaluations:

$$\tilde{O}\left(\frac{cd^{\sigma+\frac{1}{p}}}{\varepsilon}\right) \tag{4.3.1}$$

From a classical point of view, it is intuitively clear that we can never obtain a query complexity smaller than linear in d, simply because we have d entries of the gradient to determine. The algorithm presented here, though, can in certain cases have a query complexity that is better than linear in d. Especially in fields where estimating derivatives of high-dimensional functions occurs, this algorithm has the potential to provide a quadratic reduction of the query complexity, e.g., by taking $\sigma = \frac{1}{2}$ and $p = \infty$.

The most prominent of these fields is machine learning, as this field essentially revolves around fitting functions with a large number of parameters. This is why in Chapter 6, we will investigate whether we can combine this algorithm with other "quantum tricks", to obtain speed-ups within the field of quantum machine learning.

5 Optimality of Gilyén et al.'s quantum gradient estimation algorithm

Now that we have developed an algorithm that estimates gradients and discussed its query complexity, in Chapter 4, the natural question that arises is whether we can improve on this query complexity, or whether we have reached an optimal algorithm. In this chapter, we will show that under certain conditions, Gilyén et al.'s algorithm is essentially optimal, i.e., we can only improve the query complexity by at most logarithmic factors. [GAW17] gives the proof for the case $\sigma = \frac{1}{2}$ and $p = \infty$, but we generalize their approach to give a proof for all $\sigma \in [0, \frac{1}{2}]$ and $p \in [1, \infty]$.

The proof will be given in several parts. First of all, we will prove that the result holds for $\sigma = [0, \frac{1}{2}]$ and $p = 1$, in Section 5.1. Subsequently, we will use reduction arguments to deduce that optimality is also reached for all choices for $p \in [1, \infty)$, in Section 5.2. Finally, we will put all this in perspective and discuss circumstances for which optimality has either not yet been attained or proven, in Section 5.3.

5.1 Lower bound of specific cases

Before we start with proving a lower bound on the query complexity of quantum gradient estimation algorithms, we first of all fix some auxiliary notation. Throughout this section, we will frequently use the random variable that describes the outcome of a quantum gradient estimation algorithm $\mathcal{A}$. We say that whenever we estimate the gradient of a function f with algorithm $\mathcal{A}$, then $\mathcal{A}(f)$ is the random variable that denotes the output of the algorithm $\mathcal{A}$ under input f. Hence, $\mathcal{A}(f)$ is a random variable taking values in $\mathbb{R}^d$. We also introduce $T_{\mathcal{A}}(f)$, which denotes the number of calls the algorithm $\mathcal{A}$ makes to the controlled version of the phase oracle $O_{f,G}$, when the input function is f.

Now, it is time to present the key result of this section. One can directly compare the query complexity in this result with the one that is found by plugging in $p = 1$ and $\sigma = \frac{1}{2}$ into Equation 4.3.1, and observe that they are essentially equal.

Theorem 5.1.1: Lower bound on query complexity of special cases of quantum gradient estimation algorithms
Let $c, M > 0$, $d \in \mathbb{N}$, $0 < 73\varepsilon < cM$, $G \subseteq \Omega \subseteq \mathbb{R}^d$ and $\mathbf{a} \in \Omega$. Suppose that $\mathcal{A}$ is an ε-precise ℓ^1-approximate quantum gradient estimation algorithm for $\mathcal{G}^0_{d,\Omega,M,c}$ in $\mathbf{a}$ on G with success probability 17/18. Then, on every input controlled phase oracle $C(O_{f,G})$, where $f \in \mathcal{G}^0_{d,\Omega,M,c}$, the query complexity of the controlled phase oracle $C(O_{f,G})$, denoted by $T_{\mathcal{A}}(f)$, satisfies:

$$T_{\mathcal{A}}(f) \geq \frac{cd^{\frac{3}{2}}}{876\varepsilon}$$

We will postpone the proof of this theorem to the end of this section. Instead, we will first of all introduce and prove several lemmas and theorems before turning to the proof of this theorem.

In order to arrive at a good lower bound on the query complexity of a quantum gradient estimation algorithm, we find functions whose function values are close together, but whose gradients are not. Intuitively, we can think of these functions as being relatively hard to distinguish from one another, but simultaneously we require the algorithm to be able to distinguish them. We provide such functions in the following definition.

Definition 5.1.2: Test functions
Let $c, M > 0$, $d \in \mathbb{N}$, $0 < 73\varepsilon < cM$ and $\mathbf{a} \in \mathbb{R}^d$. For all $\mathbf{b} \in \{-1, 1\}^d$, we define:

$$f_{\mathbf{b},\varepsilon,c,\mathbf{a}} : \mathbb{R}^d \to \mathbb{R}, \qquad f_{\mathbf{b},\varepsilon,c,\mathbf{a}}(\mathbf{x}) = \sum_{j=1}^{d} \frac{73\varepsilon b_j}{cd} \sin(c(x_j - a_j)) \prod_{k \in [d]\setminus\{j\}} \cos(c(x_k - a_k))$$

These functions we refer to as the *test functions*. Moreover, we define the *set of test functions on an open set* $\Omega \subseteq \mathbb{R}^d$ as follows:

$$\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}} = \{f_{\mathbf{b},\varepsilon,\mathbf{c},\mathbf{a}}|_\Omega : \mathbf{b} \in \{-1, 1\}^d\}$$

These functions will only be of use to us if they are in the class of functions that we have required our algorithm to perform well for. The next theorem tells us that they are.

Theorem 5.1.3: Test functions are contained in the Gevrey-class $G^0_{d,\mathbb{R}^d,M,c}$
Let $c, M > 0$, $d \in \mathbb{N}$, $0 < 73\varepsilon < cM$ and $\mathbf{a} \in \mathbb{R}^d$. Then for all open $\Omega \subseteq \mathbb{R}^d$, $\mathcal{F}_{\Omega,\varepsilon,c} \subseteq \mathcal{G}^0_{d,\Omega,M,c}$.

Proof. The proof is rather straightforward. Let $\alpha \in \mathbb{N}_0^d$. Then, we find by simply differentiating the test functions, for all $\mathbf{b} \in \{-1, 1\}^d$ and $\mathbf{x} \in \mathbb{R}^d$:

$$D^\alpha f_{\mathbf{b},\varepsilon,c,\mathbf{a}}(\mathbf{x}) = \sum_{j=1}^{d} \frac{73\varepsilon b_j}{cd} \cdot c^{|\alpha|} (D_{\alpha_j} \sin)(c(x_j - a_j)) \cdot \prod_{k \in [d]\setminus\{j\}} (D_{\alpha_k} \cos)(c(x_k - a_k)) \tag{5.1.1}$$

Taking absolute values on both sides yields:

$$|D^\alpha f_{\mathbf{b},\varepsilon,c,\mathbf{a}}(\mathbf{x})| \leq \sum_{j=1}^{d} \frac{73\varepsilon |b_j|}{d} \cdot c^{|\alpha|-1} = 73\varepsilon c^{|\alpha|-1} < Mc^{|\alpha|}$$

Hence, we find that $f_{\mathbf{b},\varepsilon,c,\mathbf{a}} \in \mathcal{G}^0_{d,\mathbb{R}^d,M,c}$, which by Theorem 4.1.2 implies that $f_{\mathbf{b},\varepsilon,c,\mathbf{a}}|_\Omega \in \mathcal{G}^0_{d,\Omega,M,c}$. This completes the proof. □

Furthermore, we can very conveniently express the derivative of the test functions at $\mathbf{a}$, as is shown in the next theorem:

Theorem 5.1.4: Gradient of test functions
Let $c, M > 0$, $d \in \mathbb{N}$, $0 < 73\varepsilon < cM$, $\mathbf{a} \in \mathbb{R}^d$ and $\mathbf{b} \in \{-1, 1\}^d$. Then, we find:

$$\nabla f_{\mathbf{b},\varepsilon,c,\mathbf{a}}(\mathbf{a}) = \frac{73\varepsilon}{d}\mathbf{b}$$

Proof. Follows from plugging in singleton sequences in Equation 5.1.1 and observing that most terms vanish at $\mathbf{a}$. □

When we estimate the gradient of a function accurately in ℓ^1-norm, say up to accuracy ε, then the average of the error in the individual coordinates can be at most ε/d. We make this idea rigorous in the following theorem, where we show that there is always a relatively large set of coordinates where the gradients of most of the test functions are well-approximated.

Theorem 5.1.5: Coordinate-wise performance
Let $c, M > 0$, $d \in \mathbb{N}$, $0 < 73\varepsilon < cM$, $G \subseteq \Omega \subseteq \mathbb{R}^d$ and $\mathbf{a} \in \Omega$. Suppose that $\mathcal{A}$ is an ε-precise ℓ^1-approximate quantum gradient estimation algorithm for $\mathcal{G}^0_{d,\Omega,M,c}$ in $\mathbf{a}$ on G with success probability 17/18. Then, there is a set $S \subseteq [d]$ such that $|S| \geq \frac{3}{4}d$, and for all $j \in S$, there is a set $\mathcal{G}_j \subseteq \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}$, with $|\mathcal{G}_j| \geq \frac{2}{3} \cdot 2^d$ and for all $f \in \mathcal{G}_j$:

$$\mathbb{P}\left[|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \leq \frac{72\varepsilon}{d}\right] \geq \frac{2}{3}$$

Proof. For every $f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}$, we know that the algorithm $\mathcal{A}$ satisfies the following condition:

$$\mathbb{P}\left[\|\mathcal{A}(f) - \nabla f(\mathbf{a})\|_1 \leq \varepsilon\right] \geq \frac{17}{18}$$

We call the event above $\mathcal{S}_f$, so $\mathbb{P}(\mathcal{S}_f) \geq 17/18$. Now, observe trivially that the following holds:

$$\mathbb{E}\left[\sum_{j=1}^{d} |\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \,\middle|\, \mathcal{S}_f\right] = \mathbb{E}\left[\,\|\mathcal{A}(f) - \nabla f(\mathbf{a})\|_1 \,|\, \|\mathcal{A}(f) - \nabla f(\mathbf{a})\|_1 \leq \varepsilon\right] \leq \varepsilon$$

Using the linearity of the expectation, we can take the sum out, and obtain the following relation:

$$\sum_{j=1}^{d} \mathbb{E}\left[\,|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \;\,|\, \mathcal{S}_f\right] \leq \varepsilon$$

The above relation holds for all $f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}$, and hence we find:

$$\frac{1}{2^d} \sum_{f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}} \sum_{j=1}^{d} \mathbb{E}\left[\,|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \;\,|\, \mathcal{S}_f\right] \leq \varepsilon$$

But of course we are only dealing with finite sums, and hence we can swap both summations, such that we obtain:

$$\sum_{j=1}^{d} \frac{1}{2^d} \sum_{f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}} \mathbb{E}\left[\,|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \;\,|\, \mathcal{S}_f\right] \leq \varepsilon$$

Now, as we are summing d non-negative real numbers in the summation above, we cannot have that more than $\frac{1}{4}d$ of these real numbers exceed $4\varepsilon/d$, because then the total would exceed ε. Hence, there exists a set $S \subseteq [d]$ such that $|S| \geq \frac{3}{4}d$ and such that for all $j \in S$, we have:

$$\frac{1}{2^d} \sum_{f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}} \mathbb{E}\left[\,|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \;\,|\, \mathcal{S}_f\right] \leq \frac{4\varepsilon}{d}$$

We now fix this $j \in S$. Next, we invoke Markov's inequality to obtain:

$$\frac{1}{2^d} \sum_{f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}} \mathbb{P}\left[|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \geq \frac{72\varepsilon}{d} \,\middle|\, \mathcal{S}_f\right] \leq \frac{1}{2^d} \sum_{f \in \mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}} \frac{\mathbb{E}\left[|\mathcal{A}(f)_j - \nabla f(\mathbf{a})_j| \;\,|\, \mathcal{S}_f\right]}{\frac{72\varepsilon}{d}} \leq \frac{\frac{4\varepsilon}{d}}{\frac{72\varepsilon}{d}} = \frac{1}{18}$$

Now, using Bayes' rule, we obtain:

$$\begin{aligned}
&\frac{1}{2^d}\sum_{f\in\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}}\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\right]\\
=\;&\frac{1}{2^d}\sum_{f\in\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}}\left[\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\middle|\mathcal{S}_f\right]\mathbb{P}(\mathcal{S}_f)+\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\middle|\mathcal{S}_f^c\right]\mathbb{P}(\mathcal{S}_f^c)\right]\\
\leq\;&\frac{1}{2^d}\sum_{f\in\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}}\left[\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\middle|\mathcal{S}_f\right]\cdot 1+1\cdot\frac{1}{18}\right]\\
=\;&\frac{1}{2^d}\sum_{f\in\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}}\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\middle|\mathcal{S}_f\right]+\frac{1}{18}\leq\frac{1}{18}+\frac{1}{18}=\frac{1}{9}
\end{aligned}$$

Next, we find that the average of 2^d non-negative real numbers must be smaller than or equal to $\frac{1}{9}$. This implies that at most $\frac{1}{3}\cdot 2^d$ of these non-negative real numbers can be greater than or equal to $\frac{1}{3}$, because otherwise the average would exceed $\frac{1}{9}$. Hence, we find that there exists a set $\mathcal{G}_j\subseteq\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}$ such that $|\mathcal{G}_j|\geq\frac{2}{3}\cdot 2^d$, and such that for all $f\in\mathcal{G}_j$, we have:

$$\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\geq\frac{72\varepsilon}{d}\right]\leq\frac{1}{3}$$

Thus, we obtain, for all such $f\in\mathcal{G}_j$:

$$\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\leq\frac{72\varepsilon}{d}\right]\geq\frac{2}{3}$$

This completes the proof. □

Now that we know that for many coordinates, there are many test functions whose gradients are well-approximated in that coordinate, we can use this theorem to show that there always exists one particular test function that is distinguished from many of its neighbors just by looking at the one coordinate in which its gradient differs. We will refer to this test function as the *central test function*. The following theorem makes this precise.

Theorem 5.1.6: Test function center selection
Let $c, M > 0$, $d\in\mathbb{N}$, $0<73\varepsilon<cM$, $G\subseteq\Omega\subseteq\mathbb{R}^d$ and $\mathbf{a}\in\Omega$. Suppose that $\mathcal{A}$ is an ε-precise ℓ^1-approximate quantum gradient estimation algorithm for $\mathcal{G}^0_{d,\Omega,M,c}$ in $\mathbf{a}$ on G with success probability 17/18. Then, there exists a vector $\mathbf{b}^*\in\{-1,1\}^d$ and a set $U\subseteq[d]$ such that $|U|\geq\frac{1}{4}d$, and such that for all $j\in U$:

$$\mathbb{P}\left[|\mathcal{A}(f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}|_\Omega})_j-\nabla f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{a})_j|\leq\frac{72\varepsilon}{d}\right]\geq\frac{2}{3}$$

and:

$$\mathbb{P}\left[|\mathcal{A}(f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}|_\Omega})_j-\nabla f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{a})_j|\leq\frac{72\varepsilon}{d}\right]\geq\frac{2}{3}\qquad\text{where}\qquad\mathbf{b}_{(\mathbf{j})}=\mathbf{b}^*-2b_j^*\mathbf{e}_j$$

i.e., where $\mathbf{b}_{(j)}\in\{-1,1\}^d$ is the vector that differs from $\mathbf{b}^*$ only in the jth position.

Proof. From Theorem 5.1.5, we know that there exists a set $S\subseteq[d]$ such that $|S|\geq\frac{3}{4}d$ and for all $j\in S$, there exists a $\mathcal{G}_j\subseteq\mathcal{F}_{\Omega,\varepsilon,c,\mathbf{a}}$ such that $|\mathcal{G}_j|\geq\frac{2}{3}\cdot 2^d$ and such that for all $f\in\mathcal{G}_j$:

$$\mathbb{P}\left[|\mathcal{A}(f)_j-\nabla f(\mathbf{a})_j|\leq\frac{72\varepsilon}{d}\right]\geq\frac{2}{3}$$

Let's develop some intuitive picture for what the result of Theorem 5.1.5 entails. To that end, picture the cube in d dimensions, whose vertices are located at $\{-1,1\}^d$. Next, suppose that every vertex itself is made of a small, solid cube. An example is shown in Figure 5.1.

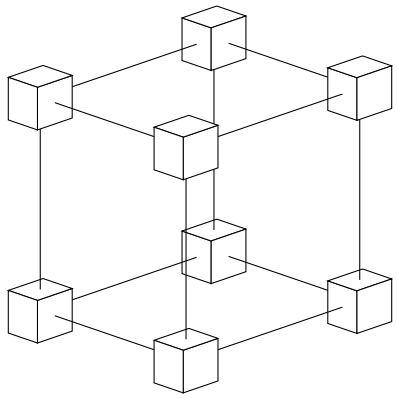


Figure 5.1: Sketch of the three-dimensional cube, whose vertices are smaller three-dimensional cubes.

For every $j \in [d]$, we easily observe that there are 2^{d-1} edges in the jth direction in this cube. Hence, for every $j \in S \subseteq [d]$, we have 2^{d-1} edges parallel to the $\mathbf{e}_j$ vector. Next, for every $f \in \mathcal{G}_j$, we can find a $\mathbf{b} \in \{-1,1\}^d$ such that $f = f_{\mathbf{b},\varepsilon,c,\mathbf{a}}$. In other words, we can associate a set of at least $\frac{2}{3} \cdot 2^d$ vertices in this cube to the set $\mathcal{G}_j$. For all these vertices, we mark the two faces of the smaller cube that faces the jth direction, (i.e., we can picture ourselves coloring these two faces blue). Note that a marked (i.e., blue) face indicates that the corresponding coordinate of the gradient of the function associated to this vertex is relatively well-approximated.

Next, we wonder how many edges are enclosed between two marked faces. For every $j \in S$, we find that the number of vertices that have marked faces in the jth direction is at least $\frac{2}{3} \cdot 2^d$. So, there are only $\frac{1}{3} \cdot 2^d$ vertices who do not have marked faces in the jth direction, which implies that there are at most $\frac{1}{3} \cdot 2^d$ edges in the jth direction which at least on one end connect to unmarked faces. Thus, we find that the number of edges in the jth direction which are enclosed between two marked faces is lower bounded by:

$$2^{d-1} - \frac{1}{3} \cdot 2^d = \left(\frac{1}{2} - \frac{1}{3}\right) 2^d = \frac{1}{6} \cdot 2^d = \frac{1}{3} \cdot 2^{d-1}$$

This holds for all $j \in S$. There are at least $\frac{3}{4} \cdot d$ such j, and hence the total number of edges that is between two marked faces in the cube is at least:

$$\frac{1}{3} \cdot 2^{d-1} \cdot \frac{3}{4} d = \frac{1}{4} d 2^{d-1}$$

As in total there are $d2^{d-1}$ edges in the cube, we have that at least a quarter of all the edges is enclosed between two marked vertices. But this implies that there is at least one vertex who connects to at least $\frac{1}{4}d$ of these edges. Denote this vertex by $\mathbf{b}^*$, and let $U \subseteq [d]$ be the set of coordinates that correspond to the directions of the edges that connect to $\mathbf{b}^*$ and are enclosed between two marked faces. Then, we find that $|U| \geq \frac{1}{4}d$, and we also have that the edges starting from $\mathbf{b}^*$ in the directions of U are enclosed between two marked vertices. Now, recall that if a face is marked, then this indicates that the corresponding vertex is well-approximated in this coordinate corresponding to the normal vector of this face. Writing out what this means in probabilities completes the proof. □

Next, we need one of the standard techniques within quantum computing to relate the above observation to query complexities of quantum algorithms. We state this result in the theorem below. The proof of the theorem, we postpone to Appendix C, specifically Theorem C.2.1.

Theorem 5.1.7: Hybrid method
Suppose that $O_0, O_1, \ldots, O_N$ are unitary operators acting on the n-qubit state space. Let $\mathcal{A}$ be a quantum algorithm whose input consists of one of the oracles O_j, with $j \in \{0, 1, \ldots, N\}$. For all $j \in \{1, \ldots, N\}$, let R_j, R_j^* be sets of outcomes of the algorithm $\mathcal{A}$ such that $R_j \cap R_j^* = \varnothing$. Suppose that for all $j \in \{1, \ldots, N\}$, we have:

$$\mathbb{P}(\mathcal{A}(O_j) \in R_j) \geq \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}(\mathcal{A}(O_0) \in R_j^*) \geq \frac{2}{3}$$

Then the worst case query complexity of $\mathcal{A}$ to the input oracle, denoted by $T_{\mathcal{A}}$, satisfies:

$$\frac{N}{9} \leq T_{\mathcal{A}}^2 \cdot \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^{n+1}} \\ \||\psi\rangle\| = 1}} \sum_{j=1}^{N} \|(O_j - O_0)\,|\psi\rangle\|^2$$

Finally, we can combine the previous two results to give a proof of Theorem 5.1.1, which is what we do below.

Proof of Theorem 5.1.1. Observe that according to Theorem 5.1.6, we can find a $\mathbf{b}^* \in \{-1, 1\}^d$ and a $U \subseteq [d]$ such that $|U| \geq \frac{1}{4}$, and for all $j \in U$, we have that:

$$\mathbb{P}\left[|\mathcal{A}(f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}|_\Omega})_j - \nabla f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{a})_j| \leq \frac{72\varepsilon}{d}\right] \geq \frac{2}{3}$$

and:

$$\mathbb{P}\left[|\mathcal{A}(f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}|_\Omega})_j - \nabla f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{a})_j| \leq \frac{72\varepsilon}{d}\right] \geq \frac{2}{3} \qquad \text{where} \qquad \mathbf{b}_{(\mathbf{j})} = \mathbf{b}^* - 2b_j^*\mathbf{e}_j$$

But now we can plug in the result from Theorem 5.1.4, and obtain that for all $j \in U$:

$$\mathbb{P}\left[\left|\mathcal{A}(f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}|_\Omega})_j - \frac{73\varepsilon b_j^*}{d}\right| \leq \frac{72\varepsilon}{d}\right] \geq \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}\left[\left|\mathcal{A}(f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}|_\Omega})_j - \frac{73\varepsilon (b_{(j)})_j}{d}\right| \leq \frac{72\varepsilon}{d}\right] \geq \frac{2}{3}$$

Next, we define the following sets:

$$R^* = \left\{\mathbf{g} \in \mathbb{R}^d : \left|g_j - \frac{73\varepsilon b_j^*}{d}\right| \leq \frac{72\varepsilon}{d}\right\} \qquad \text{and} \qquad R_{(j)} = \left\{\mathbf{g} \in \mathbb{R}^d : \left|g_j - \frac{73\varepsilon (b_{(j)})_j}{d}\right| \leq \frac{72\varepsilon}{d}\right\}$$

Suppose that $\mathbf{g} \in R^* \cap R_{(j)}$. Then, we find, using that $|b_j^* - (b_{(j)})_j| = 2$:

$$\frac{146\varepsilon}{d} = \frac{73\varepsilon}{d} \cdot |b_j^* - (b_{(j)})_j| = \left|\frac{73\varepsilon b_j^*}{d} - \frac{73\varepsilon (b_{(j)})_j}{d}\right| \leq \left|g_j - \frac{73\varepsilon b_j^*}{d}\right| + \left|g_j - \frac{73\varepsilon (b_{(j)})_j}{d}\right| \leq \frac{72\varepsilon}{d} + \frac{72\varepsilon}{d} = \frac{144\varepsilon}{d}$$

But this is a contradiction, and hence we find that $R^* \cap R_{(j)} = \varnothing$. Moreover, we have that:

$$\mathbb{P}(\mathcal{A}(f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}) \in R^*) \geq \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}(\mathcal{A}(f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,\mathbf{c},\mathbf{a}}) \in R_{(j)}) \geq \frac{2}{3}$$

Hence, we can now apply the hybrid method, Theorem 5.1.7. Let $n \in \mathbb{N}$ be the number of qubits that the phase oracles that form the input of the algorithm act on. Then, the controlled phase oracles act on $n+1$ qubits, and hence we find:

$$\frac{|U|}{9} \leq T_{\mathcal{A}}^2 \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^{n+1}} \\ \||\psi\rangle\| = 1}} \sum_{j \in U} \left\|\left(C(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}},G}) - C(O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}},G})\right)|\psi\rangle\right\|^2 \tag{5.1.2}$$

The left-hand side is clearly greater than or equal to $d/36$, as $|U| \geq d/4$. On the other hand, we can rewrite the supremum of the right-hand side as well, as follows. For all $|\psi\rangle \in \mathbb{C}^{2^{n+1}}$ with $\||\psi\rangle\| = 1$, we have:

$$\begin{aligned}
&\sum_{j\in U} \left\| \left(C(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}}) - C(O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}}) \right) |\psi\rangle \right\|^2 \\
= &\sum_{j\in U} \left\| \left[\sum_{\mathbf{x}\in G} (|0\rangle\langle 0| + |1\rangle\langle 1|) \otimes |\mathbf{x}\rangle\langle \mathbf{x}| \right] \left(C(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}}) - C(O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}}) \right) \left[\sum_{\mathbf{y}\in G} (|0\rangle\langle 0| + |1\rangle\langle 1|) \otimes |\mathbf{y}\rangle\langle \mathbf{y}| \right] |\psi\rangle \right\|^2 \\
= &\sum_{j\in U} \left\| \sum_{\mathbf{x}\in G} \sum_{\mathbf{y}\in G} (|0\rangle\langle 0| + |1\rangle\langle 1|) \otimes |\mathbf{x}\rangle\langle \mathbf{x}| \left(C(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}}) - C(O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}}) \right) (|0\rangle\langle 0| + |1\rangle\langle 1|) \otimes |\mathbf{y}\rangle\langle \mathbf{y}|\psi\rangle \right\|^2
\end{aligned}$$

As the controlled phase oracles act as the identity when the first qubit is in state $|0\rangle$, we find that their difference vanishes. Hence, we can disregard these terms in the expression above. Moreover, the phase oracles are diagonal operators, so we find that one of the sums drops out as well. What we are left with, is displayed below:

$$\begin{aligned}
\sum_{j\in U} \left\| \left(C(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}}) - C(O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}}) \right) |\psi\rangle \right\|^2 &= \sum_{j\in U} \left\| \sum_{\mathbf{x}\in G} \langle \mathbf{x}|\psi\rangle \left[\langle \mathbf{x}| \left(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}} - O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}} \right) |\mathbf{x}\rangle \right] |\mathbf{x}\rangle \right\|^2 \\
&= \sum_{j\in U} \left\| \sum_{\mathbf{x}\in G} \langle \mathbf{x}|\psi\rangle \left(e^{if_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x})} - e^{if_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})} \right) |\mathbf{x}\rangle \right\|^2
\end{aligned}$$

As the set $\{\mathbf{x} : x \in G\}$ forms an orthonormal set, this immediately rewrites to:

$$\begin{aligned}
\sum_{j\in U} \left\| \left(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}} - O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}} \right) |\psi\rangle \right\|^2 &= \sum_{j\in U} \sum_{\mathbf{x}\in G} |\langle \mathbf{x}|\psi\rangle|^2 \left| e^{if_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x})} - e^{if_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})} \right|^2 \\
&\leq \sum_{\mathbf{x}\in G} |\langle \mathbf{x}|\psi\rangle|^2 \sum_{j\in U} \left| f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,\mathbf{c},\mathbf{a}}(\mathbf{x}) \right|^2
\end{aligned}$$

where in the last inequality, we swapped the summations, and used that $|e^{i\phi} - e^{i\psi}| \leq |\phi - \psi|$, for all real ϕ and ψ. We have constructed our test functions to be very close to each other everywhere, and hence it makes sense to bound this quantity in the following manner:

$$\sum_{j\in U} \left\| \left(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}} - O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}} \right) |\psi\rangle \right\|^2 \leq \left[\sum_{\mathbf{x}\in G} |\langle \mathbf{x}|\psi\rangle|^2 \right] \cdot \left[\sup_{\mathbf{x}\in G} \sum_{j\in U} \left| f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,\mathbf{c},\mathbf{a}}(\mathbf{x}) \right|^2 \right]$$

The first factor is upper bounded by 1, because $|\psi\rangle$ is a unit vector. But now, the right-hand side no longer depends on $|\psi\rangle$, and hence:

$$\sup_{\substack{|\psi\rangle\in\mathbb{C}^{2^n} \\ \||\psi\rangle\|=1}} \sum_{j\in U} \left\| \left(O_{f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a},G}} - O_{f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a},G}} \right) |\psi\rangle \right\|^2 \leq \sup_{\mathbf{x}\in G} \sum_{j\in U} \left| f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,\mathbf{c},\mathbf{a}}(\mathbf{x}) \right|^2$$

Hence, we now obtain, plugging the last displayed equation into Equation 5.1.2:

$$dT_{\mathcal{A}}^2 \cdot \sup_{\mathbf{x}\in G} \sum_{j\in U} |f_{\mathbf{b}^*,\varepsilon,\mathbf{c},\mathbf{a}(\mathbf{x})}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})|^2 \geq 36 \tag{5.1.3}$$

So, it remains to find a good upper bound on the supremum to obtain a good lower bound on $T_{\mathcal{A}}$. To that end, we plug in the definition of the test functions, and observe that almost all terms vanish:

$$\sup_{\mathbf{x}\in G} \sum_{j\in U} |f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})|^2 \leq \sup_{\mathbf{x}\in G} \sum_{j\in U} \left(\frac{73\varepsilon |b_j^* - (b_{(j)})_j|}{cd} \right)^2 \sin^2(c(x_j - a_j)) \cdot \prod_{k\in[d]\setminus\{j\}} \cos^2(c(x_j - a_j))$$

Again, we use that for all $j \in U$, $|b^*_j - (b_{(j)})_j| = 2$. Moreover, we replace G by $\mathbb{R}^d$ and U by $[d]$, simply because we don't need to use these extra restrictions any longer. We obtain:

$$\sup_{\mathbf{x}\in G} \sum_{j\in U} |f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})|^2 \leq \sup_{\mathbf{x}\in\mathbb{R}^d} \sum_{j=1}^{d} \left(\frac{146\varepsilon}{cd}\right)^2 \sin^2(c(x_j - a_j)) \cdot \prod_{k\in[d]\setminus\{j\}} \cos^2(c(x_k - a_k))$$

Finally, observe that the sum on the right-hand side contains terms that are products with one $\sin^2$ factor, and many other $\cos^2$ factors. Rather counter-intuitively, we add many more factors, with all other combinations of $\cos^2$ and $\sin^2$ factors. Then, we obtain:

$$\begin{aligned}
\sup_{\mathbf{x}\in\mathbb{R}^d} \sum_{j\in U} |f_{\mathbf{b}^*,\varepsilon,c,\mathbf{a}}(\mathbf{x}) - f_{\mathbf{b}_{(\mathbf{j})},\varepsilon,c,\mathbf{a}}(\mathbf{x})|^2 &\leq \left(\frac{146\varepsilon}{cd}\right)^2 \sup_{\mathbf{x}\in\mathbb{R}^d} \sum_{A\subseteq[d]} \prod_{j\in A} \sin^2(c(x_j - a_j)) \prod_{k\in[d]\setminus A} \cos^2(c(x_k - a_k)) \\
&= \left(\frac{146\varepsilon}{cd}\right)^2 \cdot \sup_{\mathbf{x}\in\mathbb{R}^d} \prod_{j=1}^{d} \left(\sin^2(c(x_j - a_j)) + \cos^2(c(x_j - a_j))\right) \\
&= \left(\frac{146\varepsilon}{cd}\right)^2
\end{aligned}$$

Hence, plugging this into Equation 5.1.3, yields:

$$dT_{\mathcal{A}}^2 \cdot \left(\frac{146\varepsilon}{cd}\right)^2 \geq 36$$

And so, we find:

$$T_{\mathcal{A}} \geq \frac{cd^{\frac{3}{2}}}{876\varepsilon}$$

This completes the proof. □

In this section, we did a lot of hard work, to prove a lower bound on the query complexity of a quantum gradient estimation algorithm for one special case of parameters. In the next section, we will exploit this result to prove a similar lower bound for a much broader class of parameters.

5.2 Lower bound of more general cases

Now that we have proven a lower bound on the query complexity of quantum gradient estimation algorithms for one special case, in Theorem 5.1.1, we can use reduction arguments to prove similar lower bounds for more general cases as well. This is the aim of this section, and in particular the aim of the following theorem.

Theorem 5.2.1: Lower bound on query complexity of quantum gradient estimation algorithms
Let $c, M > 0$, $d \in \mathbb{N}$, $\sigma \geq 0$, $G \subseteq \Omega \subseteq \mathbb{R}^d$, $\mathbf{a} \in \Omega$, $p \in [1, \infty]$, $P \in (\frac{1}{2}, 1]$ and $0 < 146\varepsilon < cMd^{-1/p}$. Suppose that $\mathcal{A}$ is an ε-precise ℓ^p-approximate quantum gradient estimation algorithm for $\mathcal{G}^{\sigma}_{d,\Omega,M,c}$ in $\mathbf{a}$ on G with success probability P. Then, for every input controlled phase oracle $C(O_{f,G})$, where $f \in \mathcal{G}^{\sigma}_{d,\Omega,M,c}$, the query complexity of $C(O_{f,G})$, denoted by $T_{\mathcal{A}}(f)$, satisfies:

$$T_{\mathcal{A}}(f) \geq \min\left\{1, \frac{1}{2N}\right\} \frac{cd^{\frac{1}{2}+\frac{1}{p}}}{876\varepsilon} \qquad \text{where} \qquad N = \left\lceil \frac{18(1-P)}{(P-\frac{1}{2})^2} \right\rceil$$

Proof. We prove this in several cases. First of all, suppose that $p = 1$ and $P \geq 17/18$. Then, we know by Theorem 5.1.1 that the statement holds, and hence we are done.

Next, suppose that $p = 1$ and $\frac{1}{2} < P < 17/18$. Then, we have:

$$N = \left\lceil \frac{18(1-P)}{(P-\frac{1}{2})^2} \right\rceil \geq \frac{18(1-P)}{(P-\frac{1}{2})^2} > 0$$

Suppose that for some $f \in \mathcal{G}^{\sigma}_{d,\Omega,M,c}$, we want to determine the gradient in $\mathbf{a}$. We run algorithm $\mathcal{A}$ a total number of N times, and denote all the resulting vectors $\mathbf{g}_1, \ldots, \mathbf{g}_N$. We let X_j be the random variable that is 1 if $\|\mathbf{g}_j - \nabla f(\mathbf{a})\|_1 \leq \varepsilon$, and 0 otherwise. Then, we find $\mathbb{P}(X_j = 1) \geq P$, and hence $\mathbb{E}(X_j^2) = \mathbb{E}(X_j) = \mathbb{P}(X_j = 1) \geq P$.

Observe that all X_j's are independent random variables. Using the linearity of the expectation, we obtain:

$$\mathbb{E}\left[\sum_{j=1}^{N} X_j\right] = \sum_{j=1}^{N} \mathbb{E}[X_j] \geq NP$$

Moreover, we have:

$$\mathrm{Var}\left(\sum_{j=1}^{N} X_j\right) = \sum_{j=1}^{N} \mathrm{Var}(X_j) = \sum_{j=1}^{N} \left(\mathbb{E}(X_j^2) - E(X_j)^2\right) = \sum_{j=1}^{N} \mathbb{P}(X_j = 1) \cdot (1 - \mathbb{P}(X_j = 1)) \leq N(1-P)$$

Now, we can employ Chebyshev's inequality to obtain:

$$\begin{aligned}
\mathbb{P}\left[\sum_{j=1}^{N} X_j \leq \frac{N}{2}\right] &\leq \mathbb{P}\left[\left|\sum_{j=1}^{N} X_j - \mathbb{E}\left[\sum_{j=1}^{d} X_j\right]\right| \geq \mathbb{E}\left[\sum_{j=1}^{d} X_j\right] - \frac{N}{2}\right] \\
&\leq \mathbb{P}\left[\left|\sum_{j=1}^{N} X_j - \mathbb{E}\left[\sum_{j=1}^{d} X_j\right]\right| \geq \left[NP - \frac{N}{2}\right] \cdot \frac{\sqrt{\mathrm{Var}\left(\sum_{j=1}^{d} X_j\right)}}{\sqrt{\mathrm{Var}\left(\sum_{j=1}^{d} X_j\right)}}\right] \\
&\leq \frac{\mathrm{Var}\left(\sum_{j=1}^{d} X_j\right)}{N^2\left(P-\frac{1}{2}\right)^2} \leq \frac{N(1-P)}{N^2\left(P-\frac{1}{2}\right)^2} = \frac{1}{N} \cdot \frac{1-P}{\left(P-\frac{1}{2}\right)^2} \leq \frac{\left(P-\frac{1}{2}\right)^2}{18(1-P)} \cdot \frac{1-P}{\left(P-\frac{1}{2}\right)^2} = \frac{1}{18}
\end{aligned}$$

So, with probability at least 17/18, we obtain that more than half of the vectors $\mathbf{g}_1, \ldots, \mathbf{g}_N$ are within ℓ^1-distance ε of $\nabla f(\mathbf{a})$. If this is the case, even though we don't know this vector $\nabla f(\mathbf{a})$, we know that we must be able to find some vector $\tilde{\mathbf{g}} \in \mathbb{R}^d$ such that $\tilde{\mathbf{g}}$ is within ℓ^1-distance ε of more than half of the vectors $\mathbf{g}_1, \ldots, \mathbf{g}_N$, simply because $\nabla f(\mathbf{a})$ is one such vector $\tilde{\mathbf{g}}$. But then, by the pigeonhole principle, we find that there is at least one vector $\mathbf{g}_j$ that is within ℓ^1-distance ε of both $\nabla f(\mathbf{a})$ and $\tilde{\mathbf{g}}$, and hence we obtain:

$$\|\tilde{\mathbf{g}} - \nabla f(\mathbf{a})\|_1 \leq \|\tilde{\mathbf{g}} - \mathbf{g}_j\|_1 + \|\mathbf{g}_j - \nabla f(\mathbf{a})\|_1 \leq \varepsilon + \varepsilon = 2\varepsilon$$

So, we have now described a 2ε-precise ℓ^1-approximate quantum gradient estimation algorithm for $\mathcal{G}^{\sigma}_{d,\Omega,M,c}$ in $\mathbf{a}$ on G with success probability 17/18. By Theorem 5.1.1, we find that the query complexity of this algorithm must be at least $\frac{cd^{\frac{3}{2}}}{876 \cdot 2\varepsilon}$. But the query complexity of this algorithm was $NT_{\mathcal{A}}$, because we executed $\mathcal{A}$ a total number of N times. Hence, we find:

$$NT_{\mathcal{A}} \geq \frac{cd^{\frac{3}{2}}}{876 \cdot 2\varepsilon}$$

And so:

$$T_{\mathcal{A}} \geq \frac{1}{2N} \cdot \frac{cd^{\frac{3}{2}}}{876\varepsilon}$$

This completes the proof of the case where $p = 1$ and $P \in (1/2, 17/18)$.

Finally, we suppose that $p \in (1, \infty]$ and $P \in (\frac{1}{2}, 1]$. Then, for every $f \in \mathcal{G}^{\sigma}_{d,\Omega,M,c}$, the algorithm $\mathcal{A}$ outputs a vector $\mathbf{g} \in \mathbb{R}^d$ such that $\|\mathbf{g} - \nabla f(\mathbf{a})\|_p \leq \varepsilon$ with probability at least P. Hence, with probability at least P, it outputs a vector $\mathbf{g} \in \mathbb{R}^d$ that satisfies, using Hölder's inequality:

$$\|\mathbf{g} - \nabla f(\mathbf{a})\|_1 \leq d^{1-1/p} \|\mathbf{g} - \nabla f(\mathbf{a})\|_p \leq d^{1-1/p}\varepsilon$$

But then, we obtain, by what we have just proven for the case $p = 1$:

$$T_{\mathcal{A}} \geq \min\left\{1, \frac{1}{2N}\right\} \cdot \frac{cd^{\frac{3}{2}}}{876 \cdot d^{1-1/p}\varepsilon} = \min\left\{1, \frac{1}{2N}\right\} \cdot \frac{cd^{\frac{1}{2}+\frac{1}{p}}}{876\varepsilon}$$

This completes the proof. □

As a final note in this section, observe that the query complexity of this result is essentially equal to the query complexity portrayed in Equation 4.3.1 for $\sigma = \frac{1}{2}$. Hence, when $\sigma \in [0, \frac{1}{2}]$, we have proven essential optimality of Gilyén et al.'s quantum gradient estimation algorithm, for all values of $p \in [1, \infty]$.

In the next and last section, we will summarize these results, and put them into perspective.

5.3 Essential optimality of Gilyén et al.'s algorithm and further research

In the previous section, we have proven that for any ε-precise ℓ^p-approximate quantum gradient estimation algorithm for some class of functions with Gevrey-type at most 0, with success probability at least $\frac{1}{2}$, is at least:

$$\Omega\left(\frac{cd^{\frac{1}{2}+\frac{1}{p}}}{\varepsilon}\right)$$

However, in Section 4.3, we developed a ε-precise ℓ^p-approximate quantum gradient estimation algorithm for a class of functions with Gevrey-type at most $\sigma \in [1/2, 1]$, with success probability at least $2/3$, whose query complexity satisfies:

$$\tilde{O}\left(\frac{cd^{\sigma+\frac{1}{p}}}{\varepsilon}\right)$$

If we have a function whose Gevrey-type is smaller than $\frac{1}{2}$, then it is an element of the appropriate $\mathcal{G}^{\frac{1}{2}}$-class, though. Hence, the lower and upper bound match for all function classes $\mathcal{G}^{\sigma}$, where $\sigma \in [0, 1/2]$. This can also be viewed in Figure 5.2.

Whenever $\sigma > \frac{1}{2}$, however, the upper and lower bounds do not match. Hence, it is not obvious whether Gilyén et al.'s algorithm is optimal in this regime and the lower bound can be improved, or whether the lower bound can be improved to prove optimality of Gilyén et al.'s algorithm for this range of σ as well.

Finally, observe that using phase estimation, Algorithm 3.5.6, we can quite easily evaluate the function $f : \mathbb{R}^d \to \mathbb{R}$ at a point $\mathbf{x} \in \mathbb{R}^d$ up to ε precision with $\mathcal{O}(1/\varepsilon)$ queries to O_f. Hence, using $\tilde{\mathcal{O}}(d/\varepsilon)$ queries to O_f, we can approximate the gradient of f at some point in ℓ^{∞}-norm up to ε accuracy by employing a simple classical algorithm. This is especially relevant as in the field of quantum reinforcement learning, which we delve into in the next chapter, we naturally encounter functions whose Gevrey-type is 1. Hence, closing the optimality gap for $\sigma = 1$ is a very interesting topic of further research.

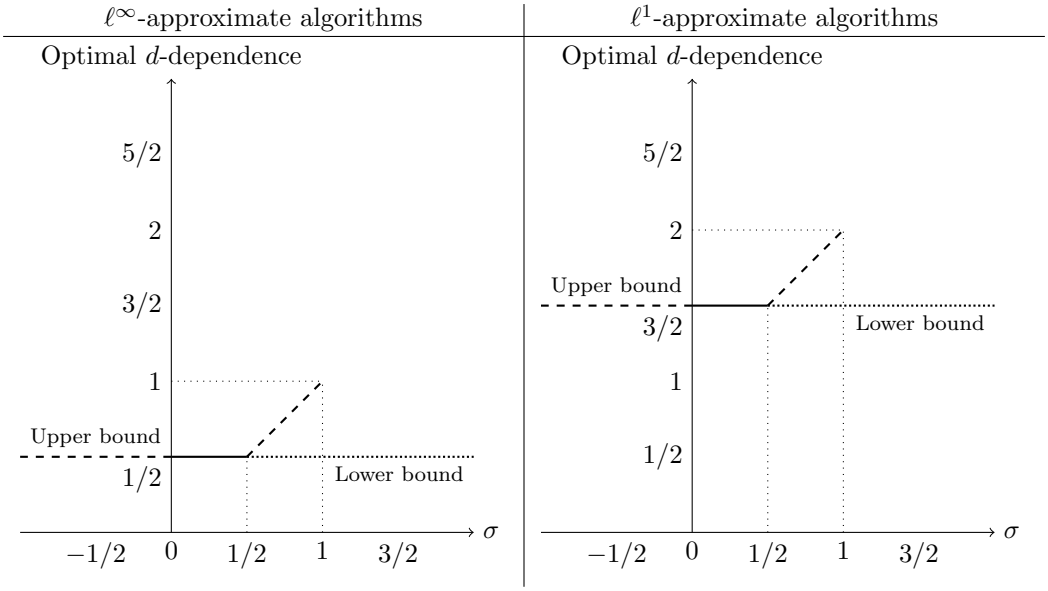


Figure 5.2: Graphical depiction of the results obtained in this chapter. On the horizontal axis, the Gevrey type σ is displayed. The dashed line represents the upper bound on the d-dependence of the optimal query complexity of a quantum gradient estimation algorithm for a class of functions whose Gevrey type is at most σ, where the value on the vertical axis should be interpreted as the power of d. Similarly, the dotted line represents the lower bound on this optimal query complexity. On the left, these bounds for ℓ^∞-approximate quantum gradient estimation algorithms are shown, and on the right for ℓ^1-approximate ones. We can see that in both cases, the upper and lower bounds coincide in the interval $\sigma \in [0, 1/2]$, and that in both cases, there still is an optimality gap for $\sigma \in (1/2, 1]$.

6 Quantum reinforcement learning

In Chapter 4, we have developed a quantum algorithm that is capable of estimating the gradient of a high-dimensional function, given a phase oracle of this function. Subsequently, in Chapter 5, we have proven that this algorithm is essentially optimal in the number of calls to this phase oracle, as long as the input functions satisfy certain smoothness conditions. In this final chapter, we will look at applications of this quantum gradient estimation algorithm. Specifically, we will look at how we can employ this algorithm to do reinforcement learning on a quantum computer.

It is important to note that the application elaborated upon in this chapter, namely quantum reinforcement learning, is not the only application of quantum gradient estimation algorithms. [GAW17] mention a couple of applications of their algorithm in their paper, but their method can be used to implement virtually any algorithm which numerically estimates the gradient on a quantum computer, most notably gradient-based optimization algorithms like gradient descent. However, the application in this chapter is in some aspects more interesting, as it involves more "quantum tricks" than simply using the gradient estimation algorithm as a subroutine in a gradient descent algorithm.

Quantum reinforcement learning is a very new field, but there are some papers by Dunjko et al. that discuss the subject. We explicitly mention [DTB16], [DB17] and [DLWT18]. First of all, these works explore the agent-environment paradigm (or AE paradigm for short) in some detail. We have this paradigm encoded in the oracles we assume to have access to.

Furthermore, these papers consider rewards that come from some discrete set, and the authors assume to have access to these rewards via binary oracles. We assume to have access to the rewards via phase oracles, and hence for our purposes, rewards do not have to come from a discrete set. It is instructive to note that their ideas can be transformed to our setting using the phase kickback trick.[1]

Moreover, these papers explore specific instances of reinforcement learning that could be sped up using quantum computing methods, whereas we investigate whether it is possible to devise quantum algorithms that speed up general reinforcement learning methods.

Another recent paper is from Montanaro [Mon15]. He essentially shows how one can perform function values from a probability oracle using the phase estimation algorithm, which is something we do as well in Algorithm 6.2.7.

To the best of our knowledge, combining the ideas of reinforcement learning with the results obtained by Gilyén et al. [GAW17], especially the interconvertibility between phase and probability oracles, is new. The core idea is to perform multiplication using probability oracles, and to perform addition using phase oracles, both in an analog way.

Section 6.1 will provide a short introduction into reinforcement learning. It will introduce the basic concepts of this relatively new research field in a concise manner. Afterwards, Section 6.2 will cover how we can perform quantum value evaluation using quantum computing. Then, we will show how the technique of quantum value evaluation can be used to perform quantum policy evaluation, in Section 6.3. Finally, we will use the quantum gradient estimation routine from Gilyén et al. in Section 6.4 to perform quantum policy optimization.

6.1 Introduction to reinforcement learning

In this section, we will review the basic concepts of reinforcement learning. All the relevant terminology from this upcoming field that we will need in subsequent sections will be introduced in this section. However, this

[1] This is the same technique that transforms Jordan's algorithm [Jor05] to Gilyen et al.'s setting [GAW17].

section is meant to be a concise introduction into the field. For a more complete introduction into the field, we refer to the reinforcement learning course taught by David Silver [Sil15], or the book that is commonly referred to as the *reinforcement learning bible*, which at the time of writing still is under development [SB18].

Within reinforcement learning, oftentimes a distinction is made between discrete and continuous state and action spaces. Before we introduce these concepts formally, we remark that in this chapter we will only consider the discrete case. Generalization to the continuous case probably is possible using similar methods, but this is left for another time.

Subsection 6.1.1 provides the definitions of the state and action spaces, and the state reward function. Next, in Subsection 6.1.2, we elaborate on how these state spaces, action spaces and reward functions combine into a Markov reward process. Furthermore, we outline how the notion of Markov reward processes can be generalized to Markov decision processes, and we also define policies and value functions there.

6.1.1 State spaces, action spaces and rewards

In this section, we will introduce the most basic concepts of reinforcement learning, namely the state and action spaces, and the reward function.

Throughout this section, we will use a running example to illustrate the definitions that we provide. To that end, suppose that we have a robot which moves around in a maze. Let's say that the maze consists of $N \times N$ squares, just like the one shown in Figure 6.1 where $N = 10$. The robot starts out at one entrance of the maze, at the square labeled S, and it can move by hopping from one square to an adjacent one, without passing through the walls of the maze, indicated by the solid lines. Its goal is to get to the final square, F, where it can exit the maze, in the fewest number of moves. To make things more interesting, we require the robot to first pass through the square labeled by X, though, before it can successfully exit the maze (we can think of the robot picking up a key located at X which it needs to open the final door).

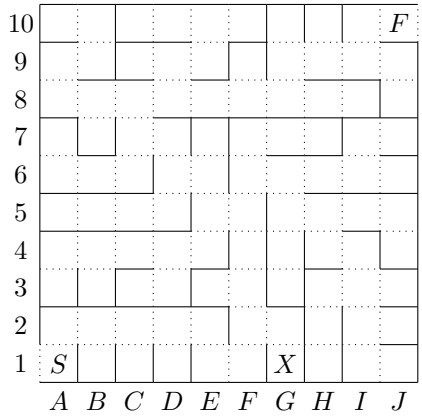


Figure 6.1: An arbitrary maze of size 10×10. A robot starts in the starting square, S, and its goal is to move through the maze, via the square X, to the finish square, F, in the fewest number of moves.

Whilst playing the game, the robot progresses through different *states*. These states are complete descriptions of the position of the game that the robot is playing. For example, we can define a *state* to be a tuple of two pieces of information, namely the position of the robot, and whether it has already passed through X or not. Hence, an example state could be $(D4, \text{No})$, indicating that the robot is currently at square $D4$, and that it has not passed through the square labeled by X yet.

The set of all states, we refer to as the *state space*. We could for example associate the following state space

to the game that we described above:

$$S = (((\{A, B, C, D, E, F, G, H, I, J\} \times \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}) \times \{\text{Yes}, \text{No}\}) \cup \{\text{Done}\}$$

Hence, all the elements of the state space S, we refer to as states. For future convenience, we included the state Done to indicate that the game is finished.

Now that we have built some intuition for the state space, let's formalize the idea below. In the formal picture, we will consider states to be very abstract mathematical objects, whose interpretation should come from context, such as the robot moving through the maze described above.

Definition 6.1.1: State space
A *state space* is a finite set,[a] whose members we refer to as *states*.

[a] In the field of reinforcement learning, it is also very common to have state spaces that have an infinite number of members. However, we will not consider such state spaces in this text.

Now, suppose that the robot is completely uncontrollable, i.e., it drives around randomly even if that causes it to run into walls. Suppose furthermore that it is in a cell which has two adjacent walls, and two adjacent cells where it can go to. Then, about a quarter of the time it will go to one of the adjacent cells, about a quarter of the time it will move to the other of the adjacent cells, and about half of the time it will run into one of the walls, causing it to not move to a different cell. Hence, it has a probability of $\frac{1}{2}$ to move to another cell, and a probability of $\frac{1}{2}$ to stay in the current cell.

These dynamics, we can concisely abbreviate in a matrix, where the rows are indexed by the current state of the robot, and the columns indicate the new states. Hence the dimensions of this matrix is $|S| \times |S|$, where $|S|$ denotes the number of elements in the state space S. The entries of this matrix indicate the probability that the robot moves from the state indexed by the row to the state indexed by the column. Such a matrix we refer to as a *state transition probability matrix*, which we formally define below.

Definition 6.1.2: State transition probability matrix
Let S be a state space, and $P : S \times S \to [0, 1]$, such that:

$$\forall s \in S, \qquad \sum_{s' \in S} P_{ss'} = 1$$

Then we refer to P as a *state transition probability matrix*.

So, if P is a state transition probability matrix, then $P_{ss'}$ indicates the probability that from state s, we transition to state s'.

Ideally, though, the robot does not move in an uncontrollable fashion, but rather it has some idea of what it is doing. In this case, whenever a robot is in a particular state, it can choose to perform one of four *actions*, i.e., to move either up, left, down or right. The set of all actions we refer to as the *action space*, i.e., the action space in this context is:

$$A = \{\text{Move up}, \text{Move left}, \text{Move down}, \text{Move right}\}$$

We can again formalize this idea:

Definition 6.1.3: Action space
An *action space* is a finite set,[a] whose members we refer to as *actions*.

[a] Again, it is perfectly legitimate to have infinite action spaces as well, but we will leave this for another time.

How these actions affect the state that the robot is in, is described by the *state-action probability matrix*. This is a matrix whose rows are indexed by all state-action pairs, i.e., the set $S \times A$. The columns, on the other hand, are indexed by the new states, and the entries again indicate the probability that the state

indexed by the row transitions into the state indexed by the column, when the action indexed by the row is performed.

Definition 6.1.4: State-action transition probability matrix
Let S be a state space and A be an action space. Let $P : S \times A \times S \to [0, 1]$, such that:

$$\forall (s, a) \in S \times A, \qquad \sum_{s' \in S} P_{sas'} = 1$$

Then, we refer to P as a *state-action transition probability matrix*.

So, when we start in state s and perform action a, $P_{sas'}$ denotes the probability that we end up in state s'.

At first sight, it might seem like the formal definitions provided here are not very well-suited to describe the maze problem introduced earlier. For example, it could happen that we pick an action that causes us to run into a wall, e.g., when we are in square $D1$ and choose the action Move down. We can however, just say that the state does not change when this happens, i.e., $P_{(D1,b),\text{Move down},(D1,b)} = 1$, where $b \in \{\text{Yes}, \text{No}\}$, hence this does not pose any difficulties.

Finally, now, we want to add some incentive for the robot to reach the final square as fast as possible. We do this by giving a reward every time the robot transitions to a new state. How big the reward is, is determined by the state reward function.

Definition 6.1.5: State reward function
Let S be a state space and $R : S \to \mathbb{R}$. Then we refer to R as a *state reward function*. Moreover, for all states $s \in S$, we refer to $R(s)$ as the *reward obtained upon exiting state s*.

In the example of the maze, we could give a reward of -1, every time the robot moves from one square to another, i.e., $R(s) = -1$ for all $s \in \{A, \ldots, J\} \times \{1, \ldots, 10\} \times \{\text{Yes}, \text{No}\}$ and $R(\text{Done}) = 0$. Then, the robot has an incentive to transition to the Done state as fast as possible, because it does not get any negative reward anymore whenever it is in the Done state.

Similarly, we can define a state-action reward function.

Definition 6.1.6: State-action reward function
Let S be a state space and A be an action space. Let $R : S \times A \to \mathbb{R}$. Then we refer to R as a *state-action reward function*. Moreover, for all $(s, a) \in S \times A$, we refer to $R(s, a)$ as the *reward obtained upon exiting state s after performing action a*.

This allows us to add incentive for taking certain actions as well.

6.1.2 Markov processes

We have introduced the state space, action space, and reward functions in the previous subsection. In this section, we will provide a way to abbreviate all these notions into a single mathematical object.

First of all, let's revisit the example in which the robot is uncontrollably moving through the maze. We can abbreviate all the defining properties of this example into one mathematical object, which we refer to as a *Markov reward process*.

Definition 6.1.7: Markov reward process
Let S be a state space, $R : S \to \mathbb{R}$ be a state reward function, $P : S \times S \to [0, 1]$ be a state transition probability matrix and $\gamma \in (0, 1)$. We say that (S, P, R, γ) is a *Markov reward process*, and we refer to γ as the *discount factor*.

Given the Markov reward process describing the robot randomly moving through the maze and the starting state $s_0 \in S$, we would like to define the expected reward that the robot will obtain throughout the process. This quantity, however, might not be finite as there could be an infinite number of steps still to come. This is why, instead, we introduce the expected *discounted* reward, which intuitively values the rewards obtained in the near future more than the rewards obtained in the distant future. We will refer to this expected discounted reward as the *value of the Markov reward process starting at* s_0. Its formal definition is given below.

Definition 6.1.8: Value of a Markov reward process
Let $M = (S, P, R, \gamma)$ be a Markov reward process. Let that $s_0 \in S$ be a state. We define the random variables $S_0, S_1, S_2, \ldots$ taking values in S such that for all $t \in \mathbb{N}$:

$$\mathbb{P}[S_0 = s] = \delta_{ss_0} \qquad \text{and} \qquad \mathbb{P}[S_t = s'] = \sum_{s \in S} P_{ss'} \mathbb{P}[S_{t-1} = s]$$

Then:

$$V_M(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

We refer to $V_M : S \to \mathbb{R}$ as the *value function of the Markov reward process* M, and this function evaluated at some state s is the *value of the Markov reward process* M *starting from* s.

So, in the example of the randomly moving robot, $V_M(s_0)$ indicates the reward that the robot on average obtains, when it wanders randomly through the maze.

On the other hand, we can also encode the action space and the state-action reward function into a single mathematical object. We refer to such objects as *Markov decision processes*. Their formal definition is provided below.

Definition 6.1.9: Markov decision process
Let S be a state space and A be an action space. Let $R : S \times A \to \mathbb{R}$ be a state-action reward function, and $P : S \times A \times S \to [0, 1]$ be a state-action transition probability matrix. Finally, let $\gamma \in (0, 1)$. Then we say that (S, A, P, R, γ) is a Markov decision process, and we refer to γ as the *discount factor*.

Whenever we have such a Markov decision process $M = (S, A, P, R, \gamma)$, we can define a function that, given a state $s \in S$ and an action $a \in A$, outputs the probability that when in state s, the action a is performed. Such a function, we call a *policy* for M.

Definition 6.1.10: Policy for a Markov decision process
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process. Let $\pi : S \times A \to [0, 1]$, satisfying:

$$\forall s \in S, \qquad \sum_{a \in A} \pi_{sa} = 1$$

Then, we say that π is a policy for M.

Given a policy for M, we can again define the expected discounted reward under the policy. This is formalized in the definition below.

Definition 6.1.11: Value of a Markov decision process
Let $M = (S, A, P, R, \gamma)$ be a Markov reward process and let π be a policy for M. Let $s_0 \in S$ be a state. We define the random variables $S_0, S_1, S_2, \ldots$ and $A_1, A_2, \ldots$, taking values in S and A, respectively, such that for all $t \in \mathbb{N}^a$:

$$\mathbb{P}[S_0 = s] = \delta_{ss_0}, \quad \mathbb{P}[S_t = s'] = \sum_{s \in S} \sum_{a \in A} P_{sas'} \mathbb{P}[A_{t-1} = a \wedge S_{t-1} = s] \quad \text{and} \quad \mathbb{P}[A_{t-1} = a | S_{t-1} = s] = \pi_{sa}$$

Then:

$$V_M^{(\pi)}(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\right]$$

We refer to $V_M^{(\pi)} : S \to \mathbb{R}$ as the *value function of M under π*. Specifically, we refer to $V_M^{(\pi)}(s_0)$ as the *value of M under π starting from s_0*.

[a] We use the convention that $\mathbb{N} = \{1, 2, \ldots, \}$.

The idea of reinforcement learning is, given a Markov decision process, to find the policy for it that maximizes the value of this Markov decision process under this policy. If we translate this to the example of the robot moving around in the maze, this boils down to finding the quickest route for the robot to exit the maze. In a similar manner, the framework provided in this section can be used to formulate a wide variety of problems in a similar way, i.e., to find the optimal policy for some given Markov decision process, where the specifics of the problem are somehow encoded into the Markov decision process. Hence, if we can find an algorithm that determines or approximates optimal policies, then we can use this framework to solve a wide variety of different problems. In the remainder of this chapter, we will investigate whether we can achieve this using quantum computing.

As a final note, observe that there is a very natural way to summarize a Markov decision process together with its policy into a Markov reward process. This, we do in the definition below:

Definition 6.1.12: Markov reward process generated by Markov decision process and policy
Let $M = (S, A, R, P, \gamma)$ be a Markov decision process and let π be a policy for M. We define the *state-action space*:

$$S' = S \times A$$

Now, observe that $R : S' \to \mathbb{R}$. Moreover, we define:

$$P^{(\pi)} : S \times A \to S \times A, \qquad P^{(\pi)}_{(sa),(s'a')} = P_{sas'} \pi_{s'a'}$$

which implies:

$$\forall (s, a) \in S \times A, \qquad \sum_{(s',a') \in S \times A} P^{(\pi)}_{(sa),(s'a')} = \sum_{s' \in S} P_{sas'} \sum_{a' \in A} \pi_{s'a'} = \sum_{s' \in S} P_{sas'} \cdot 1 = 1$$

So, $P^{(\pi)}$ is a state transition probability matrix with respect to the state space S' and hence $M^{(\pi)} = (S', R, P^{(\pi)}, \gamma)$ is a Markov reward process. We refer to $M^{(\pi)}$ as the *Markov reward process M under π*.

The following theorem motivates this embedding of the space of Markov decision processes into the space of Markov reward processes:

Theorem 6.1.13: Value of Markov reward processes under policies
Let $M = (S, A, R, P, \gamma)$ be a Markov decision process and π be a policy for M. Let $M^{(\pi)} = (S', R, P^{(\pi)}, \gamma)$ be the Markov reward process M under π. Then, we obtain, for all $s_0 \in S$:

$$V_M^{(\pi)}(s_0) = \sum_{a \in A} \pi_{s_0 a} V_{M^{(\pi)}}(s_0, a)$$

Note that the left-hand side is defined in Definition 6.1.11, and the right-hand side is defined in Definition 6.1.8.

Proof. Let $S_0, S_1, S_2, \ldots$ and $A_1, A_2, \ldots$ be the random variables taking values in S and A, respectively, as defined in Definition 6.1.11. Similarly, let $S_0^{(a)}, S_1^{(a)}, S_2^{(a)}, \ldots$ be random variables taking values in S', as defined in Definition 6.1.8 with starting state (s_0, a). We obtain:

$$V_M^{(\pi)}(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\right] = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[\mathbb{E}\left[R(S_t, A_t) \mid A_0\right]\right] = \sum_{t=0}^{\infty} \gamma^t \sum_{a \in A} \pi_{s_0 a} \mathbb{E}\left[R(S_t, A_t) \mid A_0 = a\right]$$

Similarly, we have:

$$\sum_{a \in A} \pi_{s_0 a} V_{M^\pi}(s_0, a) = \sum_{t=0}^{\infty} \gamma^t \sum_{a \in A} \pi_{s_0 a} \mathbb{E}(R(S_t^{(a)}))$$

So, it is sufficient to show that $\mathbb{P}\left[(S_t, A_t) = (s', a') \mid A_0 = a\right] = \mathbb{P}\left[S_t^{(a)} = (s', a')\right]$. To that end, observe that for $t = 0$, this result is trivial, because both sides are one if and only if $s' = s_0$ and $a' = a$. On the other hand, suppose that it holds for some $t \in \mathbb{N}$. Then, we find:

$$\begin{aligned}
& \mathbb{P}\left[(S_{t+1}, A_{t+1}) = (s', a') \mid A_0 = a\right] \\
= \; & \mathbb{P}\left[A_{t+1} = a' \mid S_{t+1} = s', A_0 = a\right] \cdot \mathbb{P}\left[S_{t+1} = s' \mid A_0 = a\right] \\
= \; & \pi_{s'a'} \cdot \sum_{(s'', a'') \in S \times A} \mathbb{P}\left[S_{t+1} = s' \mid S_t = s'', A_t = a'', A_0 = a\right] \mathbb{P}\left[S_t = s'', A_t = a'' \mid A_0 = a\right] \\
= \; & \pi_{s'a'} \cdot \sum_{(s'', a'') \in S \times A} P_{s''a''s'} \mathbb{P}\left[S_t^{(a)} = (s'', a'')\right] = \sum_{(s'', a'') \in S \times A} P^{(\pi)}_{(s''a''), (s'a')} \mathbb{P}\left[S_t^{(a)} = (s'', a'')\right] \\
= \; & \sum_{(s'', a'') \in S \times A} \mathbb{P}\left[S_{t+1}^{(a)} = (s', a') \mid S_t^{(a)} = (s'', a'')\right] \mathbb{P}\left[S_t^{(a)} = (s'', a'')\right] = \mathbb{P}\left[S_{t+1}^{(a)} = (s', a')\right]
\end{aligned}$$

And hence we complete the proof by induction. □

6.2 Quantum value evaluation

In the previous section, we have introduced the standard concepts in reinforcement learning, most notably the Markov reward and Markov decision processes. In this section, we will look at how we can approximate the value function of a Markov reward process. Then, in Section 6.3, we will see how we can use the theory developed in this section to approximate the value function of the Markov reward process generated by a Markov decision process under some policy. And finally, in Section 6.4, we will use this theory to develop a way to iteratively find better and better policies for some given Markov reward process.

In this section, though, we only look at Markov reward processes. In Subsection 6.2.1, we will consider how one approximates the value function of a Markov reward process in a classical manner, and then in Subsection 6.2.2, we will use some "quantum tricks" to speed up this process using a quantum computer. Finally, in Subsection 6.2.3, we will prove that the approach presented in this section is essentially optimal in the query complexity.

6.2.1 Classical Monte-Carlo methods

Let $M = (S, R, P, \gamma)$ be a Markov reward process, and $s_0 \in S$ be an arbitrary state. A well-studied question in the field of reinforcement learning is how we can estimate the value of M starting from s_0, i.e., how we can find an estimate of $V_M(s_0)$. Many methods exist, but the most notable ones are the Monte-Carlo method, the temporal difference method, and the $TD(\lambda)$-method [Sil15, SB18].

The Monte-Carlo method is the easiest of these methods. It randomly simulates paths through the state space, where the dynamics are governed by the state probability matrix. For all paths that are simulated, the cumulative discounted reward is calculated, and afterwards the average of all these sampled rewards provides an estimate of $V_M(s_0)$. Using this method, one always has to simulate the process through the state space until some kind of termination state has been reached, and then one looks all the way back to the starting node to update its estimate of the value function.

The temporal-difference method works in a different way. It keeps track of estimates of the value function evaluated in all states $s \in S$, i.e., $V_M(s)$. Next, it makes use of the observation that:

$$V_M(s) = R(s) + \sum_{s' \in S} P_{ss'} \gamma V_M(s')$$

And so one can use the estimates for $V_M(s')$ to update the estimate for $V_M(s)$. In this way, one updates the estimate for $V_M(s)$ by looking moving to the state s', looking back one step to figure out from which state it came, and then updating the estimate of the value function of this previous state.

Hence, summarizing, we see that Monte-Carlo methods always look back all the way to the starting node, whereas temporal-difference methods always look only one step back. $TD(\lambda)$-methods average the behavior of these two methods. $\lambda \in [0, 1]$ is a parameter that can be freely chosen, and the number of states it looks back to is roughly $1/(1 - \lambda)$. Hence, $TD(1)$ recovers Monte-Carlo learning, and $TD(0)$ recovers temporal-difference learning.

In general, $TD(\lambda)$-methods converge more quickly to a good estimate of $V_M(s_0)$ then Monte-Carlo methods. However, it is not clear how one could use $TD(\lambda)$-methods in a quantum computing setting. This would be a very interesting direction of research. For now, though, we will simply state the less-efficient Monte-Carlo algorithm here, simply because we can use it in the next subsection to develop an essentially optimal quantum algorithm in the subsection thereafter.

Algorithm 6.2.1: Monte-Carlo simulation
Description: This classical algorithm estimates the value function of a Markov reward process.
Input: Markov reward process $M = (S, P, R, \gamma)$, an initial state $s_0 \in S$ and an accuracy parameter $\varepsilon > 0$.
Derived constants:

1. $|R|_{\max} = \max_{s \in S} |R(s)|$.
2. $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.
3. $N = \left\lceil \frac{12|R|_{\max}^2}{\varepsilon^2(1-\gamma)^2} \right\rceil$.

Output: An approximation $\tilde{V}$ for $V_M(s_0)$, satisfying $|\tilde{V} - V_M(s_0)| \leq \varepsilon$.
Success probability: Lower bounded by 2/3.
Algorithm:

1. For $i = 1, \ldots, N$:
   (a) Simulate a path of length T through the state space, starting from s_0, where the dynamics are described by P. Denote the path by $s_0, s_1, s_2, \ldots, s_{T-1}$.
   (b) Calculate the cumulative discounted reward of this path:

$$r_i = \sum_{t=0}^{T-1} \gamma^t R(s_t)$$

2. Return the average of all r_i's:

$$\tilde{V} = \frac{1}{N} \sum_{i=1}^{N} r_i$$

Proof of the lower bound on the success probability. Let $S_0, S_1, \ldots, S_{T-1}$ be random variables whose dynamics are governed by P, i.e., which satisfy the following recurrence relation:

$$\mathbb{P}(S_0 = s) = \delta_{s s_0} \qquad \text{and} \qquad \forall t \in \mathbb{N}, \mathbb{P}(S_t = s) = \sum_{s' \in S} P_{s's} \mathbb{P}(S_{t-1} = s')$$

Then, we find:

$$\left| \sum_{t=0}^{T-1} \gamma^t R(S_t) \right| \leq \sum_{t=0}^{T-1} \gamma^t |R(S_t)| \leq \sum_{t=0}^{T-1} \gamma^t |R|_{\max} = \frac{|R|_{\max}}{1-\gamma}$$

So, using Popoviciu's inequality of variances [Pop35]:

$$\operatorname{Var}[r_i] = \operatorname{Var}\left[\sum_{t=0}^{T-1} \gamma^t R(S_t) \right] \leq \frac{1}{4} \left(\frac{2|R|_{\max}}{1-\gamma} \right)^2 = \frac{|R|_{\max}^2}{(1-\gamma)^2}$$

Moreover, all the individual runs of step 1 are independent, and hence:

$$\operatorname{Var}\left[\tilde{V}\right] = \frac{1}{N^2} \sum_{i=1}^{N} \operatorname{Var}[r_i] \leq \frac{|R|_{\max}^2}{N(1-\gamma)^2}$$

Thus, using Chebyshev's inequality, we obtain:

$$\mathbb{P}\left[\left| \tilde{V} - \mathbb{E}\left[\tilde{V}\right] \right| \geq \frac{\varepsilon}{2} \right] \leq \frac{4 \operatorname{Var}[\tilde{V}]}{\varepsilon^2} \leq \frac{4|R|_{\max}^2}{\varepsilon^2 N (1-\gamma)^2} \leq \frac{4}{12} = \frac{1}{3}$$

Moreover, we have:

$$\left|\mathbb{E}\left[\tilde{V}\right] - V_M(s_0)\right| = |\mathbb{E}[r_i] - V_M(s_0)| = \left|\mathbb{E}\left[\sum_{t=0}^{T-1}\gamma^t R(S_t)\right] - \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t)\right]\right| = \left|\sum_{t=T}^{\infty}\gamma^t \mathbb{E}[R(S_t)]\right|$$
$$\leq \frac{\gamma^T |R|_{\max}}{1-\gamma} \leq \frac{\gamma^{\frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)}}|R|_{\max}}{1-\gamma} = \frac{\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}|R|_{\max}}{1-\gamma} = \frac{\varepsilon}{2}$$

Hence, we obtain:

$$\mathbb{P}\left[\left|\tilde{V} - V_M(s_0)\right| \geq \varepsilon\right] \leq \mathbb{P}\left[\left|\tilde{V} - \mathbb{E}\left[\tilde{V}\right]\right| \geq \frac{\varepsilon}{2}\right] \leq \frac{1}{3}$$

Thus, the probability of the estimate being ε-precise is at least $2/3$, completing the proof. □

This completes our discussion of classical algorithms that estimate the value function of Markov reward processes. In the next subsection, we will investigate how we can use Algorithm 6.2.1 to devise a quantum algorithm that performs the same task.

6.2.2 Quantum speed-ups

In this subsection, we will develop a quantum algorithm that estimates the value function of a Markov reward process. To that end, we again assume that we have a Markov reward process $M = (S, R, P, \gamma)$ and an initial state $s_0 \in S$. The algorithm will be similar to Algorithm 6.2.1, but translated into the quantum computing framework.

The first step that we take in order to translate the problem to the quantum computing framework, is to encode the state space S of the Markov reward process into the state space of the quantum computer, $\mathbb{C}^{2^n}$, where $n \in \mathbb{N}$ is the number of qubits. To that end, we must choose a set of orthonormal states $\{|s\rangle \in \mathbb{C}^{2^n} : s \in S\}$. Note that this implies that we have to use $n \geq \lceil \log(|S|) \rceil$ qubits to encode the states of the Markov reward process as quantum states. How exactly one chooses the set of states is heavily problem-dependent, and hence we will not specify how this is done in the general setting presented in this subsection.

Next, we want to be able to access the dynamics of the Markov reward process in the quantum computing setting. This, we will do by assuming that we have access to a state transition oracle, $\mathcal{P}$, which acts on $2n$ qubits and has the following properties:

$$\forall s \in S, \mathcal{P} : |s\rangle \otimes |0\rangle^{\otimes n} \mapsto |s\rangle \otimes \left[\sum_{s' \in S} \sqrt{P_{ss'}}\,|s'\rangle\right] \tag{6.2.1}$$

The square-root is introduced to make sure the resulting state is always properly normalized.

Similarly, we want to be able to access the reward function. We assume here that we have access to the reward function by means of a phase oracle, like the one introduced in Definition 4.1.7. Hence, we assume that we have access to a controlled reward oracle $C(\mathcal{R})$, where $\mathcal{R}$ acts on n qubits and has the following action:

$$\forall s \in S, \mathcal{R} : |s\rangle \mapsto e^{i\tilde{R}(s)}\,|s\rangle \qquad \text{where} \qquad \tilde{R}(s) = \frac{(1-\gamma)R(s)}{2|R|_{\max}} \tag{6.2.2}$$

With these assumptions, we can construct an algorithm that approximates the value of the Markov decision process starting from s_0, i.e., $V_M(s_0)$. Before we do this, we construct a necessary ingredient, namely a subroutine which converts a phase oracle into a probability oracle. This routine is based on appendix B of [GAW17].

First of all, we will approximate the function $x \mapsto \arcsin(x)$ by a polynomial. This is done in the following theorem.

Theorem 6.2.2: Approximations of $\arcsin(x)$
Let $\varepsilon > 0$, $N = \lceil \log(\frac{1}{\varepsilon}) \rceil$ and let $(b_n)_{n=0}^{\infty}$ be as in Theorem 4.2.4. Then we find for all $x \in [-\frac{1}{2}, \frac{1}{2}]$:

$$\left| \frac{1}{2} \arcsin(x) - \frac{1}{2} \sum_{n=0}^{N} \frac{b_n}{n+1} x^{n+1} \right| \leq \frac{\varepsilon}{2}$$

Furthermore, we have for all $x \in [-1, 1]$:

$$\left| \frac{1}{2} \sum_{n=0}^{N} \frac{b_n}{n+1} x^{n+1} \right| \leq 1$$

Proof. Let $x \in [-\frac{1}{2}, \frac{1}{2}]$. From Lemma 4.2.6, we know that:

$$\frac{1}{\sqrt{1-x^2}} = \sum_{n=0}^{\infty} b_n x^n$$

Hence, integrating on both sides and observing that $\arcsin(0) = 0$ yields:

$$\arcsin(x) = \int_0^x \frac{1}{\sqrt{1-x^2}} \, \mathrm{d}x = \sum_{n=0}^{\infty} \frac{b_n}{n+1} x^{n+1}$$

So, we find, by Taylor's theorem and Lemma 4.2.5:

$$\left| \arcsin(x) - \sum_{n=0}^{N} \frac{b_n}{n+1} x^{n+1} \right| \leq \sum_{n=N+1}^{\infty} \frac{|b_n| \cdot |x|^{n+1}}{n+1} \leq \sum_{n=N+1}^{\infty} |x|^{n+1} = \frac{|x|^{N+1}}{1-|x|} \leq \frac{|x|}{1-|x|} \cdot \left(\frac{1}{2}\right)^N \leq \varepsilon$$

which completes the proof of the first claim of the theorem. For the second claim, we observe that for all odd n, we have that $b_n = 0$, and hence, using Lemma 4.2.5:

$$\left| \sum_{n=0}^{N} \frac{b_n}{n+1} x^{n+1} \right| \leq \sum_{n=0}^{\infty} \frac{b_n}{n+1} |x|^{n+1} \leq 1 + \frac{1}{2} \sum_{n=2}^{\infty} \frac{1}{n\sqrt{n}} \leq 1 + \frac{1}{2} \int_1^{\infty} \frac{1}{x^{\frac{3}{2}}} \, \mathrm{d}x = 1 + \frac{1}{2} \left[-\frac{2}{\sqrt{x}} \right]_1^{\infty} = 2$$

Dividing by 2 on both sides yields the desired result, completing the proof. □

Next, we will show how we can approximate the function $x \mapsto \sqrt{\frac{1}{2} + \frac{1}{4}x}$ with a polynomial. This is done in the following theorem.

Theorem 6.2.3: Approximation of $x \mapsto \sqrt{\frac{1}{2} + \frac{1}{4}x}$
Let $\varepsilon > 0$ and $N = \lceil \log(\frac{1}{\varepsilon}) \rceil$. Define $(c_n)_{n=0}^{\infty}$ in $\mathbb{R}$, as follows:

$$\forall n \in \mathbb{N}_0, \qquad c_n = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } n = 0 \\ \frac{(-1)^{n-1}(2(n-1))!}{2\sqrt{2} \cdot (2^{n-1}(n-1)!)^2 \cdot n 2^n}, & \text{otherwise} \end{cases}$$

Then, we find, for all $x \in [-1, 1]$:

$$\left| \sqrt{\frac{1}{2} + \frac{1}{4}x} - \sum_{n=0}^{N} c_n x^n \right| \leq \frac{\varepsilon}{4}$$

Moreover, we find that for all $x \in [-1, 1]$:

$$\left| \sum_{n=0}^{N} c_n x^n \right| \leq 1$$

Proof. Recall from Equation 4.2.1 that we have:

$$\frac{1}{\sqrt{1+x}} = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(2^n n!)^2} x^n$$

Moreover, we found that this series converges for all $x \in (-1, 1)$. Now, we can integrate on both sides, which yields:

$$2\sqrt{1+x} = 2 + \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(2^n n!)^2 (n+1)} x^{n+1}$$

From complex analysis, it is known that this power series has the same radius of convergence as its derivative, which implies that it converges for all $x \in (-1, 1)$. Moreover, by dividing by $2\sqrt{2}$, we obtain:

$$\sqrt{\frac{1}{2} + \frac{1}{4}x} = \frac{1}{2\sqrt{2}} \cdot 2\sqrt{1 + \frac{1}{2}x} = \frac{1}{\sqrt{2}} + \frac{1}{2\sqrt{2}} \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{(2^n n!)^2 (n+1) \cdot 2^{n+1}} x^{n+1} = \sum_{n=0}^{\infty} c_n x^n$$

And hence this converges for all $x \in (-2, 2) \supseteq [-\frac{3}{5}, \frac{3}{5}]$. Thus, we find, for all $x \in [-1, 1]$:

$$\begin{aligned} \left| \sqrt{\frac{1}{2} + \frac{1}{4}x} - \sum_{n=0}^{N} c_n x^n \right| &\leq \sum_{n=N+1}^{\infty} |c_n| |x|^n \leq \sum_{n=N+1}^{\infty} \frac{(2(n-1))!}{2\sqrt{2} \cdot (2^{n-1}(n-1)!)^2 \cdot n 2^n} \\ &= \frac{1}{2\sqrt{2}} \sum_{n=N+1}^{\infty} \frac{b_{4(n-1)}}{n} \cdot \left(\frac{1}{2}\right)^n \leq \frac{1}{4} \sum_{n=N+1}^{\infty} \frac{1}{n\sqrt{2(n-1)}} \left(\frac{1}{2}\right)^n \leq \frac{1}{4} \sum_{n=N+1}^{\infty} \left(\frac{1}{2}\right)^n \\ &= \frac{1}{4} \left(\frac{1}{2}\right)^N \leq \frac{\varepsilon}{4} \end{aligned}$$

Finally, if $\varepsilon < 4 - 2\sqrt{3}$, we have for all $x \in [-1, 1]$:

$$\left| \sum_{n=0}^{N} c_n x^n \right| \leq \left| \sum_{n=0}^{N} c_n x^n - \sqrt{\frac{1}{2} + \frac{1}{4}x} \right| + \left| \sqrt{\frac{1}{2} + \frac{1}{4}x} \right| \leq \frac{\varepsilon}{4} + \frac{\sqrt{3}}{2} \leq \frac{4 - 2\sqrt{3} + 2\sqrt{3}}{4} = 1$$

Otherwise, if $\varepsilon \geq 4 - 2\sqrt{3} \geq \frac{1}{2}$, then

$$N = \left\lceil \log\left(\frac{1}{\varepsilon}\right) \right\rceil \leq \log\left(\frac{1}{\varepsilon}\right) + 1 \leq \log(2) + 1 = 1 + 1 = 2$$

But then we have, for all $x \in [-1, 1]$:

$$\left|\sum_{n=0}^{N} c_n x^n\right| \leq \frac{1}{\sqrt{2}} + \frac{|x|}{4\sqrt{2}} + \frac{|x|^2}{32\sqrt{2}} \leq \frac{41}{32\sqrt{2}} < 1$$

This completes the proof. □

Finally, now, we show how we can merge both approximations in the above theorems to approximate the function $x \mapsto \sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)}$.

Theorem 6.2.4: Approximation of $x \mapsto \sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)}$
Let $\varepsilon > 0$ and $N = \lceil \log(\frac{1}{\varepsilon}) \rceil$. Let $(b_n)_{n=0}^{\infty}$ and $(c_n)_{n=0}^{\infty}$ be as defined as in Theorem 6.2.2 and Theorem 6.2.3, respectively. Define $(d_n)_{n=0}^{N(N+1)}$ as follows:

$$\sum_{n=0}^{N(N+1)} d_n x^n = \sum_{n=0}^{N} c_n \left(\sum_{k=0}^{N} \frac{b_k}{k+1} x^{k+1}\right)^n$$

Then, we find, for all $x \in [-\frac{1}{2}, \frac{1}{2}]$:

$$\left|\sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)} - \sum_{n=0}^{N(N+1)} d_n x^n\right| \leq \frac{\varepsilon}{2}$$

And moreover, we find for all $x \in [-1, 1]$:

$$\left|\sum_{n=0}^{N(N+1)} d_n x^n\right| \leq 1$$

Proof. From Theorem 6.2.2, observe that for all $x \in [-\frac{1}{2}, \frac{1}{2}]$, we have:

$$\left|\arcsin(x) - \sum_{k=0}^{N} \frac{b_k}{k+1} x^{k+1}\right| \leq \frac{\varepsilon}{4}$$

Moreover, observe, by the mean-value theorem, that for all $-1 \leq x < y \leq 1$, there is a $\xi \in (x, y) \subseteq [-1, 1]$ such that:

$$\left|\sqrt{\frac{1}{2} + \frac{1}{4}x} - \sqrt{\frac{1}{2} + \frac{1}{4}y}\right| = \left|\frac{1}{8\sqrt{\frac{1}{2} + \frac{1}{4}\xi}}\right| |x - y| \leq \frac{|x-y|}{8 \cdot \sqrt{\frac{1}{4}}} = \frac{|x-y|}{4}$$

Moreover, let $x \in [-\frac{1}{2}, \frac{1}{2}]$ arbitrarily. We have $|\arcsin(x)| \leq \frac{\pi}{6} < 1$, and by Theorem 6.2.2, we also have $\left|\sum_{k=0}^{N} \frac{b_k}{k+1} x^{k+1}\right| \leq 1$. We abbreviate:

$$a(x) = \sum_{k=0}^{N} \frac{b_k}{k+1} x^{k+1}$$

and hence $a(x) \in [-1, 1]$. Now, using Theorem 6.2.3, we find:

$$\left|\sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)} - \sum_{n=0}^{N(N+1)} d_n x^n\right| \leq \left|\sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)} - \sqrt{\frac{1}{2} + \frac{1}{4}a(x)}\right| + \left|\sqrt{\frac{1}{2} + \frac{1}{4}a(x)} - \sum_{n=0}^{N} c_n (a(x))^n\right|$$
$$\leq \frac{|\arcsin(x) - a(x)|}{4} + \frac{\varepsilon}{4} \leq \frac{\varepsilon}{8} + \frac{\varepsilon}{4} < \frac{\varepsilon}{2}$$

Finally, observe that as $a(x) \in [-1, 1]$, we have, according to Theorem 6.2.3:

$$\left|\sum_{n=0}^{N(N+1)} d_n x^n\right| = \left|\sum_{n=0}^{N} c_n (a(x))^n\right| \leq 1$$

This completes the proof. □

Now, finally, as we have built a polynomial approximation of the function $x \mapsto \sqrt{\frac{1}{2} + \frac{1}{4}\arcsin(x)}$, we can use it to construct a probability oracle. Here, we use techniques from Section 4.2.

Circuit 6.2.5: Probability oracle from a phase oracle
Description: This circuit constructs a probability oracle from a phase oracle. In other words, given a phase oracle O_f, where $\|f\|_\infty \leq \frac{1}{2}$:

$$O_f : |x\rangle \mapsto e^{if(x)} |x\rangle$$

this circuit implements a probability oracle Q_f approximately, i.e., with norm error at most $3\delta/2$:

$$Q_f : |x\rangle \otimes |0\rangle^{\otimes 3} \mapsto |x\rangle \otimes (\sqrt{f(x)}\,|0\rangle^{\otimes 3} + \sqrt{1-f(x)}\,|\psi(x)\rangle\,|1\rangle)$$

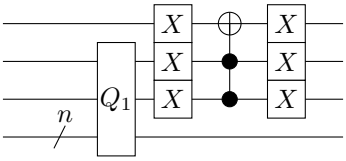
where $|\psi(x)\rangle$ is some arbitrary 2-qubit state, dependent on x.
Parameters: $\delta > 0$ is the accuracy.
Derived constants: $N = \lceil \log(\frac{1}{\delta}) \rceil$.
Oracle circuit: A phase oracle O_f acting on n qubits.
Number of qubits: $n + 3$.
Oracle query complexity: O_f is queried the following number of times:

$$4N(N+1) \leq 4\left(\log\left(\frac{1}{\delta}\right) + 1\right)\left(\log\left(\frac{1}{\delta}\right) + 2\right)$$

Circuit: We take the polynomial P, where $(d_n)_{n=0}^{N(N+1)}$ is as in Theorem 6.2.4, as follows:

$$P(x) = \sum_{n=0}^{N(N+1)} d_n x^n$$

and let $Q(x)$ and $R(x)$ be its even and odd parts, respectively. We first of all build the circuit Q_1 acting on n qubits, which implements this polynomial using Circuit 4.2.16, where we omit the S-gate such that we implement $Q(x) + R(x)$ instead of $Q(x) + iR(x)$. We can construct implementing phase vectors for Q and R using Theorem 4.2.14 and an implementation pair for both, which in turn are found using Theorem 4.2.11. The circuit $C(R(U))$ that is required by Circuit 4.2.15 is provided in Circuit 4.2.17. Next, we build the circuit Q_2, acting on $n + 3$ qubits, as follows:



Note that Q_2 performs the following mapping, up to error $\frac{\delta}{2}$ in norm:

$$Q_2 : |0\rangle^{\otimes 3} \otimes |x\rangle \mapsto \left(\frac{1}{2}\sqrt{f(x)}\,|0\rangle^{\otimes 3} + \frac{1}{2}\sqrt{4-f(x)}\,|1\rangle\,|\phi(x)\rangle\right)|x\rangle$$

where $|\phi(x)\rangle$ is some arbitrary 2-qubit state. Finally, we can simply use one step of amplitude amplification, Circuit 3.5.9, where Q_2 is the setup circuit, and the reflection circuit is XZX, acting on the first qubit. During this process, Q_2 is executed 3 times, and hence the total error is no more than $3\delta/2$ in norm, as can be seen from Lemma B.2. Finally, moving the first three qubits to the end yields the desired mapping Q_f.

Now that we have shown how we can construct a probability oracle from a phase oracle, we can show how we can approximate the value of a Markov reward process on a quantum computer. First of all, we will write down a quantum circuit that we will need in the algorithm, and afterwards we will present the algorithm itself.

Circuit 6.2.6: Markov reward process simulation
Description: Given a Markov reward process $M = (S, P, R, \gamma)$ and an initial state $s_0 \in S$, prepares a state where an estimate of $\sqrt{V_M(s_0)}$ is encoded in one of the amplitudes.
Parameters:

1. An orthogonal set of n-qubit states $\{|s\rangle : s \in S\}$.
2. An absolute bound on the reward function, $|R|_{\max} > 0$.
3. A discount factor $\gamma \in [0, 1)$.
4. An accuracy parameter $\varepsilon > 0$.
5. Circuit 6.2.5's accuracy parameter $\delta > 0$.

Derived constants:

1. $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.
2. $\delta' = \frac{\delta}{48T(\log(1/\delta)+1)(\log(1/\delta)+2)}$.

Oracle circuits:

1. A quantum circuit $\mathcal{P}$ that allows us to access the state transition probability matrix:

$$\forall s \in S, \mathcal{P} : |s\rangle \otimes |0\rangle^{\otimes n} \mapsto |s\rangle \otimes \sum_{s' \in S} \sqrt{P_{ss'}} \, |s'\rangle$$

2. A controlled reward oracle $C(\mathcal{R})$, where $\mathcal{R}$ has the following action:

$$\forall s \in S, \mathcal{R} : |s\rangle \mapsto e^{i\tilde{R}(s)} |s\rangle \qquad \text{where} \qquad \tilde{R}(s) = \frac{(1-\gamma)R(s)}{2|R|_{\max}}$$

Number of qubits: $Tn + 3$, not taking into account auxiliary qubits needed by $\mathcal{P}$ or $C(\mathcal{R})$.
Oracle query complexity: The number of queries to $\mathcal{P}$ is $T - 1$, and the number of queries to $C(\mathcal{R})$ is $\mathcal{O}(T \log(\frac{1}{\delta})^2)$.
Circuit:

1. Apply a circuit to the first register of n qubits, such that $|0\rangle^{\otimes n} \mapsto |s_0\rangle$.
2. Apply $\mathcal{P}$ to the first two registers of n qubits, then to the second and third register of n qubits, then to the third and fourth, and so on until it is applied to the $(T-2)$nd and $(T-1)$st.
3. Define the following circuit to be $\overline{\mathcal{R}}$, acting on all of the Tn qubits:
   (a) Apply $\mathcal{R}$ to the first register, then apply $\mathcal{R}^\gamma$ to the second, then $\mathcal{R}^{\gamma^2}$ to the third, etc. These $\mathcal{R}^{\gamma^t}$ are fractional phase oracles that can be constructed using Circuit 4.2.18. Use δ' as the accuracy parameter for the fractional phase query circuit.

   Note that $\overline{R}$ is a phase-oracle, implementing the function $(s_0, s_1, \ldots, s_{T-1}) \mapsto \sum_{t=0}^{T-1} \gamma^t \tilde{R}(s_t)$. In this step, though, we implement this function as a probability oracle using Circuit 6.2.5 with accuracy parameter δ, where we use the final qubit as the target qubit of the probability oracle.

Suppose that we start with the state $|0\rangle^{\otimes(Tn+1)}$, and apply the circuit above. Then, after step 1, we obtain the following state:

$$|s_0\rangle \otimes |0\rangle^{\otimes((T-1)n+3)}$$

Now, after applying $\mathcal{P}$ to the first and second register of n qubits, we obtain:

$$\left[\sum_{s_1 \in S} \sqrt{P_{s_0 s_1}} \, |s_0\rangle \, |s_1\rangle \right] \otimes |0\rangle^{\otimes((T-2)n+3)}$$

Next, after applying $\mathcal{P}$ to the second and third register of n qubits, we have that the system is in the following state:

$$\left[\sum_{(s_1, s_2) \in S^2} \sqrt{P_{s_0 s_1} P_{s_1 s_2}} \, |s_0\rangle \, |s_1\rangle \, |s_2\rangle \right] \otimes |0\rangle^{\otimes((T-3)n+3)}$$

Hence, after $T-1$ such applications of $\mathcal{P}$, we have the following state:

$$\left[\sum_{(s_1,s_2,\ldots,s_{T_1})\in S^{T-1}} \sqrt{P_{s_0s_1}P_{s_1s_2}\cdots P_{s_{T-2}s_{T-1}}}\,|s_0\rangle\,|s_1\rangle\,|s_2\rangle\cdots|s_{T-1}\rangle\right]\otimes|0\rangle^{\otimes 3}$$

Now, we apply the probability oracle. Observe that we implement the phase oracle $\overline{\mathcal{R}}$ fewer than the following number of times:

$$4\left(\log\left(\frac{1}{\delta}\right)+1\right)\left(\log\left(\frac{1}{\delta}\right)+2\right)$$

Every application of $\overline{\mathcal{R}}$ in turn calls a fractional phase query of $\mathcal{R}$ at most T times, and each of these fractional phase queries induces an error of at most $6\delta'$. Hence, using Lemma B.2, we observe that the total error introduced by these fractional phase oracles is upper bounded by:

$$T\cdot 4\left(\log\left(\frac{1}{\delta}\right)+1\right)\left(\log\left(\frac{1}{\delta}\right)+2\right)\cdot 6\delta' = \frac{\delta\cdot 24T}{48T} = \frac{\delta}{2}$$

The probability oracle also induces a norm error of $3\delta/2$. Hence, again using Lemma B.2 which shows that we can add the error norms, up to a total norm error of 2δ, we find that we obtain the following state after the probability oracle is applied:

$$\sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}} \sqrt{\xi(s_0,s_1,\ldots,s_{T-1})}\,|s_0\rangle\,|s_1\rangle\cdots|s_{T-1}\rangle\,|0\rangle^{\otimes 3} + \sqrt{1-\xi(s_0,s_1,\ldots,s_{T-1})}\,|\psi\rangle\,|1\rangle \tag{6.2.3}$$

where $|\psi\rangle$ is some $(Tn+2)$-qubit state and:

$$\xi(s_0,s_1,\ldots,s_{T-1}) = P_{s_0s_1}P_{s_1s_2}\cdots P_{s_{T-2}s_{T-1}}\cdot\left(\frac{1}{2}+\frac{1}{4}\sum_{t=0}^{T-1}\gamma^t\tilde{R}(s_t)\right)$$

The only thing that is left for us to check here is that $\left\|\sum_{t=0}^{T-1}\gamma^t\tilde{R}(s_t)\right\|_\infty \leq \frac{1}{2}$, as this is a requirement in Circuit 6.2.5. But note that we have, for all $(s_0,s_1,\ldots,s_{T-1})\in S^T$:

$$\left|\sum_{t=0}^{T-1}\gamma^t\tilde{R}(s_t)\right| \leq \sum_{t=0}^{T-1}\gamma^t|\tilde{R}(s_t)| = \frac{1-\gamma}{2|R|_{\max}}\cdot\sum_{t=0}^{T-1}\gamma^t|R(s_t)| \leq \frac{1}{2}\cdot(1-\gamma)\sum_{t=0}^{T-1}\gamma^t = \frac{1}{2}(1-\gamma^T) \leq \frac{1}{2}$$

Now, we are finally ready to present the quantum value estimation algorithm:

Algorithm 6.2.7: Quantum value estimation

Description: Given a Markov reward process $M = (S, P, R, \gamma)$ and an initial state $s_0 \in S$, this algorithm estimates the value of this Markov reward process starting from the given initial state.

Parameters:

1. An orthogonal set of n-qubit states $\{|s\rangle : s \in S\}$.
2. An absolute bound on the reward function, $|R|_{\max} > 0$.
3. The discount factor $\gamma \in [0, 1)$.
4. An accuracy parameter $\varepsilon > 0$.

Derived constants:

1. $m = \lceil \log(\frac{160|R|_{\max}\pi}{\varepsilon(1-\gamma)}) \rceil$.
2. $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.
3. $\delta = \frac{\varepsilon(1-\gamma)}{384|R|_{\max}\pi}$.

Oracle circuits:

1. A quantum circuit $\mathcal{P}$ that allows us to access the state transition probability matrix:

$$\forall s \in S, \mathcal{P} : |s\rangle \otimes |0\rangle^{\otimes n} \mapsto |s\rangle \otimes \sum_{s' \in S} \sqrt{P_{ss'}} \, |s'\rangle$$

2. A controlled reward oracle $C(\mathcal{R})$ where $\mathcal{R}$ has the following action:

$$\forall s \in S, \mathcal{R} : |s\rangle \mapsto e^{i\tilde{R}(s)} |s\rangle \qquad \text{where} \qquad \tilde{R}(s) = \frac{(1-\gamma)R(s)}{2|R|_{\max}}$$

Output: An ε-precise approximation to $V_M(s_0)$.

Number of qubits: $Tn + 3 + m$, and at least $Tn + 3$ auxilliary qubits are needed, not counting the auxilliary qubits necessary to impleemnt $\mathcal{P}$ and $C(\mathcal{R})$.

Success probability: Lower bounded by 2/3.

Oracle query complexity: The number of queries to Circuit 6.2.6 is $2^{m+1} - 1$. Consequently, the query complexity to $C(\mathcal{R})$ and $\mathcal{P}$ satisfies:

$$\tilde{\mathcal{O}}\left((2^{m+1} - 1)T\right) = \tilde{O}\left(\frac{|R|_{\max}}{\varepsilon(1-\gamma)^2}\right)$$

Algorithm:

1. Run the amplitude estimation algorithm, Algorithm 3.5.12, with the following settings:
   (a) Let the accuracy parameter be $\varepsilon(1-\gamma)/(16|R|_{\max})$.
   (b) Let the maximal fault-tolerance probability be $\frac{1}{6}$.
   (c) Let the controlled reflection circuit be the circuit that reflects through the last qubit being in state $|0\rangle$. This controlled reflection circuit can be built by implementing a controlled Z-gate, whose control qubit is the control qubit of the reflection circuit, and whose target qubit is the $(Tn + 1)$st qubit. This is off by a sign, hence we must compensate by a Z-gate applied to the control qubit.
   (d) Furthermore, let the setup circuit be Circuit 6.2.6. Take as accuracy parameter ε, and let the accuracy parameter of Circuit 6.2.5 be δ.

   Denote the output by $\tilde{\theta}$.
2. Return the following output:

$$\tilde{V} = \frac{(\sin^2(\tilde{\theta}) - \frac{1}{2}) \cdot 8|R|_{\max}}{1-\gamma}$$

Proof of the lower bound on the success probability. Observe that the setup circuit, i.e., Circuit 6.2.6, turns the all-zero state into the state displayed in Equation 6.2.3, up to an error of at most 2δ. This circuit is

invoked $2^{m+1}-1$ times, so the total norm error induced is upper bounded by:

$$2\delta \cdot (2^{m+1}-1) \leq 2\delta \cdot 2 \cdot \frac{16|R|_{\max}\pi}{\varepsilon(1-\gamma)} = \frac{64|R|_{\max}\pi}{\varepsilon(1-\gamma)} \cdot \delta = \frac{64|R|_{\max}\pi}{\varepsilon(1-\gamma)} \cdot \frac{\varepsilon(1-\gamma)}{384|R|_{\max}\pi} = \frac{1}{12}$$

Thus, this norm error induces a change in probability of at most $\frac{1}{6}$, as is proven in Lemma B.1, and hence it is sufficient to show that the success probability is lower bounded by $\frac{5}{6}$ if we assume that the output state of Circuit 6.2.5 is exactly the one displayed in Equation 6.2.3.

Under this assumption, the analysis of the amplitude estimation algorithm indicates that $\tilde{\theta}$ satisfies with probability at least $1-\frac{1}{6}=\frac{5}{6}$:

$$\left|\tilde{\theta} - \arcsin\left(\sqrt{\sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}} \xi(s_0,s_1,\ldots,s_{T-1})}\right)\right| \leq \frac{\varepsilon(1-\gamma)}{16|R|_{\max}}$$

Hence, we find, with probability at least $\frac{5}{6}$:[2]

$$\left|\sin(\tilde{\theta}) - \sqrt{\sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}} \xi(s_0,s_1,\ldots,s_{T-1})}\right| \leq \frac{\varepsilon(1-\gamma)}{16|R|_{\max}}$$

But notice that we can rewrite the summation as follows:

$$\begin{aligned}
\sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}} \xi(s_0,\ldots,s_{T-1}) &= \sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}} P_{s_0s_1}P_{s_1s_2}\cdots P_{s_{T-2}s_{T-1}} \cdot \left(\frac{1}{2} + \frac{1}{4}\cdot\sum_{t=0}^{T-1}\gamma^t\tilde{R}(s_t)\right) \\
&= \frac{1}{2} + \frac{1}{4}\cdot\sum_{t=0}^{T-1}\sum_{(s_1,\ldots,s_t)\in S^t} P_{s_0s_1}P_{s_1s_2}\cdots P_{s_{t-1}s_t}\gamma^t\tilde{R}(s_t) \\
&= \frac{1}{2} + \frac{1}{4}\mathbb{E}\left[\sum_{t=0}^{T-1}\gamma^t\tilde{R}(S_t)\right] = \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}}\mathbb{E}\left[\sum_{t=0}^{T-1}\gamma^t R(S_t)\right] \\
&= \frac{1}{2} + \frac{(1-\gamma)V_M(s_0)}{8|R|_{\max}}
\end{aligned}$$

Thus, we obtain:

$$\begin{aligned}
\left|\tilde{V} - V_M(s_0)\right| &= \left|\frac{(\sin^2(\tilde{\theta})-\frac{1}{2})\cdot 8|R|_{\max}}{1-\gamma} - V_M(s_0)\right| = \frac{8|R|_{\max}}{1-\gamma}\cdot\left|\sin^2(\tilde{\theta}) - \frac{1}{2} - \frac{(1-\gamma)V_M(s_0)}{8|R|_{\max}}\right| \\
&= \frac{8|R|_{\max}}{1-\gamma}\cdot\left|\sin^2(\tilde{\theta}) - \sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}}\xi(s_0,s_1,\ldots,s_{T-1})\right| \\
&\leq \frac{8|R|_{\max}}{1-\gamma}\cdot 2\cdot\left|\sin(\tilde{\theta}) - \sqrt{\sum_{(s_1,\ldots,s_{T-1})\in S^{T-1}}\xi(s_0,s_1,\ldots,s_{T-1})}\right| \\
&\leq \frac{16|R|_{\max}\cdot\varepsilon(1-\gamma)}{16|R|_{\max}\cdot(1-\gamma)} = \varepsilon
\end{aligned}$$

Hence, with probability at least $\frac{2}{3}$, $\tilde{V}$ indeed is an ε-precise approximation of $V_M(s_0)$. This completes the proof. □

[2] Proof via mean value theorem.

Proof of the query complexity of the Algorithm 6.2.7. The number of calls to Circuit 6.2.6 is $2^{m+1}-1$, as this is the number of calls to the setup circuit in Algorithm 3.5.12. Circuit 6.2.6 makes $T-1$ calls to $\mathcal{P}$, and $\tilde{\mathcal{O}}(T)$ to $\mathcal{R}$. Hence, the query complexity of both is indeed:

$$\tilde{\mathcal{O}}\left((2^{m+1}-1)T\right) \tag{6.2.4}$$

We can pretty easily substitute m, as we find:

$$2^{m+1} = 2 \cdot \left\lceil \frac{160|R|_{\max}}{\varepsilon(1-\gamma)} \right\rceil = \mathcal{O}\left(\frac{|R|_{\max}}{\varepsilon(1-\gamma)}\right)$$

Now, let's focus on T. We have by definition:

$$T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$$

We write out what is in the ceil in terms of $\zeta = 1/(1-\gamma)$, where we note that $\gamma = 1 - 1/\zeta$:

$$\begin{aligned}
\frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} &= \frac{\ln\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\ln(\gamma)} = \frac{-\ln\left(\frac{2|R|_{\max}}{\varepsilon(1-\gamma)}\right)}{\ln(\gamma)} = \frac{-\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right)}{\ln(1-\frac{1}{\zeta})} = \frac{\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right)}{\frac{1}{\zeta}+\mathcal{O}(\frac{1}{\zeta^2})} = \frac{\zeta\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right)}{1+\mathcal{O}(\frac{1}{\zeta})} \\
&= (\zeta+\mathcal{O}(1))\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right) = \zeta\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right) + \mathcal{O}\left(\ln\left(\frac{2|R|_{\max}}{\varepsilon}\zeta\right)\right) \\
&= \tilde{O}(\zeta) = \tilde{O}\left(\frac{1}{1-\gamma}\right)
\end{aligned}$$

And hence, we find:

$$T = \tilde{O}\left(\frac{1}{1-\gamma}\right)$$

Thus, substituting into Equation 6.2.4 yields:

$$\tilde{O}\left(\frac{|R|_{\max}}{\varepsilon(1-\gamma)^2}\right)$$

This completes the proof. □

Finally, we can take a step back and put the properties of the algorithm that we just described into perspective. In the classical algorithm, Algorithm 6.2.1, we can see that the number of samples from the Markov reward process, N, scales quadratically in $|R|_{\max}$, $1/\varepsilon$ and $1/(1-\gamma)$. Its quantum counterpart, however, i.e., Algorithm 6.2.7, scales only linearly in ε and $|R|_{\max}$, but still quadratically in $1/(1-\gamma)$, aside from some logarithmic factors. With respect to the precision parameter ε, though, we can clearly see that the quantum value estimation algorithm provides a quadratic speed-up over the Monte Carlo method. This quadratic speed-up can be attributed to the use of Grover's iterate in the amplitude estimation step.

In the next subsection, we will prove that we cannot hope to achieve a quantum algorithm with similar input circuits which has a better scaling than linear in ε and $|R|_{\max}$, and quadratic in $1/(1-\gamma)$.

6.2.3 Essential optimality of query complexity

In the last subsection, we have developed a quantum value estimation algorithm, Algorithm 6.2.7, which estimates the value of a Markov reward process starting at some initial state. We obtained that the scaling in the parameters $1/\varepsilon$, $1/(1-\gamma)$ and $|R|_{\max}$ was linear. In this subsection, we will prove that this scaling is

essentially optimal, i.e., that up to factors that scale logarithmically in these parameters, we cannot improve Algorithm 6.2.7.

We will again use methods introduced in Section 5.1 to arrive at this result. Specifically, we will be using the Hybrid method, i.e., Theorem 5.1.7. To that end, we should identify some examples of Markov reward processes that we want our algorithm to be able to distinguish, but which at the same time are hard to distinguish. This, we do in the definition below.

Definition 6.2.8: Test Markov reward processes
Let $S = \{s_0, s_1\}$, $\delta \in [0,1]$, $\eta \geq 0$, $R_0 > 0$ and $\gamma \in [0,1)$ be arbitrary parameters. Let the state transition probability matrix P_δ and the state reward function $R_{R_0,\delta,\eta}$ be defined as follows:

$$P_\delta = \begin{bmatrix} 1-\delta & \delta \\ 0 & 1 \end{bmatrix} \qquad \text{and} \qquad R_{R_0,\delta,\eta} = \begin{bmatrix} \eta \\ R_0 \end{bmatrix}$$

Here, the matrix and vector are indexed by (s_0, s_1), so e.g., $(P_\delta)_{s_0 s_1} = \delta$. Finally, define the Markov reward process $M_{R_0,\delta,\eta,\gamma} = (S, P_\delta, R_{R_0,\delta,\eta}, \gamma)$. We refer to these Markov reward processes as the *test Markov reward processes*.

In order to get some intuition for the Markov reward processes that we defined in this way, we have graphically depicted the dynamics of this Markov reward process in Figure 6.2.

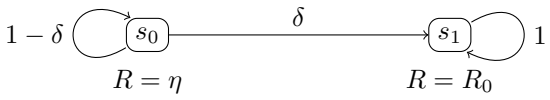


Figure 6.2: The Markov reward process $M_{R_0,\delta,\eta,\gamma}$. The states are shown as nodes in the directed graph above, and the possibly non-zero probability matrix entries are shown as arcs in this graph, and they are annotated with the corresponding probability. Moreover, the rewards that one obtains upon exiting the states are displayed.

Next, let's determine the value of the test Markov reward processes $M_{R_0,\delta,\eta,\gamma}$ starting at s_0.

Lemma 6.2.9: Value of the test Markov reward processes, starting at s_0
Let $\delta \in [0,1]$, $R_0 > 0$, $\eta \geq 0$ and $\gamma \in [0,1)$. Then:

$$V_{M_{R_0,\delta,\eta,\gamma}}(s_0) = \frac{\eta}{1-\gamma(1-\delta)} + \frac{R_0}{1-\gamma} - \frac{R_0}{1-\gamma(1-\delta)}$$

Proof. Let $(S_t)_{t=0}^\infty$ be the random variables taking values in S, such that $\mathbb{P}[S_t = s]$ denotes the probability that we arrive at state s after t steps in the Markov reward process. Observe that we start in state s_0, hence $\mathbb{P}[S_0 = s] = \delta_{s s_0}$. Whenever we traverse to s_1, we will stay there forever, hence in order for us to be in s_0 at time step t, we must have stayed in s_0 throughout all time steps before t. Thus, $\mathbb{P}[S_t = s_0] = (1-\delta)^t$, and then automatically, we find $\mathbb{P}[S_t = s_1] = 1 - (1-\delta)^t$. Thus:

$$\begin{aligned} V_{M_{R_0,\delta,\eta,\gamma}}(s_0) &= \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t R(S_t)\right] = \sum_{t=0}^\infty \gamma^t \left[(1-\delta)^t \eta + (1-(1-\delta)^t) R_0\right] \\ &= \eta \sum_{t=0}^\infty (\gamma(1-\delta))^t + R_0 \sum_{t=0}^\infty \gamma^t - R_0 \sum_{t=0}^\infty (\gamma(1-\delta))^t \\ &= \frac{\eta}{1-\gamma(1-\delta)} + \frac{R_0}{1-\gamma} - \frac{R_0}{1-\gamma(1-\delta)} \end{aligned}$$

This completes the proof. □

In the remainder of this subsection, we will denote ε-precise quantum value estimation algorithms with maximal reward $|R|_{\max}$, and discount factor γ to be algorithms that use the circuits $\mathcal{P}$ and $\mathcal{R}$, as described by Equation 6.2.1 and Equation 6.2.2, to produce, with probability at least $2/3$, an ε-precise estimation of the value of a given Markov reward process whose reward is absolutely bounded by $|R|_{\max}$ and whose discount factor is γ. Now, we can provide lower bounds on the query complexities of such algorithms to $\mathcal{P}$ and $\mathcal{R}$. This we do in the following theorems.

Theorem 6.2.10: Lower bound on the query complexity of a quantum value estimation algorithm to $\mathcal{R}$
Let $\gamma \in [0,1)$, $|R|_{\max} > 0$ and $0 < \varepsilon < \frac{|R|_{\max}}{2(1-\gamma)}$. Let $T_{\mathcal{R}}$ be the query complexity to the oracle circuit $\mathcal{R}$, which acts on n qubits, of an ε-precise quantum value estimation algorithm with maximal reward $|R|_{\max}$ and discount factor γ. Then:

$$T_{\mathcal{R}} \geq \frac{|R|_{\max}}{3\varepsilon(1-\gamma)^2}$$

Proof. Let $\eta = 2\varepsilon(1-\gamma)$, $R_0 = |R|_{\max} > \eta$ and $\delta = 0$. We find, according to Lemma 6.2.9:

$$V_{M_{R_0,0,0,\gamma}}(s_0) = 0 \qquad \text{and} \qquad V_{M_{R_0,0,\eta,\gamma}}(s_0) = \frac{\eta}{1-\gamma} = \frac{2\varepsilon(1-\gamma)}{1-\gamma} = 2\varepsilon$$

Hence, the algorithm should be able to distinguish between the two Markov reward processes. But the oracle circuit $\mathcal{P}$ is equal for both Markov reward processes. We let $\mathcal{R}_{R_0,\delta,\eta}$ be the reward oracle $\mathcal{R}$ associated to the Markov reward process $M_{R_0,\delta,\eta,\gamma}$. Then, using the hybrid method, Theorem 5.1.7, we find that:

$$\begin{aligned}
\frac{1}{9T_{\mathcal{R}}^2} &\leq \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^n} \\ \||\psi\rangle\|=1}} \|(\mathcal{R}_{R_0,0,0} - \mathcal{R}_{R_0,0,\eta})\,|\psi\rangle\|^2 = \|\mathcal{R}_{R_0,0,0} - \mathcal{R}_{R_0,0,\eta}\|^2 \\
&= \sup_{s\in S} \left| e^{i\tilde{R}_{R_0,0,0}(s)} - e^{i\tilde{R}_{R_0,0,\eta}(s)} \right|^2 \leq \sup_{s\in S} \left| \tilde{R}_{R_0,0,0}(s) - \tilde{R}_{R_0,0,\eta}(s) \right|^2 \\
&= \frac{(1-\gamma)^2}{4|R|_{\max}^2} \cdot \sup_{s\in S} |R_{R_0,0,0}(s) - R_{R_0,0,\eta}(s)|^2 = \frac{(1-\gamma)^2}{4|R|_{\max}^2} \cdot |\eta|^2 = \frac{\varepsilon^2(1-\gamma)^4}{|R|_{\max}^2}
\end{aligned}$$

Hence, we find:

$$T_{\mathcal{R}} \geq \frac{|R|_{\max}}{3\varepsilon(1-\gamma)^2}$$

This completes the proof. □

Theorem 6.2.11: Lower bound on the query complexity of a quantum value estimation algorithm to $\mathcal{P}$
Let $\gamma \in [\frac{1}{2}, 1)$, $|R|_{\max} > 0$ and $0 < \varepsilon < \frac{1}{32} R_0 \gamma$. Let $T_{\mathcal{P}}$ be the query complexity to the oracle circuit $\mathcal{P}$, which acts on $2n$ qubits, of an ε-precise quantum value estimation algorithm with maximal reward $|R|_{\max}$ and discount factor γ. Then:

$$T_{\mathcal{P}} \geq \frac{R_0}{\sqrt{2} \cdot 96\varepsilon(1-\gamma)^2}$$

Proof. Let $\eta = 0$, $R_0 = |R|_{\max}$ and:

$$\delta = \frac{16(1-\gamma)^2\varepsilon}{R_0\gamma}$$

Then, we find, according to Lemma 6.2.9, we have $V_{M_{R_0,0,\gamma}}(s_0) = 0$ and:

$$V_{M_{R_0,\delta,0,\gamma}}(s_0) = \frac{R_0}{1-\gamma} - \frac{R_0}{1-\gamma(1-\delta)} = R_0 \cdot \frac{(1-\gamma(1-\delta)) - (1-\gamma)}{(1-\gamma)(1-\gamma(1-\delta))} = \frac{\gamma\delta R_0}{(1-\gamma)^2 + \delta\gamma(1-\gamma)}$$
$$= \frac{\gamma \cdot \frac{16(1-\gamma)^2\varepsilon}{R_0\gamma} \cdot R_0}{(1-\gamma)^2 + \frac{16(1-\gamma)^2\varepsilon}{R_0\gamma} \cdot \gamma(1-\gamma)} = \frac{16\varepsilon}{1 + \frac{16\varepsilon}{R_0\gamma} \cdot \gamma(1-\gamma)} \geq \frac{16\varepsilon}{1 + 16 \cdot \frac{1}{32} \cdot \frac{1}{4}} = \frac{16\varepsilon}{1 + \frac{1}{8}} \geq 2\varepsilon$$

Hence, the algorithm should be able to distinguish the Markov reward processes $M_{R_0,0,0,\gamma}$ and $M_{R_0,\delta,0,\gamma}$. But the oracles $\mathcal{R}$ are the same, and hence we must use the $\mathcal{P}$ oracles to distinguish the two. So, let's denote by $\mathcal{P}_0$ and $\mathcal{P}_\delta$ the $\mathcal{P}$ oracles of $M_{R_0,0,0,\gamma}$ and $M_{R_0,\delta,0,\gamma}$, respectively. Now, we can invoke the hybrid method, i.e., Theorem 5.1.7, to deduce that:

$$\frac{1}{9T_{\mathcal{P}}^2} \leq \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^{2n}} \\ \||\psi\rangle\| = 1}} \|(\mathcal{P}_\delta - \mathcal{P}_0)|\psi\rangle\|^2 = \|\mathcal{P}_\delta - \mathcal{P}_0\|^2 = \left\| \begin{bmatrix} -\delta & \delta \\ 0 & 0 \end{bmatrix} \right\|^2 = 2\delta^2 = 2\left(\frac{16(1-\gamma)^2\varepsilon}{R_0\gamma}\right)^2$$

Hence, we obtain:

$$T_{\mathcal{P}} \geq \frac{R_0\gamma}{\sqrt{2} \cdot 48\varepsilon(1-\gamma)^2} \geq \frac{R_0}{\sqrt{2} \cdot 96\varepsilon(1-\gamma)^2}$$

This completes the proof. □

Hence, we have proven that Algorithm 6.2.7 is an essentially optimal quantum value estimation algorithm in all its parameters, meaning that it is optimal up to logarithmic factors. This also means that attempting to transport other classical value evaluation algorithms, such as temporal difference learning and $TD(\lambda)$, will not result in a better algorithm.

This concludes our discussion on quantum value estimation algorithms. In the next section, we will show how we can use the theory developed here to evaluate policies for a Markov decision process.

6.3 Quantum policy evaluation

In this section, we will show how we can use the quantum value estimation algorithm to perform policy evaluation. This means that given a Markov decision process $M = (S, A, P, R, \gamma)$, an initial state $s_0 \in S$, and a policy π for M, we will determine the value function of M under π starting from s_0, i.e., $V_M^{(\pi)}(s_0)$.

The key ingredient will be Theorem 6.1.13, which states that for all such choices of M, s_0 and π, we have:

$$V_M^{(\pi)}(s_0) = V_{M^{(\pi)}}(s_0)$$

Hence, we might just as well perform quantum value evaluation on the Markov reward process generated by the Markov decision process and the policy π, i.e., $M^{(\pi)}$.

So, suppose that we have a Markov decision process $M = (S, A, P, R, \gamma)$ and a policy π for it. Suppose that we have two orthogonal sets of n_S- and n_A-qubit states, respectively, which we label by the elements from the state and action space of the Markov decision process M:

$$\{|s\rangle \in \mathbb{C}^{2^{n_S}} : s \in S\} \qquad \text{and} \qquad \{|a\rangle \in \mathbb{C}^{2^{n_A}} : a \in A\}$$

Next, suppose that we have access to the state-action transition probability matrix by means of the following oracle circuit, acting on $2n_S + n_A$ qubits:

$$\mathcal{P} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_S} \mapsto \sum_{s' \in S} \sqrt{P_{sas'}}\, |s\rangle \otimes |a\rangle \otimes |s'\rangle$$

Similarly as before, suppose that we have access to the state-action reward function by means of a phase oracle acting on $n_S + n_A$ qubits, defined as follows:

$$\mathcal{R} : |s\rangle \otimes |a\rangle \mapsto e^{i\tilde{R}(s,a)} |s\rangle \otimes |a\rangle \qquad \text{where} \qquad \tilde{R}(s,a) = \frac{1-\gamma}{2|R|_{\max}} R(s,a)$$

where again $|R|_{\max} = \max_{(s,a)\in S\times A} |R(s,a)|$. Finally, suppose that we have access to the policy via the following oracle, acting on $n_S + n_A$ qubits:

$$\Pi : |s\rangle \otimes |0\rangle^{\otimes n_A} \mapsto \sum_{a\in A} \sqrt{\pi_{sa}} |s\rangle \otimes |a\rangle$$

Now, consider the Markov reward process generated by M under π, which we denote by $M^{(\pi)} = (S', P', R, \gamma)$. Recall that $S' = S \times A$, and hence we can define the following orthonormal set of $(n_A + n_S)$-qubit states:

$$\{|s\rangle \otimes |a\rangle \in \mathbb{C}^{2^{n_A+n_S}} : (s,a) \in S \times A\} = \{|s\rangle : s \in S\} \otimes \{|a\rangle : a \in A\}$$

So, there is a very natural embedding of the state space $S' = S \times A$ into the $(n_S + n_A)$-qubit state space. Moreover, we can construct the following circuit $\mathcal{P}'$, acting on $2(n_S + n_A)$ qubits:

$$\mathcal{P}' = (I_2^{\otimes(n_S+n_A)} \otimes \Pi)\mathcal{P} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes(n_S+n_A)} \mapsto \sum_{(s',a')\in S'} \sqrt{P_{sas'}\pi_{s'a'}} |s\rangle \otimes |a\rangle \otimes |s'\rangle \otimes |a'\rangle$$

But this is the circuit that implements the state probability matrix for the Markov reward process $M^{(\pi)}$. Hence, we have supplied all the ingredients necessary to use the quantum value estimation algorithm, Algorithm 6.2.7, to estimate $V_{M^{(\pi)}}(s_0)$, which equals $V_M^\pi(s_0)$ according to Theorem 6.1.13.

Note that in order to implement $\mathcal{P}'$, we use $\mathcal{P}$ and Π once. Hence, the query complexity to $\mathcal{P}$ and Π will be equal to the query complexity of $\mathcal{P}'$.

For convenience, the full algorithm is stated below.

Algorithm 6.3.1: Quantum policy evaluation

Description: Given a Markov decision process $M = (S, A, P, R, \gamma)$, a starting state $s_0 \in S$ and a policy π for it, this algorithm returns an ε-precise approximation to the value of the Markov decision process M under π starting from s_0.

Parameters:

1. Orthogonal sets of n_A- and n_S-qubit states, $\{|s\rangle : s \in S\}$ and $\{|a\rangle : a \in A\}$, respectively.
2. An absolute bound on the reward function, $|R|_{\max} > 0$.
3. The discount factor, $\gamma \in [0, 1)$.
4. An accuracy parameter, $\varepsilon > 0$.

Derived constants:

1. $m = \lceil \log(\frac{160|R|_{\max}\pi}{\varepsilon(1-\gamma)}) \rceil$.
2. $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.

Oracle circuits:

1. A quantum circuit $\mathcal{P}$ that implements the state-action probability matrix:

$$\mathcal{P} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_S} \mapsto \sum_{s' \in S} \sqrt{P_{sas'}} \, |s\rangle \otimes |a\rangle \otimes |s'\rangle$$

2. A controlled quantum circuit $C(\mathcal{R})$, where $\mathcal{R}$ implements the state-action reward function as a phase oracle:

$$\mathcal{R} : |s\rangle \otimes |a\rangle \mapsto e^{i\tilde{R}(s,a)} \, |s\rangle \otimes |a\rangle \qquad \text{where} \qquad \tilde{R}(s,a) = \frac{1-\gamma}{2|R|_{\max}} R(s,a)$$

3. A quantum circuit Π that implements the policy, as follows:

$$\Pi : |s\rangle \otimes |0\rangle^{\otimes n_A} \mapsto \sum_{a \in A} \sqrt{\pi_{sa}} \, |s\rangle \otimes |a\rangle$$

Number of qubits: $T(n_S + n_A) + 3 + m$.

Success probability: Lower bounded by 2/3.

Oracle query complexity: The oracle query complexity to $\mathcal{P}$, $C(\mathcal{R})$ and Π satisfies:

$$\tilde{O}\left(\frac{|R|_{\max}}{\varepsilon(1-\gamma)^2}\right)$$

Algorithm: Run the quantum value estimation algorithm, Algorithm 6.2.7, with the following settings:

1. The state probability matrix circuit is $\mathcal{P}' = (I_2^{\otimes(n_S+n_A)})\mathcal{P}$.
2. The controlled reward oracle is just $C(\mathcal{R})$.
3. Use the same values for the parameters ε, γ and $|R|_{\max}$.

The only modification one should make is in Circuit 6.2.6. Here, in step 1, one should not only map the all zeros-state to $|s_0\rangle$, but one should call the circuit Π afterwards as well, such that the state of the first two registers after step 1 of the circuit becomes:

$$|s_0\rangle \otimes \sum_{a \in A} \sqrt{\pi_{sa}} \, |a\rangle$$

This concludes our discussion of quantum policy evaluation. In the next and final section of this chapter, we will use this algorithm as one of the steps in performing quantum policy optimization.

6.4 Quantum policy optimization

In this section, we will elaborate on how we can use the theory from the previous section to perform quantum policy optimization. This means that given a Markov decision process and an initial state, we try to find a policy for this Markov decision process such that its value under this policy starting from the given initial state is maximized. In the notation introduced in Section 6.1, this can be concisely expressed as given some Markov decision process $M = (S, A, P, R, \gamma)$ and initial state $s_0 \in S$, finding a policy π for M such that $V_M^{(\pi)}(s_0)$ is maximized.

Recently, many classical policy optimization algorithms have been developed. The most notable ones are SARSA, Q-learning and the policy gradient method [Sil15, SB18]. With SARSA and Q-learning it is not clear how these can be implemented in a quantum algorithm in an efficient way, as they are in some respects similar to temporal difference methods for quantum value evaluation. That's why in this chapter, we will develop a quantum algorithm that can be viewed as the quantum analogue of the policy gradient method. For this algorithm, we will need to estimate gradients of high-dimensional functions, and this is where the gradient estimation algorithm of Gilyén et al. that we covered in the previous chapters comes into play.

It must be noted in advance that the algorithm presented in this chapter is not necessarily better than a classical version of this algorithm. This is because we will be using Gilyén et al.'s algorithm, Algorithm 4.3.9, with the parameter $\sigma = 1$, which has a query complexity that is linear in the dimension of the domain of the function. There exist classical algorithms that also have a query complexity linear in the dimension, and hence the method presented here does not necessarily provide a speed-up. However, we have not proven optimality of Algorithm 4.3.9 with $\sigma = 1$, and hence it might be possible to improve it. If this is indeed possible, then as a result the ideas in this section provide a speed-up over the classical case.

In Subsection 6.4.1, we will first of all have a look at how the partial derivatives of a composition of high-dimensional functions can be rewritten into the partial derivatives of the individual functions. We arrive at a generalization of Faà di Bruno's formula, and subsequently use it to prove that compositions of functions that are of Gevrey-type at most 1 are again of Gevrey-type at most 1. Then, in Subsection 6.4.2, we prove that the function $\pi \mapsto V_M^{\pi}(s_0)$ is of Gevrey-type 1 if it is composed with some natural embedding into the space of policies. This result allows us to find the gradient of this function through Gilyén et al.'s gradient estimation algorithm, which we prove in Subsection 6.4.3. We also describe how this gradient estimation procedure can be used to perform policy optimization. Finally, in Subsection 6.4.4, we briefly discuss in what kind of applications we can usefully employ the algorithm developed in this section.

6.4.1 The class of functions of Gevrey-type 1 is closed under composition

In this section, we will very frequently encounter high-dimensional functions, i.e., functions acting on a high-dimensional domain. To get a better grip on these functions, we first of all prove how the partial derivatives of compositions of these kind of functions can be expressed in terms of the partial derivatives of the individual functions. We achieve this in the following lemma, which is a generalization of Faà di Bruno's formula [dB55]. It is not a very well-known formula, but it's not new either, as it can for example be found in [Gzy86].

Lemma 6.4.1: Multidimensional Faà di Bruno's formula
Let $d, n \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$, $\Psi \subseteq \mathbb{R}^n$ open, and let $f : \Omega \to \Psi$ and $g : \Psi \to \mathbb{R}$ be smooth functions. Moreover, let $k \in \mathbb{N}$ and $\alpha \in [d]^k$. Let $\Pi_{|\alpha|}$ be the set of all partitions of $\{1, 2, \ldots, |\alpha|\}$ and let α_π be the subsequence of α indexed by the set $\pi \subseteq \{1, 2, \ldots, |\alpha|\}$. Then, we have for all $\mathbf{x} \in \Omega$:

$$\partial_\alpha (g \circ f)(\mathbf{x}) = \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} \partial_{(\ell_1, \ldots, \ell_{|\pi|})} g(f(\mathbf{x})) \cdot \prod_{j=1}^{|\pi|} \partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x})$$

Proof. We will prove this by means of induction to k. To that end, first of all, suppose that $k = 1$. Then, we find $\alpha = (i)$ for some $i \in [m]$. We find, using the chain rule:

$$\partial_\alpha (g \circ f)(\mathbf{x}) = \partial_i (g \circ f)(\mathbf{x}) = \sum_{\ell=1}^{n} \partial_\ell g(f(\mathbf{x})) \partial_i f_\ell(\mathbf{x})$$

Now observe that $\Pi_{|\alpha|} = \Pi_1 = \{\{\{1\}\}\}$, and hence in the outermost summation on the right-hand side in the statement of the theorem, we obtain just one term, where $\pi = \{\{1\}\}$. Then we directly find that $|\pi| = 1$, and hence the right-hand side reduces to:

$$\sum_{\ell=1}^{n} \partial_\ell g(f(\mathbf{x})) \cdot \partial_i f_\ell(\mathbf{x})$$

which is exactly equal to the expression for $\partial_\alpha (g \circ f)(\mathbf{x})$ we obtained above. Hence, the formula holds when $k = 1$.

Now suppose that the formula holds for some $k \in \mathbb{N}$. Take $\alpha \in [d]^k$, $i \in [d]$, and define $\beta = (\alpha, i) \in [d]^{k+1}$. Then, we obtain, again invoking the chain rule:

$$\begin{aligned}
\partial_\beta (g \circ f)(\mathbf{x}) = \partial_i \partial_\alpha (g \circ f)(\mathbf{x}) &= \partial_i \left(\sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} \partial_{(\ell_1,\ldots,\ell_{|\pi|})} g(f(\mathbf{x})) \cdot \prod_{j=1}^{|\pi|} \partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x}) \right) \\
&= \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} \left(\sum_{\ell_{|\pi|+1}=1}^{n} \partial_{(\ell_1,\ldots,\ell_{|\pi|+1})} g(f(\mathbf{x})) \cdot \partial_i \partial_{\ell_{|\pi|+1}} f(\mathbf{x}) \cdot \prod_{j=1}^{\pi} \partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x}) \right. \\
&\quad \left. + \partial_{(\ell_1,\ldots,\ell_{|\pi|})} g(f(\mathbf{x})) \cdot \sum_{j'=1}^{\pi} \partial_i \partial_{\alpha_{\pi_{j'}}} f_{\ell_j}(\mathbf{x}) \cdot \prod_{\substack{j=1 \\ j \neq j'}}^{|\pi|} \partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x}) \right)
\end{aligned}$$

In the last expression of the above equation, the terms on the top line can be obtained by making partitions $\pi' \in \Pi_{|\beta|} = \Pi_{|\alpha|+1}$, by adding a new set to the partition, i.e., setting $\pi' = \pi \cup \{|\beta|\}$. The terms on the bottom line can be obtained by adding $|\beta|$ to the j'th set in the partition π. As all partitions $\pi \in \Pi_{|\beta|}$ can be constructed in this way, we obtain that we can rewrite the above equation into:

$$\partial_\beta (g \circ f)(\mathbf{x}) = \sum_{\pi \in \Pi_{|\beta|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} \partial_{(\ell_1,\ldots,\ell_{|\pi|})} g(f(\mathbf{x})) \cdot \prod_{j=1}^{|\pi|} \partial_{\beta_{\pi_j}} f_{\ell_j}(\mathbf{x})$$

And hence the formula also holds for all $\beta \in [d]^{k+1}$. We can now complete the proof with induction. □

The formula we derived in the previous lemma can be used to analyze the smoothness properties of compositions of functions of Gevrey-type at most 1. This is done in the following theorem.[3]

Theorem 6.4.2: Composition of functions of Gevrey-type at most 1
Let $d, n \in \mathbb{N}$, $\Omega \subseteq \mathbb{R}^d$, $\Psi \subseteq \mathbb{R}^n$ open, and let $f : \Omega \to \Psi$ and $g : \Psi \to \mathbb{R}$ be smooth functions. Let $A, B, C, D \geq 0$. Suppose that for all $j \in [n]$, $f_j \in \mathcal{G}^1_{d,\Omega,A,B}$, and that $g \in \mathcal{G}^1_{n,\Psi,C,D}$. Then, we find:

$$g \circ f \in \mathcal{G}^1_{d,\Omega,C,B(1+nAD)}$$

[3]This theorem is at the moment not used in the remainder of this document. However, it is quite likely that it might become useful in further research on this topic. Moreover, it is a theorem that is not easily found in literature, especially not with the constants explicitly written out, so this is why it is included here.

Proof. Take $k \in \mathbb{N}$, $\alpha \in [d]^k$ and $\mathbf{x} \in \Omega$ arbitrarily. According to Lemma 6.4.1, we have:

$$\partial_\alpha(g \circ f)(\mathbf{x}) = \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} \partial_{(\ell_1,\ldots,\ell_{|\pi|})} g(f(\mathbf{x})) \cdot \prod_{j=1}^{|\pi|} \partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x})$$

Hence, by taking absolute values on both sides, we can bound the right-hand side as follows:

$$\begin{aligned}
|\partial_\alpha(g \circ f)(\mathbf{x})| &\leq \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|\pi|}=1}^{n} |\partial_{(\ell_1,\ldots,\ell_{|\pi|})} g(f(\mathbf{x}))| \cdot \prod_{j=1}^{|\pi|} |\partial_{\alpha_{\pi_j}} f_{\ell_j}(\mathbf{x})| \\
&\leq \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1=1}^{n} \cdots \sum_{\ell_{|pi|}=1}^{n} CD^{|\pi|}|\pi|! \cdot \prod_{j=1}^{|\pi|} AB^{|\pi_j|}|\pi_j|! \\
&= \sum_{\pi \in \Pi_{|\alpha|}} n^{|\pi|} CD^{|\pi|}|\pi|! \cdot A^{|\pi|} B^{\sum_{j=1}^{|\pi|} |\pi_j|} \cdot \prod_{j=1}^{|\pi|} |\pi_j|! \\
&= CB^{|\alpha|} \sum_{\pi \in \Pi_{|\alpha|}} (nAD)^{|\pi|}|\pi|! \cdot \prod_{j=1}^{|\pi|} |\pi_j|!
\end{aligned}$$

Now, we rewrite the right-hand side of the last equation. To that end, we consider all partitions in $\Pi_{|\alpha|}$ that have r sets, and then sum over r. This results in the following inequality:

$$|\partial_\alpha(g \circ f)(\mathbf{x})| \leq CB^{|\alpha|} \sum_{r=1}^{|\alpha|} \sum_{\substack{\pi \in \Pi_{|\alpha|} \\ |\pi|=r}} (nAD)^r r! \cdot \prod_{j=1}^{r} |\pi_j|!$$

Next, we fix $r \in \{1, \ldots, |\alpha|\}$, and we take a sequence $(k_1, \ldots, k_r) \in \mathbb{N}^r$ such that $k_1 + k_2 + \cdots + k_r = |\alpha|$. If we now want to distribute all the elements of $|\alpha|$ over r sets with sizes k_1, k_2, …, k_r, then we know that there are $\binom{|\alpha|}{k_1\ k_2\ \cdots\ k_r}$ ways to do this if the order of the sets matters. However, in a partition, the order of the sets do not matter, and hence the total number of partitions with sets that have sizes k_1, k_2, …, k_r is given by:

$$\binom{|\alpha|}{k_1\ k_2\ \cdots\ k_r} \cdot \frac{1}{r!}$$

Using this, we can rewrite the last inequality in the following form:

$$\begin{aligned}
|\partial_\alpha(g \circ f)(\mathbf{x})| &\leq CB^{|\alpha|} \sum_{r=1}^{|\alpha|} \sum_{\substack{(k_1,\ldots,k_r) \in \mathbb{N}^r \\ k_1+\cdots+k_r=|\alpha|}} \binom{|\alpha|}{k_1\ k_2\ \cdots\ k_r} \cdot \frac{1}{r!} \cdot (nAD)^r r! \cdot \prod_{j=1}^{r} k_j! \\
&= CB^{|\alpha|} \sum_{r=1}^{|\alpha|} \sum_{\substack{(k_1,\ldots,k_r) \in \mathbb{N}^r \\ k_1+\cdots+k_r=|\alpha|}} |\alpha|! \cdot (nAD)^r
\end{aligned}$$

Next, observe that in order to find a sequence $(k_1, k_2, \ldots, k_r) \in \mathbb{N}^r$ such that $k_1 + k_2 + \cdots + k_r = |\alpha|$, we have to divide $|\alpha| - r$ units in r boxes. Thus, we can find $\binom{|\alpha|-r+r-1}{r-1}$ such vectors $(k_1, k_2, \ldots, k_r) \in \mathbb{N}^r$.

Hence, the inequality can be rewritten as follows:

$$\begin{aligned}|\partial_\alpha(g\circ f)(\mathbf{x})| &\leq CB^k \sum_{r=1}^{|\alpha|} \binom{|\alpha|-r+r-1}{r-1} |\alpha|!(nAD)^r = CB^{|\alpha|}|\alpha|! \cdot \sum_{r=1}^{|\alpha|} \binom{|\alpha|-1}{r-1}(nAD)^r \\ &= CB^{|\alpha|}|\alpha|! \cdot \sum_{r=0}^{|\alpha|-1} \binom{|\alpha|-1}{r}(nAD)^{r+1} = nACDB^{|\alpha|}|\alpha|! \cdot (1+nAD)^{|\alpha|-1} \\ &= \frac{nACD}{1+nAD}(B(1+nAD))^{|\alpha|}|\alpha|! \leq C(B(1+nAD))^{|\alpha|}|\alpha|!\end{aligned}$$

Finally, observe trivially that $|g(f(\mathbf{x}))| \leq C$. Hence, we find that $g \in \mathcal{G}^1_{d,\Omega,C,B(1+nAD)}$. This completes the proof. □

The above theorem proves that whenever we compose two functions that are of Gevrey-type at most 1, then the resulting composition is again of Gevrey-type at most 1. The interested reader might wonder whether similar results hold true for functions that are of at most Gevrey-type σ as well, where $\sigma \in [0,1)$. The answer is no, as one can for example take the function $f : \mathbb{R} \to \mathbb{R}$, $f(x) = e^x$. This is a function of Gevrey-type 0, but $f \circ f$ is a function of Gevrey-type 1, providing a counterexample for all choices of $\sigma \in [0,1)$.

This completes our discussion on the composition of two functions of Gevrey-type 1.

6.4.2 Smoothness properties of the policy evaluation function of a Markov decision process

Suppose that we have some Markov decision process $M = (S, A, P, R, \gamma)$ and an initial state $s_0 \in S$. In this section, we will look at the function $\pi \mapsto V_M^{(\pi)}(s_0)$, i.e., the function that maps a policy for M to the value of M under the policy, starting from state s_0. We call this function *the policy evaluation function*, denoted by $E_{M,s_0}(\pi) = V_M^{(\pi)}(s_0)$. We formally define it in the definition below.

Definition 6.4.3: Policy evaluation function
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process and $s_0 \in S$. We define $E_{M,s_0} : X \to \mathbb{R}$, where $X \subseteq \mathbb{R}^{S\times A}$ is defined as follows:

$$X = \left\{ \pi \in [0,1]^{S\times A} : \forall s \in S, \sum_{a\in A} \pi_{sa} = 1 \right\}$$

and E_{M,s_0} sends a policy $\pi \in X$ to the value of M under π:

$$E_{M,s_0}(\pi) = V_M^{(\pi)}(s_0)$$

We will refer to X as the *policy space* and E_{M,s_0} as the *policy evaluation function of M from s_0*.

The goal of this subsection is to prove some smoothness properties of this policy evaluation function E_{M,s_0}. As a first step towards this goal, we prove that it is bounded in the lemma below.

Lemma 6.4.4: Boundedness of the policy evaluation function
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process and $s_0 \in S$. Moreover, define $|R|_{\max} = \max_{(s,a)\in S\times A} |R(s,a)|$. Then, for all policies $\pi \in X$:

$$|E_{M,s_0}(\pi)| \leq \frac{|R|_{\max}}{1-\gamma}$$

Proof. Let $\pi \in X$ be a policy for M, and let $(S_t^\pi)_{t=0}^\infty$ be random variables taking values in S, which model the dynamics of the Markov decision process under π. Then, we find:

$$|E_{M,s_0}(\pi)| = |V_M^{(\pi)}(s_0)| = \left| \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t^\pi) \right] \right| \leq \sum_{t=0}^{\infty} \gamma^t |R|_{\max} = \frac{|R|_{\max}}{1-\gamma}$$

This completes the proof. □

Next, note that X is a closed set, whose interior is empty. In particular, if $S = \{s_0\}$ and $A = \{a_0, a_1\}$, we can sketch X as in Figure 6.3. Due to X not having any interior, it is not trivial to meaningfully define derivatives of E_{M,s_0}. So, if we want to say something about the smoothness of E_{M,s_0}, we will have to do a little bit more work.

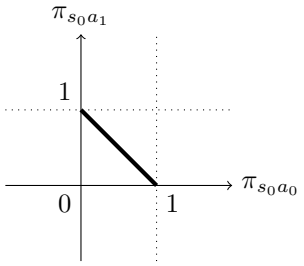


Figure 6.3: The policy space X, for $S = \{s_0\}$ and $A = \{a_0, a_1\}$, is depicted as the thick oblique line. The endpoints are included in this set.

In the example portrayed in Figure 6.3, we can see that it is enough to specify the horizontal coordinate, $\pi_{s_0a_0}$, to uniquely determine where we are in the space X. Hence, we can create a function κ, which takes $\pi_{s_0a_0}$ as input, and outputs the corresponding point in X, i.e., $(\pi_{s_0a_0}, 1 - \pi_{s_0a_0})$.

We can generalize this idea when S and A are larger, where our function κ acts on higher-dimensional spaces. To that end, we pick one particular action $a^* \in A$ arbitrarily, and we choose all the coefficients of π that do not involve a^*, i.e., the coefficients π_{sa} where $(s,a) \in S \times (A \setminus \{a^*\})$, such that they are located in the interval $[0,1]$, and such that for all $s \in S$, the sum of the coefficients π_{sa} over all a in $A \setminus \{a^*\}$ is also located in $[0,1]$. Then, for all $s \in S$, we can define $\pi_{sa^*} \in [0,1]$ to be such that the coefficients of π, for every $s \in S$, sum to 1. In this context, we let κ be the function that sends the π without a^* to the π with a^*. Hence, κ is the function that appropriately normalizes the policy π by setting the policy parameters involving a^*.

We formalize the function κ in the definition below.

Definition 6.4.5: Canonical embedding in the policy space
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process. We let $a^* \in A$ and define:

$$\Omega \in \left\{ x \in [0,1]^{S \times (A \setminus \{a^*\})} : \forall s \in S, 1 - \sum_{a \in A \setminus \{a^*\}} x_{sa} \in [0,1] \right\}$$

Then, we define $\kappa : \Omega \to X$ as follows, for all $s \in S$:

$$\forall a \in A \setminus \{a^*\}, \kappa(x)_{sa} = x_{sa} \qquad \text{and} \qquad \kappa(x)_{sa^*} = 1 - \sum_{a \in A \setminus \{a^*\}} x_{sa}$$

We refer to κ as the *canonical embedding in the policy space of* M.

It can easily be seen that κ indeed maps into X, as for all $s \in S$ and $x \in \Omega$, we have:

$$\sum_{a \in A} \kappa(x)_{sa} = \sum_{a \in A \setminus \{a^*\}} \kappa(x)_{sa} + \kappa(x)_{sa^*} = \sum_{a \in A \setminus \{a^*\}} \kappa(x)_{sa} + 1 - \sum_{a \in A \setminus \{a^*\}} \kappa(x)_{sa} = 1$$

Furthermore, it is not difficult to show that κ is infinitely smooth, as the derivatives are very easy to calculate. Note that as the coefficients of a vector $x \in (0,1)^{S \times A \setminus \{a^*\}}$ are indexed by elements from $S \times (A \setminus \{a^*\})$, we have derivatives with respect to these indices, i.e., derivatives ∂_{sa} where $sa \in S \times (A \setminus \{a^*\})$. These derivatives of κ are calculated in the lemma below.

Lemma 6.4.6: Smoothness of the canonical embedding in the policy space
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process, and let κ be the canonical embedding in the policy space of M. Then, we find, for all $(s,a), (s',a') \in S \times (A \setminus \{a^*\})$:

$$\partial_{sa}(\kappa_{s'a'}) = \delta_{ss'}\delta_{aa'} \qquad \text{and} \qquad \partial_{sa}(\kappa_{s'a^*}) = -\delta_{ss'}$$

and moreover, for all $k \in \mathbb{N} \setminus \{1\}$ and $\alpha \in (S \times (A \setminus \{a^*\}))^k$, we find for all $(s,a) \in S \times A$ that $\partial_\alpha(\kappa_{sa}) = 0$.

Proof. Observe that for all $x \in (0,1)^{S \times (A \setminus \{a^*\})}$, and $(s,a) \in S \times (A \setminus \{a^*\})$, we have $\kappa(x)_{sa} = x_{sa}$, and hence $\partial_{sa}(\kappa_{s'a'})$ is only non-zero if $s = s'$ and $a = a'$, in which case it is 1. On the other hand, if $s, s' \in S$ and $a \in A \setminus \{a^*\}$, then x_{sa} appears in $\kappa(x)_{s'a^*}$ with a minus sign if and only if $s = s'$, and hence $\partial_{sa}(\kappa_{s'a^*})$ is -1 if $s = s'$, and 0 otherwise. Finally, as all components of κ are linear functions, all higher-order derivatives vanish, completing the proof. □

Hence, we have shown that κ provides us with a smooth embedding of Ω into X. Moreover, Ω does have an open interior, and hence it makes sense to look at the derivatives of $E_{M,s_0} \circ \kappa$, and investigate the smoothness of this function. In order to do this, we need to explicitly express E_{M,s_0} in terms of its policy parameters, which is what we do in the following lemma.

Lemma 6.4.7: Dependence of the policy evaluation function on the policy parameters
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process, and let $s_0 \in S$. For every policy π for M, let $(S_t^\pi)_{t=0}^\infty$ and $(A_t^\pi)_{t=0}^\infty$ be random variables taking values in S and A, modeling the dynamics of the Markov decision process under the policy π. We define the following shorthand notation, for all $t \in \mathbb{N}_0$, $s \in S$ and $a \in A$:

$$P_{\pi,t,s,a} = \mathbb{P}[S_t^\pi = s, A_t^\pi = a]$$

Then:

$$E_{M,s_0}(\pi) = \sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t P_{\pi,t,s,a}$$

Moreover, for all $t \in \mathbb{N}_0$, $s \in S$ and $a \in A$, $P_{\pi,t,s,a}$ is a multivariate polynomial in the coefficients of π of total degree $t+1$.

Proof. Let $\pi \in X$. The first claim is easily verified:

$$E_{M,s_0}(\pi) = V_M^{(\pi)}(s_0) = \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t R(S_t^\pi)\right] = \sum_{t=0}^\infty \gamma^t \sum_{(s,a) \in S \times A} R(s,a) \mathbb{P}\left[S_t^\pi = s, A_t^\pi = a\right]$$

$$= \sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t P_{\pi,t,s,a}$$

Next, observe that we have, for all $s \in S$ and $a \in A$:

$$P_{\pi,0,s,a} = \mathbb{P}\left[S_0^\pi = s, A_0^\pi = a\right] = \delta_{ss_0}\pi_{sa}$$

Hence, $P_{\pi,0,s,a}$ is indeed a polynomial in the coefficients of π, of total degree 1. Moreover, we can rewrite $P_{\pi,t,s,a}$ recursively, as follows, for all $t \in \mathbb{N}$, $s \in S$ and $a \in A$:

$$\begin{aligned} P_{\pi,t,s,a} &= \mathbb{P}\left[A_t^\pi = a, S_t^\pi = s\right] = \mathbb{P}\left[A_t^\pi = a | S_t^\pi = s\right] \cdot \mathbb{P}\left[S_t^\pi = s\right] \\ &= \pi_{sa} \cdot \sum_{(s',a') \in S \times A} \mathbb{P}\left[S_t^\pi = s | S_{t-1}^\pi = s', A_{t-1}^\pi = a'\right] \cdot \mathbb{P}\left[S_{t-1}^\pi = s', A_{t-1}^\pi = a'\right] \\ &= \pi_{sa} \cdot \sum_{(s',a') \in S \times A} P_{s'a's} \cdot P_{\pi,t-1,s',a'} \end{aligned}$$

And so by induction, we find that $P_{\pi,t,s,a}$ is a polynomial in terms of the coefficients of π of total degree $t+1$ for all $t \in \mathbb{N}_0$. □

We can use this last lemma to prove some smoothness bounds on $E_{M,s_0} \circ \kappa$. However, in order for these smoothness bounds to hold globally, we must shrink the domain of this function. Specifically, we must make sure that the coefficients of the resulting policy are bounded away from 0 and 1. This is why we formally introduce the following set, where $\eta \geq 0$:

$$\Omega_\eta = \left\{ x \in (\eta, 1-\eta)^{S \times (A \setminus \{a^*\})} : \forall s \in S, 1 - \sum_{a \in A \setminus \{a^*\}} x_{sa} \in (\eta, 1-\eta) \right\}$$

Note that $\overline{\Omega_0}$ recovers the set Ω used in Definition 6.4.5. Now, we can prove some smoothness properties of $E_{M,s_0} \circ \kappa$, in the following theorem:

Theorem 6.4.8: Smoothness properties of the policy evaluation function under the canonical embedding in the policy space
Let $M = (S, A, P, R, \gamma)$ be a Markov decision process and $s_0 \in S$. Moreover, let κ be the canonical embedding of $\overline{\Omega_0}$ in the policy space X. Let $\eta \in (0, \frac{1}{2})$. For all $x \in \Omega_\eta$, $k \in \mathbb{N}_0$ and $\alpha \in (S \times (A \setminus \{a^*\}))^k$, we have:

$$|\partial_\alpha (E_{M,s_0} \circ \kappa)(x)| \leq \frac{|S||A||R|_{\max}}{\gamma(1-\gamma)} \cdot \left(\frac{2\gamma}{\eta(1-\gamma)} \right)^k \cdot k!$$

In other words, we find:

$$E_{M,s_0} \circ \kappa \in \mathcal{G}^1_{S \times (A \setminus \{a^*\}), \Omega_\eta, \frac{|S||A||R|_{\max}}{\gamma(1-\gamma)}, \frac{2\gamma}{\eta(1-\gamma)}}$$

Proof. Take $x \in \Omega_\eta$ arbitrarily, and let $k \in \mathbb{N}_0$ and $\alpha \in (S \times (A \setminus \{a^*\}))^k$. We take $s_1, \ldots, s_{|\alpha|} \in S$ and $a_1, \ldots, a_{|\alpha|} \in A$ such that for all $j \in \{1, \ldots, |\alpha|\}$, we have the state-action pair $s_j a_j = \alpha_j \in S \times A$. We find, using Lemma 6.4.7:

$$\partial_\alpha (E_{M,s_0} \circ \kappa)(x) = \partial_\alpha \left(\sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t (P_{\pi,t,s,a} \circ \kappa)(x) \right) = \sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t \partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x)$$

Next, due to Lemma 6.4.1, observe that for all $t \in \mathbb{N}_0$, $s \in S$ and $a \in A$:

$$\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x) = \sum_{\pi \in \Pi_{|\alpha|}} \sum_{\ell_1 \in S \times A} \cdots \sum_{\ell_{|\pi|} \in S \times A} \partial_{(\ell_1, \ldots, \ell_{|\pi|})} P_{\pi,t,s,a}(\kappa(x)) \cdot \prod_{j=1}^{|\pi|} \partial_{\alpha_{\pi_j}} \kappa_{\ell_j}(x)$$

Recall from Lemma 6.4.6 that all second and higher-order derivatives of κ vanish, and hence the only term of the outermost summation that does not vanish is the term where π only contains singletons. Hence, we obtain:

$$\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x) = \sum_{s'_1 a'_1 \in S \times A} \cdots \sum_{s'_k a'_k \in S \times A} \partial_{(s'_1 a'_1, \ldots, s'_k a'_k)} P_{\pi,t,s,a}(\kappa(x)) \cdot \prod_{j=1}^{|\alpha|} \partial_{s_j a_j} \kappa_{s'_j a'_j}(x)$$

The only terms that remain are the ones where for all $j \in \{1, \ldots, |\alpha|\}$, $s'_j a'_j = s_j a_j$, and $s'_j a'_j = s_j a^*$. Hence, we find:

$$\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x) = \sum_{a'_1 \in \{a_1, a^*\}} \cdots \sum_{a'_k \in \{a_k, a^*\}} \partial_{(s_1 a'_1, \ldots, s_k a'_k)} P_{\pi,t,s,a}(\kappa(x)) \cdot \prod_{j=1}^{|\alpha|} \partial_{s_j a_j} \kappa_{s_j a'_j}$$

Taking the absolute value on both sides and using the triangle inequality yields, using Lemma 6.4.6:

$$|\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x)| \leq \sum_{a'_1 \in \{a_1, a^*\}} \cdots \sum_{a'_k \in \{a_k, a^*\}} |\partial_{(s_1 a'_1, \ldots, s_k a'_k)} P_{\pi,t,s,a}(\kappa(x))| \cdot 1$$

But now observe that $P_{\pi,t,s,a}$ is a polynomial in the coefficients of π of total degree at most $t+1$, according to Lemma 6.4.7. Hence, we find for all $\beta \in (S \times A)^k$ and $y \in \kappa(\Omega_\eta)$:[4]

$$|\partial_\beta P_{\pi,t,s,a}(y)| \leq (t+2-k)_k \cdot \frac{|P_{\pi,t,s,a}(y)|}{\prod_{j=1}^k y_{\beta_j}} \leq \frac{(t+2-k)_k |P_{\pi,t,s,a}(y)|}{\eta^k} \leq \frac{(t+2-k)_k}{\eta^k}$$

Plugging this in yields:

$$|\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x)| \leq 2^k \cdot \frac{(t+2-k)_k}{\eta^k}$$

And so, plugging this into the expression for the derivatives of $E_{M,s_0} \circ \kappa$, we obtain:

$$\begin{aligned}
|\partial_\alpha (E_{M,s_0} \circ \kappa)(x)| &\leq \sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t |\partial_\alpha (P_{\pi,t,s,a} \circ \kappa)(x)| \\
&\leq \sum_{(s,a) \in S \times A} R(s,a) \sum_{t=0}^\infty \gamma^t \cdot \left(\frac{2}{\eta}\right)^k \cdot (t+2-k)_k \\
&\leq |S||A||R|_{\max} \cdot \left(\frac{2}{\eta}\right)^k \cdot \sum_{t=0}^\infty (t+2-k)_k \gamma^t
\end{aligned}$$

But the summation we can calculate, as follows:

$$\begin{aligned}
\sum_{t=0}^\infty (t+2-k)_k \gamma^t &= \gamma^{k-1} \sum_{t=0}^\infty (t+2-k)_k \gamma^{t+1-k} = \gamma^{k-1} \frac{\mathrm{d}^k}{\mathrm{d}\gamma^k} \left[\sum_{t=0}^\infty \gamma^{t+1}\right] = \gamma^{k-1} \frac{\mathrm{d}^k}{\mathrm{d}\gamma^k} \left[\frac{\gamma}{1-\gamma}\right] \\
&= \gamma^{k-1} \cdot \frac{\mathrm{d}^k}{\mathrm{d}\gamma^k} \left[\frac{1}{1-\gamma} - 1\right] = \gamma^{k-1} \cdot \left[\frac{k!}{(1-\gamma)^{k+1}} - \delta_{k0}\right] \\
&= \frac{1}{\gamma(1-\gamma)} \cdot \left(\frac{\gamma}{1-\gamma}\right)^k \cdot k! - \frac{\delta_{k0}}{\gamma}
\end{aligned}$$

And hence, we obtain:

$$|\partial_\alpha (E_{M,s_0} \circ \kappa)(x)| \leq \frac{|S||A||R|_{\max}}{\gamma(1-\gamma)} \cdot \left(\frac{2\gamma}{\eta(1-\gamma)}\right)^k \cdot k! - \frac{\delta_{k0}|S||A||R|_{\max}}{\gamma}$$

If $k = 0$, we can simplify the expression, which implies:

$$|E_{M,s_0} \circ \kappa)(x)| \leq \frac{|S||A||R|_{\max}}{\gamma} \left[\frac{1}{1-\gamma} - 1\right] = \frac{|S||A||R|_{\max}}{\gamma} \cdot \frac{\gamma}{1-\gamma} = \frac{|S||A||R|_{\max}}{1-\gamma} \leq \frac{|S||A||R|_{\max}}{\gamma(1-\gamma)}$$

[4]Here, we use the Pochhammer symbol. For every $a \in \mathbb{C}$ and $k \in \mathbb{N}$, this is defined as $(a)_k = a(a+1)(a+2)\cdots(a+k-1)$.

And so, for all $k \in \mathbb{N}_0$, we have:

$$|\partial_\alpha(E_{M,s_0} \circ \kappa)(x)| \leq \frac{|S||A||R|_{\max}}{\gamma(1-\gamma)} \cdot \left(\frac{2\gamma}{\eta(1-\gamma)}\right)^k \cdot k!$$

This completes the proof. □

The attentive reader might wonder why it is necessary to bound the coefficients of the policy away from 0 and 1 in the definition of Ω_η. It appears to be hard to provide an example of a Markov decision process for which the derivatives blow up near the boundary of Ω_0, indicating that bounding the coefficients away from 0 and 1 might not be necessary. Hence, it is very well possible that the above result can be improved upon, however doing so does not appear to be an easy task. This is why we will stick to this result for now, but improving on this result would be a very interesting topic of further research.

So, to recap, in this subsection we introduced the policy evaluation function, E_{M,s_0}, and we composed it with a function κ that naturally embeds a set with non-empty interior in the policy space. Then we investigated the smoothness of the composition $E_{M,s_0} \circ \kappa$, and we found the particular Gevrey-class of which the function is a member, whenever we restrict it to a smaller domain that bounds away the coefficients of 0 and 1.

Note that the image of $\kappa : \Omega_\eta \to X$ does not cover all of X, but it does cover all policies whose coefficients are in the interval $(\eta, 1-\eta)$. Hence, optimizing the function $E_{M,s_0} \circ \kappa$ over Ω_η should still yield a good approximation of the optimal policy for M starting from s_0. In the next subsection, we will attempt to use a gradient ascent algorithm to perform this optimization procedure. To that end, though, we have to estimate the gradient, which we will do using Gilyén et al.'s gradient estimation algorithm, Algorithm 4.3.9.

6.4.3 Quantum algorithm for quantum policy optimization

In this section, we will devise the quantum policy optimization algorithm. Its main component will be the gradient estimation algorithm, i.e., Algorithm 4.3.9, applied to the function $E_{M,s_0} \circ \kappa$, introduced in the previous subsection. The smoothness bounds proven there indicate that we will need to use $\sigma = 1$ in this algorithm.

The gradient estimation algorithm requires a phase oracle implementation of the objective function, though. Recall that in Section 6.3, we have already constructed a quantum circuit that evaluates the value function as a probability oracle. Hence, we will first of all construct a circuit that converts a probability oracle into a phase oracle.

Gilyén et al. state a way to convert a probability oracle into a phase oracle in [GAW17], Chapter 4. Our construction will be mainly based on theirs, and hence will also include techniques from [CKS17]. The first step is a polynomial approximation of the function $x \mapsto e^{i\sin^2(x)}$, provided in the lemma below.

Lemma 6.4.9: Approximation of $x \mapsto e^{i\sin^2(x)}$
Let $\varepsilon > 0$ and $N = \lceil \max\{2e, \log(\frac{1}{\varepsilon})\} \rceil$. Define:

$$\beta_n = \sum_{k=|n|}^{N} \binom{2k}{k+n} \frac{(-1)^n i^k}{k! 2^{2k}}$$

Then, we find, for all $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$:

$$\left| e^{i\sin^2(x)} - \sum_{n=-N}^{N} \beta_n e^{2inx} \right| \leq \varepsilon$$

Moreover:

$$\sum_{n=-N}^{N} |\beta_n| \leq 2(1 + e^{e/2}) e^{1/4}$$

Proof. For all $y \in \mathbb{R}$, we have:

$$e^{iy} = \sum_{n=0}^{\infty} \frac{i^n y^n}{n!}$$

Hence, we obtain, for all $y \in [-1, 1]$:

$$\begin{aligned} \left| e^{iy} - \sum_{n=0}^{N} \frac{i^n y^n}{n!} \right| &\leq \sum_{n=N+1}^{\infty} \frac{|y|^n}{n!} \leq \sum_{n=N+1}^{\infty} \frac{1}{n!} \leq \frac{1}{N!} \cdot \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n = \frac{1}{N!} \leq \frac{1}{\sqrt{2\pi} N^{N+\frac{1}{2}} e^{-N}} \leq \left(\frac{e}{N}\right)^N \\ &\leq \left(\frac{1}{2}\right)^{\log(1/\varepsilon)} = \varepsilon \end{aligned}$$

And so, for all $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, we have:

$$\left| e^{i\sin^2(x)} - \sum_{n=0}^{N} \frac{i^n \sin^{2n}(x)}{n!} \right| \leq \varepsilon$$

Moreover, we have, by elementary rewriting:

$$\begin{aligned} \sum_{n=0}^{N} \frac{i^n \sin^{2n}(x)}{n!} &= \sum_{n=0}^{N} \frac{i^n}{n!} \left(\frac{e^{ix} - e^{-ix}}{2i}\right)^{2n} = \sum_{n=0}^{N} \frac{i^n}{n!(2i)^{2n}} \sum_{k=0}^{2n} \binom{2n}{k} e^{ikx} \cdot (-1)^{2n-k} e^{-i(2n-k)x} \\ &= \sum_{n=0}^{N} \frac{(-i)^n}{2^{2n} n!} \sum_{k=0}^{2n} \binom{2n}{k} (-1)^k e^{(-2n+2k)ix} = \sum_{n=0}^{N} \frac{i^n}{2^n n!} \sum_{k=-n}^{n} \binom{2n}{k+n} (-1)^k e^{2kix} \\ &= \sum_{k=-N}^{N} e^{2ikx} \sum_{n=|k|}^{N} \binom{2n}{n+k} \frac{(-1)^k i^n}{2^{2n} n!} = \sum_{k=-N}^{N} \beta_k e^{2ikx} \end{aligned}$$

And so, putting everything together, we obtain:

$$\left| e^{i\sin^2(x)} - \sum_{n=-N}^{N} \beta_n e^{2inx} \right| \leq \varepsilon$$

Finally, recall that for all binomial coefficients, we have the following upper bound, for all $a, b \in \mathbb{N}_0$:

$$\binom{a}{b} \leq \left(\frac{ea}{b}\right)^b$$

Hence, for the non-negative integers $k, n \in \mathbb{N}_0$ with $k > n$, we find:

$$\binom{2k}{k+n} \cdot \frac{1}{k!2^{2k}} = \binom{2k}{k-n} \cdot \frac{1}{k!2^{2k}} \leq \left(\frac{2ek}{k-n}\right)^{k-n} \cdot \frac{1}{k!2^{2k}} = \left(\frac{2e}{1-\frac{n}{k}}\right)^{k-n} \cdot \frac{1}{k!2^{2k}} \leq (2e)^{-n} \cdot \frac{(e/2)^k}{k!}$$

Thus, we obtain, for all $n \in \{-N, -N+1, \ldots, N-1, N\}$:

$$\begin{aligned} |\beta_n| &\leq \sum_{k=|n|}^{N} \binom{2k}{k+|n|} \cdot \frac{1}{k!2^{2k}} \leq \frac{1}{|n|! \cdot 2^{2|n|}} + \sum_{k=|n|+1}^{N} \frac{(e/2)^k}{(2e)^{|n|}k!} \leq \frac{1}{|n|!2^{2|n|}} + \frac{(e/2)^{|n|}}{|n|!(2e)^{|n|}} \sum_{k=0}^{\infty} \frac{(e/2)^k}{k!} \\ &= \frac{1}{|n|!4^{|n|}} + \frac{e^{e/2}}{4^{|n|}|n|!} = \frac{1+e^{e/2}}{|n|!4^{|n|}} \end{aligned}$$

And so, we have:

$$\sum_{n=-N}^{N} |\beta_n| \leq 2\sum_{n=0}^{N} |\beta_n| \leq 2(1+e^{e/2}) \sum_{n=0}^{N} \frac{1}{n!} \cdot \left(\frac{1}{4}\right)^n = 2(1+e^{e/2})e^{1/4}$$

This completes the proof. □

Now, we show how we can use this approximation to construct a phase oracle from a probability oracle. This result is due to [GAW17], Chapter 4, and it uses techniques from [CKS17], specifically Lemma 6.

Circuit 6.4.10: Phase oracle from a probability oracle
Description: This circuit approximately implements a phase oracle, using a number of queries to a probability oracle that is logarithmic in the precision.
Parameters: $\delta > 0$.
Derived constants:
1. $N = \lceil \max\{2e, \log(\frac{21}{\delta})\} \rceil$.
2. $(\beta_k)_{k=-N}^{N}$, as defined in Lemma 6.4.9.

Oracle circuits: A probability oracle P, acting on $n+1$ qubits, having the following action:

$$P : |0\rangle^{\otimes(n+1)} \mapsto \sqrt{p}\,|\psi\rangle\,|0\rangle + \sqrt{1-p}\,|\phi\rangle\,|1\rangle$$

where $|\phi\rangle$ and $|\psi\rangle$ are arbitrary n-qubit quantum states.
Number of qubits: $2N + n + 2$.
Oracle query complexity: P is queried a total number of $4N \cdot 21$ times.
Circuit: Denote the following circuit by Q, acting on all of the $2N + n + 2$ qubits.
1. We implement the following mapping on the first $2N$ qubits.

$$V : |0\rangle^{\otimes(2N)} \mapsto \frac{1}{\sqrt{\|\beta\|_1}} \sum_{k=-N}^{N} \sqrt{\beta_k} \begin{cases} |1\rangle^{\otimes k} \otimes |0\rangle^{\otimes(2N-k)}, & \text{if } k \geq 0 \\ |0\rangle^{\otimes(2N+k)} \otimes |1\rangle^{\otimes(-k)}, & \text{otherwise} \end{cases}$$

2. Next, we define the following circuit, acting on $n+1$ qubits, and controlled by one extra qubit, which we denote by $C(G)$:
   (a) Apply the probability oracle, P.
   (b) Apply XZX to the last qubit, where the Z is controlled by the control qubit.
   (c) Apply the circuit of step (a) in reverse.
   (d) Apply the circuit $C(R_0)$, Circuit 3.5.10, to all of the $n+1$ qubits, controlled by the control qubit.

   Now, apply $C(G)$ to the last $n+1$ qubits, controlled by the first qubit. Repeat this on the second, third, etc., up to the Nth qubit. Do the same for the second batch of N qubits, but now implement the reverse, $C(G)^*$.
3. We rotate in the phases, i.e., conditional on the first k qubits being in state $|1\rangle$, we multiply by a factor $e^{i\,\mathrm{Arg}(\beta_k)}$, and similarly for the last k qubits of the first $2N$ qubits being in state 1, we multiply by a factor $e^{i\,\mathrm{Arg}(\beta_{-k})}$.[a]
4. Apply the reverse of V to the first $2N$ qubits.
5. Apply the following mapping to the $(2N+1)$st qubit:

$$R : |0\rangle \mapsto \sin\left(\frac{\pi}{42}\right)\|\beta\|_1\,|0\rangle + \sqrt{1 - \left(\sin\left(\frac{\pi}{42}\right)\|\beta\|_1\right)^2}\,|1\rangle$$

We use this circuit as the setup circuit in the amplitude amplification algorithm, Circuit 3.5.9, and we run it with $k = 10$. The reflection circuit, we take to be R_0, applied to the first $2N+1$ qubits.

[a] For all $z \in \mathbb{C} \setminus \{0\}$, $\mathrm{Arg}(z)$ denotes the principle value of the argument of z, i.e., if we write $z = re^{i\phi}$, with $r > 0$ and $\phi \in (-\pi, \pi]$, then $\mathrm{Arg}(z) = \phi$.

Proof that this indeed implements a phase oracle. Define $\theta = \arcsin(\sqrt{p})$. We observe that P implements the following mapping:

$$P : |0\rangle^{\otimes(n+1)} \mapsto \sin(\theta)\,|\psi\rangle\,|0\rangle + \cos(\theta)\,|\phi\rangle\,|1\rangle$$

Hence, $G = R_0 P^* (I_2^{\otimes n} \otimes XZX) P$ has the following mapping, for all $\phi \in \mathbb{R}$:

$$G : P^*(\sin(\phi)\,|\psi\rangle\,|0\rangle + \cos(\phi)\,|\phi\rangle\,|1\rangle) \mapsto P^*(\sin(\phi + 2\theta)\,|\psi\rangle\,|0\rangle + \cos(\phi + 2\theta)\,|\phi\rangle\,|1\rangle)$$

which we can prove in exactly the same way as in the proof of Circuit 3.5.8. But this implies that G is a rotation operator on the subspace of the $(n+1)$-qubit state space, spanned by the vectors $P^*(|\psi\rangle|0\rangle)$ and $P^*(|\phi\rangle|1\rangle)$. Hence, with respect to a suitable basis in this subspace, we can rewrite G as follows:

$$G = \begin{bmatrix} e^{2i\theta} & 0 \\ 0 & e^{-2i\theta} \end{bmatrix}$$

The terms in the state after step 1 of circuit Q, when applied to the state $|0\rangle^{\otimes(2N+n+2)}$, have some interesting properties. If k is positive, then the first k qubits are set to state $|1\rangle$, and if k is negative, then the last k qubits of the first $2N$ qubits are set to $|1\rangle$. Hence, we can implement G^k by calling either $C(G)$ or $C(G)^*$, which is what we do in step 2. Hence, the state of the entire system after step 2 is given by:

$$\frac{1}{\sqrt{\|\beta\|_1}} \sum_{k=-N}^{N} \sqrt{\beta_k} \left[\begin{cases} |1\rangle^{\otimes k} \otimes |0\rangle^{\otimes(2N-k)}, & \text{if } k \geq 0 \\ |0\rangle^{\otimes(2N+k)} \otimes |1\rangle^{\otimes(-k)}, & \text{otherwise} \end{cases} \right] \otimes |0\rangle \otimes G^k |0\rangle^{\otimes(n+1)}$$

Now, after step 3, the state becomes:

$$\frac{1}{\sqrt{\|\beta\|_1}} \sum_{k=-N}^{N} \sqrt{\beta_k} \cdot e^{i \arg(b_k)} \left[\begin{cases} |1\rangle^{\otimes k} \otimes |0\rangle^{\otimes(2N-k)}, & \text{if } k \geq 0 \\ |0\rangle^{\otimes(2N+k)} \otimes |1\rangle^{\otimes(-k)}, & \text{otherwise} \end{cases} \right] \otimes |0\rangle \otimes G^k |0\rangle^{\otimes(n+1)}$$

And after step 4, we obtain:

$$\frac{1}{\|\beta\|_1} \sum_{k=-N}^{N} \beta_k |0\rangle^{\otimes(2N)} \otimes |0\rangle \otimes G^k |0\rangle^{\otimes(n+1)} + |\Psi\rangle$$

where $\left\| ((|0\rangle\langle 0|)^{\otimes(2N+1)} \otimes I_2^{\otimes(n+1)}) |\Psi\rangle \right\| = 0$. Finally, observe that $\sin(\pi/42)\|\beta\|_1 \leq \sin(\pi/42) \cdot 2(1 + e^{e/2})e^{1/4} < 1$ and hence step 5 is legitimate. The final state of the system becomes, after step 5:

$$\sin\left(\frac{\pi}{42}\right) |0\rangle^{\otimes(2N+1)} \otimes \sum_{k=-N}^{N} \beta_k G^k |0\rangle^{\otimes(n+1)} + |\Psi\rangle$$

But note that in the appropriate subspace, where $|0\rangle^{\otimes(n+1)}$ is located in, we can write G as a diagonal matrix with entries $e^{2i\theta}$ and $e^{-2i\theta}$. But this implies that we can rewrite, with respect to this basis:

$$\sum_{k=-N}^{N} \beta_k G^k = \sum_{k=-N}^{N} \beta_k \begin{bmatrix} e^{2i\theta} & 0 \\ 0 & e^{-2i\theta} \end{bmatrix}^k = \begin{bmatrix} \sum_{k=-N}^{N} \beta_k e^{2ik\theta} & 0 \\ 0 & \sum_{k=-N}^{N} \beta_k e^{-2ik\theta} \end{bmatrix}$$

Up to operator norm $\delta/21$, this is equal to:

$$\sum_{k=-N}^{N} \beta_k G^k \approx \begin{bmatrix} e^{i\sin^2(\theta)} & 0 \\ 0 & e^{i\sin^2(-\theta)} \end{bmatrix} = e^{ip} I$$

But that means that, up to operator norm $\delta/21$, this summation equals the identity operator, shifted by p in phase. Hence the resulting state of the system after step 5 is:

$$\sin\left(\frac{\pi}{42}\right) e^{ip} |0\rangle^{\otimes(2N+1)} \otimes |0\rangle^{\otimes(n+1)} + |\Psi\rangle$$

So, we have shown that the circuit Q has the following action, up to norm error $\delta/21$:

$$Q : |0\rangle^{\otimes(2N+n+2)} \mapsto \sin\left(\frac{\pi}{42}\right) e^{ip} |0\rangle^{\otimes(2N+1)} \otimes |0\rangle^{\otimes(n+1)} + |\Psi\rangle$$

This circuit, we can now use in the amplitude amplification circuit. As we take 10 steps, we query Q a total of 21 times. Thus, we obtain that the resulting circuit has the following action, up to norm error δ:

$$|0\rangle^{\otimes(2N+n+2)} \mapsto \sin\left(\frac{(2\cdot 10+1)\pi}{42}\right) e^{ip}\,|0\rangle^{\otimes(2N+n+2)} = e^{ip}\,|0\rangle^{\otimes(2N+n+2)}$$

This completes the proof. □

Note that we can very easily implement the above quantum circuit in a controlled manner. The idea is to implement the circuits V and R in a controlled manner, which does not influence the oracle query complexity. Note that step 2 does not do anything if step 1 does not modify the state $|0\rangle^{\otimes(2N)}$. Step 3 only adds an extra phase $\exp(i\beta_0)$, and hence we will have to correct for this phase at the control qubit. This is enough to implement Q in a controlled manner.

In order to perform the amplitude amplification in a controlled manner as well, we can use a controlled version of Q in the setup step of the amplitude amplification algorithm. Then, in the amplification step, we can repeatedly use controlled versions of the Grover iterate, which we provided in Circuit 3.5.11. These modifications do not increase the number of times we call the probability oracle, and hence the oracle query complexity shown in Circuit 6.4.10 remains valid in the controlled case as well.

Now that we have shown how we can implement a phase oracle query from a probability oracle query, we can use it to build a phase oracle that evaluates our objective function $E_{M,s_0} \circ \kappa$. This we do in the quantum circuit below.

Circuit 6.4.11: Phase oracle of a composition of the policy evaluation function
Description: Given a Markov decision process $M = (S, A, P, R, \gamma)$ and an initial state $s_0 \in S$, this circuit implements a phase oracle of the function

$$x \mapsto \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}}(E_{M,s_0} \circ \kappa)(x)$$

Parameters:
1. The accuracy parameter of the resulting state, $\delta > 0$.
2. The accuracy parameter of the function to evaluate, $\varepsilon > 0$.
3. The discount factor $\gamma \in [0, 1)$.
4. An absolute bound on the reward function, $|R|_{\max}$.
5. The bound on the parameters of the policy, $\eta > 0$.
6. A set of orthogonal K-qubit states $\{|x\rangle : x \in G \subseteq \Omega_\eta\}$.
7. Orthogonal sets of n_S-qubit states $\{|s\rangle : s \in S\}$ and n_A-qubit states $\{|a\rangle : a \in A\}$.

Derived constants:
1. $N = \lceil \max\{2e, \log(\frac{42}{\delta})\} \rceil$.
2. $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.

Oracle circuits:
1. A quantum circuit $\mathcal{P}$ that implements the state-action probability matrix:

$$\mathcal{P} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_S} \mapsto \sum_{s' \in S} \sqrt{P_{sas'}} \, |s\rangle \otimes |a\rangle \otimes |s'\rangle$$

2. A controlled quantum circuit $C(\mathcal{R})$, where $\mathcal{R}$ implements the state-action reward function as a phase oracle:

$$\mathcal{R} : |s\rangle \otimes |a\rangle \mapsto e^{i\tilde{R}(s,a)} |s\rangle \otimes |a\rangle \qquad \text{where} \qquad \tilde{R}(s,a) = \frac{1-\gamma}{2|R|_{\max}} R(s,a)$$

3. A quantum circuit Π, acting on $K + n_S + n_A$ qubits, that implements the policy $\kappa(x)$ for all $x \in G \subseteq \Omega$, as follows:

$$\Pi : |x\rangle \otimes |s\rangle \otimes |0\rangle^{\otimes n_A} \mapsto |x\rangle \otimes \sum_{a \in A} \sqrt{\kappa(x)_{sa}} \, |s\rangle \otimes |a\rangle$$

Number of qubits: $K + 2N + T(n_S + n_A) + 4$.
Oracle complexity: The query complexity of all three circuits satisfies:

$$\tilde{O}(T) = \tilde{\mathcal{O}}\left(\frac{1}{1-\gamma}\right)$$

Circuit: Circuit 6.4.10, with the following settings.
1. Let the probability oracle circuit be Circuit 6.2.6, with the following settings:
   (a) Let the state probability oracle $\mathcal{P}$ in that circuit be the operation $(I_2^{\otimes(n_S+n_A)} \otimes \Pi)(\mathcal{P} \otimes I_2^{\otimes n_A})$.
   (b) Let the reward circuit $\mathcal{R}$ be the same as in this circuit.
   (c) Modify step 1. Instead of constructing $|s_0\rangle$ from the all-zero state, construct $|s_0\rangle \otimes \sum_{a \in A} \sqrt{\kappa(x)_{sa}} \, |a\rangle$ instead. This can be done using one call to Π.
   (d) Let the accuracy parameter of Circuit 6.2.5 be $\delta/(168N)$. Furthermore, use the same ε, γ and $|R|_{\max}$.
2. Let accuracy parameter be $\delta/2$.

Proof that the error induced is no more than δ. Observe that N is the same constant as the one that is being chosen in Circuit 6.4.10, when the accuracy parameter is set to $\delta/2$. Hence, Circuit 6.2.6 is called $42N$ times and hence from Lemma B.2, we can deduce that the total error there is upper bounded by:

$$2 \cdot \frac{\delta}{168N} \cdot 42N = \frac{\delta}{2}$$

And the error induced by Circuit 6.4.10 itself is also upper bounded by $\delta/2$. Hence, again using Lemma B.2, the total error is upper bounded by δ. This completes the proof. □

Proof of the oracle query complexity. The circuits $\mathcal{P}$ and $\mathcal{R}$ and Π are called $\tilde{O}(T)$ times in Circuit 6.2.6. Circuit 6.4.10 only introduces extra logarithmic factors, hence the total query complexity to all oracle circuits is $\tilde{O}(T)$. We have proven in the analysis of the query complexity of Algorithm 6.2.7 that $\tilde{O}(T) = \tilde{O}(1/(1-\gamma))$, which completes the proof. □

Proof that Circuit 6.4.11 indeed implements the function that is stated in the description. Observe that we have proven before that Circuit 6.4.10 indeed implements a phase oracle from a probability oracle. The probability oracle supplied is Circuit 6.2.6, and this process implements the following function up to accuracy ε:

$$\frac{1}{2} + \frac{1}{4} \sum_{t=0}^{\infty} \gamma^t \sum_{(s,a) \in S \times A} \mathbb{P}\left[S_t = s, A_t = a\right] \tilde{R}(s,a) = \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\right] = \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}} V_M^{(\kappa(\mathbf{x}))}(s_0)$$

Hence, indeed, the function that was stated in the description of the quantum circuit is indeed implemented up to accuracy ε, up to δ-norm. This completes the proof. □

Note that this circuit is essentially an implementation of Circuit 6.4.10. Hence, with the modifications provided directly after the introduction of that circuit, we can implement this circuit in a controlled fashion as well, without changing the oracle query complexity from the one presented in the box above.

Rather than the function $E_{M,s_0} \circ \kappa$, the quantum circuit above implements the following function:

$$x \mapsto \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}} (E_{M,s_0} \circ \kappa)(x)$$

This phase oracle, we can now use in Gilyén et al.'s quantum gradient estimation algorithm, Algorithm 4.3.9. The details are shown in the following box.

Algorithm 6.4.12: Gradient estimation of composition of policy evaluation function
Description: Given a Markov decision process $M = (S, A, P, R, \gamma)$ and a starting state $s_0 \in S$, this algorithm determines the gradient of the function $E_{M,s_0} \circ \kappa$ in some point $\mathbf{a} \in \Omega_\eta$ up to ε accuracy in the ℓ^∞-norm.
Parameters:
1. The dimension of the space input, $d > 0$.
2. The accuracy of the gradient estimate, $\varepsilon > 0$.
3. The discount factor, $\gamma \in [\frac{1}{2}, 1)$.
4. The point where the gradient is to be evaluated, $\mathbf{a} \in \Omega_\eta$.
5. The bound on the parameters of the policy, $\eta > 0$.
6. An absolute bound on the reward function, $|R|_{\max}$.
7. Orthogonal sets of K-, n_S- and n_A-qubit states $\{|\mathbf{x}\rangle : \mathbf{x} \in G \subseteq \Omega\}$, $\{|s\rangle : s \in S\}$ and $\{|a\rangle : a \in A\}$.

Derived constants:
1. $c = \frac{4\gamma}{\eta(1-\gamma)}$, $\varepsilon' = \frac{\gamma(1-\gamma)\varepsilon}{8|R|_{\max}|S||A|}$, $m = \lceil \log(\frac{cd}{\varepsilon'}) \rceil$, $r = \frac{(81 \cdot 8 \cdot 42\pi cmd/\varepsilon)^{-\frac{1}{2m}}}{9cmd}$.
2. $S = 2\pi \cdot 2^{\lceil \log(\frac{4}{r\varepsilon'}) \rceil}$, $\delta = \frac{1}{2000 \cdot (12mS + 288S(2\ln(m)+2))}$, $N = \lceil \frac{2\log(3d)}{\log(3)} \rceil$.
3. $n = \lceil \log\left(\frac{4}{r\varepsilon'}\right) \rceil + \lceil \log(3rc) \rceil$, $\delta' = \frac{1}{12S[2m+48\cdot(2\ln(m)+2)]\cdot[12+12\max\lceil \log(1/\delta)\rceil,1]}$.
4. $\delta_1 = \frac{\delta'}{4(12+12\max\{\lceil \log(1/\delta_3)\rceil,1\})}$, $\delta_2 = \frac{\delta'}{4(12+12\max\{\lceil \log(1/\delta_3)\rceil,1\})}$, $\delta_3 = \frac{\delta'}{12}$.
5. $N_0 = \lceil \max\{2e, \log(\frac{42}{\delta_1})\} \rceil$, $T = \left\lceil \frac{\log\left(\frac{\varepsilon(1-\gamma)}{2|R|_{\max}}\right)}{\log(\gamma)} \right\rceil$.

Oracle circuits: The same ones as in Circuit 6.4.11.
Output: A vector $\mathbf{g} \in \mathbb{R}^d$ such that $\|\mathbf{g} - \nabla(E_{M,s_0} \circ \kappa)(\mathbf{a})\|_\infty \leq \varepsilon$.
Number of qubits: $nd + (\lceil \log(m) \rceil + n)d + K + 2N_0 + T(n_S + n_A) + 4$.
Oracle complexity: The number of times $\mathcal{P}$, $\mathcal{R}$ and Π are queried satisfies:

$$\tilde{O}\left(\frac{|S|^2|A|^2|R|_{\max}}{\varepsilon\eta(1-\gamma)^3}\right)$$

Success probability: Lower bounded by 7/12.
Algorithm: We use the gradient estimation algorithm, Algorithm 4.3.9, with the following settings:
1. The controlled phase oracle circuit is the fractional phase query circuit, Circuit 4.2.18, run with the following settings:
   (a) The controlled phase oracle circuit is Circuit 6.4.11, implemented in a controlled manner, with the following settings:
      i. The phase is altered by a factor $e^{-i/2}$, to get rid of the term $\frac{1}{2}$ in the description of Circuit 6.4.11. This can for example be achieved by adding a gate that acts as $|j\rangle \mapsto e^{ij/2}|j\rangle$ for all $j \in \{0, 1\}$, to the control qubit of the controlled phase oracle circuit.
      ii. The accuracy parameter of the resulting state is δ_1.
      iii. The accuracy parameter of the function to evaluate is δ_2.
      iv. Use the same values for the discount factor γ and the absolute bound on the reward function $|R|_{\max}$.
   (b) The multiplication factor is $\frac{\gamma}{|S||A|}$.
   (c) The accuracy parameter to the resulting state is δ_3.
2. Use the parameters $p = \infty$, $\sigma = 1$. Use the same values for c, d, and $\mathbf{a}$, and take for the value of ε in Algorithm 4.3.9 the value ε' defined here.

Multiply the output vector by $\frac{8|S||A||R|_{\max}}{\gamma(1-\gamma)}$ and return the result.

Proof of the oracle complexity. According to Algorithm 4.3.9 with $p = \infty$ and $\sigma = 1$, the number of calls to the fractional phase query circuit satisfies:

$$\tilde{O}\left(\frac{cd}{\varepsilon'}\right) = \tilde{O}\left(\frac{\gamma|S|(|A|-1)|R|_{\max}|S||A|}{\varepsilon\eta\gamma(1-\gamma)^2}\right) = \tilde{O}\left(\frac{|S|^2|A|^2|R|_{\max}}{\varepsilon\eta(1-\gamma)^2}\right)$$

Every fractional phase query queries the phase oracle a number of times that scales only logarithmically in the input parameters. Moreover, the phase oracle is implemented using only $\tilde{O}(T) = \tilde{O}(1/(1-\gamma))$ applications of $\mathcal{P}$, $\mathcal{R}$ and Π. Hence, their total query complexity becomes:

$$\tilde{O}\left(\frac{|S|^2|A|^2|R|_{\max}}{\varepsilon\eta(1-\gamma)^3}\right)$$

This completes the proof. □

Proof of the lower bound on the success probability of Algorithm 6.4.12. Note that the probability oracle implements the following function:

$$\mathbf{x} \mapsto \frac{1}{2} + \frac{1-\gamma}{8|R|_{\max}} E_{M,s_0}(\kappa(\mathbf{x}))$$

But the term $\frac{1}{2}$ is removed in part (a). Hence, the resulting phase oracle implements the function:

$$\mathbf{x} \mapsto \frac{1-\gamma}{8|R|_{\max}} E_{M,s_0}(\kappa(\mathbf{x}))$$

And so the fractional phase query implements the following function:

$$\mathbf{x} \mapsto \frac{\gamma(1-\gamma)}{8|R|_{\max}|S||A|} E_{M,s_0}(\kappa(\mathbf{x}))$$

Thus, if Algorithm 4.3.9 succeeds, then it returns a vector $\mathbf{h}$ which differs from the gradient of the above function evaluated in $\mathbf{a}$ by at most $\varepsilon' = \frac{\gamma(1-\gamma)\varepsilon}{8|R|_{\max}|S||A|}$ in the ℓ^∞-norm. Hence, the resulting vector $\mathbf{g} = \frac{8|R|_{\max}|S||A|}{\gamma(1-\gamma)}\mathbf{h}$ differs from the gradient of $E_{M,s_0} \circ \kappa$ by at most ε in the ℓ^∞-norm.

We already observed that $E_{M,s_0} \circ \kappa \in \mathcal{G}^1_{S\times(A\setminus\{a^*\}),\Omega_{\frac{\eta}{2}},\frac{|S||A||R|_{\max}}{\gamma(1-\gamma)},\frac{4\gamma}{\eta(1-\gamma)}}$. Hence, using Theorem 4.1.4, we find that the above function is an element of $\mathcal{G}^1_{S\times(A\setminus\{a^*\}),\Omega_{\frac{\eta}{2}},\frac{1}{8},c}$. Moreover, observe, as $\gamma \geq \frac{1}{2}$:

$$r = \frac{(81\cdot 8\cdot 42\pi cmd/\varepsilon)^{-\frac{1}{2m}}}{9cmd} \leq \frac{1}{9cmd} = \frac{\eta(1-\gamma)}{36md\gamma} \leq \frac{\eta}{36md}$$

Hence, the function will be only be evaluated at points in a square lattice with side length $\eta/(36d)$ around $\mathbf{a}$. This means that the points $\mathbf{x}$ where the function is evaluated satisfy $\|\mathbf{x}-\mathbf{a}\|_1 \leq \eta/36 < \eta/2$, and hence $\mathbf{a} \in \Omega_\eta \Rightarrow \mathbf{x} \in \Omega_{\eta/2}$, which is in the domain of $E_{M,s_0} \circ \kappa$ for which the appropriate smoothness bound holds.

Now, observe that the controlled phase oracle circuit, Circuit 6.4.11, is called $12 + 12\max\{\lceil\log(1/\delta_3)\rceil, 1\}$ times by the fractional phase oracle circuit. Every time it is called, it introduces a norm error of $\delta_1 + \delta_2$, as the error in the function value can just as well be viewed as a norm error. Hence, the total error introduced by the fractional phase oracle circuit is, using Lemma B.2:

$$\begin{aligned}(12 + 12\max\{\lceil\log(1/\delta_3)\rceil, 1\})\cdot(\delta_1+\delta_2) + 6\delta_3 &= \frac{\delta'}{4} + \frac{\delta'}{4} + 6\cdot\frac{\delta'}{12} = \delta' \\ &= \frac{1}{12S[2m + 48\cdot(2\ln(m)+2)]\cdot[12 + 12\max\lceil\log(1/\delta)\rceil, 1]}\end{aligned}$$

This fractional phase query is called a total number of $S[2m + 48\cdot(2\ln(m)+2)]\cdot[12 + 12\max\lceil\log(1/\delta)\rceil, 1]$ times. So, the total additive norm error is, again using Lemma B.2:

$$\frac{S[2m + 48\cdot(2\ln(m)+2)]\cdot[12 + 12\max\lceil\log(1/\delta)\rceil, 1]}{12S[2m + 48\cdot(2\ln(m)+2)]\cdot[12 + 12\max\lceil\log(1/\delta)\rceil, 1]} \leq \frac{1}{12}$$

And hence, using Lemma B.1, we observe that the success probability of Algorithm 4.3.9 is altered by at most $\frac{1}{12}$, which implies that the success probability is lower bounded by:

$$\frac{2}{3} - \frac{1}{12} = \frac{7}{12}$$

This completes the proof. □

Hence, the above quantum algorithm, Algorithm 6.4.12, provides us with a way to estimate the gradient of the function $E_{M,s_0} \circ \kappa$, up to precision ε in ℓ^∞-norm. Moreover, the query complexity is polynomial in the cardinality of the state and action spaces.

Note that in order for the domain, Ω_η, to be non-empty, we must choose $\eta \leq 1/(|S||A|)$. Hence, we will obtain an oracle query complexity that scales at least as follows:

$$\tilde{O}\left(\frac{|S|^3|A|^3|R|_{\max}}{\varepsilon(1-\gamma)^3}\right)$$

Recall that at the end of Subsection 6.4.2, we argued that it might be possible to improve the bounds on the higher-order derivatives of $E_{M,s_0} \circ \kappa$. This means that it might be possible to remove the dependence on η from the above query complexity, and it would also allow for an algorithm that is capable of estimating the gradient on the entire domain of $E_{M,s_0} \circ \kappa$. Moreover, it would decrease the dependence of the query complexity of the algorithm on $|S|$ and $|A|$ by one order.

Now that we have a quantum algorithm that estimates the gradient of the function $E_{M,s_0} \circ \kappa$, we can use this algorithm as a subroutine in a gradient ascent algorithm. The idea is to randomly select a vector $\mathbf{x}$ in Ω_0. Next, one can invoke the gradient estimation algorithm, Algorithm 6.4.12, to estimate the direction of steepest ascent in Ω_0, say $\mathbf{g} \in \mathbb{R}^{S\times(A\setminus\{a^*\})}$. Then, one can adjust $\mathbf{x}$ into this direction, i.e., set $\mathbf{x} := \mathbf{x} + \alpha\mathbf{g}$, where α is some small parameter, commonly referred to as the *learning rate*. This procedure can be iterated, to obtain policies π that yield higher and higher values of M under π.

In every step, though, as $\mathbf{x}$ moves one can choose η again. With this approach, whenever $\mathbf{x}$ converges closer to the boundary of the domain Ω_η, the number of oracle queries will increase in the gradient estimation step. On the other hand, one can choose to stay in the domain Ω_η, and look for the optimal policy that one can find by selecting vectors $\mathbf{x}$ in this domain.

In addition, one has the option of composing the function $E_{M,s_0} \circ \kappa$ with another function, say $\lambda : \Phi \to \Omega_\eta$, where $\Phi \subseteq \mathbb{R}^d$ with $d \in \mathbb{N}$. Then, one can employ Theorem 6.4.2 to obtain smoothness conditions for this composite function. For example, one could pick λ to be some neural-network type of function, which achieves high non-linearity with a sharp decrease in the number of variables, i.e., d. The chances of this being very beneficial are debatable, though, as neural networks are very convenient for calculating gradients analytically, but in this algorithm, we substitute the gradient estimation step by a numerical quantum algorithm anyway. However, it might be that choosing a very specific problem-dependent λ increases the learning time, but this will have to be investigated separately for the different problems.

Finally, one can combine the algorithm outlined here with other quantum tricks to speed up the search for an optimal policy. For example, one can use a number of initial policies, and apply the optimization algorithm in superposition. Then, Grover's search can be applied to find the optimization procedure that terminated with the best value of the objective function. This is equivalent to performing a number of random restarts in a classical setting, say N, but with a query complexity that scales just with $\sqrt{N}$. This technique cannot be combined with choosing η depending on where the current vector $\mathbf{x}$ is located, though.

The theory of the quantum optimization procedure is not quite complete. We already mentioned that it might be possible to remove the scaling of $1/\eta$ from the query complexity of Algorithm 6.4.12. However, it is also unclear if the resulting dependence is anywhere near optimal. It would be nice to apply the hybrid method, i.e., Theorem 5.1.7, to the gradient estimation problem of the policy evaluation problem as well,

and prove some lower bounds on the query complexity, in the same sense as we proved lower bounds on the query complexity of the quantum value evaluation algorithm in Subsection 6.2.3. Finding an algorithm that achieves such a lower bound would be ideal and this would be a very interesting topic of further research.

Furthermore, observe that the gradient estimation procedure provided here uses Algorithm 4.3.9 with the parameter $\sigma = 1$. At the end of Chapter 5, we remarked that we have proven optimality whenever $\sigma \in [0, 1/2]$, but that it is still an open question whether Algorithm 4.3.9 is optimal in the case $\sigma \in (1/2, 1]$.

If Algorithm 4.3.9 is optimal, then Algorithm 6.4.12 does not provide any speed-up over using Algorithm 6.2.7 in conjunction with some numerical classical gradient estimation technique. However, we might be able to use the Markov decision process framework to construct example functions that have Gevrey-type 1 and are difficult to distinguish, such that we can prove optimality of Algorithm 4.3.9. Whether this is feasible is hard to predict in advance.

On the other hand, if Algorithm 4.3.9 is not optimal for $\sigma = 1$, though, we might be able to improve it, and hence we could improve Algorithm 6.4.12 as well. Using all the ideas from this section, but with a more efficient gradient estimation subroutine, we could then reduce the query complexity of Algorithm 6.4.12, such that it scales sublinearly in d. As every classical algorithm scales at least linearly in d, as it has to do at least $d + 1$ function evaluations, we then obtain a provable speed-up over classical algorithms. Hence, closing the optimality gap portrayed at the end of Chapter 5 remains an important open question in this field of research.

This concludes our discussion of the quantum policy optimization algorithm. In the following and final subsection, we will briefly elaborate on some of the settings in which we can apply this algorithm.

6.4.4 Applications

Reinforcement learning has become a very fast-growing field of research in the last decade, especially since the amount of computational power available has increased and its cost has decreased dramatically. A few major breakthroughs have been reached recently, most notably the defeat of one of the world-leading Go players, Lee Sedol, by Google Deep Mind's program Alpha Go in March 2016 [SSS+17], which was trained using reinforcement learning techniques, and more recently the defeat of chess engine Stockfish 8 by AlphaZero in December 2017 [SHS+17]. These results were obtained by making a computer play against itself, and learn from these games. Here, we show how a similar technique can be employed using the algorithm presented in the previous section.

Let's take the game of chess as a running example. We can take the positions on the board as elements in the state space. As for every piece, it can be either on or off the board, and there are 64 squares on the board where it can be placed, we can encode the position on the chess board into $32 \cdot (\log(64)+1) = 32 \cdot (6+1) = 224$ qubits. We will need to store some extra information as well, e.g., whether castling and taking en passant is possible, but even taking these things into account, we should be able to encode the elements of the state space in less than 500 qubits.

The action space consists of moving one of the 32 pieces to one of the 64 squares. Hence, we can simply store which piece we want to move, i.e., a number between 1 and 32, and to which square we want to move it. This takes $\log(32) + \log(64) = 5 + 6 = 11$ qubits.

We will give a player a reward of 1 if the player makes a move which checkmates the opponent. Otherwise, the reward will be 0 and we will give a reward of $\frac{1}{2}$ to either player when one of the players makes a drawing move. This way, we will value policies that eventually lead to checkmating the opponent more than those that don't, and we will value the policies that lead to drawing the opponent more than those that lead to defeat.

The idea is to play against a given strategy. Suppose that black plays according to some not necessarily deterministic strategy, and that we want to optimize white's strategy. We can do so by applying quantum

policy optimization, specifically using the gradient estimation algorithm Algorithm 6.4.12. We can encode black's strategy into the input oracle $\mathcal{P}$. Moreover, if white picks an action that corresponds to an illegal move, we can simply not modify the state, or penalize this by some negative reward.

In principle, we could simply take this algorithm, described in the previous paragraph, and update the vector x according to the gradients that this gradient estimation algorithm outputs. However, the state and actions spaces are very large in this case, leading to very large oracle complexities. Below, we will elaborate on some considerations that aim at decreasing the number of oracle queries.

Observe that the relation between the state and action spaces in chess is somewhat more specific than the framework that we introduced in Section 6.1. From any arbitrary state, namely, there is only a small section of the action space that constitutes a legal move. Hence, we can impose extra restrictions on the policy, enforcing that the probability of selecting any of the illegal moves is 0. The maximum number of legal moves from any given position appears to be 218.[5] If we enforce the policy oracle Π to only select legal moves, then we can replace $|A|$ by 218 in the query complexity.

Moreover, if we select two states from the state space at random, it is probably not possible to move from one to the other immediately; if possible at all, it generally takes a lot of moves to do so. Hence, if we consider the state space to be a directed graph, connected by arcs that correspond to legal moves, then the degree of the states is far lower than the number of vertices. Of course if the number of legal moves is upper bounded by 218, then the out-degree is also upper bounded by 218. If we take this into account in the query complexity analysis, then we can replace the quantity $|S|$ in the query complexity by 218 as well.

We can also use that a chess game can be at most 6000 moves in length. Hence, we can set $T = 6000$, which drops one of the $1 - \gamma$ factors in the denominator of the query complexity. Moreover, we can drop the $1 - \gamma$ factor in the denominator of the scaling of the objective function, as the total reward throughout the entire game cannot exceed 1.

With all of these optimizations, we can significantly reduce the query complexity of the gradient estimation step. It would be nice to figure out the exact elementary gate complexity for this algorithm. It would be pretty involved, but it would provide better insight in the number of elementary operations and qubits necessary to implement Algorithm 6.4.12 in a meaningful setting on a quantum computer.

Finally, once we can optimize white's strategy, given black's strategy, we can terminate the optimization procedure once white wins, say, 90% of its games. Then, we can fix white's strategy and optimize black's strategy, until it wins 90% of its games over white. Determining these percentages can be done using the policy evaluation algorithm, Algorithm 6.3.1. Iterating this process hopefully yields better and better strategies for both players[6], allowing us to *learn* to play chess well using a quantum computer.[7]

The ideas presented here can be used to generate policies for other games as well, like Go. Perhaps it might be possible to build a proof of concept that learns tic-tac-toe once the first quantum computers with a reasonable number of qubits are built. In any case, it would be fruitful to figure out how many qubits and elementary operations are needed to learn the game of tic-tac-toe using the algorithm presented in this chapter.

As a final note, observe that the hard work in this chapter, was to convert phase oracles to probability oracles and back, which was invented by Gilyén et al. [GAW17]. In addition, note that by repeated applications of phase oracles, we can perform addition of phases, and with repeated applications of probability oracles, we can perform multiplication of amplitudes. As these addition and multiplication operations do not depend on binary representations of the values that are being added or multiplied, we could refer to these operations

[5] This has never been formally proven, so it might be that this number must be increased by a few units. This is not particularly interesting for the discussion here, though.

[6] It could also be that whenever one of the strategies learns how to beat the other, it forgets the previous things it learned, hence there might be some trial and error process here in trying to get the quantum computer to actually learn.

[7] A tiny remark is that it is not necessarily very hard to come up with good initial strategies, as with complete random play, numerical analysis shows that white and black win about 7% of their games.

as “analog operations”. As [GAW17] showed that there is a way to convert between these addition and multiplication oracles, we can multiply and add in an analog manner, which forms the basic idea of the gradient estimation algorithm, [GAW17], and also of the algorithm described in this chapter. These methods of analog computation may have far bigger applications than just these two used in this document, and it would probably be very fruitful to standardize these kind of conversions into some easy well-documented circuits.

All in all, the algorithm presented in this section consists of many individual components. Of some of these components, we have proven optimality, but of others we have not. Hence, there are still a lot of points where the theory can be improved, but in essence this algorithm shows that it is possible to port reinforcement learning techniques to a quantum computing setting, and hence that the field of quantum reinforcement learning might become a very interesting field in the decades to come.

7 Conclusion

This report covers two main research directions. The first one concerns solving the gradient estimation problem with a quantum computer. First of all, the ideas of Gilyén et al.'s gradient estimation algorithm, [GAW17], are expanded upon. Specifically, the algorithm is extended such that larger function classes, whose elements satisfy less stringent regularity constraints, can be used as input. Moreover, optimality of Gilyén et al.'s algorithm has been proven for classes of functions satisfying more stringent regularity constraints, and for approximating gradients with respect to different ℓ^p-norms, where $p \in [1, \infty]$, rather than just the ℓ^∞-norm.

The main open question in this direction of research is whether the extension of Gilyén et al.'s gradient estimation algorithm, presented in this document, can be improved upon. Specifically, there remains an optimality gap when functions whose Gevrey-type is bigger than $1/2$, are used as input to the gradient estimation algorithm. Especially function classes containing functions of Gevrey-type 1 are of special interest, as they arise naturally in the field of quantum reinforcement learning.

The second research direction investigates how reinforcement learning can be performed on a quantum computer. It is shown that extensions of Montanaro's ideas for quantum Monte-Carlo simulation, [Mon15], in the setting of quantum reinforcement learning are essentially optimal in the query complexity to all input oracles, giving rise to essentially optimal quantum value estimation and quantum policy evaluation algorithms. Finally, it is shown how these ideas can be used as subroutines in Gilyén et al.'s quantum gradient estimation algorithm, to obtain a policy optimization algorithm.

There remain some very important open questions in this direction of research. First of all, the regularity bounds that are proven on the policy evaluation function are not known to be tight, and it is not known whether shrinking the domain to exclude the edges is necessary in the proof of these bounds either. Improving these two results would directly improve the query complexity of the resulting quantum policy optimization algorithm, such that it is better than using a classical gradient estimation step with the quantum policy evaluation routine. Hence, it would be interesting to figure out whether such improvements are feasible.

Finally, both the gradient estimation algorithm and the quantum algorithms that are used in quantum reinforcement learning in some sense rely on techniques that collectively can be referred to as "analog computation", as they do not use binary representations of numbers to perform mathematical operations. As these methods are very general, they have the potential to be applicable in a wide variety of settings, and hence a very interesting topic of future research would be to find other use cases of these techniques.

Bibliography

[AB98] Charalambos D. Aliprantis and Owen Burkinshaw. *Principles of real analysis*. Acadamic Press, 3rd edition edition, 1998.

[BBBV97] Charles H. Bennet, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on computing*, 26(5):1510–1523, 1997. arXiv:quant-ph/9701001.

[Bea97] Robert Beals. Quantum computation of fourier transforms over symmetric groups. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 48–53, May 1997. 10.1145/258533.258548.

[BEST96] Adriano Barenco, Artur Ekert, Kalle-Antti Suominen, and Päivi Törmä. Approximate quantum fourier transform and decoherence. *Physical Review A*, 54(139), 1996. arXiv:quant-ph/9601018.

[BHMT00] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Proceedings of 5th Israeli symposium on Theory of Computing and Systems*, pages 12–23, May 2000. arXiv:quant-ph/0005055.

[CKS17] Andrew M. Childs, Robin Kothari, and Rolando D. Somma. Quantum linear systems algorithm with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46:1920–1950, 2017. arXiv:1511.02306.

[Cor16] Arjan Cornelissen. Quantum computation - shor's algorithm, July 2016. Bachelor's Thesis, Delft University of Technology repository uuid:79ca8e64-d05e-4d0d-b77b-c72faa885490.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, September 1965.

[dB55] Francesco Faà di Bruno. Sullo sviluppo delle funzioni. *Annali di Scienze Matematiche e Fisiche*, 6:479–480, 1855.

[DB17] Vedran Dunjko and Hans J. Briegel. Machine learning & artificial intelligence in the quantum domain. September 2017. arXiv:1709.02779.

[DLWT18] Vedran Dunjko, Yi-Kai Liu, Xingyao Wu, and Jacob M. Taylor. Exponential improvements for quantum-accessible reinforcement learning. August 2018. arXiv:1710.11160.

[Dra00] Thomas G. Draper. Addition on a quantum computer. August 2000. arXiv:quant-ph/0008033.

[DTB16] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Physical Review Letters*, 117, October 2016. arXiv:1610.08251.

[GAW17] András Gilyén, Srinivasan Arunachalam, and Nathan Wiebe. Optimizing quantum optimization algorithms via faster quantum gradient computation. November 2017. arXiv:1711.00465v2 [quant-ph].

[Gri16] David J. Griffiths. *Introduction to Quantum Mechanics*. Cambridge University Press, August 2016. 2nd edition.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of 28th ACM STOC*, pages 212–219, May 1996. arXiv:quant-ph/9605043.

[GSLW18] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. June 2018. arXiv:1806.01838v1 [quant-ph].

[Gzy86] Henryk Gzyl. Multidimensional extension of faa di bruno's formula. *Journal of Mathematical Analysis and Applications*, 116(2):450–455, June 1986.

[Haa18] Markus Haase. The Dore Venni theorem. *Internet Seminar on Functional Calculus 21 Proceedings*, 2018. https://www.math.uni-kiel.de/isem21/en/course/phase1/isem21-2018-02-02-chapter13.

[Har05] Aram W. Harrow. *Applications of coherent classical communication and the Schur transform to quantum information theory*. PhD thesis, Massachusetts Institute of Technology, August 2005. arXiv:quant-ph/0512255.

[Hoe63] Wassily Hoeffing. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[Jor] Stephen P. Jordan. Quantum algorithm zoo. `http://math.nist.gov/quantum/zoo`. Accessed: August 20th, 2018.

[Jor05] Stephen P. Jordan. Fast quantum algorithm for numerical gradient estimation. *Physical Review Letters*, 95:050501, January 2005. arXiv:quant-ph/0405146.

[Kry08] Nikolai W. Krylov. *Lectures on Elliptic and Parabolic Equations in Sobolev Spaces*. The American Mathematical Society, August 2008.

[Mon15] Ashley Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society of London A*, 417(2181), April 2015. arXiv:1504.06987.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Ng13] Chi-Keung Ng. On genuine infinite algebraic tensor products. *Revista Matemática Iberoamericana*, 29(1):329–356, January 2013. arXiv:1112.3128 [math.RA].

[Pop35] Tiberiu Popoviciu. Sur les quations algbriques ayant toutes leurs racines relles. *Mathematica (Cluj)*, 9:129–145, 1935.

[Rya02] Raymond A. Ryan. *Introduction to Tensor Products of Banach Spaces*. Springer, 2002.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 2nd edition, 2018. http://incompleteideas.net/book/the-book-2nd.html.

[SHS+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Denis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. December 2017. arXiv:1712.01815.

[Sil15] David Silver. Ucl course on rl, 2015. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, October 2017.

[Tas13] Steven Taschuk. A short introduction to tensor products of vector spaces, 2013. www.amotlpaa.org/math/vecten.pdf.

[VBE96] Vlatko Vredal, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54:147–153, July 1996. arXiv:quant-ph/9511018.

A Mathematical background of tensor products of Hilbert spaces

In this section, we give a rigorous definition of the *Hilbert space tensor product*. The construction that we present in this section is quite involved, and will first of all require the introduction of the *vector space tensor product* (sometimes also called the algebraic tensor product). It is important to note in advance that these are two different concepts. Generally, they are both denoted by $\otimes$, but here, in an attempt to avoid confusion, we will explicitly differentiate between them. We will denote the vector space tensor product by $\otimes_{VS}$ and we will write $\otimes$ only whenever we refer to the Hilbert space tensor product.

In this section we explicitly construct the Hilbert space tensor product, because that is the mathematical concept that we need in the other parts of this text. The construction presented here, though, can be easily modified to construct, e.g., the Banach space tensor product. At the point where this modification has to take place, we will hint on how this can be done.

Note that the construction presented here is not the only way in which we can define the Hilbert space tensor product. Another method considers the Hilbert space dual of the space of bilinear functions. Some additional references include [Tas13], who elaborates on both ways in the vector space setting, and [Rya02], who uses a bilinear approach to define the Banach space tensor product.

We kick off the construction of the vector space tensor product with the introduction of the free vector space.

Definition A.1: Free vector space
Let S be a set and F be a field. We define $F(S)$ to be the set of all functions from S to F that have finite support. That is, if $f \in F(S)$, then $f : S \to F$, where $|f^{-1}(F \setminus \{0_F\})| < \infty$. Here 0_F is the additive identity element of F.
The addition operation on $F(S)$ adds two functions mapping S to F pointwise. The scalar multiplication of an element of $F(S)$ with a scalar from F is defined to act pointwise on the function mapping S to F as well. This turns the free vector space $F(S)$ into a vector space over F.

The attentive reader might note that we did not specifically prove here that $F(S)$ becomes a vector space. This, however, comes down to simply checking that the vector space axioms are satisfied, which we leave as an exercise for the reader.

Interestingly, there is a very convenient way to associate all the elements of the set S with elements of the free vector space $F(S)$. We will refer to this association mapping as the canonical embedding of S in $F(S)$. Below, we give its definition.

Definition A.2: Canonical embedding in the free vector space
Let S be a set and F a field. We identify every element $s \in S$ with the function $\delta_s \in F(S)$, defined by:

$$\delta_s : S \to F \qquad \text{with} \qquad \delta_s(t) = \begin{cases} 1_F, & s = t \\ 0_F, & s \neq t \end{cases}$$

Here, 0_F is the additive identity element of F, and 1_F is the multiplicative identity element of F. We refer to the resulting mapping $s \mapsto \delta_s$ as the *canonical embedding of S in $F(S)$*.

So, informally speaking, we identify every element $s \in S$ with a Dirac delta function mapping S into F, where the peak is located at s.

At this point, it is important to remark that in general free vector spaces are huge. For example, if we were to consider $\mathbb{R}(\mathbb{R})$, this is the set of all functions mapping $\mathbb{R}$ to $\mathbb{R}$, which is a space bigger than $\mathbb{R}^n$ for any $n \in \mathbb{N}$. Tensor product spaces are much smaller quotient spaces of free vector spaces. Intuitively, one can think of the tensor product space of two vector spaces as the quotient space of the free vector space of their Cartesian product in which all the unnecessary degeneracy is removed, effectively reducing the number of elements. The following definition makes that precise.

Definition A.3: Vector space tensor product
Let V and W be vector spaces over a field F. We define a subset D of the free vector space $F(V \times W)$ such that $x \in D$ if and only if at least one of the following conditions is satisfied:

1. There exist $v_1, v_2 \in V$ and $w \in W$ such that $x = \delta_{(v_1,w)} + \delta_{(v_2,w)} - \delta_{(v_1+v_2,w)}$.
2. There exist $v \in V$ and $w_1, w_2 \in W$ such that $x = \delta_{(v,w_1)} + \delta_{(v,w_2)} - \delta_{(v,w_1+w_2)}$.
3. There exist $v \in V$, $w \in W$ and $f \in F$ such that $x = f\delta_{(v,w)} - \delta_{(fv,w)}$.
4. There exist $v \in V$, $w \in W$ and $f \in F$ such that $x = f\delta_{(v,w)} - \delta_{(v,fw)}$.

Let $N = \text{Span}(D) \subseteq F(V \times W)$. Then, we define the *tensor product space* of V and W to be quotient space of $F(V \times W)$ divided by N, i.e.

$$V \otimes_{VS} W = F(V \times W)/N$$

Moreover, for any $v \in V$ and $w \in W$, we define $v \otimes_{VS} w$ to be the image of $\delta_{(v,w)}$ under the quotient map $\xi : F(V \times W) \to F(V \times W)/N$. Hence $v \otimes_{VS} w \in V \otimes_{VS} W$. Conversely, we refer to the elements of $V \otimes_{VS} W$ that can be written as $v \otimes_{VS} w$, for some vectors $v \in V$ and $w \in W$, as *simple tensors*, *pure tensors*, or *elementary tensors*.

Before we move on to define the Hilbert space tensor product, we will first of all prove some results about the vector space tensor product.

Theorem A.4: Vector space tensor product properties
Let V and W be vector spaces over some field F. Then, their vector space tensor product space $V \otimes_{VS} W$ satisfies the following properties.

1. $V \otimes_{VS} W$ is a vector space over F.
2. For all $v_1, v_2 \in V$ and $w \in W$, we have $v_1 \otimes_{VS} w + v_2 \otimes_{VS} w = (v_1 + v_2) \otimes_{VS} w$.
3. For all $v \in V$ and $w_1, w_2 \in W$, we have $v \otimes_{VS} w_1 + v \otimes_{VS} w_2 = v \otimes_{VS} (w_1 + w_2)$.
4. For all $v \in V$, $w \in W$ and $f \in F$, we have $f(v \otimes_{VS} w) = (fv) \otimes_{VS} w = v \otimes_{VS} (fw)$.
5. The mapping $A \times B \to A \otimes_{VS} B$, defined by $(a, b) \mapsto a \otimes_{VS} b$ is bilinear.

Proof. First, we prove statement 1. To that end, observe that $V \otimes_{VS} W$ is defined as the quotient space of the free vector space $F(V \times W)$, which is itself a vector space over F. Hence, the vector space structure of $F(V \times W)$ carries over to $V \otimes_{VS} W$. This completes the proof of statement 1.

For statement 2, suppose that $v_1, v_2 \in V$ and $w \in W$. Then, we find, by property 1 in Definition A.3, that $\delta_{(v_1,w)} + \delta_{(v_2,w)} - \delta_{(v_1+v_2,w)} \in N$. But trivially $0 \in N$, and hence the image of $\delta_{(v_1,w)} + \delta_{(v_2,w)} - \delta_{(v_1+v_2,w)}$ under the quotient map is $0 \in V \otimes_{VS} W$. But the quotient map is linear, hence $v_1 \otimes_{VS} w + v_2 \otimes_{VS} w - (v_1+v_2) \otimes_{VS} w = 0$. Rearranging yields $v_1 \otimes_{VS} w + v_2 \otimes_{VS} w = (v_1 + v_2) \otimes_{VS} w$, completing the proof of statement 2.

The proof of property 3 is very similar to the proof of property 2. The only difference is that now one has to use property 2 of Definition A.3, instead of property 1.

Next, we turn to the proof of property 4. To that end, take $v \in V$, $w \in W$ and $f \in F$ arbitrarily. Then, we find that $f\delta_{(v,w)} - \delta_{(fv,w)} \in N$, by property 3 in Definition A.3. Again using $0 \in N$ and invoking the linearity of the quotient map, allows us to deduce that $f(v \otimes_{VS} w) - (fv) \otimes_{VS} w = 0$. Rearranging yields the first equality. The second equality can be obtained in a similar manner, but then one has to use property 4 of Definition A.3. This completes the proof of statement 4.

Finally, observe that statements 2, 3 and 4 imply statement 5. This completes the proof. □

Lemma A.5: Bilinear operators on vector spaces
Let V, W, D and N be as in Definition A.3. Let $\phi : V \to F$ and $\psi : W \to F$ be linear maps. Define $F_{\phi,\psi}$ to be the linear extension of the following mapping:

$$F_{\phi,\psi}(\delta_{(v,w)}) = \phi(v)\psi(w)$$

Then, any $x \in N$ satisfies $F_{\phi,\psi}(x) = 0_F$.

Proof. First of all, suppose that $v_1, v_2 \in V$ and $w \in W$. Then, we find:

$$\begin{aligned} F_{\phi,\psi}(\delta_{(v_1,w)} + \delta_{(v_2,w)} - \delta_{(v_1+v_2,w)}) &= F_{\phi,\psi}(\delta_{(v_1,w)}) + F_{\phi,\psi}(\delta_{(v_2,w)}) - F_{\phi,\psi}(\delta_{(v_1+v_2,w)}) \\ &= \phi(v_1)\psi(w) + \phi(v_2)\psi(w) - \phi(v_1+v_2)\psi(w) = [\phi(v_1+v_2) - \phi(v_1+v_2)]\psi(w) = 0_F \end{aligned}$$

If $v \in V$ and $w_1, w_2 \in W$, then we completely analogously check that:

$$\begin{aligned} F_{\phi,\psi}(\delta_{(v,w_1)} + \delta_{(v,w_2)} - \delta_{(v,w_1+w_2)}) &= F_{\phi,\psi}(\delta_{(v,w_1)}) + F_{\phi,\psi}(\delta_{(v,w_2)}) - F_{\phi,\psi}(\delta_{(v,w_1+w_2)}) \\ &= \phi(v)\psi(w_1) + \phi(v)\psi(w_2) - \phi(v)\psi(w_1+w_2) = \phi(v)[\psi(w_1+w_2) - \psi(w_1+w_2)] = 0_F \end{aligned}$$

Furthermore, if $v \in V$, $w \in W$ and $f \in F$, we find:

$$F_{\phi,\psi}(f\delta_{(v,w)} - \delta_{(fv,w)}) = F_{\phi,\psi}(f\delta_{(v,w)}) - F_{\phi,\psi}(\delta_{(fv,w)}) = f\phi(v)\psi(w) - \phi(fv)\psi(w) = [f\phi(v) - f\phi(v)]\psi(w) = 0_F$$

Similarly, we check:

$$F_{\phi,\psi}(f\delta_{(v,w)} - \delta_{(v,fw)}) = F_{\phi,\psi}(f\delta_{(v,w)}) - F_{\phi,\psi}(\delta_{(v,fw)}) = f\phi(v)\psi(w) - \phi(v)\psi(fw) = \phi(v)[f\phi(w) - f\phi(w)] = 0_F$$

So, we find that for all $x \in D$, we have $F_{\phi,\psi}(x) = 0_F$. By linearity, we find that for all $x \in N$, we have $F_{\phi,\psi}(x) = 0$, as required. □

Lemma A.6: Additive identity of the vector space tensor product
Let V and W be vector spaces over some field F. Then, $v \otimes_{VS} w = 0$ if and only if $v = 0$ or $w = 0$.

Proof. Let V, W, D and N be as in Definition A.3. First of all, suppose that $v = 0$. Then, we find, using property 3 of Definition A.3:

$$\delta_{(v,w)} = \delta_{(0\cdot v,w)} - 0\delta_{(v,w)} \in D \subseteq N$$

Similarly, suppose that $w = 0$. Then, we find, using property 4 of Definition A.3:

$$\delta_{(v,w)} = \delta_{(v,0\cdot w)} - 0\delta_{(v,w)} \in D \subseteq N$$

This proves the "if" part.

For the "only if" part, let $\delta_{(v,w)} \in N$. Furthermore, let $\phi : V \to F$ and $\psi : W \to F$ be linear. We find, with $F_{\phi,\psi}$ as in Lemma A.5:

$$0_F = F_{\phi,\psi}(\delta_{(v,w)}) = \phi(v)\psi(w) = \psi(\phi(v)w)$$

This holds for all linear functions $\psi : W \to F$. So, we find that $\phi(v)w = 0$. So, either $w = 0$, or $\phi(v) = 0_F$. But if $w = 0$, we are done, and if $w \neq 0$, we find that $\phi(v) = 0_F$ for all $\phi : V \to F$. This implies that $v = 0$, and hence we are done too. This completes the proof. □

Theorem A.7: Basis of the vector space tensor product space
Let V and W be vector spaces over a field F. Suppose that $\mathcal{B}_V$ and $\mathcal{B}_W$ are bases for V and W. Let:

$$\mathcal{B}_{V\otimes_{VS}W} = \{a \otimes_{VS} b : a \in \mathcal{B}_V, b \in \mathcal{B}_W\}$$

Then $\mathcal{B}_{V\otimes_{VS}W}$ is a basis for $V \otimes_{VS} W$.

Proof. As in the statement of the theorem, suppose that $\mathcal{B}_V$ and $\mathcal{B}_W$ are bases of the vector spaces V and W, respectively. Now, in order to prove that $\mathcal{B}_{V\otimes_{VS}W}$ is a basis of $V \otimes_{VS} W$, we will first prove that every vector in $V \otimes_{VS} W$ can be written as a linear combination of vectors in $\mathcal{B}_{V\otimes_{VS}W}$, and secondly that every finite subset of $\mathcal{B}_{V\otimes_{VS}W}$ is a set of linearly independent vectors.

So, first of all, we prove that every element in $V \otimes_{VS} W$ can be written as a linear combination of $\mathcal{B}_{V\otimes_{VS}W}$. To that end, take $x \in V \otimes_{VS} W$ arbitrarily. Next, take any element $y \in F(V \times W)$ in the pre-image of the element x under the quotient map. Observe that we can write this element y as follows. We can take a finite sequence of vectors $a_1, \ldots, a_N \in V$ and $b_1, \ldots, b_N \in W$ and a finite sequence of scalars $f_1, \ldots, f_N \in F$, such that:

$$y = \sum_{i=1}^{N} f_i \delta_{(a_i,b_i)} \tag{A.0.1}$$

Now, take $i \in \{1, \ldots, N\}$ arbitrarily. As $\mathcal{B}_V$ and $\mathcal{B}_W$ are bases for V and W, respectively, we can find sets of vectors $\{v_1, \ldots, v_n\} \subseteq \mathcal{B}_V$ and $\{w_1, \ldots, w_m\} \subseteq \mathcal{B}_W$ and finite sequences of scalars $g_{i1}, \ldots, g_{in} \in F$ and $h_{i1}, \ldots, h_{im} \in F$, such that:

$$a_i = \sum_{j=1}^{n} g_{ij} v_j \qquad \text{and} \qquad b_i = \sum_{k=1}^{m} h_{ik} w_k \tag{A.0.2}$$

Next, we make the following crucial observation. For any vectors $v_1, v_2 \in V$ and $w \in W$, we have, according to property 1 of Definition A.3, the following relation, where Δ is some element of N:

$$\delta_{(v_1+v_2,w)} = \delta_{(v_1,w)} + \delta_{(v_2,w)} + \Delta$$

Similarly, we can employ the other properties of Definition A.3 to obtain other useful relations of this form. Moreover, as N, being a linear subspace of $F(V \times W)$, is a vector space in its own right, we can iteratively apply these relations and merge all the resulting Δ5s into one. Plugging Equation A.0.2 into Equation A.0.1, and using the described relations, we obtain, for some $\Delta \in N$:

$$y = \sum_{i=1}^{N} f_i \delta_{(a_i,b_i)} = \sum_{i=1}^{n} f_i \delta_{\left(\sum_{i=1}^{m} g_{ij} v_j, \sum_{k=1}^{m} h_{ik} w_k\right)} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m} f_i g_{ij} h_{ik} \delta_{(v_j,w_k)} + \Delta$$

Now, we can apply the quotient map on both sides. Note that this map is linear, and that it maps all elements in N to 0. In particular, this means that Δ vanishes. Thus, we obtain:

$$x = \sum_{j=1}^{n}\sum_{k=1}^{m} \left[\sum_{i=1}^{N} f_i g_{ij} h_{ik}\right] (v_j \otimes_{VS} w_k)$$

Note that we also changed the order of summation here, to make it apparent that $x \in \text{Span}(\mathcal{B}_{V\otimes_{VS}W})$. Hence, indeed, we can write every element of $V \otimes_{VS} W$ as a finite linear combination of elements in $V \otimes_{VS} W$. This completes the first part of the proof.

Now, it remains to prove that every finite subset $\mathcal{B} \subseteq \mathcal{B}_{V\otimes_{VS}W}$ is a linearly independent subset of $V \otimes_{VS} W$. Then, we can find finite subsets $\{v_1, \ldots, v_n\} \subseteq \mathcal{B}_V$ and $\{w_1, \ldots, w_m\} \subseteq \mathcal{B}_W$ such that $\mathcal{B} \subseteq \{v_i \otimes_{VS} w_j : i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\} = C$. We prove the slightly stronger result that C is linearly independent, which trivially implies that $\mathcal{B}$ is linearly independent as well.

To that end, for all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, let $\lambda_{ij} \in F$ and suppose:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \lambda_{ij}(v_i \otimes_{VS} w_j) = 0$$

In order to prove that C is linearly independent, we must prove that all λ_{ij}s are 0. As a first step towards proving this, we deduce from the definition of $V \otimes W$:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\delta_{(v_i,w_j)} \in N$$

Now, we apply Lemma A.5. To that end, let $\phi : V \to F$ and $\psi : W \to F$ be linear functions. Then, we find:

$$F_{\phi,\psi}\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\delta_{(v_i,w_j)}\right) = 0$$

Now, using the linearity of $F_{\phi,\psi}$ and its definition, we obtain:

$$0 = F_{\phi,\psi}\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\delta_{(v_i,w_j)}\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_{ij}F_{\phi,\psi}\left(\delta_{(v_i,w_j)}\right) = \sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\phi(v_i)\psi(w_j)$$

But now we can invoke the linearity of ψ to obtain:

$$0 = \sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\phi(v_i)\psi(w_j) = \psi\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\lambda_{ij}\phi(v_i)w_j\right)$$

It is important to remark that the above relation holds *for all* linear functions $\psi : W \to F$. This implies that the expression within the parentheses is 0. Hence, after changing the order of summation, we obtain:

$$\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_{ij}\phi(v_i)w_j = 0$$

But recall that the w_j's form a subset of $\mathcal{B}_W$, and hence they are linearly independent. So, by definition of linear independence, we find that, for all $j \in \{1,\ldots,m\}$, we have:

$$\sum_{i=1}^{n}\lambda_{ij}\phi(v_i) = 0$$

This time, we can invoke the linearity property of the mapping $\phi : V \to F$, to obtain that for all such linear mappings ϕ, we have:

$$\phi\left(\sum_{i=1}^{n}\lambda_{ij}v_i\right) = 0$$

This time, the above holds for all linear functions $\phi : V \to F$. So, again we obtain:

$$\sum_{i=1}^{n}\lambda_{ij}v_i = 0$$

Now, finally, recall that all v_i's are elements of $\mathcal{B}_V$, and hence they are linearly independent. Thus, we find that for all $i \in \{1,\ldots,n\}$, we have $\lambda_{ij} = 0$. Thus, in summary, we have found:

$$\forall i \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,m\}, \qquad \lambda_{ij} = 0$$

This is exactly what we set out to prove. Hence, we find that C is indeed a set of linearly independent vectors. Thus, we have completed the second part of the proof as well, and can now conclude that $\mathcal{B}_{V\otimes_{VS}W}$ is indeed a basis of $V \otimes_{VS} W$. □

Now, we are ready to define the Hilbert space tensor product.

Definition A.8: Hilbert space tensor product
Let A and B be Hilbert spaces over a field $\mathbb{K}$, where $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Let $\langle \cdot, \cdot \rangle_A$ and $\langle \cdot, \cdot \rangle_B$ be their corresponding inner products, respectively. On the vector space tensor product $A \otimes_{VS} B$, we now define the following inner product $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$, which we define for all $a_1, a_2 \in A$ and $b_1, b_2 \in B$:

$$\langle a_1 \otimes_{VS} b_1, a_2 \otimes_{VS} b_2 \rangle_{A \otimes_{VS} B} = \langle a_1, a_2 \rangle_A \cdot \langle b_1, b_2 \rangle_B$$

and subsequently extend linearly in the second variable, and conjugate-linearly in the first. Next, we define the Hilbert space tensor product to be the completion of the vector space tensor product $A \otimes_{VS} B$, with respect to the norm induced by the inner product $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$. The resulting space we denote by $A \otimes B$, and the resulting inner product we define as $\langle \cdot, \cdot \rangle_{A \otimes B}$. We refer to the resulting Hilbert space as the *Hilbert space tensor product of A and B*.

Proof of well-definedness of Definition A.8. At this point, we should check that $A \otimes B$ is indeed well-defined. To that end, we must check whether the inner product that we define here, $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$, is a well-defined inner product.

First of all, we will check whether $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$ is a well-defined mapping from $A \otimes_{VS} B \to \mathbb{K}$. To that end, we note that $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$ is defined as the extension of its action on the pure tensors, where its action on the second variable is extended linearly, and its action on the first variable is extended conjugate linearly. So, suppose that:

$$\sum_{i=1}^{I} e_i (p_i \otimes_{VS} q_i) = \sum_{j=1}^{J} f_j (r_j \otimes_{VS} s_j) \qquad \text{and} \qquad \sum_{k=1}^{K} g_k (t_k \otimes_{VS} u_k) = \sum_{\ell=1}^{L} h_\ell (v_\ell \otimes_{VS} w_\ell)$$

for some finite sequences

$$(e_i)_{i=1}^I, (f_j)_{j=1}^L, (g_k)_{k=1}^K, (h_\ell)_{\ell=1}^L \text{ in } \mathbb{K}, \qquad (p_i)_{i=1}^I, (r_j)_{j=1}^J, (t_k)_{k=1}^K, (v_\ell)_{\ell=1}^L \text{ in } A,$$
$$\text{and} \qquad (q_i)_{i=1}^I, (s_j)_{j=1}^J, (u_k)_{k=1}^K, (w_\ell)_{\ell=1}^L \text{ in } B$$

Now, we make the following observations. We have:

$$\sum_{i=1}^{I} e_i \delta_{(p_i, q_i)} - \sum_{j=1}^{J} f_j \delta_{(r_j, s_j)} \in N$$

Moreover, let $k \in \{1, \ldots, K\}$. Then, we find that the following maps are linear (note that if $\mathbb{K} = \mathbb{R}$, the conjugation is redundant):

$$\phi : a \mapsto \overline{\langle a, t_k \rangle_A} \qquad \text{and} \qquad \psi : b \mapsto \overline{\langle b, u_k \rangle_B}$$

So, we obtain, by Lemma A.5:

$$F_{\phi,\psi} \left(\sum_{i=1}^{I} e_i \delta_{(p_i, q_i)} - \sum_{j=1}^{J} f_j \delta_{(r_j, s_j)} \right) = 0$$

Thus, writing out the definitions, we obtain:

$$\sum_{i=1}^{I} e_i \overline{\langle p_i, t_k \rangle_A \langle q_i, u_k \rangle_B} = \sum_{j=1}^{J} f_j \overline{\langle r_j, t_k \rangle_A \langle s_j, u_k \rangle_B}$$

So, we find, taking the complex conjugate on both sides:

$$\sum_{i=1}^{I} \overline{e_i} \langle p_i \otimes_{VS} q_i, t_k \otimes_{VS} u_k \rangle_{A \otimes_{VS} B} = \sum_{j=1}^{J} \overline{f_j} \langle r_j \otimes_{VS} s_j, t_k \otimes_{VS} u_k \rangle_{A \otimes_{VS} B}$$

We could have also taken $\ell \in \{1, \ldots, L\}$ to obtain:

$$\sum_{i=1}^{I} \overline{e_i} \langle p_i \otimes_{VS} q_i, t_\ell \otimes_{VS} u_\ell \rangle_{A \otimes_{VS} B} = \sum_{j=1}^{J} \overline{f_j} \langle r_j \otimes_{VS} s_j, t_\ell \otimes_{VS} u_\ell \rangle_{A \otimes_{VS} B}$$

Similarly, we have that:

$$\sum_{k=1}^{K} g_k \delta_{(t_k, u_k)} - \sum_{\ell=1}^{L} h_\ell \delta_{(v_\ell, w_\ell)} \in N$$

and this implies, using the same argument as before, that for all $i \in \{1, \ldots, I\}$:

$$\sum_{k=1}^{K} g_k \langle p_i \otimes_{VS} q_i, t_k \otimes_{VS} u_k \rangle_{A \otimes_{VS} B} = \sum_{\ell=1}^{L} h_\ell \langle p_i \otimes_{VS} q_i, v_\ell \otimes_{VS} w_\ell \rangle_{A \otimes_{VS} B}$$

Putting everything together, we obtain:

$$\begin{array}{rcl} \displaystyle\sum_{i=1}^{I} \sum_{k=1}^{K} \overline{e_i} g_k \langle p_i \otimes_{VS} q_i, t_k \otimes_{VS} u_k \rangle_{A \otimes_{VS} B} & = & \displaystyle\sum_{j=1}^{J} \sum_{k=1}^{K} \overline{f_j} g_k \langle r_j \otimes_{VS} s_j, t_k \otimes_{VS} u_k \rangle_{A \otimes_{VS} B} \\ \| & & \\ \displaystyle\sum_{i=1}^{I} \sum_{\ell=1}^{L} \overline{e_i} h_\ell \langle p_i \otimes_{VS} q_i, v_\ell \otimes_{VS} w_\ell \rangle_{A \otimes_{VS} B} & = & \displaystyle\sum_{j=1}^{J} \sum_{\ell=1}^{L} \overline{f_j} h_\ell \langle r_j \otimes_{VS} s_j, v_\ell \otimes_{VS} w_\ell \rangle_{A \otimes_{VS} B} \end{array}$$

Hence, we now obtain the last vertical equality for free. Thus the mapping we defined is indeed linear in the second variable and conjugate linear in the first one. Hence, the mapping $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$ is well-defined, without any ambiguity.

Now, it remains to check that this mapping $\langle \cdot, \cdot \rangle_{A \otimes_{VS} B}$ is indeed an inner product. This is not very hard, though, as we have the linearity in the first variable and the conjugate linearity by construction. The conjugate symmetry carries over directly from the inner products that we used to define this new inner product. Moreover, we also get immediately from the definition that $\langle x, x \rangle_{A \otimes_{VS} B} \geq 0$ for all $x \in A \otimes_{VS} B$. So, all that is left is to check that $\langle x, x \rangle_{A \otimes_{VS} B} = 0 \Leftrightarrow x = 0$. The direction to the left follows directly from e.g. the linearity in the second variable (and the well-definedness). The direction to the right is also obvious, once we take the result of Lemma A.6 into account. This completes the proof of the well-definedness of $A \otimes B$. □

Now that we know that $A \otimes B$ is well-defined, let's prove some of its properties.

Theorem A.9: Properties of the Hilbert space tensor product space
Let A and B be Hilbert spaces over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Then, we find:

1. $A \otimes B$ is a Hilbert space.
2. For all $a \in A$ and $b \in B$, we have $\|a \otimes b\|_{A \otimes B} = \|a\|_A \cdot \|b\|_B$.
3. The map $A \times B \to A \otimes B$, given by $(a, b) \mapsto a \otimes b$ is jointly continuous.
4. For all $a_1, a_2 \in A$ and $b \in B$, we have $a_1 \otimes b + a_2 \otimes b = (a_1 + a_2) \otimes b$.
5. For all $a \in A$ and $b_1, b_2 \in B$, we have $a \otimes b_1 + a \otimes b_2 = a \otimes (b_1 + b_2)$.
6. For all $a \in A$, $b \in B$ and $k \in \mathbb{K}$, we have $k(a \otimes b) = (ka) \otimes b = a \otimes (kb)$.
7. The map $A \times B \to A \otimes B$, given by $(a, b) \mapsto a \otimes b$ is bilinear.
8. Let $\mathcal{B}_A$ and $\mathcal{B}_B$ be orthonormal bases of A and B, respectively. Then, $\mathcal{B}_{A \otimes B} = \{a \otimes b : a \in \mathcal{B}_A, b \in \mathcal{B}_B\}$ is an orthonormal basis of $A \otimes B$.
9. $\dim(A \otimes B) = \dim(A) \dim(B)$.

Proof. Statement 1 is true by construction. For statement 2, take $a \in A$ and $b \in B$ arbitrarily. We find:

$$\|a \otimes b\|^2_{A\otimes B} = \langle a \otimes b, a \otimes b\rangle_{A\otimes B} = \langle a, a\rangle_A \cdot \langle b, b\rangle_B = \|a\|^2_A \cdot \|b\|^2_B$$

This proves statement 2. Statement 3 follows trivially from 2. Statements 4 through 7 all directly follow from Theorem A.4. Statement 8 follows directly from Theorem A.7. Statement 9 follows directly from 8. □

As $A \otimes B$ is again a Hilbert space, of course we could tensor it again with another Hilbert space C, to obtain the composite tensor product space $(A \otimes B) \otimes C$. But we could of course also first tensor $B \otimes C$ and then tensor it with A to obtain $A \otimes (B \otimes C)$. As these spaces are not equal as sets, they are not equal as Hilbert spaces. However, they are isomorphic, as we prove in the following theorem.

Theorem A.10: Associative isomorphism of Hilbert space tensor product spaces
Let A, B and C be Hilbert spaces over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Then $(A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$.

Proof. We define $\psi : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ by the linear extension of its action on the simple tensors:

$$\psi : (a \otimes b) \otimes c \mapsto a \otimes (b \otimes c)$$

This map is well-defined and obviously linear. Moreover, it preserves the inner product, as:

$$\begin{aligned}&\langle (a_1 \otimes b_1) \otimes c_1, (a_2 \otimes b_2) \otimes c_2\rangle_{(A\otimes B)\otimes C} = \langle a_1, a_2\rangle_A \cdot \langle b_1, b_2\rangle_B \cdot \langle c_1, c_2\rangle_C \\ &= \langle a_1 \otimes (b_1 \otimes c_1), a_2 \otimes (b_2 \otimes c_2)\rangle_{A\otimes(B\otimes C)} = \langle \psi((a_1 \otimes b_1) \otimes c_1), \psi((a_2 \otimes b_2) \otimes c_2)\rangle_{A\otimes(B\otimes C)}\end{aligned}$$

Thus, we have explicitly constructed an isomorphism between $(A \otimes B) \otimes C$ and $A \otimes (B \otimes C)$, proving that they are isomorphic. □

Now that we know that $A \otimes (B \otimes C)$ and $(A \otimes B) \otimes C$ are isomorphic, we can identify them with each other. *In principle*, namely, as an artifact of our construction, the tensors $(a \otimes b) \otimes c$ and $a \otimes (b \otimes c)$ are not equal, and neither are the tensor product spaces in which they live, $(A \otimes B) \otimes C$ and $A \otimes (B \otimes C)$. These spaces are isomorphic, though, and the explicit isomorphism is given by $(a \otimes b) \otimes c \mapsto a \otimes (b \otimes c)$. So, we can identify the spaces $(A \otimes B) \otimes C$ and $A \otimes (B \otimes C)$ with each other, and simply drop the parentheses to obtain:

$$(A \otimes B) \otimes C = A \otimes B \otimes C = A \otimes (B \otimes C)$$

Similarly, we will identify the tensors $(a \otimes b) \otimes c$ and $a \otimes (b \otimes c)$ with each other and again we will drop the parentheses. So, we write:

$$(a \otimes b) \otimes c = a \otimes b \otimes c = a \otimes (b \otimes c)$$

Similarly, we can write $a \otimes b \otimes c \otimes d$ for $a \otimes (b \otimes c \otimes d)$ and $(a \otimes b \otimes c) \otimes d$. Continuing in this manner, we obtain arbitrary finite tensor products, $a_1 \otimes \cdots \otimes a_n$.

B Error-propagation lemmas

Throughout this document, we use little arguments to bound the propagation of errors in norms and probabilities. We included the arguments here for convenience. Note that they are special cases of more well-known theorems, but we don't need the more general theorems anywhere in this document.

We start with how an error in norm of a quantum state influences the probability distribution of a measurement. We phrase it in terms of n-qubit states, but the result holds in a more general setting as well, where the quantum system is not necessarily made up of qubits.

Lemma B.1: Relation between the probability distribution and the norm error
Let $\varepsilon > 0$ and $\delta \in (0,1)$. Suppose that we have two n-qubit states, $|\psi\rangle$ and $|\phi\rangle$, with $\| |\psi\rangle - |\phi\rangle \| \leq \varepsilon$. Moreover, suppose that if we measure $|\psi\rangle$ in the computational basis, then the probability of obtaining an outcome in the set $A \subseteq \{0,1\}^n$ is lower bounded by δ. Then the probability of obtaining an outcome in the set A by measuring $|\phi\rangle$ in the computational basis is lower bounded by $\delta - \varepsilon$.

Proof. We define, for all $a \in \{0,1\}^n$, the following constants:

$$\psi_a = \langle a|\psi\rangle \qquad \text{and} \qquad \phi_a = \langle a|\phi\rangle$$

Then, we find, as the states $|\psi\rangle$ and $|\phi\rangle$ are normalized:

$$\sum_{a\in\{0,1\}^n} |\psi_a|^2 = \sum_{a\in\{0,1\}^n} |\phi_a|^2 = 1$$

And so:

$$\begin{aligned}\sum_{a\in\{0,1\}^n\setminus A} |\psi_a|^2 - \sum_{a\in\{0,1\}^n\setminus A} |\phi_a|^2 &= \sum_{a\in\{0,1\}^n} |\psi_a|^2 - \sum_{a\in A} |\psi_a|^2 - \sum_{a\in\{0,1\}^n} |\phi_a|^2 + \sum_{a\in A} |\phi_a|^2 \\ &= 1 - \sum_{a\in A} |\psi_a|^2 - 1 + \sum_{a\in A} |\phi_a|^2 = \sum_{a\in A} |\phi_a|^2 - \sum_{a\in A} |\psi_a|^2\end{aligned}$$

We can use the above relation in the following way:

$$\begin{aligned}\left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| &= \frac{1}{2}\left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| + \frac{1}{2}\left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| \\ &= \frac{1}{2}\left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| + \frac{1}{2}\left|\sum_{a\in\{0,1\}^n\setminus A} |\psi_a|^2 - \sum_{a\in\{0,1\}^n\setminus A} |\phi_a|^2\right| \\ &\leq \frac{1}{2}\sum_{a\in\{0,1\}^n} \left||\psi_a|^2 - |\phi_a|^2\right| = \frac{1}{2}\sum_{a\in\{0,1\}^n} |\,|\psi_a| - |\phi_a|\,|\cdot|\,|\psi_a| + |\phi_a|\,|\end{aligned}$$

Now, we can employ the reversed triangle inequality, Cauchy-Schwarz and the triangle inequality to obtain:

$$\begin{aligned}\left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| &\leq \frac{1}{2}\sum_{a\in\{0,1\}^n} |\psi_a - \phi_a|\cdot(\,|\psi_a| + |\phi_a|\,) \\ &\leq \frac{1}{2}\sqrt{\sum_{a\in\{0,1\}^n} |\psi_a - \phi_a|^2}\cdot\sqrt{\sum_{a\in\{0,1\}^n} (\,|\psi_a| + |\phi_a|\,)^2} \\ &\leq \frac{1}{2}\,\| |\psi\rangle - |\psi\rangle \|\cdot\left[\sqrt{\sum_{a\in\{0,1\}^n} |\psi_a|^2} + \sqrt{\sum_{a\in\{0,1\}^n} |\phi_a|^2}\right] \\ &\leq \frac{1}{2}\cdot\varepsilon\cdot[\,\| |\psi\rangle \| + \| |\phi\rangle \|\,] = \frac{1}{2}\cdot\varepsilon\cdot 2 = \varepsilon\end{aligned}$$

And so, we find:

$$\sum_{a\in A} |\langle a|\phi\rangle|^2 \geq \sum_{a\in A} |\langle a|\psi\rangle|^2 - \left|\sum_{a\in A} |\psi_a|^2 - \sum_{a\in A} |\phi_a|^2\right| \geq \delta - \varepsilon$$

This completes the proof. □

The next statement is a very simple corollary of the triangle inequality.

Lemma B.2: Error-propagation of multiple consecutive errors
Let $\delta_1, \delta_2 \geq 0$. Suppose that $|\psi\rangle$ and $|\phi\rangle$ are quantum states, and that U and V are unitary that act on the state space of which $|\psi\rangle$ and $|\phi\rangle$ are members. Moreover, suppose that we have the following properties:

$$\||\psi\rangle - |\phi\rangle\| \leq \delta_1 \qquad \text{and} \qquad \|U - V\| \leq \delta_2$$

Then we have:

$$\|U|\psi\rangle - V|\phi\rangle\| \leq \delta_1 + \delta_2$$

Hence, consecutive errors accumulate in an additive manner.

Proof. The result follows easily from the triangle inequality:

$$\begin{aligned}\|U|\psi\rangle - V|\phi\rangle\| &= \|U|\psi\rangle - V|\psi\rangle + V|\psi\rangle - U|\phi\rangle\| \leq \|(U-V)|\psi\rangle\| + \|V(|\psi\rangle - |\phi\rangle)\| \\ &\leq \|U - V\|\,\||\psi\rangle\| + \|V\|\,\||\psi\rangle - |\phi\rangle\| \leq \delta_2 \cdot 1 + 1 \cdot \delta_1 = \delta_1 + \delta_2\end{aligned}$$

This completes the proof. □

C Hybrid method

In Chapter 5, we make use of the hybrid method. As this is a very general result in quantum computing, whose proof is unrelated to the problem of gradient estimation, we prove the validity of this method in this appendix.

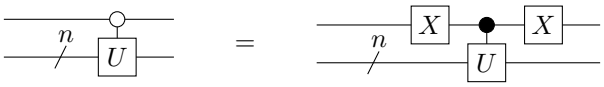
The method was first introduced in [BBBV97]. Subsequently, the method has been generalized, and the approach taken here follows the one given in [GAW17], but we highlight a few more details.

First of all, we will provide the proof of a version of the principle of deferred measurement, in Section C.1. Afterwards, we will prove the main result that we need in the main body of this text, namely the hybrid method, in Section C.2.

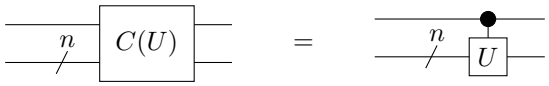
C.1 Principle of deferred measurement

The principle of deferred measurement is a result that shows that for every quantum algorithm, we can construct an algorithm with equal success probability, which only prescribes measurements after all quantum circuits have been executed. In order to prove this result, we first of all prove a lemma, Lemma C.1.1, which provides the induction step of the main result, Theorem C.1.2.

In the lemma below, we use the following shorthand notation, where U is an arbitrary unitary operator acting on n qubits:



Recall that in Section 3.3, we introduced the controlled version of U, denoted by $C(U)$, which was denoted as follows:



and recall that the action of $C(U)$ was as follows:

$$C(U) = I_2^{\otimes n} \otimes |0\rangle\langle 0| + U \otimes |1\rangle\langle 1|$$

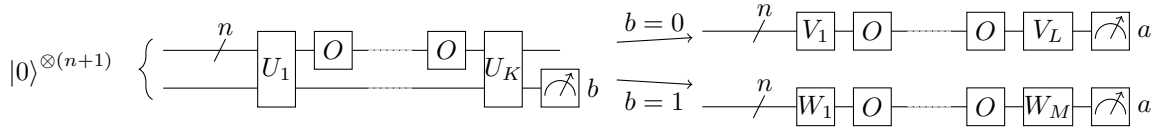
i.e., if the state of the first qubit is $|1\rangle$, the operator U is applied to the lower n qubits, and if the state of the first qubit is $|0\rangle$, then it acts as the identity.

Now, we are ready to present the lemma, which we do below.

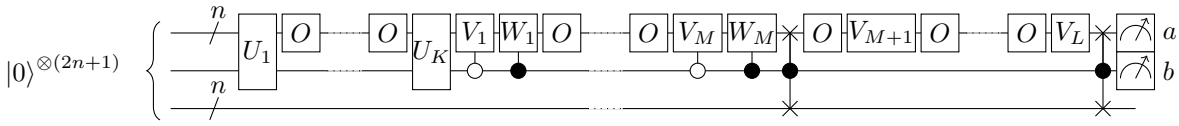
Lemma C.1.1: Principle of deferred measurement - induction step
Suppose that we have a quantum algorithm, acting on $n+1$ qubits with $n \in \mathbb{N}$, which makes queries to an oracle O' acting on $m \leq n$ qubits. We denote:

$$O = O' \otimes I_2^{\otimes(n-m)}$$

and hence, O acts on n qubits. Suppose now that the quantum algorithm performs one intermediate measurement of the $(n+1)$st qubit in the computational basis, and bases the circuit that is applied after the measurement on the outcome of the measurement. Without loss of generality, we can say that it starts in the state $|0\rangle^{\otimes(n+1)}$, and first executes the circuits U_1 through U_K, alternating with $O \otimes I_2$ before the measurement. Then, if the measurement returned 0, we can similarly say without loss of generality that it applies the circuits V_1 through V_L, alternating with O, and otherwise it applies W_1 through W_M, alternating with O, to the remaining n qubits, before measuring them. We can graphically depict this algorithm as follows:



Then, assuming that $L \geq M$, the following algorithm, using $2n+1$ qubits, has the same query complexity of O and the outcomes of the measurements have the same probability distribution:



If $M \geq L$, one can do a similar construction. If $M = L$, we do not need any auxiliary qubits at all. If we have access to a controlled version of O, we only have to add the number of auxiliary qubits required to implement this controlled oracle query.

Proof. First of all, we prove that we can without loss of generality assume that the oracle O' acts on the first m qubits. If it does not, then we can always conjugate the oracle with some appropriate SWAP-gates, i.e., directly precede and succeed it with appropriate SWAP-gates, such that it can be applied to the first m qubits. If we now summarize all unitary operations that are being performed in between the oracle queries into one unitary operation, then we obtain the form $U_K(O \otimes I_2) \cdots (O \otimes I_2)U_1$. Finally, we can absorb a circuit setting up the initial state from $|0\rangle^{\otimes(n+1)}$ in U_1, which completes the proof of the first claim in the theorem.

Next, we prove that the probability distributions of the measurements are equal for both algorithms. To that end, we first of all focus on the first circuit. Let $|\psi\rangle$ be the state prior to the intermediate measurement. Then, we find:

$$|\psi\rangle = U_K(O \otimes I_2) \cdots (O \otimes I_2)U_1 \, |0\rangle^{\otimes(n+1)}$$

Thus, we obtain the following probability distribution of the outcomes of the intermediate measurement:

$$\mathbb{P}[b=0] = \left\| \left(I_2^{\otimes n} \otimes |0\rangle\langle 0|\right) |\psi\rangle \right\|^2 \qquad \text{and} \qquad \mathbb{P}[b=1] = \left\| \left(I_2^{\otimes n} \otimes |1\rangle\langle 1|\right) |\psi\rangle \right\|^2$$

Moreover, given the value of b, the initial state for the next part of the algorithm, denoted by $\left|\psi^{(b)}\right\rangle$ is:

$$\left|\psi^{(b)}\right\rangle = \frac{\left(I_2^{\otimes n} \otimes |b\rangle\langle b|\right) |\psi\rangle}{\left\| \left(I_2^{\otimes n} \otimes |b\rangle\langle b|\right) |\psi\rangle \right\|}$$

And hence, we find that the state before the final measurement, given the value of b, which we denote by $\left|\psi_f^{(b)}\right\rangle$, is given by:

$$\left|\psi_f^{(b)}\right\rangle = \begin{cases} \frac{(V_L\otimes I_2)(O\otimes I_2)\cdots(O\otimes I_2)(V_1\otimes I_2)(I_2^{\otimes n}\otimes|0\rangle\langle 0|)|\psi\rangle}{\left\|\left(I_2^{\otimes n}\otimes|0\rangle\langle 0|\right)|\psi\rangle\right\|}, & \text{if } b=0 \\ \frac{(W_M\otimes I_2)(O\otimes I_2)\cdots(O\otimes I_2)(W_1\otimes I_2)(I_2^{\otimes n}\otimes|1\rangle\langle 1|)|\psi\rangle}{\left\|\left(I_2^{\otimes n}\otimes|1\rangle\langle 1|\right)|\psi\rangle\right\|}, & \text{if } b=1 \end{cases} = \begin{cases} \frac{((V_L O\cdots OV_1)\otimes|0\rangle\langle 0|)|\psi\rangle}{\left\|\left(I_2^{\otimes n}\otimes|0\rangle\langle 0|\right)|\psi\rangle\right\|}, & \text{if } b=0 \\ \frac{((W_M O\cdots OW_1)\otimes|1\rangle\langle 1|)|\psi\rangle}{\left\|\left(I_2^{\otimes n}\otimes|1\rangle\langle 1|\right)|\psi\rangle\right\|}, & \text{if } b=1 \end{cases}$$

And so, the probability of measuring some output $a_0 \in \{0,\ldots,2^n-1\}$, given some value for b, can be calculated as follows:

$$\mathbb{P}\left[a=a_0 \mid b\right] = \left\|\left(|a_0\rangle\langle a_0|\otimes I_2\right)\left|\psi_f^{(b)}\right\rangle\right\|^2 = \begin{cases} \frac{\|(|a_0\rangle\langle a_0|\otimes|0\rangle\langle 0|)((V_L O\cdots OV_1)\otimes I_2)|\psi\rangle\|^2}{\left\|\left(I_2^{\otimes n}\otimes|0\rangle\langle 0|\right)|\psi\rangle\right\|^2}, & \text{if } b=0 \\ \frac{\|(|a_0\rangle\langle a_0|\otimes|1\rangle\langle 1|)((W_M O\cdots OW_1)\otimes I_2)|\psi\rangle\|^2}{\left\|\left(I_2^{\otimes n}\otimes|1\rangle\langle 1|\right)|\psi\rangle\right\|^2}, & \text{if } b=1 \end{cases}$$

Hence, we can write down our final probability distribution as:

$$\mathbb{P}[a,b] = \mathbb{P}[a|b]\mathbb{P}[b] = \begin{cases} \left\|\left(|a\rangle\langle a|\otimes|0\rangle\langle 0|\right)\left((V_L O\cdots OV_1)\otimes I_2\right)|\psi\rangle\right\|^2, & \text{if } b=0 \\ \left\|\left(|a\rangle\langle a|\otimes|1\rangle\langle 1|\right)\left((W_M O\cdots OW_1)\otimes I_2\right)|\psi\rangle\right\|^2, & \text{if } b=1 \end{cases}$$

Now, we have to check that the probability distribution of the measurement outcomes of the second algorithm is equal to the one of the first algorithm. To that end, we track the state through the second algorithm. Again, we have that after the application of the circuit U_K, the state of the system is:

$$(U_K\otimes I_2^{\otimes n})(O\otimes I_2^{\otimes(n+1)})\cdots(O\otimes I_2^{\otimes(n+1)})(U_1\otimes I_2^{\otimes n})\left|0\right\rangle^{\otimes(2n+1)} = |\psi\rangle\otimes|0\rangle^{\otimes n}$$

We rewrite $|\psi\rangle$ as follows:

$$|\psi\rangle = \left(\left(I_2^{\otimes n}\otimes\langle 0|\right)|\psi\rangle\right)\otimes|0\rangle + \left(\left(I_2^{\otimes n}\otimes\langle 1|\right)|\psi\rangle\right)\otimes|1\rangle = |\psi_0\rangle\otimes|0\rangle + |\psi_1\rangle\otimes|1\rangle$$

where $|\psi_b\rangle = \left(I_2^{\otimes n}\otimes\langle b|\right)|\psi\rangle$ are not necessarily normalized, but we do have $\||\psi_0\rangle\|^2 + \||\psi_1\rangle\|^2 = 1$. Now, the application of V_1 controlled on the $(n+1)$st qubit being in state $|0\rangle$ brings the system in the following state:

$$\left[\left(V_1\left|\psi_0\right\rangle\right)\otimes|0\rangle + |\psi_1\rangle\otimes|1\rangle\right]\otimes|0\rangle^{\otimes n}$$

Afterwards, we apply the circuit W_1 controlled on the $(n+1)$st qubit being in the state $|1\rangle$. This yields the following state of the system:

$$\left[\left(V_1\left|\psi_0\right\rangle\right)\otimes|0\rangle + \left(W_1\left|\psi_1\right\rangle\right)\otimes|1\rangle\right]\otimes|0\rangle^{\otimes n}$$

Next, we apply O to the first n qubits. Hence, we obtain the following state of the system:

$$\left[\left(OV_1\left|\psi_0\right\rangle\right)\otimes|0\rangle + \left(OW_1\left|\psi_1\right\rangle\right)\otimes|1\rangle\right]\otimes|0\rangle^{\otimes n}$$

Iterating this procedure, we find that after the application of the controlled version of W_M, we have:

$$\left[\left(V_M O\cdots OV_1\left|\psi_0\right\rangle\right)\otimes|0\rangle + \left(W_M O\cdots OW_1\left|\psi_1\right\rangle\right)\otimes|1\rangle\right]\otimes|0\rangle^{\otimes n}$$

Now, we implement the controlled SWAP, which yields the following state:

$$\left(V_M O\cdots OV_1\left|\psi_0\right\rangle\right)\otimes|0\rangle\otimes|0\rangle^{\otimes n} + |0\rangle^{\otimes n}\otimes|1\rangle\otimes\left(W_M O\cdots OW_1\left|\psi_1\right\rangle\right)$$

Now, the remaining applications of the oracle O and the circuits $V_{M+1},\ldots,V_L$ yield the following state:

$$\left(V_L O\cdots OV_1\left|\psi_0\right\rangle\right)\otimes|0\rangle\otimes|0\rangle^{\otimes n} + \left(V_L O\cdots V_{M+1}O\left|0\right\rangle^{\otimes n}\right)\otimes|1\rangle\otimes\left(W_M O\cdots OW_1\left|\psi_1\right\rangle\right)$$

And finally, the remaining controlled SWAP brings the system into the final state before measurement:

$$(V_L O \cdots O V_1 |\psi_0\rangle) \otimes |0\rangle \otimes |0\rangle^{\otimes n} + (W_M O \cdots O W_1 |\psi_1\rangle) \otimes |1\rangle \otimes (V_L O \cdots V_{M+1} O |0\rangle^{\otimes n})$$

So, we obtain:

$$\mathbb{P}[a,b] = \left\| (|a\rangle\langle a| \otimes |b\rangle\langle b| \otimes I_2^{\otimes n}) |\psi_f\rangle \right\|^2 = \begin{cases} \left\| |a\rangle\langle a| V_L O \cdots O V_1 |\psi_0\rangle \right\|^2, & \text{if } b = 0 \\ \left\| |a\rangle\langle a| W_M O \cdots O W_1 |\psi_1\rangle \right\|^2, & \text{if } b = 1 \end{cases}$$

Substituting back for $|\psi_0\rangle$ and $|\psi_1\rangle$, we obtain:

$$\mathbb{P}[a,b] = \begin{cases} \left\| (|a\rangle\langle a| \otimes |0\rangle\langle 0|)((V_L O \cdots O V_1) \otimes I_2) |\psi\rangle \right\|^2, & \text{if } b = 0 \\ \left\| (|a\rangle\langle a| \otimes |1\rangle\langle 1|)((W_M O \cdots O W_1) \otimes I_2) |\psi\rangle \right\|^2, & \text{if } b = 1 \end{cases}$$

Hence, we obtain that both probabilities are equal, which completes the second part of the proof.

If $M \geq L$, the construction can very easily be modified, in several ways. We could perform the SWAP operation controlled on the $(n+1)$st qubit being in the state $|0\rangle$ instead of $|1\rangle$, and replace the circuits V_j between the controlled SWAP's with W_j's. Or, we could apply X-gates to the $(n+1)$st qubit after the circuit U_K and just before the final measurement, and reverse the roles of V and W in between these two X-gates.

If $M = L$, we can see from the above constructions that the two SWAP-operations are executed directly after each other, rendering them redundant. Hence, we can simply remove them, removing the need for the n auxiliary qubits as well.

Finally, if we have access to a controlled version of O, there is no need to use the SWAP-gates, because we can simply execute all the final circuits V_j and oracle calls O controlled on the $(n+1)$st qubit being in state $|0\rangle$. This completes the proof. □

Theorem C.1.2: Principle of deferred measurement
Suppose that we have a quantum algorithm which queries an oracle O, and which performs intermediate measurements. Then, we can construct a quantum algorithm with equal success probability, with the same worst-case query complexity to O, and without any intermediate measurements.

Proof. First of all, we start with the last intermediate measurement. We can move it all the way to the end using the construction used in Lemma C.1.1. Then, we have a new quantum algorithm, with 1 fewer intermediate measurements. This procedure, we can repeat until there are no intermediate measurements left. Moreover, the probabilities of the final measurements equal the ones of the original algorithm, as proven in Lemma C.1.1, and hence the success probability of the algorithm is left unchanged. This completes the proof. □

Note that the construction presented in Lemma C.1.1 essentially doubles the number of qubits necessary to run the algorithm, every time an intermediate measurement is removed. So, intermediate measurements can still be very beneficial in keeping the number of qubits required small, but the point of this section is that it can never be used to reduce the number of queries necessary to an oracle circuit.

C.2 Hybrid method

Now that we have introduced the principle of deferred measurement, we can use it to prove the hybrid method. This is what we do below.

Theorem C.2.1: Hybrid method
Suppose that $O_0, O_1, \ldots, O_N$ are unitary operators acting on the n-qubit state space. Let $\mathcal{A}$ be a quantum algorithm whose input consists of one of the oracles O_j, with $j \in \{0, 1, \ldots, N\}$. For all $j \in \{1, \ldots, N\}$, let R_j, R_j^* be sets of outcomes of the algorithm $\mathcal{A}$ such that $R_j \cap R_j^* = \varnothing$. Suppose that for all $j \in \{1, \ldots, N\}$, we have:

$$\mathbb{P}(\mathcal{A}(O_j) \in R_j) \geq \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}(\mathcal{A}(O_0) \in R_j^*) \geq \frac{2}{3}$$

Then the worst case query complexity of $\mathcal{A}$ to the input oracle, denoted by $T_{\mathcal{A}}$, satisfies:

$$\frac{N}{9} \leq T_{\mathcal{A}}^2 \cdot \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^{n+1}} \\ \||\psi\rangle\| = 1}} \sum_{j=1}^{N} \|(O_j - O_0)\,|\psi\rangle\|^2$$

Proof. Using Theorem C.1.2, without loss of generality we can assume that $\mathcal{A}$ consists of one quantum circuit, acting on say $m \in \mathbb{N}$ qubits, a measurement in the computational basis, and a classical post-processing step. Without loss of generality, we can also assume that the oracle circuit supplied to the algorithm, which we refer to as O, is only applied to the first n qubits, because we can conjugate it with appropriate SWAP-circuits to move it to the first n qubits if necessary. Hence, we can write the action of this circuit as:

$$U_{T_{\mathcal{A}}}(O \otimes I_2^{\otimes(m-n)})U_{T_{\mathcal{A}}-1} \cdots U_1(O \otimes I_2^{\otimes(m-n)})U_0$$

where $U_0, U_1, \ldots, U_{T_{\mathcal{A}}}$ are quantum circuits acting on m qubits that do not query the oracle circuit O.

Suppose that the initial state of the algorithm is $|\psi\rangle \in \mathbb{C}^{2^n}$. For all $j \in \{0, \ldots, N\}$, we define the shorthand notation $O_j' = O_j \otimes I_2^{\otimes(m-n)}$, and we define the final states prior to measurement of the algorithm:

$$|\psi_j\rangle = U_{T_{\mathcal{A}}} O_j' U_{T_{\mathcal{A}}-1} \cdots U_1 O_j' U_0\,|\psi\rangle$$

Moreover, for all $j \in \{1, \ldots, N\}$, let A_j be the random variable that has the probability distribution of the outcome of the final measurement, when the input is the oracle circuit O_j. Note that we have, for all $j \in \{1, \ldots, N\}$:

$$\mathbb{P}\left(\mathcal{A}(O_0) \in R_j^*\right) \geq \frac{2}{3}, \qquad \mathbb{P}\left(\mathcal{A}(O_j) \in R_j\right) \geq \frac{2}{3} \qquad \text{and} \qquad R_j \cap R_j^* = \varnothing$$

Hence, there are corresponding sets S_j^* and S_j which are subsets of the space of possible outcomes of the final measurement, such that:

$$\mathbb{P}\left(A_0 \in S_j^*\right) \geq \frac{2}{3}, \qquad \mathbb{P}\left(A_j \in S_j\right) \geq \frac{2}{3} \qquad \text{and} \qquad S_j \cap S_j^* = \varnothing$$

In other words, we obtain:

$$\mathbb{P}\left(A_0 \in S_j^*\right) = \left\|\text{proj}_{S_j^*}\,|\psi_0\rangle\right\|^2 \geq \frac{2}{3} \qquad \text{and} \qquad \mathbb{P}\left(A_j \in S_j\right) = \left\|\text{proj}_{S_j}\,|\psi_j\rangle\right\|^2 \geq \frac{2}{3}$$

And so, as S_j^* and S_j are disjoint:

$$\begin{aligned}
\||\psi_j\rangle - |\psi_0\rangle\|^2 &\geq \left\|\left(\text{proj}_{S_j^*} + \text{proj}_{S_j}\right)(|\psi_j\rangle - |\psi_0\rangle)\right\|^2 = \left\|\text{proj}_{S_j^*}(|\psi_j\rangle - |\psi_0\rangle)\right\|^2 + \left\|\text{proj}_{S_j}(|\psi_j\rangle - |\psi_0\rangle)\right\|^2 \\
&\geq \left(\left\|\text{proj}_{S_j^*}\,|\psi_0\rangle\right\| - \left\|\text{proj}_{S_j^*}\,|\psi_j\rangle\right\|\right)^2 + \left(\left\|\text{proj}_{S_j}\,|\psi_j\rangle\right\| - \left\|\text{proj}_{S_j}\,|\psi_0\rangle\right\|\right)^2 \\
&\geq \left(\sqrt{\frac{2}{3}} - \sqrt{\frac{1}{3}}\right)^2 + \left(\sqrt{\frac{2}{3}} - \sqrt{\frac{1}{3}}\right)^2 = 2\left(\frac{2}{3} + \frac{1}{3} - 2 \cdot \frac{\sqrt{2}}{3}\right) = 2 - \frac{4\sqrt{2}}{3} > \frac{1}{9}
\end{aligned}$$

On the other hand, we define for all $j \in \{0, \dots, N\}$ and $t \in \{0, 1, \dots, T_{\mathcal{A}}\}$:

$$\left|\psi_j^{(t)}\right\rangle = U_t O_j' U_{t-1} \cdots U_1 O_j' U_0 \left|\psi\right\rangle$$

We find, for all $j \in \{1, \dots, N\}$:

$$\left\| \left|\psi_j^{(0)}\right\rangle - \left|\psi_0^{(0)}\right\rangle \right\| = \left\| U_0 \left|\psi\right\rangle - U_0 \left|\psi\right\rangle \right\| = 0$$

Moreover, for all $j \in \{1, \dots, N\}$ and $t \in \{1, \dots, T_{\mathcal{A}}\}$, we find using the triangle inequality:

$$\begin{aligned}
\left\| \left|\psi_j^{(t)}\right\rangle - \left|\psi_0^{(t)}\right\rangle \right\| &= \left\| U_t O_j' \left|\psi_j^{(t-1)}\right\rangle - U_t O_0' \left|\psi_0^{(t-1)}\right\rangle \right\| = \left\| O_j' \left|\psi_j^{(t-1)}\right\rangle - O_0' \left|\psi_0^{(t-1)}\right\rangle \right\| \\
&= \left\| O_j' \left(\left|\psi_j^{(t-1)}\right\rangle - \left|\psi_0^{(t-1)}\right\rangle \right) + \left(O_j' - O_0'\right) \left|\psi_0^{(t-1)}\right\rangle \right\| \\
&\leq \left\| O_j' \left(\left|\psi_j^{(t-1)}\right\rangle - \left|\psi_0^{(t-1)}\right\rangle \right) \right\| + \left\| \left(O_j' - O_0'\right) \left|\psi_0^{(t-1)}\right\rangle \right\| \\
&= \left\| \left|\psi_j^{(t-1)}\right\rangle - \left|\psi_0^{(t-1)}\right\rangle \right\| + \left\| \left(O_j' - O_0'\right) \left|\psi_0^{(t-1)}\right\rangle \right\|
\end{aligned}$$

Hence, by induction, we obtain:

$$\left\| \left|\psi_j\right\rangle - \left|\psi_0\right\rangle \right\| \leq \sum_{t=1}^{T_{\mathcal{A}}} \left\| \left(O_j' - O_0'\right) \left|\psi_0^{(t-1)}\right\rangle \right\|$$

Squaring on both sides and using Cauchy-Schwarz's inequality yields:

$$\frac{1}{9} \leq \left\| \left|\psi_j\right\rangle - \left|\psi_0\right\rangle \right\|^2 \leq \left(\sum_{t=1}^{T_{\mathcal{A}}} \left\| \left(O_j' - O_0'\right) \left|\psi_j^{(t-1)}\right\rangle \right\| \right)^2 \leq T_{\mathcal{A}} \cdot \sum_{t=1}^{T_{\mathcal{A}}} \left\| \left(O_j' - O_0'\right) \left|\psi_j^{(t-1)}\right\rangle \right\|^2$$

This holds for all $j \in \{1, \dots, N\}$, and hence we can sum the above expression over j. This implies:

$$\frac{N}{9} \leq T_{\mathcal{A}} \sum_{t=1}^{T_{\mathcal{A}}} \sum_{j=1}^{N} \left\| \left(O_j' - O_0'\right) \left|\psi_j^{(t-1)}\right\rangle \right\|^2 \leq T_{\mathcal{A}}^2 \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^m} \\ \| |\psi\rangle \| = 1}} \sum_{j=1}^{N} \left\| \left(O_j' - O_0'\right) \left|\psi\right\rangle \right\|^2$$

Finally, let's take $|\psi\rangle \in \mathbb{C}^{2^m}$ arbitrarily. Using the Schmidt decomposition, we can find some $M \in \mathbb{N}$, $(\alpha_k)_{k=1}^M$ in $\mathbb{C}$ and orthonormal vectors $(|\phi_k\rangle)_{k=1}^M$ in $\mathbb{C}^{2^n}$ and $(|\chi_k\rangle)_{k=1}^M$ in $\mathbb{C}^{2^{n-m}}$ such that:

$$\left|\psi\right\rangle = \sum_{k=1}^{M} \alpha_k \left|\phi_k\right\rangle \otimes \left|\chi_k\right\rangle$$

Then, we find:

$$\begin{aligned}
\sum_{j=1}^{N} \left\| \left(O_j' - O_0'\right) \left|\psi\right\rangle \right\|^2 &= \sum_{j=1}^{N} \left\| \sum_{k=1}^{M} \alpha_k \left(O_j - O_0\right) \left|\phi_k\right\rangle \otimes \left|\chi_k\right\rangle \right\|^2 = \sum_{j=1}^{N} \sum_{k=1}^{M} |\alpha_k|^2 \cdot \left\| \left(O_j - O_0\right) \left|\phi_k\right\rangle \right\|^2 \\
&\leq \sup_{\substack{|\phi\rangle \in \mathbb{C}^{2^n} \\ \| |\phi\rangle \| = 1}} \sum_{j=1}^{N} \left\| \left(O_j - O_0\right) \left|\phi\right\rangle \right\|^2 \cdot \sum_{k=1}^{N} |\alpha_k|^2 = \sup_{\substack{|\phi\rangle \in \mathbb{C}^{2^n} \\ \| |\phi\rangle \| = 1}} \sum_{j=1}^{N} \left\| \left(O_j - O_0\right) \left|\phi\right\rangle \right\|^2
\end{aligned}$$

And so, we obtain:

$$\frac{N}{9} \leq T_{\mathcal{A}}^2 \sup_{\substack{|\psi\rangle \in \mathbb{C}^{2^n} \\ \| |\psi\rangle \| = 1}} \sum_{j=1}^{N} \left\| \left(O_j - O_0\right) \left|\phi\right\rangle \right\|^2$$

This completes the proof. □

This completes our discussion of the hybrid method.